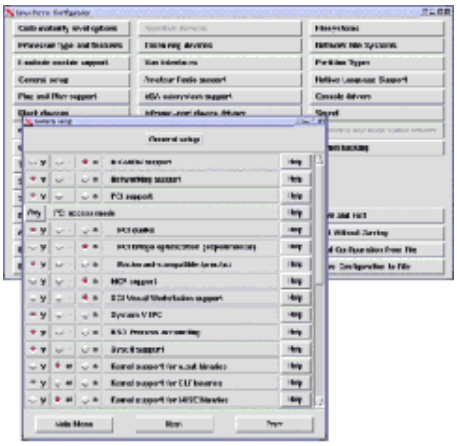
LinuxFocus article number 252
http://linuxfocus.org

by Philip de Groot
<philipg/at/authors.linuxfocus.org>

Compile your own Linux kernel



Abstract:

You will presently find out that you too can get, configure, compile, and install your very own kernel.

About the author:

I expect to receive my PhD from the Universiteit van Nijmegen during this year. My thesis is on what we call chemometrics. I'm presently working on computer technology applied to biology, at the Academic Medical Center in Amsterdam, the Netherlands. In addition I maintain my own page for Linux newbies (in Dutch): it is one of the many initiatives by and for the linux community. I love working with Linux and I frequently report on my experiences.

Translated to English by:
Nino R. Pereira
<pereira/at/speakeasy.org>

Introduction

Why would you even want to compile and install a new kernel all your own? Possible reasons are:

- The new kernel has better hardware support.
- The new kernel offers certain advantages, such as better support for multiple-processor machines

(SMP), or support for the USB. This applies to the 2.4.x kernels.
- The new kernel lacks old bugs.
- Your own kernel lacks superfluous elements and is therefore faster and more stable.

It is a problem that compiling ("rolling") your own kernel demands a fair amount of computer savvy. Therefore a new Linux user will not attempt to get into compiling kernels lightly. This article shows screen dumps of the way to do compile the kernel using the command 'make xconfig'. With this command the user handles the kernel through a GUI, a Graphical User Interface, and the mouse. There are about 40 screen dumps, which clarify why you do or do not choose certain options in particular situations. Discussing these 40 screen dumps may seem excessive, but it is the best way to clarify the kernel's internal workings and the how and why of certain kernel options. The screen dumps are based on kernel-2.4.6. The newest kernel is now 2.4.19, but apart from a few extra entries in the menus (e.g., support for new hardware) the screen shots are the same and the process of compiling the kernel is too. One word of advice: print this page out before you start, so that you always have access to all the necessary information!

The article is structured as follows. First it discusses where you can find the source code on the Internet, and how to install the source code, followed by the kernel's graphical configuration using the screen dumps. Once the kernel is configured it must be compiled, but even a newly compiled kernel is not yet ready for use. First, the new kernel must be installed with the boot manager 'lilo', and before using 'lilo' you must make the file '/etc/lilo.conf'. You can also copy the compiled kernel to a partition from where you can start Linux with a DOS/Windows program called 'loadlin'. In addition there are a slew of specific points that need to be addressed, such as PCMCIA-support as needed by laptops. PCMCIAs, small inserts that often handle networking tasks and look like fat credit cards, are supported by the kernel itself only since the 2.4.x series kernels. Older kernels can support PCMCIA through a separate compilation and installation. SuSE linux has another problem, namely support for sound through the ALSA drivers. These drivers are not part of the kernel and must be separately compiled and installed, because the original drivers usually no longer work. To make matters worse, going from one series kernel to another, say from the 2.2 series to the 2.4 series, may be accompanied by problems with certain kernel utilities, the so-called 'modutils'. These contain the code needed to load a kernel module: Figure 3 explains what a module is. Sometimes the new kernel does not know what to do with the old 'modutils', so that you must compile and install a more recent modutils version. Problems like these are rare but they do occur, and it is only fair to point them out in advance.

But, if you faithfully follow the procedures in this article there is almost nothing that can go wrong. The new kernel is added to 'lilo', or copied to the 'loadlin' partition. Therefore, in emergencies you can still restart the original kernel. Then, working under your original kernel you can try to solve the problem with the new kernel. Even when you might have difficulties with a new kernel's 'modutils' it is still possible to restart the old kernel and then fix the problem by e.g., compiling and installing them separately: all new versions of 'modutils' are downwards compatible with the older kernels, so that the new 'modutils' work fine with the old kernels.

Installing the kernel source code

Everything you do here demands root privileges, so you must begin by logging in as root. First and foremost you must install the kernel source code, e.g., from the installation CD. In SuSE the source is

located in part ’d’ (for ’development’) as the packet ’lx_kernel’. It is advisable to install the kernel source code that comes with your distribution, because the various GUIs are then automatically installed too. Once this is done the tarball with the newest Linux kernel, e.g.,the file ’linux-2.4.6.tar.bz2’ can be downloaded from (http://www.kernel.org/pub/linux/kernel/v2.4/) and installed. The corresponding modutils are on http://www.kernel.org/pub/linux/utils/kernel/modutils/v2.4/. Notice that the version numbers of ’modutils’ do not have to agree with those of the kernel: just download the most recent versions. Compiling and installing modutils is discussed later on, under the heading ’Installing modutils’. First we’ll do the kernel itself.

The source code for the kernel that is on your machine right now is in the directory ’/usr/src/linux/’.It is wise to keep this particular source code safe, for example by renaming the linux directory as follows:

```
cd /usr/src
mv linux linux-2.2.19 (if the original source code is for 2.2.19).
```

Only after you have safely stored the original kernel should you unpack the newest kernel: you will see that the file linux-2.4.6.tar.bz2 unpacks everything in the ’linux’ directory by default. It is overwritten if this directory already exists, and then you have problems: you can no longer recompile the original kernel, you lost the original configuration, etc. In this example I rename the ’linux’ directory right after unpacking the kernel source code ’linux-2.4.6’, and I create a symbolic link with the name ’linux’ to the directory ’linux-2.4.6’. The advantage of this procedure is that you can see the system’s present kernel version immediately. In addition it is easier to install a kernel upgrade. The commands are (as root, remember):

```
cd /usr/src
cp ~/linux-2.4.6.tar.bz2 ( assuming that the tarball was downloaded )
                         ( to your home-directory (’~’)              )
bzip2 -d linux-2.4.6.tar.bz2 (this may take a while                  )
tar -xvf linux-2.4.6.tar
mv linux linux-2.4.6
ln -s /usr/src/linux-2.4.6 /usr/src/linux
```

Once this is done you go the kernel’s directory, and you do:

```
cd /usr/src/linux
make xconfig (see Figure 1)
```

