KAPITEL 8
Installation der Software

Nun ist der große Augenblick gekommen. Sie sitzen fiebrig vor dem PC, neben sich ein Stapel CDs und vielleicht das Handbuch Ihrer Distribution, und möchten loslegen. Endlich dürfen Sie mit der Installation beginnen. Normalerweise würde es jetzt genügen, von der Installations-CD zu booten und einige Fragen zu beantworten, um kurz darauf ein voll funktionsfähiges Linux vorzufinden.

Die Installation, die ich hier beschreibe, unterscheidet sich allerdings ziemlich von einer solchen Standardinstallation. Vieles, das in einer Standardinstallation mit einem Mausklick erledigt wird, muß hier mühsam mit einem Texteditor konfiguriert werden. Dies liegt daran, daß Bequemlichkeit oft mit einem Verzicht auf Sicherheit erkauft wird. Wenn Sie sich an einem System unter einer graphischen Oberfläche anmelden können, so bedeutet dies auch, daß auf ihm ein X-Server läuft, der als Netzwerkdienst auch aus dem Internet angesprochen werden kann. Wenn auf CD-ROMs durch normale Benutzer zugegriffen werden kann, so heißt dies auch, daß diese Verzeichnisse mounten können, was normalerweise aus gutem Grund `root` vorbehalten ist. Sie müssen dabei ein Programm starten können, das Rootrechte besitzt. Enthält dieses einen Programmierfehler, so kann die Situation entstehen, daß ganz normale Benutzer die komplette Kontrolle über das System übernehmen können.

Schließlich führt das hier beschriebene Vorgehen auch dazu, daß Sie Ihr System deutlich besser kennenlernen und sich so ein Verständnis erwerben, das Ihnen fehlen würde, wenn Sie nur die benutzerfreundlichen Administrationswerkzeuge benutzen.

Beginnen Sie also damit, daß Sie vor dem Boot des Rechners alle Ethernet- und ISDN-Kabel sowie alle Verbindungen zu Modems vom Rechner abziehen. So vermeiden Sie, daß Ihr Rechner aus dem Internet angreifbar wird, bevor Sie ihn fertig konfiguriert haben.

Nun können wir mit der eigentlichen Installation beginnen.

Installation der benötigten Software

Es ist an dieser Stelle nicht möglich, für alle gebräuchlichen Distributionen eine ausführliche Installationsanleitung zu geben. Hierzu konsultieren Sie bitte die mit den Distributionen gelieferte Dokumentation. Es existieren allerdings einige Punkte, in denen eine Firewall-Installation von der Installation eines normalen Arbeitsplatzrechners abweicht. Diese wollen wir im folgenden am Beispiel von SuSE und Debian betrachten.

Besonders ist darauf zu achten, welche Software installiert wird. Hier gilt auf jeden Fall: »Weniger ist mehr.« Jedes installierte Programm ist eine potentielle Sicherheitslücke. Dies gilt insbesondere für aktive Serverdienste. Aus Sicherheitsgründen sollte auf der Firewall auf WWW-, FTP-, NNTP-, Netbios-, Telnet-, R- (Rsh, Rlogin, ...), NFS-, NetBIOS-Server, den Portmapper und die X Window-Oberfläche verzichtet werden.

SuSE

Die Installation von SuSE-Linux erfolgt, indem entweder von den mitgelieferten CD-ROMs oder DVDs gebootet wird.

Wenn Sie zur Installation der benötigten Software kommen, sollten Sie im Menü Software-Auswahl das Minimal-System auswählen. Unter Erweiterte Auswahl sollten Sie dann die zur Auswahl stehenden Pakete noch einmal einzeln betrachten und entscheiden, welche zusätzlich nötig sind.

Sie können sich die Pakete in verschiedenen Arten gegliedert anzeigen lassen. Wählen Sie »Selektionen«, so werden Ihnen nicht alle Pakete angezeigt. Bestimmte Pakete werden einfach als notwendig vorausgesetzt und können nicht einzeln an- oder abgewählt werden. Aus diesem Grund sollten Sie sie lieber als hierarchisch gegliederte Paket-Gruppen (z. B. »Produktivität/Editoren« oder »Hardware/Modem«) anzeigen lassen.

Tabelle 8-1 gibt an, welche Pakete zusätzlich sinnvoll ausgewählt werden können bzw. welche besser abgewählt werden. Allerdings kann die dort präsentierte Auswahl nur ein Vorschlag sein. Wenn Sie z. B. keine ISDN-Karte besitzen, sind die Pakete i4l und i4lfirm reine Speicherplatzverschwendung.

Pakete, die auf der Firewall besser nicht installiert werden sollten, sind in der zweiten Spalte mit »deinst.« gekennzeichnet. So sollte eine Firewall z. B. feste IP-Adressen eingetragen haben und diese nicht von einem DHCP-Server beziehen, weshalb der `dhcpd` nicht installiert werden sollte.

Pakete, die nur auf der DVD, nicht aber den CDs enthalten sind, sind mit »(DVD)« gekennzeichnet.

Das Paket `exim` kollidiert mit dem Paket `postfix`, weshalb Sie `postfix` abwählen sollten. Sie werden sich jetzt vielleicht fragen, warum ich Ihnen empfehle, einen anderen Mailserver zu installieren, als von SuSE vorgesehen. Die Firewall sollte normalerweise nicht als Mailserver dienen. Dazu sollte man besser einen eigenen Server aufsetzen. Auch wenn der `postfix` als sehr sicher gilt, ist es keine gute Idee, zusätzliche Netzwerkdienste auf einer Firewall zu betreiben.

Tabelle 8-1: Pakete der SuSE-Installation (Version 9.3)

Serie / Paket	Aktion	Bemerkung
Dokumentation		
binddoc	inst.	Doku zum Bind
books	inst.	einige grundlegende Bücher über Linux (DVD)
fhs	inst.	der File Hierarchy Standard (DVD)
howto	inst.	die HOWTOs (DVD)
man-pages	inst.	zusätzliche Manpages
rfc	inst.	Internet-Standards
suselinux-adminguide_de	inst.	Administrationshandbuch
suselinux-userguide_de	inst	Benutzerhandbuch
Entwicklung		
dbus-1-glib	deinst.	Bibliothek zu dbus-1
gcc	inst.	C Compiler
gcc-c++	inst.	C++ Compiler
hex	inst.	Hexdumps erstellen (DVD)
kernel-source	inst.	von SuSE modifizierte Kernelquellen
ncurses-devel	inst.	für die Kompilierung von Menüprogrammen
Hardware / Modem		
minicom	inst.	Terminalprogramm
Hardware / ISDN		
i4l	inst.	ISDN-Unterstützung
i4lfirm	inst.	Firmware für aktive ISDN-Karten
Produktivität / Archivierung		
mt_st	inst.	Steuerung von Bandlaufwerken (DVD)
unzip	inst.	Entpacken von Zip-Archiven
zip	inst.	Erzeugung von Zip-Archiven
Produktivität / Datei-Dienstprogramme		
mc	inst.	Midnight Commander
Produktivität / Editoren		
joe	deinst.	Editor
Produktivität / Multimedia		
cdrecord	inst.	CD-Images brennen
dvd+rw-tools	inst.	DVDs brennen
mkisofs	inst.	CD-Images erstellen
Produktivität / Netzwerk		
bind	inst.	DNS-Server
bind-chrootenv	inst.	Chroot-Umgebung für den Bind
dhcpd	deinst.	IP-Adr. autom. beziehen
exim	inst.	Mailserver (Konflikt: postfix deinst.)
iptraf	inst.	Überwachung des Netzwerkverkehrs
lynx	inst.	textbasierter Browser
mutt	inst.	einfaches E-Mail-Programm
ncftp	inst.	FTP-Klient, besser als `ftp` (DVD)
netcat	inst.	Port Scan, Probeverbindungen
netdiag	inst.	Netzwerkkartendiagnose
nmap	inst.	Port Scans
portmap	deinst.	Portmapper
postfix	deinst.	Mailserver (kollidiert m. exim)

Tabelle 8-1: Pakete der SuSE-Installation (Version 9.3) (Forts.)