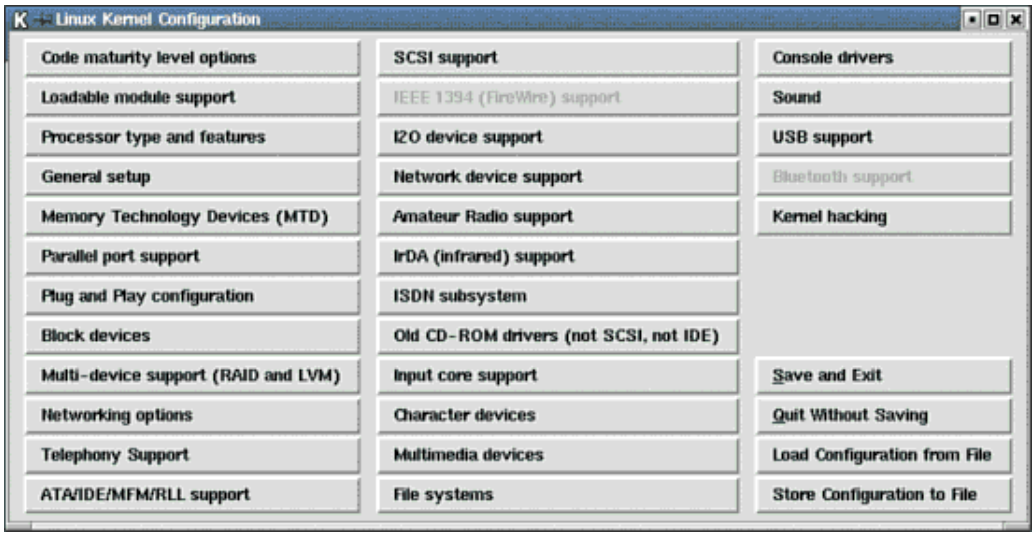


Figure 1: The graphical interface for defining your Linux kernel after the command 'make xconfig'.

This is the main menu used to define the kernel. To do so you must click on the different options. Clicking 'Save and Exit' stores your choices on disk, and once this is done you can finally compile and install the kernel (as in Figure 40). But we are not there yet.

Configuring the kernel

Below I reproduce the various screen-dumps with my selections filled in. Each picture is accompanied by an explanation as to why I have chosen my specific options. Carefully reading these examples you will understand the reasons for my choices, and you will also understand better what option you should pick in your specific situation. The 'help' feature gives similar information. You can see 'help' by doing your own 'make xconfig' for your own Linux distribution. Then, click on 'Help'. The text that appears usually recommends what option you should choose.

The examples can of course not discuss all the hardware that you might have. However, they should clarify how you handle your specific hardware, and how you can look in the kernel itself to find out if your hardware is supported.

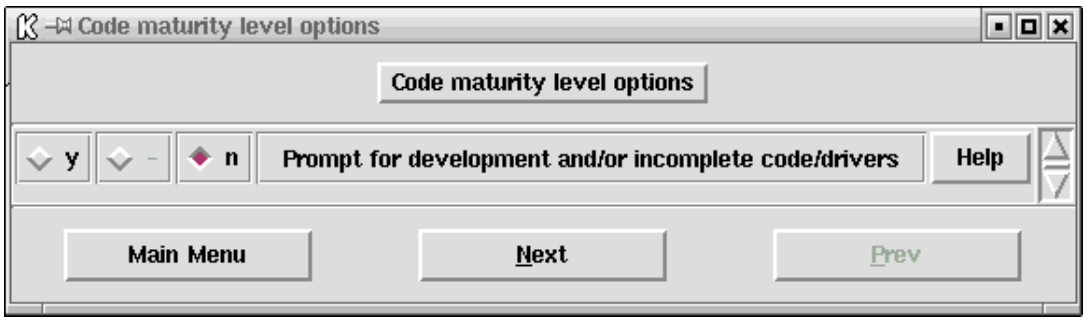


Figure 2: Selection of the 'code maturity level options'.

In this section you can allow the use of the kernel's experimental options. Sometimes these are necessary, for example to get support for new types of cards. However, in most cases you deselect this option since experimental code results in a less stable kernel. In Figure 1 you see the grayed-out options 'IEEE 1394 (FireWire) support' and 'Bluetooth support'. With the present setup you can not yet select these two because the corresponding code is still experimental.

Figure 3: Support for loadable modules.

(from now on the screen-dumps will be linked, you can open them in a new window yourself to take a look)

Modules are snippets of kernel code, e.g., drivers, that are compiled separately but ideally at the same time as the kernel itself is compiled. Therefore, this code is not part of the kernel, but it can be loaded and hence become available when you need it. The general recommendation is to compile the kernel code as a module whenever possible, because this results in a small and stable kernel. One warning: never compile the file system as a module, see Figure 32. If you make this mistake and compiled the file system as a module, the resulting kernel can not read its own file system. Then, the kernel can not even load its own configuration files, something that is obviously a prerequisite for correctly starting linux. You will see that I use modules sparingly: I like my kernel to be able to talk directly the all the hardware without having to load modules, but this is only my own preference.

Figure 4: Selecting processor type and features.

Here you pick the type of processor that you have, and you indicate whether the various options apply. In general the '/dev/cpu' options are rather advanced and should not be selected by most users. 'High Memory Support' becomes necessary only if your computer has more than 1 gigabyte of RAM (not disk space). Most computers have 64 to 512 megabytes of ram (and from 8 to 60 Gb on the hard disk), so 'High Memory Support' is usually deselected. You must turn on the option 'Math Emulation' if you use linux on a 386 of 486SX system. These older systems lack the math coprocessor that linux counts on, so in this case you must select 'Math Emulation'. Virtually all modern processors have a coprocessor built in, so you can usually keep this option turned off. The option 'MTRR' allows faster communication through a PCI or AGP bus. Since all modern systems have their videocard on a PCI or AGP bus you should usually select 'MTRR': in any case, it is always safe to turn on this option even if your system does not use the PCI or AGP bus for the video card. Symmetric multi-processing support (SMP) applies only to motherboards with more than one main processor, e.g., a motherboard with two Pentium II processors. SMP ensures that the kernel loads both processors optimally. The last option (APIC) usually applies to multiprocessor systems too, and is normally turned off.

Figure 5: General Kernel Options.

Here you specify certain general options to the kernel. Everyone always selects 'Networking support' because you always need it, e.g., for the Internet. Linux is heavily Internet-oriented and it can not run properly without networking. In addition, network support is also needed for all kinds of other actions that do not seem to have much to do with networking. It is even possible that the kernel does not compile without network support. In short: include network support. All modern systems use the PCI bus, so you select these options too. The grayed-out text 'PCMCIA/CardBus support' shows that this option is not available, because you have earlier indicated that you do not want to use experimental code (see Figure 2.). If you use a laptop you need PCMCIA/CardBus support in the kernel to allow the use of a network or a modem (see also under the heading 'pcmcia support (laptops)' later on). 'System V IPC' allows programs to communicate and to synchronize, 'BSD process accounting' keeps e. g., the error code when processes end, and 'Sysctl support' allows programs to modify certain kernel options without

recompiling the kernel or restarting the system. These options are usually left on. Modern linux distributions have their 'kernel core (/proc/kcore/) format' selected as 'ELF': this is the standard format of various system libraries, i.e., code snippets that are available to the system and used by programs. 'ELF' is the successor to the obsolete 'a.out' format, and similar to windows .dll files. All modern linux programs use the ELF libraries, but unfortunately some older programs also demand support of the 'a.out' format. One example is 'Word Perfect 8 for X Windows: this native X Window System/Linux application is only available in the the 'a.out' format, so that 'xwp' simply does not run without support of the 'a.out' format. Include 'a.out' as a module if you might want to use 'xwp'. I also carry 'MISC' as a module. In principle I do not use it, but it comes in handy to have the code available if you are a frequent user of java, python, or the DOS emulator DOSEMU. I have selected 'Power Management support' and 'Advanced Power Management BIOS support' (not shown in Figure 5). These two options are the minimum needed with modern ATX motherboards to allow the kernel to turn off the computer automatically when closing linux. The other power management functions are turned off because they do not normally work under XFree 86 (which is my standard when using linux). KDE and Gnome have their own standard power management functions that you can select.

Figure 6: Configuring Memory Technology Devices.
You need this option to make linux read for example flash cards. Flash cards are frequently used in digital cameras. With this option linux can read flash cards (from the necessary hardware) and copy the photos as .jpg files to disk. Unless you know you need it I would deselect this option: if you find you need it you can always add it later on

Figure 7: Configuring the parallel port.
Before USB technology existed the parallel port was commonly used to connect the computer to printers and scanners. My printer hangs off a parallel port, so I want the port to be available under linux. Note that configuring the parallel port is not the same as configuring printing: this is done later on, at Figure 28.

Figure 8: Configuring Plug & Play.
Almost everyone has a 'Plug & Play' system and therefore wants this option supported. Turning this on allows the kernel to configure the 'Plug & Play' devices and to make them available to the system. Sometimes it is necessary to turn on the option 'Plug & Play OS' in the BIOS, because otherwise linux (and also Windows) can not configure the 'Plug & Play' devices. The option 'ISA Plug & Play support' refers to ISA cards that are 'Plug & Play' but use the ISA bus. One example is the AWE64 sound blaster. The ISA bus never had a 'Plug & Play' standard, which makes it difficult to configure these cards. Earlier, before kernel 2.4.x, linux users had to call the program 'isapnp' (package isapnptools, rpm -qil isapnptools to see all files) during the boot process. 'isapnp' read the file '/etc/isapnp.conf'. This file contained all the ports, addresses and interrupts used by the various cards. If the information in '/etc/isapnp.conf was not correct, or if 'isapnp' was not called, the card was not accessible to linux and the modem, network card or sound card did not work. Selecting the option 'ISA Plug & Play support' supersedes the old procedure: the file '/etc/isapnp.conf' is no longer used. Instead, the configurations are detected automatically. Under SuSE 7.1 I had to rename the file '/etc/isapnp.conf' to e.g., '/etc/isapnp.conf.old' after compiling 2.4.x, because both the kernel as 'isapnp' claimed the same resources, with disastrous consequences. The problem is that SuSE 7.1 (and older versions) automatically activates 'isapnp' during boot, even though the kernel already contains the necessary support. This is however only relevant for older Linux systems, newer ones don't use isapnp by default.

Figure 9: Configuring block devices.

Virtually everyone will want to use a floppy disk, so the top option is turned on (or, in my case, selected as a module). On a request to access the floppy the kernel automatically loads the necessary module, provided that the file '/etc/modules.conf' or '/etc/conf.modules' is properly configured in your distribution as is normally the case. As user you should have no problems with this if you have selected the correct options in Figure 3. To access the floppy the kernel must of course be able to read the floppy's file system. Therefore you must also copy Figure 32 correctly. The other options can be important if you use IDE storage media through your parallel port, but they are normally turned off. One possible exception is 'loopback device support'. Before burning CDs under linux you usually make an image of the CD, and the 'loopback device' is needed to look at the image's content. I've selected this option (5'th line from below) as a module (not shown in Figure 9).

Figure 10: Configuring multiple devices.
Small users of linux normally do not need raid or LVM support. 'Raid' implies that the system uses two or more hard disks to store the information in parallel. If one disk crashes the other keeps on going, and the system continues to work. LVM makes it possible to add a hard disk in such a way that an existing partition seems to get larger. In practice this means that you do not have to repartition or to copy a smaller partition to a larger one. Path names remain the same too. This possibility is really handy, but it is not usually needed for most small users.

Figure 11: Configuring networking options.
You need the 'Packet Socket' option to communicate with network elements without implementing a network protocol in the kernel. Here I can be brief: always select this one. Most of the other options are off, unless you need their specific support. For example, I have selected 'Network packet filtering (replaces ipchains)' because I use SuSE's standard firewall. A firewall protects your computer against attacks from the outside, e.g., through the Internet, at least when you have configured the firewall properly. Firewall protection at kernel level is obviously very advantageous. Further choices in configuring 'network packet filtering' are explained in Figure 12. You need 'Unix domain sockets' to make network connections, but also elsewhere: The X Window System automatically uses Unix sockets, so that you can not use X without this option. Always turn this one on. 'TCP/IP networking' contains the protocols needed for the Internet, and also for internal networks. Normally you want to activate TCP/IP support. In case of doubt about selecting a particular option, try the help texts. If you still don't know it is always possible to include the support, and then remove it later on during testing. Compiling certain options as modules is also a good possibility.

Figure 12: Configuring the IP netfilter (firewall).
For proper operation of its firewall SuSE Linux demands backwards support for ipchains. Hence, for SuSE I select this option. If you use a firewall in other distributions or installations, consult their manual.

Figure 13: Configuring telephony support.
This is only needed if you have a telephone card in your computer, e.g., to make phone calls over the Internet. Most small users do not have this card, and do not need the option.

Figure 14: Configuring ATA, IDE, MFM en RLL support (communication protocols for hard disks).
Almost everyone needs these protocols, with as sole exception those few with systems that have only SCSI disks but no other disk types. Therefore, most users select this option. Clicking on the line right under it gives a sub-menu with another set of options. These are discussed below. Because of their

importance there are not one but three screen shots. Fill these out very carefully: they are extremely important.

Figure 15: Configuring ATA, IDE, MFM en RLL support: screenshot 1.
The top option is needed by everyone who addresses a piece of hardware through an IDE/ATAPI interface. These include hard disks, but also tape drives, ZIP disks, and CD readers and burners. Basically, all modern computers use an IDE/ATAPI interface, hence this option is on. The option 'include IDE/ATA-2 DISK support' is needed to support the system's hard disks. Hence, this option must be turned on too, except if you have only a SCSI system.