Serie / Paket	Aktion	Bemerkung
ppp	inst.	Modemnutzung
privoxy	inst.	WWW-Proxy
proxy-suite	inst.	FTP-Proxy (DVD)
rp-pppoe	inst.	DSL-Unterstützung (DVD)
squid	inst.	FTP/WWW-Proxy
tin	inst.	Newsreader (DVD)
Produktivität / Publishing		
texinfo	inst.	Programmdokumentationen
Produktivität / Text		
patch	inst.	Einspielen von Patches
System / Basis		
acct	inst.	Programmaufrufe protokollieren
System / Daemons		
at	deinst.	Programme einmalig zu einer bestimmten Uhrzeit starten
dbus-1	deinst.	Message-Bus, wird von hal benötigt
hal	deinst.	Daemon, der Hardware-Informationen sammelt. Wird von SuSE für das Hotplugging benutzt und von submount benötigt.
nscd	deinst.	Name Service Caching Daemon macht NIS und LDAP effizienter
openslp-server	deinst.	unnötiger Server
powersave	deinst.	Stromspar-Daemon
System / Dateisysteme		
mtools	inst.	Zugriff auf DOS-Disketten
submount	deinst.	automatisches Mounten von Datenträgern

Der `postfix` hatte in meinen Versuchen mit SuSE den Nachteil, daß der Dienst grundsätzlich aktiv sein mußte. D. h., es war immer ein Netzwerkdienst auf Port 25 aktiv. Deaktivierte man den Dienst, so war nicht einmal eine lokale Zustellung von E-Mails möglich. Lokale E-Mails werden aber z. B. von Cron-Jobs verwendet, um `root` über aufgetretene Probleme zu informieren.

Mit `exim` war dies kein Problem. Beim Senden einer lokalen Mail wurde `exim` kurz gestartet, stellte die E-Mail zu und beendete sich wieder, ohne aber auf Port 25 Verbindungen entgegenzunehmen. Darüber hinaus kam er sehr viel besser mit der Tatsache zurecht, daß mein selbstkompilierter Kernel kein IPv6 unterstützte.

Es war allerdings nötig, die Konfiguration so zu ändern, daß `exim` auch E-Mails an `root` zustellte. Aber dazu später mehr.

Nach der Installation versucht SuSE, Ihre Netzwerkkarten zu konfigurieren. Zum jetzigen Zeitpunkt ist es dafür aber noch zu früh. Wählen Sie deshalb unter »Netzwerkkonfiguration« den Knopf »Ändern«, und löschen Sie dann unter »Netzwerkschnittstellen« alle Netzwerkkarten.

Unter »Authentifikationsmethode« wählen Sie bitte »lokal (/etc/passwd)«.

Debian

Die Installation von Debian beginnt, indem Sie von den Installationsmedien booten.

Wenn Sie wie beschrieben alle Netzwerkkabel abgezogen haben, dann wird Debian feststellen, daß kein DHCP-Server zur Verfügung steht, um die Netzwerkkarten automatisch mit IP-Adressen zu versorgen.

Wenn Sie dann gefragt werden, welche `Netzwerk-Konfigurationsmethode` Sie verwenden wollen, antworten Sie am besten:

```
Netzwerk unkonfiguriert lassen
```

Kurz darauf werden Sie gefragt, ob Sie eine `PPP-Verbindung für die Installation verwenden` wollen. Auch hier sagen Sie besser

```
Nein
```

Schließlich müssen Sie sich für eine Methode zur `Debian Software-Auswahl` entscheiden. Nehmen Sie hier

```
Manuelle Paketwahl
```

Damit wird nur ein Minimalsatz an Paketen installiert. Welche das sind, wird Ihnen im folgenden angezeigt. Durch Drücken der Taste »g« bestätigen Sie die Auswahl. Wir werden sie später noch genauer an unsere Bedürfnisse anpassen.

Werden wir schließlich noch gefragt, welche `E-Mail-Konfigurationsart` wir wünschen, so sollten wir

```
Nur lokale E-Mailzustellung (keine Netzverbindung)
```

wählen.

Nachdem die Installation abgeschlossen ist und wir als `root` am System angemeldet sind, sollten wir noch die installierten Pakete durchgehen und die Pakete deinstallieren, die unnötige Dienste bereitstellen. Auch fehlen noch Pakete, die nützliche Funktionen bereitstellen. Tabelle 8-2 kann Ihnen bei der Entscheidung helfen, welche Pakete Sie unter Debian 3.1 näher betrachten sollten.

Am einfachsten bearbeiten Sie dabei die Pakete mit dem Programm `aptitude`. Wenn Sie es aufrufen, sehen Sie im Hauptbereich der Anwendung eine Baumstruktur, die Sie durch einfaches Drücken von <Return> auf einem Unterzweig auf- und zuklappen können. Drücken Sie dagegen <Return> auf einem einzelnen Paket, so werden Ihnen weitere Informationen angezeigt.

Für die weitere Navigation sind folgende Befehle nützlich:

- **?** Öffnet ein Hilfefenster.
- \+ Wählt ein Paket an.
- \- Wählt ein Paket ab.

Tabelle 8-2: Pakete der Debian-Installation (Version 3.1)

Bereich	Abschn.	Paket	Aktion	Bemerkung
admin	main	acct	inst.	Protokollierung aufgerufener Programme
		at	deinst.	Programme einmalig zu einer bestimmten Uhrzeit starten
		debfoster	inst.	Deinstallation überflüssiger Pakete
		hdparm	inst.	Konfiguration von Festplatten
		mt-st	inst.	Benutzung von Bandlaufwerken
		tct	inst.	Finden gelöschter Dateien
comm	main	minicom	inst.	Terminalprogramm
devel	main	autoconf	inst.	wird beim Kompilieren mancher Programme benötigt
		kernel-source-2.4.27	inst.	Kernel-Quellen Version 2.4
		oder		bzw.
		kernel-source-2.6.8	inst.	Version 2.6
doc	main	bind9-doc	inst.	Dokumentation zum DNS-Server
		cdrtools-doc	inst.	Dokumentation zu mkisofs und cdrecord
editors	main	joe	inst.	Editor
		vim	inst.	Editor
misc	main	gpm	inst.	Maus-Unterstützung
net	main	bind9	inst.	DNS-Server
		ftp-proxy	inst.	FTP-Proxy
		iptraf	inst.	Überwachung des Netzverkehrs
		links2	inst.	textbasierter Browser
		lpr	deinst.	Druckerserver
		ncftp	inst.	FTP-Klient
		netcat	inst.	Port Scans, Probeverbindungen
		netdiag	inst.	Netzwerkkartendiagnose
		nfs-common	deinst.	Network File System, Unix-Gegenstück zum Windows-Netzwerk, ähnlich unsicher
		nmap	inst.	Port Scans
		pidentd	deinst.	Ident-Daemon, gibt unnötigerweise vertrauliche Daten preis
		portmap	deinst.	Portmapper, Metadaemon für diverse unsichere Dienste
		tcpdump	inst.	Überwachung des Netzverkehrs
news	main	tin	inst.	Newsreader
otherfs	main	cdrecord	inst.	CDs brennen
		mkisofs	inst.	CDs und DVDs brennen
utils	main	bzip2	inst.	Packer
		dvd+rw-tools	inst.	DVDs brennen
		hex	inst.	Hexdumps erstellen
		isdnutils-base	inst.	ISDN
		ipppd	inst.	ISDN
		mc	inst.	Midnight Commander
		tripwire	inst.	Dateiänderungen überwachen
		zip	inst.	Packer
web	main	links	inst.	textbasierter Browser
		lynks	inst.	textbasierter Browser
		privoxy	inst.	Webproxy
		squid	inst.	Webproxy

/ Erlaubt es, nach einem Begriff zu suchen.

***n, *** Beide Tasten setzen die Suche fort.

g Installiert die ausgewählten Pakete. Unter Umständen werden vorher noch automatisch hinzugefügte oder vorgeschlagene Pakete angezeigt. In diesem Fall muß noch einmal mit <g> bestätigt werden.

q Beendet das Programm.