Figure 16: Configuring ATA, IDE, MFM en RLL support: screenshot 2.
The option 'include IDE/ATAPI CDROM support' is usually selected if you have an ATAPI CDROM drive. However, ATAPI CD burners must be accessed through SCSI emulation. The SCSI-emulation can be used to access both the CD player and CD burner. However, you can run into problems if you mount your CDs througt the SCSI-emulation, such as error messages when mounting the CD, or when starting the CD player to listen to audio CDs. The best solution is to turn on both 'include IDE/ATAPI CDROM support' and 'SCSI emulation support' as shown in Figure 16. The device that needs SCSI emulation, usually the CD burner, can be defined in '/etc/lilo.conf' by adding the line 'append="hdd=ide-scsi":' this is discussed further below under the heading 'Lilo configuration.' Since I have an internal ZIP drive that communicates with the motherboard through an ATAPI interface, I have selected the option 'include IDE/ATAPI FLOPPY support.' You need the same option to access other floppy-like drives, such as an LS120 drive. Most motherboards use 'PCI IDE' to access hard disks, CDROMs and floppies, hence this option is usually turned on. Likewise the two possibilities to enable DMA. DMA gives your hardware direct access to the computer's internal memory, without the intervention of the processor. As a result the IDE disks can be accessed faster. This you want. The option 'sharing PCI IDE interrupts support' is off because usually you should not need it. True, some IDE controllers allow sharing of interrupts with another computer device, for example an exotic network card. Unfortunately, sharing IDE interrupts decreases the performance of the shared disks, so normally you want to share interrupts only when this is the only way to solve certain serious hardware problems.

Figure 17: Configuring ATA, IDE, MFM en RLL support: screenshot 3.
My motherboard has a Pentium II and an Intel chipset, so of course I want to use the specific support for this particular chipset. When you fill in your own kernel options you will see other chipsets that are not shown in Figure 17.

Figure 18: Configuring SCSI support.
If you have a SCSI card you must of course select the options that you need. The screenshot only shows the options needed for your ATAPI CD burner if you have selected 'SCSI emulation support' (Figure 16).

Figure 19: Configuring I2O device support.
You must select this option if you have an I2O interface in your computer. Most people have not, and in this case you simply switch it off.

Figure 20: Configuring network device support.
I have never been able to compile a kernel without network device support. You should therefore always select this option. You should also select the dummy driver, either as part of the kernel or as a module. Linux demands such a dummy driver even when the actual, physical network is absent, as is the case

with many home users. Even when there is a network linux uses the dummy driver frequently. In this menu you can select your type network and network card, as shown by the example in Figure 21. Note that you need to do more if you want to access the Internet through a modem: you must turn on ppp support by choosing either 'PPP support for async serial ports' (for COM ports) or 'PPP support for sync tty ports' (for fast connections through e.g., a SyncLink adapter). If you forget to do this the kernel will tell you that the ppp module does not exist, even though you have made it, an error message that does not help in finding the real problem. You can choose both compression methods without problems: if the kernel needs them they are used, otherwise not.

Figure 21: Configuring the ethernet device.
My ethernet card is a 3COM /100 MBit card that uses the 3c509/3c529 chipset. Since I do not have a physical connection with a network (I have a network card, but I'm connected to the network through a modem) I compile this driver as a module, just is case I might need the card in the future. You will of course select the type network and network card in your machine. In addition, you must configure the network connection with a linux configuration program, such as 'yast2' under SuSE.

Figure 22: Configuring amateur radio support.
You select this option if you want to use amateur radio support, and you turn on the necessary driver. Most people do not use this option.

Figure 23: Configuring support for infrared (wireless) communication.
You turn on the infrared communication option if you have a wireless device, e.g., a wireless mouse or a wireless keyboard. Most desktop systems do not have this and do not need the option.

Figure 24: Configuring ISDN support.
Here you select support for an ISDN-card that you might have in your system. It is important to know which card you have, including the chipset: you need this information to pick the right driver.

Figure 25: Configuring old CDROM drivers.
In older 486 and even 386 systems the CDROM is not connected through the hard disk IDE (ATAPI) controller, but through a sound card or a special card. Using these old CDs demands selecting the corresponding driver. This option is superseded in modern systems, and hence superfluous.

Figure 26: Configuring input core support.
This refers to one of the most important additions to the 2.4.x kernels: USB support. Input core support is a layer in between the kernel and some USB devices. Figure 38 shows the various USB devices you can select, and the help text for some of these indicates which ones need 'input core support': see Figure 38. You must turn on 'input core support' here if one of your USB devices needs it. All modern motherboards have a USB connection, so as a rule you should turn it on. But, to be honest, I know I will not need USB support in my system so I have turned it off.

Figure 27: Configuring character devices: screenshot 1.
The upper option ('virtual terminal') enables the possibility of opening an xterm (using X11) or to use the text mode for login. Normally this option is always turned on. The second option ('support for console on virtual terminal') tells the kernel where it should send messages, such as warnings about lacking or incorrectly working modules, problems with the kernel itself, and startup messages. Under X Window System you often set apart a special window for the kernel messages, but in text mode they typically go to the first virtual terminal ('CTRL+ALT+F1'). Leave this option on. You can also choose

to send these messages to the serial port, e.g., to a printer or to another terminal (the fourth option). To send the messages to the printer you must also activate the port through the third option. Likewise, you must activate this port if you want to use it for a 'serial mouse'. Again, usually the third option ('standard/generic (8250/16550 and compatible UARTs) serial support') is on. In my own system I have chosen to compile this as a module. The reason is that during startup SuSE complains about the lack of the 'serial support' module, and including the support as a module is an elegant way to avoid the message by ensuring that the module does exist. Configuring the 'character devices' is extremely important. If you fail to do this properly you can end up with a non-working system. Hence that Figures 28 to 30 discuss a few more options.

Figure 28: Configuring the 'character devices': screenshot 2.
If you want to use an xterm on your own machine from a remote site, for example through 'telnet' of 'ssh', you must turn on the option 'unix98 PTY support'. It might seem that a standalone desktop system would not need this option, but a number of processes in the background also use this option. Therefore, it is a good idea to activate this option in any case, if only to avoid error messages (at least from SuSE) during startup. Everyone who connects a printer through a parallel port needs of course 'Parallel printer support'. Still, not everyone needs the parallel port: modern USB printers do not. Kernel messages can go through the parallel printer by turning on 'Support for console on line printer': usually, you do not want this. You need the option 'support for user-space parallel port device drivers' if you have certain pieces of equipment hanging off the parallel port, but normally you do not. Likewise, usually you do not need 'I2C support': it is needed for certain cards that handle video, but if you find out you need it you can always add it to the kernel later on, once you know the kernel works fine. You select support for mouse and joy stick when you use them, but not all mice use this driver (see further below, at Figure 29). Present-day CD burners have made the tape drives that need 'QIC-02 Tape support' largely obsolete, hence this option is usually off.

Figure 29: Configuring the 'character devices': Mice.
You do not need anything from this option if you have a serial mouse, but for all other mouse types you must configure certain parameters here. If you use an ORIGINAL bus mouse you must select the upper option, with below it the corresponding type or make of the bus mouse. Many computers nowadays have another type of mouse, usually (and erroneously) called 'busmouse' or 'PS/2 mouse'. These mice are often connected to '/dev/aux' and plugged in through a small connector similar to those used for keyboards. Often this type of mouse uses the keyboard to connect to the computer. To make these mice work properly you must select the options shown in Figure 29, 'mouse support (not serial and bus mice)' and 'PS/2 mouse (aka "auxiliary device" support)'.

Figure 30: Configuring the 'character devices': screenshot 3.
The options for configuring the kernel in between those of Figure 28 and Figure 30 are not discussed here. They are normally off. The option 'Ftape, the floppy tape device driver' refers to support for tape drives c9nnected through the floppy controller. Even if you have such a tape drive it is not essential to compile support for it, at least not in the first go-around. The other options refer to modern 3D video cards. If you have a video card that is connected through an AGP-bus, you could turn on AGP support, and also the specific driver for your video card (under '/dev/agpgart (AGP support)'). Note that it is possible to have a correctly functioning kernel without these options, but no necessarily! People who have an integrated videocard on their motherport, such as in the intel i815 chipset, MUST use the kernel driver! If not, XFree86 version 4 or higher (used with most recent distributions) won't work. My system does have an AGP card, an NVidia TNT2, but this card is not supported by a specific kernel module (NVidia refuses to share hardware specifications, necessary to develop these drivers). Unfortunately, in

my case it makes therefore little sense to turn on AGP support. Despite this particular problem, I can use XFree86 version 4 without the kernel-driver. 'Direct rendering support' is for an option in XFree86 starting with version 4 to accelerate graphics performance through the kernel. To make use of this option your specific video card must be supported, and you must use XFree86 4.0 or higher. In addition you must also activate 'AGP support'. Still, you can safely deselect these options and come out with a correctly functioning linux kernel.

Figure 31: Configuring the 'multimedia devices.
This option is turned on if you have a card that handles video or radio. As before, this option is not essential for the kernel's proper functioning.

Figure 32: Configuring the 'file systems': screenshot 1.
Here you specify the file systems to be read by the linux kernel. You may want to make a kernel that can read Windows disks or Windows floppies, but you must make sure that the kernel can read linux' own ext2 file system, or the newer ReiserFS file system. Linux can not even start if you fail to do this, because then the kernel can not read from its own boot disk (as discussed earlier around Figure 3). To read DOS/Windows floppies and disks you need to turn on the option 'DOS FAT support': however, to read Windows NT/Windows 2000 disks you need a separate read-only driver that can be selected in this menu later on. To read and also to write DOS/Windows disks and diskettes you need the option 'MSDOS fs support'. Virtually everyone wants this, so most people turn these options on. 'VFAT' refers to support for the use of long file names under Windows 95 of 98. My own system is a so-called dual boot system, in which I can start both Windows 98 and linux (using the linux boot manager lilo, see under 'Configuring lilo'). Therefore I have activated 'VFAT'. You need to include support for ISO 9660 to read CDs in the standard format. Below this is the 'Joliet extensions' option, which allows longer file names than the MS-DOS 8.3 that is the limit in the ISO 9660 standard. Almost everyone wants to read present-day CDs, so these options are normally on. Figure 33 clarifies some additional options, among which is Linux' ext2 file system.

Figure 33: Configuring the file systems: screenshot 2.
The files in the '/proc' directory contain information about the status of the system, e.g., which interrupts are in use. Normally you always turn this option on. The 'Second extended fs support' refers to linux' (still) standard file system. You absolutely MUST compile this in (NOT as module)! Figures 32 and 33 do not show the 'ReiserFS' option that can be selected here too: ext2's anoited successor, ReiserFS is better able to deal with damage to the file system due to power failures and similar problems. At this moment ReiserFS is still under development and therefore marked as experimental code. Even so most recent distributions support the use of ReiserFS already, but even though ReiserFS is supposed to replace 'ext2' in the future I would not recommend it as file system for all partitions at this time. You need 'UDF file system support' if you use (under Windows) the program 'packetCD', which allows you to copy CD files on the fly as from a slow hard disk. It is very handy to exchange data with other PCs. Reading these packet CDs also under linux is possible by mounting them with the 'udf' file system, e.g., with a command like 'mount -t udf /dev/scd0 /cdrom'. This part also contains items like 'Network file systems', 'partition types' and 'Native language support'. You do not have to deal with 'Network File Systems' unless your computer is part of a large network, in which case you need to activate 'NFS File System Support' and perhaps also 'SMB file support', but for a standalone computer you do not need these options. The option 'Partition Types' is quite advanced but not necessary for effective use of the linux kernel. It is best to turn this one off. Figures 34 and 35 further explain 'Native Language Support'.

Figure 34: Configuring 'native language support': screenshot 1.
In this menu you pick which code table is to be used by linux to deal with file names under DOS and Windows. Figure 34's code tables are for the usual DOS file names. The NLS tables in Figure 35 are needed to use the longer file names. The top option in Figure 34, 'Default NLS option', determines which symbols will be standard in linux. Figure 35 depicts and explains the option 'iso8859-15'.

Figure 35: Configuring 'native language support': screenshot 2.
You need the option 'NLS ISO 8859-15' to reproduce Windows' FAT and the Joliet extensions to the CD file systems correctly, something that is always a good idea. The selection 'NLS ISO 8859-15' is appropriate for the Western languages, and it includes the Euro symbol. Therefore this code table is almost always compiled in. The table 'NLS ISO 8859-1' is the previous table for the Western languages but without the Euro symbol.

Figure 36: Configuring the console drivers.
The 'VGA text console' is to enable text mode with VGA resolution. Almost everyone wants this, so this option is almost always on. Only a few of the older 386 computers lack a VGA-compatible card, while most modern computers have not the slightest problem with this choice. The second option, 'video mode selection support', makes it possible to select the resolution in text mode during the boot process. This is sometimes handy if you want to have more letters on a line, but usually you leave this one off. The final two options are experimental, and I advise you to deselect them.

Figure 37: Sound configuration.
In this section you configure sound. If your distribution uses the ALSA sound drivers (like SuSE 6.3 and higher), it suffices to select 'sound card support' as MODULE. The ALSA drivers are compiled and linked later on (see further under the heading 'SuSE and the ALSA sound drivers'). If your distribution uses the kernel's standard sound drivers you must now select the right one for your sound card. Virtually all sound card brands are named here, so in principle the selection of the right driver is not a problem. If your sound card works well with your distribution's standard kernel you can also use configuration programs (such as SuSE's 'yast2') to find out which drivers your specific sound card needs. It is reassuring to know that sound is not critical: you lack sound if something goes wrong here, but the kernel itself works fine.