Wenn wir bei der Installation gefragt werden: »`Sollen die cdrecord-Programme als SUID-root installiert werden?`«, so sollten wir dies ablehnen. Auf der Firewall werden nur in seltenen Fällen CDs gebrannt (z. B. für Backups). Dies kann man dann auch als Benutzer `root` tun.

Konfiguration des Mailservers Exim

Ist die Installation abgeschlossen und sie sind als `root` am System angemeldet, so müssen Sie jetzt noch konfigurieren, wohin E-Mails geschickt werden, die an `root` gesandt wurden. Grundsätzlich ist es der Wille der Exim-Schöpfer, daß die E-Mails nicht im Postkorb von `root`, sondern von einem normalen Anwender landen. Vorzugsweise sollte es sich dabei um die Person handeln, die für die Administration des Rechners zuständig ist.

Aus diesem Grund ist in der Konfiguration üblicherweise eine Zeile eingetragen, die es verbietet, `root` E-Mails zuzustellen. Unter SuSE 9.3 reicht es, diese Zeile zu ändern, damit `root` wieder E-Mails empfangen kann. Unter Debian 3.1 sind dagegen ein paar mehr Schritte nötig.

SuSE 9.3

Hier reicht es, eine Zeile in der `exim`-Konfigurationsdatei zu ändern. Öffnen Sie dazu */etc/exim/exim.conf*. Dort finden Sie die folgende Zeile:

```
never_users = root
```

Diese Zeile bewirkt, daß E-Mails an `root` nicht zugestellt werden. Da wir dies aber wollen, sollten wir sie folgendermaßen ändern:

```
never_users =
```

Debian 3.1

Im Gegensatz zu SuSE hat man bei Debian den Exim mit Standardeinstellungen kompiliert. Daher kennt der Dienst eine feste Liste von Benutzerkonten, an die unabhängig vom Inhalt der Konfigurationsdatei niemals direkt E-Mails zugestellt werden. Diese Liste enthält standardmäßig `root`.

Um E-Mails, die an root adressiert sind, zustellen zu können, muß ein Benutzerkonto definiert werden, das alle E-Mails erhält, die für root bestimmt sind. Welches Benutzerkonto dies ist, wird in der Datei */etc/aliases* definiert. Sie enthält Zeilen der Art:

```
Alias: Benutzerkonto
```

Mail, die an *Alias* adressiert ist, wird dabei an *Benutzerkonto* zugestellt.

Wir müssen hier nach der Zeile suchen, die definiert, was mit der E-Mail für root geschieht:

```
# grep 'root' /etc/aliases
root: user
```

In diesem Fall wird die E-Mail an das normale Benutzerkonto weitergeleitet, das wir während der Installation angelegt haben. Wenn wir uns regelmäßig unter diesem Benutzerkonto anmelden, dann können wir hier abbrechen.

Falls wir uns aber in der Regel nur kurz am System anmelden, um es zu administrieren, so kann es sein, daß wir das normale Benutzerkonto gar nicht verwenden. In diesem Fall kann es sinnvoll sein, eine Mailbox für root als symbolischen Link des Benutzerkontos anzulegen:

```
# cd /var/spool/mail
# ln -s user root
```

Testen der geänderten Konfiguration

Wir können einfach testen, ob unsere Änderungen funktionieren, indem wir uns selbst eine E-Mail schicken:

```
# sendmail root
subject: test

Test, 1, 2, 3
.
```

Die Leerzeile ist dabei wichtig. Wir müssen auch darauf achten, daß die letzte Zeile nur aus einem Punkt ohne zusätzliche Leerzeichen besteht.

Nun können wir nachsehen, ob die E-Mail angekommen ist. Dazu verwenden wir den Befehl mail. Dieses E-Mail-Programm ist zwar archaisch und für normale Zwecke kaum zu ertragen, für einen kurzen Check aber ausreichend:

```
# mail
Mail version 8.1.2 01/15/2001.  Type ? for help.
"/var/mail/root": 21 messages 1 new 21 unread
>N 21 root@gonzales      Tue Jul 26 21:53   15/397   test
&
```

Die E-Mail ist angekommen[1]. Mit dem Befehl q oder quit können wir nun mail verlassen.

1 Das Beispiel stammt von einem Debian-Rechner, daher werden Sie auf einem SuSE-Rechner vermutlich eine andere Versionsnummer sehen, ansonsten verhalten sich die mail-Implementationen aber gleich.

Der Bootvorgang

Später werden wir Skripte schreiben, die automatisch gestartet werden sollen. Um dies zu erreichen, ist es notwendig zu verstehen, was passiert, wenn Linux bootet. Darüber hinaus stellt der Bootvorgang die Achillesferse eines jeden PC-basierten Serversystems dar. Wer in ihn eingreifen kann, ist kurz davor, Administrator des Systems zu werden. Aus diesem Grund empfiehlt es sich, Server in einem eigenen Raum mit einem soliden Schloß unterzubringen.

Das BIOS

Der Vorgang beginnt, indem das BIOS nach einem *Bootloader* sucht. Dieser kann prinzipiell auf einer Festplatte, Diskette, CD und neuerdings auch einem ZIP-Medium stehen. Ist das BIOS so konfiguriert, daß es als erstes überprüft, ob eine bootfähige Diskette oder CD eingelegt ist, so reicht der Besitz einer Installations-CD oder Rettungsdiskette aus, um ein Linux-Rettungssystem zu laden, mit dem man das Linux-System auf der Festplatte mounten und manipulieren kann. Dabei könnte der Angreifer z. B. das Paßwort für `root` in der Datei */etc/shadow* entfernen, um sich so ohne Paßwort anmelden zu können.

Eine gängige Lösung besteht darin, die Bootreihenfolge auf »nur C« zu setzen und die BIOS-Einstellungen mit einem Paßwort zu versehen. Dies ist zwar eine sinnvolle Maßnahme gegen versehentlich eingelegte Disketten mit Bootviren[2], kann von einem Angreifer aber leicht außer Kraft gesetzt werden. Fast jedes Motherboard kennt einen Weg, ein vergessenes BIOS-Paßwort zu umgehen. Dabei kann es sich z. B. um einen Jumper auf dem Motherboard oder ein Masterpaßwort handeln. Schlimmstenfalls muß die Batterie des CMOS-RAMs entfernt werden.

Ist ihm dies zu aufwendig, kann er notfalls auch die Festplatte ausbauen und in einen anderen Rechner einbauen, wo er dann unbeschränkten Zugriff hat. Es gibt mittlerweile Miniatur-PCs in der Größe einer Zigarrenschachtel, die über einen Anschluß für Tastatur, Maus, Monitor sowie eine zusätzliche Festplatte verfügen.

Der Bootloader

Wird von Festplatte gebootet, so wird der Bootloader aktiv. Er wird nun nach der Partition suchen, auf der seine Konfigurationsdaten installiert sind. Findet er diese, so wird er einen Prompt oder ein Bootmenü anzeigen.

Nun kann der Benutzer zwischen verschiedenen vordefinierten Konfigurationen auswählen, in denen jeweils festgelegt ist, welcher Kernel bzw. welches Betriebssystem gestartet werden soll und auf welcher Partition sich das Wurzelverzeichnis des zu bootenden Systems befindet. Des weiteren können diverse Parameter angegeben werden, die die Arbeit des Kernels beeinflussen.

2 Obwohl ein DOS-Bootvirus unter Linux wohl kaum aktiv werden kann, besteht doch die Gefahr, daß er versucht, sich in den MBR zu schreiben, bevor Linux gebootet ist. Dabei würde er den Bootloader löschen, womit Linux nicht mehr starten könnte.

Wurde eine Konfiguration ausgewählt, so wird der festgelegte Kernel geladen und ausgeführt. Dieser beginnt nun damit, alle fest einkompilierten Treiber zu starten und gegebenenfalls zu initialisieren. Ist er damit fertig, so startet er ein Programm namens `init`.