Figure 38: Configuring 'USB support'.
My motherboard has a USB port, but I do not use it. However, if I were to turn off all USB support SuSE gives me an an error message during boot. Of course SuSE supports the USB and therefore tries to load the necessary module(s), hence my selection of 'Support for USB' as a module. Even though this error message is not important for me, I solve it in an elegant way by compiling the driver needed for the USB ports on my motherboard. To do this the minimum is setting the 'Preliminary USB device filesystem' to 'y' and to load a specific USB driver. Since my Pentium II motherboard is rather old I have picked the 'UHCI (Intel PIIX4, VIA, ...)' driver as a module. But, you must select the 'UHCI Alternate Driver (JE) support' module if you have a recent motherboard with the Intel chip set, while for e.g., Compaq computers you should choose 'OHCI support' as a module. In principle you need only one of these three modules, but in case of doubt you can select all three. Your linux distribution has already found out which of those it needs, and it automatically loads the correct one.

Simply enabling your motherboard's USB ports in not enough, you need to specify also the drivers (modules) of the USB equipment that is connected to your computer. The list that comes up under 'USB Device Class drivers' has the various choices. All this is rather straightforward and little can go wrong:

still, in case of doubt read the help texts.

Figure 39: Configuring 'kernel hacking'.
This one is easy: do NOT select!. It is an option useful to programmers who want to find the reason for a kernel crash or to read the hard drive's cache: the option is completely useless for the typical user.

Figure 40: Save and Exit.
Pffft, we made it. All that remains is to compile and install the kernel as discussed below.

Compiling the kernel

By clicking on 'Save and Exit' you store your configuration choices in the file './.config' (or '/usr/src/linux/.config' if you compile in /usr/src/linux). As an aside, it is handy to save this file and to copy it to the new kernel source directory if you do only a minor upgrade, from kernel 2.4.5 to 2.4.6 for example. This way you can (usually) keep all your old configuration choices, which can save a lot of work. In a similar way you can start off by using the config file of the 'standard' kernel of your distribution, which in some distro's is called at /boot/config (copy it to ./.config to start using it). But, if you upgraded your kernel source and you get weird problems when compiling the kernel this is of course the first file you remove! As a matter of course you have, for security's sake, written down and carefully stored the configuration of the kernel that works correctly. The procedure to compile the kernel is as follows:

```
make dep
make clean (for older kernels)
make bzImage
make modules
make modules_install
```

Figure 40 already indicated the need for the 'make dep' command. Of course you execute these commands in linux' source directory, usually '/usr/src/linux'. Kernels in the 2.0.x series or older also need the command 'make clean', which removes earlier files before the compilation of a new kernel. The 'make clean' command prevented strange error messages that were difficult to track down, but were presumably caused by older object (.o) files that were not overwritten. The command 'make bzImage' compiles the new kernel, but does not yet install it. You can also compile the kernel with other 'make'

commands, e.g., ’make bzlilo’ or ’make zImage,’ but these commands can give unexpected problems. Most kernels are too large to allow a properly executed ’make zImage’: you get an error message during the compile, and you end up without a kernel. With the command ’make bzlilo’ everything must be configured properly in files such as ’/etc/lilo.conf,’ but this is not always the case. Hence it is safer to avoid these latter commands. The command ’make modules’ compiles the modules: they are installed with the command ’make modules_install’. This command puts the modules in the directory ’/lib/modules/2.4.6/’ if the present kernel version is 2.4.6: it changes when you compile another kernel version. This way the modules corresponding to a particular kernel end up automatically in a separate directory, thereby avoiding conflicts with obsolete modules and similar problems. During boot the linux kernel now knows in which directory it can find the right modules. But, the files in ’/lib/modules/2.4.6/’ are overwritten and older modules remain if you already compiled kernel 2.4.6 earlier and now recompile it. Then, older modules may hang around even though they are no longer needed in the newer kernel. Normally this is not a problem, but it is always a good idea to take the time to remove the older modules first before installing the new ones.

To avoid problems in installing the kernel you must also ensure that lilo’s configuration ’/etc/lilo.conf’ is correct, and you must copy the kernel and the file ’System.map’ into the correct location. After all this you must also execute the command ’lilo’ too. An alternative is the use of ’loadlin’, which allows booting a linux kernel under Dos/Windows. Both options are discussed below.

Configuring lilo

You can find lilo’s configuration file usually in the ’/etc’ directory as ’/etc/lilo.conf’. Open it in a text window with a simple ASCII editor: you may have installed a full-fledged editor like XEmacs (then, use ’xemacs /etc/lilo.conf &’), a simple full screen editor like kedit or gedit (then, use ’k(g)edit /etc/lilo.conf’) or a primitive line editor such as pico or nano (then, use ’pico /etc/lilo.conf’). The file lilo.conf will look something like this:

```
boot    = /dev/hda
vga     = normal
read-only
menu-scheme = Wg:kw:Wg:Wg
lba32
prompt
timeout = 300
message = /boot/message

  other  = /dev/hda1
  label  = win98

  image  = /boot/bzImage
  label  = linux-2.4.6
  root   = /dev/hda3
  append = "parport=0x378,7 hdd=ide-scsi"

  image  = /boot/vmlinuz.suse
  label  = suse
  root   = /dev/hda3
  append = "hdd=ide-scsi"
  initrd = /boot/initrd.suse
```

The detailed content of the file lilo.conf may well differ from the above on each system and between distributions. Therefore I will now walk you through this file. The top 8 lines are already in good shape and you need not change them, usually. The command ’boot’ in the first line indicates the physical hard disk from which booting starts, that is, ’boot’ points to the location of the ’master boot record’. In my case booting starts from /dev/hda, the first physical hard disk. The option ’vga’ indicates that the boot uses a standard VGA text mode, with 80x25 characters. The option ’read-only’ means that the booting process first mounts the linux partition read-only. During linux’ boot the partitions are checked for errors: only afterwards are they re-mounted with both read and write permissions. The line ’menu-scheme’ sets the colors of the ’lilo’ boot menu in text mode. With ’lda32’ it is possible to boot the operating system after the 1024’th cylinder, provided that this is supported in the BIOS. All moderns systems support ’lba32’. Problems with this you can solve through a BIOS upgrade, something that is almost a necessity with the large hard disks that are now available. The command ’prompt’ forces ’lilo’ to give a prompt that allows the user to choose the desired operation system. The option ’timeout’ gives the number of milliseconds that ’lilo’ waits for input after the prompt before starting the standard operating system. If ’lilo.conf’ does not give a standard operating system, as in the example here, the boot process starts the first operating system encountered. In my case this is Windows98, so that people who do not know linux yet will end up in a Windows environment. The ’message’ option shows a message while ’lilo’ is executing. Under SuSE this is Tux, linux’ cute penguin mascot, with (of course) the text ’SuSE Linux 7.1’. You can see this message by typing ’xv /boot/message’ or ’gv /boot/message’ (sometimes even ’gimp /boot/message’) : ’xv’ and ’gv’ (ghostview) are shareware programs with which you can look at various kinds of picture formats. Please note that the file /boot/message does not exist on systems without a graphical login screen (e.g. older distributions), in this case the ’boot message’ is just a text message. It is in principle possible to show your own preferred image during boot, but I have not yet tried out this possibility. All options to ’lilo’ are of course documented in the ’man’ pages, which you access by the commands ’man lilo’ and ’man lilo.conf’.