Allerdings bieten die gängigen Bootloader dem Benutzer nicht nur die Möglichkeit, vorher definierte Konfigurationen zu booten, er kann statt dessen auch eigene Parameter übergeben. Ein denkbarer Parameter wäre dabei

```
init=/bin/sh
```

Damit würde der Kernel direkt nach dem Booten statt `init` eine Shell mit Rootrechten starten, ohne vorher nach einem Paßwort zu fragen. Um dies zu verhindern, muß der Bootloader so konfiguriert werden, daß nur vorher festgelegte Konfigurationen gestartet, nicht aber eigene Parameter übergeben werden können.

Im folgenden sehen wir, wie wir dies bei den derzeit gängigen Bootloadern tun können.

Grub

Der Grub wird über eine zentrale Datei gesteuert. Üblicherweise handelt es sich dabei um */etc/boot/grub/menu.lst*. Jede Änderung, die hier vorgenommen wird, wird dabei automatisch beim nächsten Bootvorgang wirksam.

Die Datei ist dabei folgendermaßen aufgebaut:

```
<Globale Optionen>

title <Konfigurationsname1>
<Konfigurationsoptionen>

title <Konfigurationsname2>
<Konfigurationsoptionen>

...
```

Normalerweise kann ein Benutzer eine Eingabezeile aufrufen, auf der er in einer Art Shell diverse Befehle absetzen kann, die es nicht nur erlauben, beliebige Kernel mit beliebigen Parametern zu booten, sondern z. B. auch beliebige Dateien auf der Festplatte auslesen. Außerdem enthält der Grub einen Editor, mit dem man das Bootmenü umkonfigurieren kann.

Um dies zu verhindern, muß man ein Passwort festlegen, das eingegeben werden muß, um diese Funktionen freizuschalten. Dies geschieht, indem man als erste globale Option die folgende Zeile einfügt:

```
password <Passwort>
```

Zusätzlich können wir auch bestimmte Konfigurationen sperren, indem wir direkt hinter der jeweiligen `title`-Zeile eine neue Zeile mit dem Befehl `lock` einfügen. Wird dieser Menüpunkt ausgewählt, ohne daß vorher das Paßwort eingegeben wurde, so wird die Bearbeitung des Menüpunkts abgebrochen, und der Benutzer findet sich im Menü wie-

der. Dabei ist es wichtig, daß sich der Befehl direkt in der Zeile unter `title` befindet. Die Zeilen werden der Reihe nach abgearbeitet. Stehen also Befehle vor `lock`, so werden diese ausgeführt.

Wir werden in Kapitel 8, Unterabschnitt *Grub*, ab Seite 157 noch einmal auf den Grub zurückkommen. Dort wird es dann noch einmal um das Anlegen eigener Konfigurationen gehen.

LiLo

Um den LiLo zu sichern, muß die Konfigurationsdatei */etc/lilo.conf* angepaßt werden. In ihren globalen Optionen sind die Zeilen

```
password=<Passwort>
restricted
```

nachzutragen. Damit können nur dann Optionen eingegeben werden, wenn der Benutzer das Paßwort kennt. Da dieses nur selten gebraucht und darüber hinaus im Klartext gespeichert wird, sollte es sich dabei keinesfalls um das Root-Paßwort handeln. Auch sollte die Datei */etc/lilo.conf* nur für `root` lesbar sein.

Beachten Sie aber bitte, daß die Konfiguration erst wirksam wird, wenn Sie den Aufruf

```
# /sbin/lilo
```

ausführen. Ohne diesen Befehl findet der Bootloader nach einer Änderung seine Konfigurationsdatei nicht mehr, und der Bootvorgang mißlingt.

Zusätzliche Details zu LiLo und der */etc/lilo.conf* finden Sie in Kapitel 8, Unterabschnitt *LiLo*, ab Seite 159.

Init

`Init` ist der erste Prozeß, der vom Kernel gestartet wird. Er startet dann direkt oder indirekt alle weiteren Prozesse. Modernere Versionen von `init` erlauben es, mehrere Varianten vorzugeben, die beschreiben, welche Prozesse gestartet werden sollen. Diese werden *Runlevel* genannt.

Jedem Runlevel ist der Aufruf eines Skripts zugeordnet. Dieses sorgt dafür, daß diejenigen Dienste gestartet werden, die diesem Runlevel zugeordnet sind. Es ist auch möglich, im laufenden Betrieb mit `init <Runlevel>` in einen anderen Runlevel zu wechseln. Dies führt dazu, daß die Dienste, die dem alten Runlevel zugeordnet sind, beendet und die Dienste des neuen Runlevels gestartet werden.

Des weiteren wird für jeden Runlevel angegeben, welche Terminals gestartet werden sollen. Dabei kann es sich um die normalen virtuellen Terminals handeln, die man zum lokalen Arbeiten nutzt, es können auf diese Weise aber auch Modemzugänge auf den seriellen Anschlüssen aktiviert werden.

Gesteuert wird der ganze Vorgang über die Datei */etc/inittab*. In dieser wird unter anderem festgelegt, in welchen Runlevel das System nach dem Booten schalten soll, welches

Skript zur Initialisierung gleich nach dem Start von `init` ausgeführt und welches Skript beim Wechsel in einen bestimmten Runlevel gestartet werden soll. Darüber hinaus können auch noch Programme definiert werden, die bei bestimmten Ereignissen oder Tastatureingaben aktiviert werden (z. B. Stromausfall, *<Ctrl><Alt><Del>*).

Das jeweilige Skript für einen Runlevel dient dazu, bestimmte Dienste zu starten bzw. nicht mehr benötigte Dienste des vorherigen Runlevels zu beenden. Normalerweise existiert dazu für jeden Runlevel ein eigenes Unterverzeichnis. In diesem befinden sich Skripte, deren Name mit einem »K« (für kill) oder »S« (für start) beginnt.

Soll das System nun in einen anderen Runlevel wechseln, so wird das Skript des Runlevels in der Regel alle diejenigen Skripte aufrufen, deren Name mit »K« beginnt und die sich entweder im Verzeichnis des aktuellen Runlevels (SuSE) oder im Verzeichnis des neuen Runlevels befinden (Debian). Als Parameter wird `stop` übergeben. Danach werden alle diejenigen Skripte ausgeführt, die im Verzeichnis des neuen Runlevels liegen und deren Name mit »S« beginnt. Als Parameter wird dabei `start` übergeben.

K-Skripte dienen dazu, Dienste zu beenden. S-Skripte dagegen starten Dienste. Der zusätzliche Parameter erlaubt es, ein einziges Skript zu schreiben, das jeweils die gewünschte Aktion ausführt. Die K- und S-Skripte sind dabei in der Regel nur symbolische Links auf das eigentliche Skript, das in einem eigenen Verzeichnis untergebracht ist.

SuSE 9.3

Seit Version 8.0 befinden sich die Runlevel-Skripte im Verzeichnis */etc/init.d*. Dies wurde nötig, um die in der Linux Standard Base Specification [1] festgelegten Regeln zu erfüllen. Vorher wurde */sbin/init.d* benutzt. Die Runlevel-Verzeichnisse heißen */etc/init.d/rc<Runlevel>.d* statt bisher */sbin/init.d/rc<Runlevel>.d*.

Alle Runlevel werden vom Skript `/etc/init.d/rc` bearbeitet, das mit dem Runlevel als Argument aufgerufen wird und die K-Skripte des **alten** Runlevels gefolgt von den S-Skripten des neuen Runlevels aufruft. Üblicherweise existiert für jeden Dienst in dem Verzeichnis des Runlevels, in dem er gestartet werden soll, sowohl ein K-Skript als auch ein S-Skript. So wird der Dienst beim Übergang in den Runlevel gestartet und beim Verlassen des Runlevels beendet.

Dies steht im Gegensatz zu Debian, bei denen üblicherweise für einen Dienst in jedem Runlevel-Verzeichnis nur ein Skript steht. Dabei stehen K-Skripte in Runleveln, in denen er nicht gestartet werden soll, während sich S-Skripte in Verzeichnissen von Runleveln befinden, in denen der Dienst gewünscht ist.

Unter SuSE-Linux werden die folgenden Runlevel verwendet:

boot Dies ist der Zustand, in dem sich das System beim ersten Start von `init` befindet. Er stellt einen Sonderfall dar, da es keinen Runlevel `boot` gibt, den man mit `init <Level>` aufrufen kann. Trotzdem ist auch ihm ein Verzeichnis */etc/init.d/boot.d* zugeordnet, das Skripte enthalten kann, die durch das Skript `/etc/init.d/boot` ausgeführt werden. Anschließend startet `boot` noch `/etc/init.d/boot.local`. Dieses Skript dient in

erster Linie dazu, eigene Programme zu definieren, die beim Systemstart vor dem Start aller anderen Dienste ausgeführt werden sollen.[3]

Runlevel 0 In diesen Zustand wechselt das System, wenn es heruntergefahren werden soll (`sync;halt`). Im zugehörigen Vezeichnis */etc/init.d/rc0.d/* befindet sich nur ein Link auf das Skript `/etc/init.d/halt`. Dieses führt seinerseits vor dem Beenden von Linux noch das Skript `/etc/init.d/halt.local` aus, welches in seiner Funktion das Gegenstück zu `boot.local` darstellt.

Runlevel 6 In diesen Zustand wechselt das System, wenn es neu gestartet werden soll (`sync;reboot`). Das Skript `/etc/init.d/reboot` wird über einen Link in */etc/init.d/rc6.d* ausgeführt. Nachdem es das Skript `/etc/init.d/halt.local` aufgerufen hat, führt es einen Neustart durch.

Runlevel S Wird benutzt, wenn nach dem Booten in den *Single User Mode* geschaltet werden soll. In diesem Zustand ist das Netzwerk abgeschaltet, und es kann nicht zwischen virtuellen Konsolen umgeschaltet werden.

Runlevel 1 Entspricht Runlevel S. Er wird dann benutzt, wenn aus dem normalen Betrieb (Runlevel 2, 3 oder 5) in den Single User Mode geschaltet werden soll.

Runlevel 2 Entspricht Runlevel 1, allerdings werden mehrere virtuelle Konsolen unterstützt.

Runlevel 3 Der normale Zustand mit voller Netzwerkunterstützung.

Runlevel 5 In unserem Fall uninteressant. Der Rechner startet gleich mit einer graphischen Oberfläche.

Auch hier wurden Anpassungen vorgenommen, um der Linux Standard Base Specification zu genügen. So wurde früher Runlevel 3 für die graphische Anmeldung benutzt, 2 für ein netzwerkfähiges System ohne graphische Anmeldung, 1 für ein System ohne Netzwerk, und S wurde immer für den Single User Mode benutzt.

Ein Beispiel für ein Runlevel-Skript ist `/etc/init.d/gpm`. Dieses Skript startet und stoppt den `gpm`, einen Dienst, der die Benutzung der Maus im Textmodus ermöglicht. Von seinem Aufbau her ist es repräsentativ für die meisten Runlevel-Skripte, die Serverdienste starten. Es lohnt sich daher, es etwas näher zu betrachten:

```
#! /bin/sh
# Copyright (c) 1995-1998 S.u.S.E. GmbH Fuerth, Germany.
#
# Author:
#
# /etc/init.d/gpm
#
#   and symbolic its link
#
# /sbin/rcgpm
#
### BEGIN INIT INFO
# Provides:       gpm
# Required-Start: $remote_fs $syslog
# Required-Stop:  $remote_fs $syslog
```

3 Das Konzept entspricht der *autoexec.bat* unter DOS.

```
# Default-Start:  2 3
# Default-Stop:   0 1 5 6
# Description:    Start gpm to support mouse on console
### END INIT INFO
```

Nach dem normalen Header, wie man ihn in den meisten Shellskripten findet, folgt eine Reihe von Zeilen mit Verwaltungsinformationen. Diese wurden erst vor einiger Zeit durch die Linux Standard Base Specification (LSB) definiert und erlauben es, die S- und K-Links auf ein Skript automatisch zu generieren.

Die Linux Standard Base kennt die folgenden Header:

Provides Eine Facility, die das Skript zur Verfügung stellt.

Required-Start Diese Facilities müssen bereits zur Verfügung stehen, wenn das Skript gestartet wird. Das heißt, Skripte, die diese Facilities bereitstellen, werden vor diesem Skript gestartet. Existiert für eine der angegebenen Facilities kein Skript, so kann dieses Skript nicht gestartet werden.

Should-Start (optional) Wie bei `Required-Start` werden Skripte, die diese Facilities bereitstellen, vor diesem Skript gestartet. Existiert aber kein Skript, das die jeweilige Facility bereitstellt, so wird sie ignoriert und das Skript trotzdem gestartet.

Required-Stop Wenn das Skript gestoppt wird, sollten diese Facilities immer noch aktiv sein. Das heißt, erst wird dieses Skript gestoppt, dann die Skripte, welche die genannten Facilities bereitstellen.

Should-Stop (optional) Wie bei `Required-Stop` werden Skripte, die diese Facilities bereitstellen, nach diesem Skript gestoppt. Existiert aber kein Skript, das die jeweilige Facility bereitstellt, so wird sie ignoriert.

Default-Start In diesen Runleveln soll das Skript gestartet werden.

Default-Stop In diesen Runleveln soll das Skript nicht laufen.

Short-Description (optional) Eine einzeilige Beschreibung der Funktion.

Description Eine ausführliche Beschreibung. Diese kann über mehrere Zeilen gehen. Folgezeilen müssen mit »#<Tab>« beginnen.

Header, die mit »(optional)« gekennzeichnet sind, dürfen dabei fehlen.

Facilities stehen dabei für Systemdienste. Grundsätzlich können alle Runlevel-Skripte mit dem `Provides:`-Header erklären, daß sie eine bestimmte Facility bereitstellen.

Eine Sonderrolle spielen die System-Facilities. Hierbei handelt es sich um Dienste, deren Name standardisiert ist. Sie stellen Standarddienste bereit, die von vielen Anwendungen benötigt werden.

Die Linux Standard Base definiert in der Version 1.2 die folgenden System-Facilities:

$local_fs Die lokalen Dateisysteme (z. B. Festplatten) wurden gemountet.

$network Das Netzwerk ist aktiv.

$portmap Der Portmapper ist gestartet.

$remote_fs Die Netzwerklaufwerke wurden gemountet.

$syslog Die Systemprotokollierung ist gestartet.

$time Die Systemzeit ist gesetzt.

Die Namen von System-Facilities beginnen mit einem »$«. Üblicherweise werden diese Namen nicht direkt im Runlevel-Skript definiert, sondern den Namen von System-Facilities werden eine oder mehrere normale Facilities zugeordnet. Unter SuSE geschieht dies über die Datei */etc/insserv.conf*.

Das Neue an diesem Vorgehen liegt darin, daß man nun nicht mehr selbst entscheiden muß, in welchen Verzeichnissen man Links unter welchem Namen anlegt, um sicherzustellen, daß alle Dienste auch wirklich in der richtigen Reihenfolge gestartet werden. Dies ist insbesondere dann schwierig, wenn man einen Dienst geschrieben hat und ihn den Benutzern verschiedener Distributionen anbieten will.

Nicht nur benutzen alle Distributionen unterschiedliche Pfade für die S- und K-Links. Teilweise variiert sogar das Verständnis, was diese bedeuten. Darüber hinaus werden die Links in alphabetischer Reihenfolge abgearbeitet. Um einen Dienst nach einem bestimmten anderen Dienst starten zu lassen, muß man dem jeweiligen Link einen Namen geben, der im Alphabet hinter dem des vorher zu startenden liegt. Die Linknamen der Standarddienste unterscheiden sich aber ebenfalls von Distribution zu Distribution. So entspricht z. B. `S80sendmail` unter Red Hat 7.0 dem `S20exim` unter Debian 2.2.