The other options direct the boot of the various operating systems. At most you can boot up sixteen different operating systems or kernels. Normally this is ample. You pick the operating system with the line ’label=’. The default for Windows98 (and also older Windows versions and DOS, but not Windows NT or Windows 2000) is to have them on the first, primary partition. Therefore these operating systems only need a line ’other’ and a line ’label’. The second section, starting with ’image=/boot/bzImage’, starts the new kernel with the label ’linux-2.4.6’. My linux root directory is ’/dev/hda3’. The line ’append = "parport=0x378,7 hdd=ide-scsi"’ tells the kernel the address and interrupt for the parallel port (port 0x378, interrupt 7) , and specifies that my CD burner ’hdd’ must be addressed through SCSI-emulation. The names for the CDs depend on the system: in mine it is ’hdd’, but in yours it can be another name. The use of an interrupt is mostly a question of personal preference. An interrupt speeds up printing, but you can leave out this command if you do not have an interrupt available for the parallel (printer) port. Linux’ default is the slower so-called ’polling’, which allows the kernel to use the parallel port without an interrupt. The last section, starting with ’image = /boot/vmlinuz.suse’, contains lilo’s configuration made during SuSE’s configuration process: I have added the line append="hdd=ide-scsi" by hand. The file ’boot/vmlinuz.suse’ is the standard kernel that comes with the distribution. Preferably, you should ALWAYS save this kernel for emergencies. The line ’initrd = /boot/initrd.suse’ applies only to the standard installation kernel: it specifies the loading of a so-called ’ramdisk’ image, a virtual disk that is loaded in memory (or random access memory, RAM). The ’ramdisk’ contains the modules needed for the correct booting of linux: a distribution kernel must of course be able to access an enormous variety of hardware, something that is only feasible by using many modules.

Hopefully it is now clear where you must store the new linux kernel before executing lilo. In this example, the right commands are:

```
cp /usr/src/linux/arch/i386/boot/bzImage /boot
cp /usr/src/linux/System.map /boot/System.map-2.4.6
lilo
```

Under SuSE 7.3 you can also do the second copy like this:

```
cp /usr/src/linux/System.map /boot
```

(the original System.map has already been renamed)
If you already have a kernel with the name ’bzImage’ and you want to keep this kernel, you can copy the new kernel to ’/boot/bzImage-2.4.6’ and make the corresponding change in /etc/lilo.conf: change /boot/bzImage to /boot/bzImage-2.4.6. The compilation always recreates the file System.map, which contains the names and the configuration of important kernel variables. The command ’depmod -a’ creates a file that contains all the dependencies of the kernel modules, that is, the various relations between the kernel and the modules, between the modules themselves, and also the information in the file ’/etc/modules.conf’. Most linux distributions, including SuSE, make sure that ’depmod -a’ is executed during boot, but it is wise to ensure explicitly that the file /boot/System.map exists, and that it corresponds to the proper kernel version. The command ’lilo’ installs the new configurations of an old kernel, or the new kernel. If you already have a file ’/boot/bzImage’ that you overwrite with a freshly compiled kernel but without executing lilo, you will end up with a new kernel that can not boot. The old, original distribution kernel that you stored safely away still works, because this one has not been overwritten. Keeping the old kernel makes it possible to compile and test new kernels in a responsible way.

Using loadlin

If you use ’loadlin’ to boot linux you undoubtedly know how to find this handy program, viz., in C:\loadlin. Virtually all linux distributions have ’loadlin’ on the first CD in the ’dosutils’ directory. You must also copy the new linux kernel to the hard disk in ’C:\loadlin’, perhaps with a unique name. By first starting Windows98 in DOS mode you can then boot linux with the command:
loadlin bzImage
Under normal circumstances most configurations that are stored in the kernel itself, such as the location of the root partition, are correct if you have compiled your own kernel, and in this case the above command is sufficient to boot linux without problems. By typing (in DOS) the command ’loadlin | more’ you will get a help screen that includes a link to a ’loadlin-HowTo’ on the Internet. With ’loadlin’ you can try out a newly compiled kernel even if you are reluctant to diddle with ’lilo’: namely, ’loadlin’ and ’bzImage together fit on a 1.44 MB diskette. There is absolutely nothing that can go wrong if you first boot DOS with a boot diskette (with support for EMM386), and then boot linux from the diskette with ’loadlin’ and ’bzImage’. But, you must of course have a DOS boot diskette.

SuSE and the ALSA sound drivers

SuSE uses the standard ALSA (Advanced Linux Sound Architecture) sound drivers. These drivers are better quality than the drivers of the OSS project that are standard in the kernel. If you are serious about sound under linux you should absolutely use the ALSA drivers. These drivers are not part of the (older) kernel source code, which implies that the drivers must be compiled and installed separately. Source code for the ALSA drivers are located in the part 'zq' of the SuSE distribution: read on if you do not know what this means. Install ALSA through YaST or YaST2 so that you have the source code available in the directory '/usr/src/packages/.' To compile and install the ALSA drivers you do the following:

```
rpm -bb /usr/src/packages/SPECS/alsa.spec
cd /usr/src/packages/BUILD/alsa/alsa-driver-<version number>/
./configure
make install
```

The first line installs the source code, including the drivers, in the directory '/usr/src/packages/BUILD/' directory. In addition the ALSA libraries and utilities are directly compiled as rpm-files. Unfortunately, the ALSA drivers are not compiled in by default. You must compile and install them separately by hand using the bottom two commands. De command './configure' finds the necessary configurations and files in your system and puts them in a configuration file. The command 'make install' compiles all ALSA drivers and installs them at the same time for use by the kernel in the directory '/lib/modules/2.4.6/misc/'. Now, when booting SuSE the desired sound driver is installed automatically. I admit that the procedure suggested here is somewhat cumbersome, but you must know exactly what you are doing to find the necessary drivers and to include sound support in a new kernel in a more direct way.