In einem LSB-konformen System reicht der Aufruf

```
# /usr/lib/lsb/install_initd /etc/init.d/gpm
```

aus, um die nötigen Links zu erzeugen, damit das Skript

- in den Runleveln 2 und 3 aktiviert wird,
- in den anderen Runleveln inaktiv ist und
- nach den *Facilities* `$remote_fs` und `$syslog` gestartet bzw. vor diesen beendet wird.

Der Aufruf

```
# /usr/lib/lsb/remove_initd /etc/init.d/gpm
```

führt umgekehrt dazu, daß alle Links wieder gelöscht werden.

Unter SuSE brauchen wir uns aber nicht diesen komplizierten Befehl zu merken, es geht auch mit

```
# insserv /etc/init.d/gpm
```

Um den Dienst wieder zu deinstallieren, rufen Sie den Befehl einfach mit dem Parameter `-r` auf:

```
# insserv -r /etc/init.d/gpm
```

Achtung: Seit SuSE 9.3 sollten Sie keine Links mehr von Hand anlegen. Verwenden Sie immer `insserv`. SuSE verwendet ein neues System, bei dem die Dateien */etc/init.d/.depend.boot*, */etc/init.d/.depend.start* und */etc/init.d/.depend.stop* festlegen, in welcher Reihenfolge Dienste gestartet werden. Die Links werden nur noch benutzt, um herauszufinden, ob ein Dienst, der in den Dateien aufgeführt ist, auch tatsächlich in einem Runlevel verwendet werden soll. Legen Sie also nur die Links an, ohne die Dateien anzupassen, dann wird Ihr Dienst nicht gestartet.

```
GPM_BIN=/usr/sbin/gpm
test -x $GPM_BIN || exit 5
```

Nun wird überprüft, ob das aufzurufende Programm existiert. Wenn nicht, wird das Skript beendet.

```
# Check for existence of needed config file and read it
GPM_SYSCONFIG=/etc/sysconfig/mouse
test -r $GPM_SYSCONFIG || exit 6
. $GPM_SYSCONFIG

# Check for existence of needed values from $GPM_SYSCONFIG
test "$MOUSEDEVICE" -a "$MOUSETYPE" || test "$GPM_PARAM" || exit 6
```

Das Skript bindet die Datei */etc/sysconfig/mouse* ein. In der Konfigurationsdatei werden die Variablen *MOUSEDEVICE*, *MOUSETYPE* und *GPM_PARAM* definiert. Dabei kann in der ersten Variablen der Mauseingang festgelegt werden, während die zweite Variable das Kommunikationsprotokoll mit der Maus[4] enthalten kann. Alternativ kann auch ein kompletter Satz Parameter in *GPM_PARAM* eingetragen werden.

```
# Shell functions sourced from /etc/rc.status:
[...]
. /etc/rc.status
```

Anschließend wird das Skript `rc.status` eingebunden, das Funktionen definiert, mit denen ein Rückgabewert verwaltet werden kann. So überprüfen `rc_check` und `rc_status` den Rückgabewert des zuletzt aufgerufenen Programms und speichern diesen in einer internen Variablen. `rc_status` kann dabei auf Wunsch den numerischen Status auch als Klartext ausgeben (Option `-v`). `rc_exit` dient andererseits dazu, das Skript zu beenden und den gespeicherten Rückgabewert an das aufrufende Programm zurückzugeben.

```
# First reset status of this service
rc_reset
```

Als erste der neu definierten Funktionen wird aber `rc_reset` aufgerufen, um den gespeicherten Rückgabewert auf einen definierten Ausgangszustand »alles o. k.« zu setzen.

```
case "$1" in
    start)
        echo -n "Starting console mouse support (gpm)"
        if test "$GPM_PARAM" ; then
          startproc $GPM_BIN $GPM_PARAM
```

4 Zum Beispiel `ms` für Microsoft-Mäuse, `ps2` für ps/2-Mäuse oder `imps2` für Intellimouse-Modelle

```
            else
                ADD_PARAMS=""
                if test "$GPM_REPEAT" ; then
                    ADD_PARAMS="$ADD_PARAMS -R$GPM_REPEAT"
                fi
                startproc $GPM_BIN -m $MOUSEDEVICE -t $MOUSETYPE $ADD_PARAMS
            fi
            rc_status -v
            ;;
        stop)
            echo -n "Shutting down console mouse support (gpm)"
            killproc -TERM $GPM_BIN
            rc_status -v
            ;;
        try-restart)
            $0 status >/dev/null &&  $0 restart
            rc_status
            ;;
        restart)
            $0 stop
            $0 start
            rc_status
            ;;
        force-reload)
            echo -n "Reload console mouse support (gpm)"
            $0 stop  &&  $0 start
            rc_status
            ;;
        reload)
            echo -n "Reload console mouse support (gpm)"
            rc_failed 3
            rc_status -v
            ;;
        status)
            echo -n "Checking for console mouse support (gpm): "
            checkproc $GPM_BIN
            rc_status -v
            ;;
        probe)
            GPM_RESTART="False"
            for file in $GPM_SYSCONFIG /etc/gpm/* ; do
                test $file -nt /var/run/gpm.pid && GPM_RESTART="True"
            done
            test "$GPM_RESTART" = "True" && echo restart
            ;;
        *)
            echo Üsage: $0 {start|stop|status|try-restart\
            |restart|force-reload|reload|probe}"
            exit 1
            ;;
    esac
```

Schließlich kennt das Skript eine Vielzahl von Kommandos, von denen aber im Normalbetrieb nur `start` und `stop` benötigt werden, um den Dienst zu starten oder zu beenden.

Eine Fallunterscheidung mit `case` überprüft, welcher Befehl dem Skript als Parameter mitgegeben wurde. War es `start`, so wird der Dienst mit dem Befehl `startproc` gestartet, nachdem eventuell vorhandene Instanzen des Dienstes beendet wurden. Gelingt dies

nicht, liefert `startproc` einen Fehlerwert zurück, welcher durch `rc_status` als Text ausgegeben und intern gespeichert wird.

Wurde das Skript mit `stop` aufgerufen, so ist der Ablauf ähnlich, es wird allerdings `killproc` statt `startproc` aufgerufen, wodurch alle Instanzen des `gpm` beendet werden.

Im Falle, daß das Skript schließlich mit unbekannten Parametern aufgerufen wurde, wird eine Meldung ausgegeben, die alle diejenigen Befehle aufzählt, die das Skript kennt.

```
rc_exit
```

Abschließend wird noch ein Wert zurückgegeben, der signalisiert, ob der Start erfolgreich war.

Dieses Skript funktioniert so allerdings nur unter SuSE-Linux, insbesondere da `startproc`, `checkproc`, `killproc` und die `rc`-Funktionen in anderen Distributionen nicht enthalten sind. Davon abgesehen kann das Skript leicht für eigene Zwecke modifiziert werden.[5]

Debian 3.1

Debian benutzt eine andere Verzeichnisstruktur. Bei dieser Distribution befinden sich die Runlevel-Skripte im Verzeichnis */etc/init.d*. Die Runlevel-Verzeichnisse mit den symbolischen Links sind aber nicht etwa Unterverzeichnisse davon, sondern von */etc*. Sie heißen also */etc/rc<Runlevel>.d*. Alle Runlevel außer S werden vom Skript `/etc/init.d/rc` bearbeitet, das mit dem Runlevel als Argument aufgerufen wird und die K-Skripte des **neuen** Runlevels gefolgt von den S-Skripten des neuen Runlevels aufruft. Dienste werden also nur dann beendet, wenn dies für den neuen Runlevel so vorgesehen ist. Soll ein Dienst nur in einem Runlevel laufen, so muß sichergestellt sein, daß in allen anderen Runlevel-Verzeichnissen K-Skripte vorhanden sind.

Debian benutzt die folgenden Runlevel:

Runlevel S Dieser Runlevel sollte unter Debian nicht direkt aufgerufen werden. Statt dessen sollte Runlevel 1 gewählt werden, wenn man in den Single User Mode gehen will.

Das Verzeichnis */etc/rcS.d* ist dem Bootvorgang zugeordnet. Es kann Skripte enthalten, die beim Bootvorgang durch `/etc/init.d/rcS` ausgeführt werden. Anschließend führt `/etc/init.d/rcS` noch Skripte in */etc/rc.boot* aus. Dieses Verzeichnis existiert aber nur noch aus Kompatibilitätsgründen.

Runlevel 0 In diesen Zustand wechselt das System, wenn es heruntergefahren werden soll (`sync;halt`).

Runlevel 6 In diesen Zustand wechselt das System, wenn es neu gestartet werden soll (`sync;reboot`).

Runlevel 1 In diesem Zustand ist das Netzwerk abgeschaltet, und es kann nicht zwischen virtuellen Konsolen umgeschaltet werden (*Single User Mode*).

Runlevel 2 – 5 Der normale Zustand mit voller Netzwerkunterstützung. In der Grund-

5 Die Syntax von Shellskripten findet sich unter `man bash`.

installation sind alle diese Runlevel gleich eingerichtet und können nach eigenem Geschmack mit Leben erfüllt werden. Standardmäßig startet das System im Runlevel 2.

Die Runlevel-Skripte sind auch unter Debian nach dem schon beschriebenen Prinzip aufgebaut. Sie werden beim Start eines Dienstes mit »`start`« als Parameter aufgerufen, beim Beenden desselben mit »`stop`«. `startproc` und `killproc` unter SuSE entspricht bei Debian der Befehl `start-stop-daemon`.

Betrachten wir nun einmal das Skript `/etc/init.d/gpm` von Debian:

```
#!/bin/sh
#
# Start Mouse event server

PIDFILE=/var/run/gpm.pid
GPM=/usr/sbin/gpm
CFG=/etc/gpm.conf

test -x $GPM || exit 0

if [ "$(id -u)" != "0" ]
then
  echo "You must be root to start, stop or restart gpm."
  exit 1
fi

cmdln=
niceness=0

if [ -f $CFG ]; then
  . $CFG
  if [ -n "$device" ]; then cmdln="$cmdln -m $device"; fi
  if [ -n "$type" ]; then cmdln="$cmdln -t $type"; fi
  if [ -n "$responsiveness" ]; then cmdln="$cmdln -r $responsiveness"; fi
  if [ -n "$sample_rate" ]; then cmdln="$cmdln -s $sample_rate"; fi
  # Yes, this /IS/ correct! There is no space after -R!!!!!!
  # I reserve the right to throw manpages at anyone who disagrees.
  if [ -n "$repeat_type" ] && [ "$repeat_type" != "none" ]; then
    cmdln="$cmdln -R$repeat_type"
  fi
  if [ -n "$append" ]; then cmdln="$cmdln $append"; fi
  # If both the second device and type are specified, use it.
  if [ -n "$device2" ] && [ -n "$type2" ] ; then
    cmdln="$cmdln -M -m $device2 -t $type2"
  fi
fi

gpm_strace () {
  echo -n "Running mouse interface server under strace: gpm"
  eval strace -T -o /root/gpm.strace $GPM -V -D -e $cmdln > /root/gpm.out 2>&1
  echo "."
  return 0
}
```

```
gpm_start () {
  echo -n "Starting mouse interface server: gpm"
  eval start-stop-daemon --start --quiet --nicelevel $niceness --exec $GPM \
    -- $cmdln || { echo " start failed."; return 1; }
  echo "."
  return 0
}

gpm_stop () {
  echo -n "Stopping mouse interface server: gpm"
  $GPM -k || { echo " not running."; return 1; }
  echo "."
}


case "$1" in
  strace)
     gpm_strace || exit 1
     ;;
  start)
     gpm_start || exit 1
     ;;
  stop)
     gpm_stop || exit 1
     ;;
  force-reload|restart)
     gpm_stop && sleep 3
     gpm_start || exit 1
     ;;
  *)
     echo "Usage: /etc/init.d/gpm {start|stop|restart|force-reload|strace}"
     exit 1
esac

exit 0
```

Das Skript beginnt, indem es überprüft, ob der `gpm` installiert ist und der Aufrufer tatsächlich Rootrechte besitzt.

Debian verwendet eine eigene Konfigurationsdatei für jeden Dienst (hier: */etc/gpm.cfg*), die, falls vorhanden, eingebunden wird. Das Fehlen der Datei führt aber nicht dazu, daß der Dienst als nicht konfiguriert angesehen und deswegen nicht gestartet wird. Vielmehr wird für jeden Konfigurationsparameter getestet, ob er existiert. Aus diesen getesteten Parametern wird dann eine Kommandozeile für den `gpm` aufgebaut.

Als nächstes werden Funktionen definiert, die es erlauben, den Dienst zu starten oder zu stoppen. Diese werden in der obligatorischen Fallunterscheidung nach dem übergebenen Parameter benutzt. Die einzelnen Fälle sind knapp gehalten und beschränken sich auf eine kurze Meldung und den Aufruf bzw. das Beenden des Dienstes.

Allerdings wird hier weder durch eine Ausgabe noch durch einen Rückgabewert mitgeteilt, ob der Start des Dienstes gelungen ist.

Hardware-Integration durch Kompilation

Das eigentliche Betriebssystem stellt unter Linux der *Kernel* dar. Hierbei handelt es sich gewissermaßen um das Programm, das die Ausführung aller anderen Programme kontrolliert und diesen auch standardisierte Schnittstellen zur Hardware bietet. So braucht ein normales Programm nicht den Unterschied zwischen einer SCSI- oder einer IDE-Festplatte zu kennen. Es reicht ihm, daß es auf Dateien zugreifen kann. Um die Details kümmert sich der Kernel, der den eigentlichen Hardwarezugriff vornimmt.

Im folgenden werden wir daher einen Kernel generieren, der ganz gezielt auf unsere Bedürfnisse zugeschnitten ist. Allerdings macht es hierbei doch einen Unterschied, ob wir einen Kernel der Serie 2.2 kompilieren, der immer noch von Debian verwendet wird, oder einen Kernel der Serie 2.6, wie er mitlerweile von SuSE eingesetzt wird. Aus diesem Grund habe ich die Konfiguration des Kernels dreimal beschrieben: je eine Version für Kernel der Serie 2.2, 2.4 und 2.6. Sie können getrost die Abschnitte ignorieren, die nicht zu dem von Ihnen eingesetzten Kernel passen.

Bevor wir aber beginnen, hier noch ein paar Hinweise, die auf meinen eigenen Erfahrungen beruhen:

- Gehen Sie immer von einer möglichst simplen Grundkonfiguration aus. Es ist möglich, die Konfiguration so zu verbasteln, daß danach z. B. das Abschalten der Unterstützung für Module dazu führt, daß der Kernel beim Booten plötzlich ohne jegliche Meldung abbricht und das System einfach stehenbleibt. Verwenden Sie also `make mrproper` beim Kernel 2.2 oder 2.4 und die Befehle `make clean` und `make defconfig` beim Kernel 2.6.
- Bei Problemen sollten Sie erst einmal einen aktuellen Original-Kernel von *ftp://ftp.kernel.org* oder *ftp://ftp.de.kernel.org* verwenden. Die Distributionen liefern teilweise stark modifizierte Quellen aus. Während z. B. SuSE diverse experimentelle Zusatzfunktionen in den Kernel integriert, entfernt Debian Teile des Kernels, die nicht der Debian-Philosophie entsprechen.
- Debian 3.0 (Woody) ist nicht für die Kompilation eines 2.6er Kernels vorbereitet. Es spricht aber nichts dagegen, einen 2.4er Kernel zu verwenden. Wollen Sie einen 2.6er Kernel verwenden, so müssen Sie mehrere Softwarepakete auf den aktuellen Stand bringen oder Sarge (Version 3.1) verwenden.
- SuSE-Versionen seit 8.3 benutzen Kernel 2.6 und können teilweise keine älteren Versionen kompilieren.[6]
- Der ISDN-Support unter Kernel 2.6 ist derzeit für einen Umbau vorgesehen. Noch ist es möglich, die altbekannten HiSax-Treiber zu verwenden. In Zukunft wird angestrebt, auf CAPI-Treiber umzusteigen. Für einige aktive ISDN-Karten von AVM und Eicon Networks ist dies bereits möglich, für klassische passive Karten wie z. B. die Fritz!Card ist die Unterstützung derzeit experimentell und nicht Teil des Kernels. Wenn Sie diese nutzen wollen, müssen Sie erst die Kernelsourcen modifizieren[7].