If you do not use SuSE, or if you want to use a more recent version of the ALSA drivers, you can download these drivers and the corresponding libraries and utilities from http://www.alsa-project.org. This site's start page gives the latest news on the ALSA project (e.g., February 2002's integration of the ALSA drivers in the official 2.5 series kernel source tree) and links to the various files to download. Below I will show how to compile the ALSA drivers: you can follow the analogous steps for the libraries and the utilities. Unpack the drivers in a convenient directory, e.g., '/usr/local/.' Move to this directory, in this case '/usr/local/alsa-driver-<version-number>/' and execute the above commands starting with './configure'. It is possible that you must perform some additional steps to get the drivers to work if your distribution does not use the ALSA drivers as a standard. Unfortunately, these problems are outside the scope of this already quite extensive paper, but you can get further help from the ALSA FAQ (Frequently Asked Questions) that you can download too.

Support for pcmcia (laptops)

Support for pcmcia is standard in kernels from 2.4.x onward. However, the official PCMCIA HOWTO argues that the kernel version of PCMCIA should preferably not be used. For now, the pcmcia source, including the scripts and the drivers can work with all the kernel trees: 2.0, 2.2, and 2.4. It is my experience that the pcmcia drivers stop to work after recompiling the kernel, and that they must be recompiled too. There are two solutions depending on your distribution. The first is to use the source code that comes with the distribution. Installing and compiling the source code to an rpm file gives you a file that you can install. The other solution is to download, unpack, compile, and install the most recent pcmcia version from http://sourceforge.net/projects/pcmcia-cs/), like this:

```
cp /etc/rc.d/pcmcia /etc/rc.d/pcmcia.SuSE
cp ~/pcmcia-cs-3.1.?.tar.gz /usr/src
cd /usr/src
tar -zxf ./pcmcia-cs-3.1.?.tar.gz
make config
make all
make install
cp /etc/rc.d/pcmcia.SuSE /etc/rc.d/pcmcia
```

The first and last lines solve a SuSE-specific problem. SuSE's pcmcia-initialization script '/etc/rc.d/pcmcia' is overwritten by the command 'make install', which cause the script to fail under SuSE. The problem is solved by copying the original script back after 'make install'. If you have mistakenly overwritten the original SuSE script you must re-install the pcmcia packet anew beginning with the part 'a1', copy the original script to another file, execute 'make install' again, and finally copy the original script back.

The new rpm file for pcmcia support you get under SuSE as follows:

```
rpm -i /cdrom/suse/zq1/pcmcia-3.1.?.spm
cd /usr/src/packages
rpm -bb ./SPECS/pcmcia-3.1.?.spec
cd /RPMS/i386/
rpm -i --force ./pcmcia-3.1.?.rpm
SuSEconfig
```

In the first line I assume that you will install the pcmcia drivers by themselves from the sixth or seventh CD and that the CD-ROM reader is already mounted on /cdrom. The command 'rpm -i' installs the source code and the command 'rpm -bb' compiles the pcmcia rpm file. Then you install this particular rpm file as you do all rpm files. Note that you must use the option '--force' because otherwise the rpm program will tell you (correctly) that 'pcmcia' is already installed, and will therefore ignore the new file. As always you must execute the program SuSEconfig (note the CAPITALS and lower case letters) when you have installed rpm files by hand under SuSE, to activate the configuration changes. The latter is done automatically by SuSE's setup programs YaST or YaST2 after they have installed or modified new packets. Then, it is no longer necessary to do the activation by hand.

To be able to use pcmcia support properly you must leave on 'network support' during compilation, but you must turn off all other drivers for network cards. And, as has already been discussed with Figure 11, you must of course turn on 'TCP/IP support' too if you want to use the Internet.

Installing 'modutils'

As already mentioned the kernel uses the small programs in 'modutils' to manage kernel modules. These programs include:

```
insmod (which installs a module),
rmmod  (to remove a module, and)
lsmod  (showing all modules in use),
```

among many others. With commands like ’man lsmod’ you can find out how to work the various commands, something I will not get into here.

Compiling and installing ’modutils’ is straightforward. Simply do:

```
cd /usr/src
cp ~/modutils-2.4.6.tar.bz2 . ( assuming that the file sits in your )
                              ( home directory, '~'                  )
bzip2 -d modutils-2.4.6.tar.bz2 ( unzip: this may take a while       )
tar -xvf modutils-2.4.6.tar
cd modutils-2.4.6 ( go to the directory in which 'modutils' have     )
                  ( just been unpacked                               )
./configure (find system-specific configurations                     )
make ( compile 'modutils': since it is small the compilation can be  )
     ( surprisingly fast                                             )
make install (install 'modutils' in the directory '/sbin/'           )
```

This is all you have to do to make ’modutils’ ready for use. Note again that in this example ’modutils’ happen to have the same version number as the kernel, but this is not always the case.

Does the kernel work correctly?

The new kernel is configured, compiled, and probably also installed through lilo. You restart the system and ask yourself: how can I figure out whether the new kernel will work correctly? You will easily find hardware that does not work when you try things, but in addition the kernel puts lots of useful information to the screen during the boot, even before you start the graphical login. This information contains things such as configurations that are detected automatically (ports, IRQs, etc.), but also error messages in case a certain driver can not be initialized directly, either as module or as part of the kernel. Watching these messages gives an early warning about certain problems. For SuSE linux this is very easy: each part of the kernel that boots puts on the right side of the screen the (green) message ’done’ if all worked well, or ’failed’ if something is wrong. Since the messages can scroll across the screen at a furious pace, any error messages are summarized above the login prompt. You find the login prompt under ’<Ctrl>+<Alt>+F1’ if you get the graphical login screen automatically. This way you have at least a hint as to where the problem occurs if indeed the error message is relevant to you. Error messages that look like ’Cannot find module’ or ’Cannot load module’ usually indicate that you have not included certain parts that you can not miss. Fixing the kernel configuration and recompiling usually solves the problem. Note that it is not necessary to recompile the whole kernel. If you just forgot some modules it is sufficient to re-execute ’make modules’ and ’make modules_install’. It is not even a problem if you must recompile the entire kernel. Most kernel code is already compiled, and the only parts that need to be recompiled are the parts that were missing. In summary, simply modifying the kernel a little and recompiling goes very quickly, in contrast to the initial kernel compilation. Then you can safely get some coffee.

Another way to inspect the kernel’s boot messages is with the command ’dmesg’. Simply executing this command calls up the messages that scrolled off the screen earlier. Piping them to a file, with ’dmesg > temp’, allows you to read the error messages at your own pace (with ’more temp’, or your favorite editor).

Conclusion

With this guide in hand you can now begin your experimentation with the kernel in a reasonably sensible way. Hopefully the threshold to start playing with the kernel is lowered enough that you are eager to get started. Most of your time will be spent deciding on the correct kernel configuration, while during compilation you can safely play ’freecell’ or do other computer work.

If you still get into trouble and know no way out, you could make use of the various linux-related mailing lists and web sites where you can ask questions. These exist in all different languages, not only English. A reasonably short time later you usually get a helpful answer that enables you to solve your problem. The best way to find these lists and web sites is with a search engine.

2005-01-14, generated by lfparser_pdf version 2.51