6 Dies trifft zumindest auf 2.4er Kernel unter SuSE 8.3, 9.0 und 9.1 zu.

7 Vgl. das »Linux 2.6 and mISDN HowTo« unter *http://rcum.uni-mb.si/~uvp00845b/*

Grundlagen: Modularer vs. monolithischer Kernel

Man unterscheidet zwei verschiedene Arten, einen Kernel zu kompilieren. Zum einen können die benötigten Treiber in den Kernel hineinkompiliert werden (*monolithischer Kernel*), zum anderen können Treiber bei Bedarf nachgeladen werden (*modularer Kernel*). Letzteres ist sogar im laufenden Betrieb möglich, ohne daß neu gebootet werden müßte. (Der Menüpunkt Hardware im System integrieren im yast ist so ein Beispiel für das dynamische Einbinden von Modulen.)

Weiß man im voraus, welche Hardware man einsetzt, so ist ein schlanker monolithischer Kernel, der genau die benötigten Treiber enthält, sicherlich die effizienteste Lösung. Dies bedeutet aber auch, daß der Kernel bei jeder neu installierten Hardware (Drucker, Zip-Laufwerk, SCSI-Adapter oder ähnlichem) neu kompiliert werden muß.

Alternativ könnte man zu Beginn einen modularen Kernel kompilieren, wobei man die Unterstützung für jede Art von Hardware, die man später einmal anschaffen könnte, als Modul kompiliert. Dies erlaubt es, die Unterstützung für bestimmte Geräte im laufenden Betrieb an- oder abzuschalten. Dies geschieht, indem Code dynamisch geladen wird, der dann als Teil des Kernels arbeitet, solange er gebraucht wird. Wird er nicht mehr benötigt, wird der von ihm benötigte Speicherplatz wieder freigegeben.

Auf diese Weise kann man einen generischen Kernel zu erstellen, der auf einer Vielzahl von Systemen einsetzbar ist, indem man einfach bei Bedarf die nötigen Module nachlädt, um spezielle Hardware zu unterstützen. Die Alternative wäre, einen riesigen Kernel zu kompilieren, der alle nur denkbaren Treiber bereits enthält. Gerade für Rechner mit begrenztem Hauptspeicher ist diese Lösung aber nicht besonders günstig. Daher erfreuen sich modulare Kernel bei den Herstellern von Distributionen großer Beliebtheit.

Dieses Vorgehen hat aber auch Nachteile. So existieren Module, die die Sicherheit des Systems aushöhlen, indem sie ähnliche Funktionen erfüllen wie ein Rootkit (siehe Kapitel 4, Unterabschnitt *Sicherungsmaßnahmen*, ab Seite 42). Sie sind allerdings noch schwerer zu entdecken, da hier nicht normale Programme ausgetauscht werden, um z. B. verdächtige Prozesse zu verbergen, sondern das Betriebssystem selbst manipuliert wird. Wird auf einem System nur das Kommando `ps` ausgetauscht, so kann man immer noch ein eigenes Programm zur Anzeige der Prozesse benutzen. Verbirgt aber das Betriebssystem einen Prozeß, so gibt es keinen Weg, seine Existenz nachzuweisen.

Aus diesem Grund wäre es eigentlich sinnvoll, keine Unterstützung für das Laden von Modulen in den Kernel zu kompilieren. Bei Kerneln der Serie 2.2 bleibt uns dieser Weg leider verschlossen, da bestimmte Funktionalitäten nur als Modul kompiliert werden können (z. B. das FTP-Masquerading). Immerhin kann verhindert werden, daß der Kernel selbständig Module lädt, wenn ihm dies nötig erscheint. Wird die Einstellung »Unterstützung des Loaders für Kernelmodule« deaktiviert, so ist ein explizites Laden der Module nötig. Andernfalls[8] startet der Kernel immer dann ein externes Programm[9], wenn er auf ein Gerät zugreifen soll, für das im Moment keine Unterstützung besteht. Dieses muß dann dafür sorgen, daß das geeignete Modul nachgeladen wird.

8 Dies ist die Grundeinstellung der meisten Distributionen.

9 Standardmäßig wird `modprobe` aufgerufen. Es kann aber auch ein anderes Programm konfiguriert werden.

Im Kernel der Serie 2.4 oder 2.6 besteht diese Einschränkung nicht, falls nicht eine Hardwarekomponente konfiguriert werden muß, für die der nötige Treiber nur als Modul kompiliert werden kann.

Konfiguration eines Kernels der Serie 2.2

Um nun einen Kernel zu erzeugen, sollten wir als erstes seine gewünschten Eigenschaften festlegen. Dazu wechseln wir mit

```
# cd /usr/src/linux
```

in das Verzeichnis mit den Kernelquellen. *linux* ist dabei ein symbolischer Link auf das eigentliche Verzeichnis mit den aktuellen Quellen. Dieses hat dann normalerweise einen Namen der Art *linux-<Versionsnr.>*. Eine Ausnahme bildet hierbei Debian. Dort heißt das Verzeichnis *kernel-source-<Versionsnummer>*, und ein symbolischer Link fehlt.

Haben Sie schon einmal eine Version des Kernels kompiliert und wollen nun wieder zu den Standardeinstellungen zurückkehren, so können Sie dies mit dem folgenden Befehl erreichen:

```
# make mrproper
```

Nun können Sie den Kernel an Ihre Bedürfnisse anpassen. Dazu existiert ein menügesteuertes Konfigurationsprogramm, das Sie mit

```
# make menuconfig
```

aufrufen können.

Es empfiehlt sich, die Menüs Punkt für Punkt durchzugehen und gezielt nur Unterstützung für diejenigen Komponenten auszuwählen, die auch tatsächlich benötigt werden. Ist dies bei einer Einstellung unklar, so kann mit »?« ein Hilfetext angezeigt werden. Dort wird in der Regel eine kurze Erklärung der Option und eine Empfehlung der Art »If unsure say ‚Yes' « gegeben.

Hier einige Optionen, die für Firewalls von besonderer Wichtigkeit sind:

```
Code maturity level options
    Prompt for development [...] drivers                                  Ja
Loadable module support
    Enable loadable module support                                        Ja
    kernel module loader support                                        Nein
General setup
    Networking Support                                                    Ja
    BSD Process Accounting                                                Ja
    Sysctl Support                                                        Ja
```

```
Networking options
    Packet Socket                                                    Ja
    Kernel/User network link driver                                  Ja
    Routing messages                                                 Ja
    Network firewalls                                                Ja
    Unix domain sockets                                              Ja
    TCP/IP networking                                                Ja
    IP: advanced router                                              Ja
    IP: verbose route monitoring                                     Ja
    IP: kernel level autokonfiguration                             NEIN!
    IP: firewalling                                                  Ja
    IP: firewall netlink device                                      Ja
    IP: transparent proxying                                         Ja
    IP: masquerading                                                 Ja
    IP: ICMP masquerading                                          Nein
    IP: masquerading special module support                          Ja
    IP: ipautofw masquerading support                                Ja
    IP: ipportfw masquerading support                                Ja
    IP: ipmarkfw masquerading support                                Ja
    IP: optimize as router not host                                  Ja
    IP: SYN flood protection                                         Ja
    Fast switching                                                 NEIN!
Network device support
    Network device support?                                          Ja
    Dummy net driver support                                         Ja
    Treiber für eingebaute Netzwerkkarten                 Ja (u.U. als Modul)
    PPP (point-to-point) support                                     Ja
ISDN Subsystem
    ISDN support                                          Ja (u.U. als Modul)
    Support Synchronous PPP                                          Ja
    Nötige Treiber                                                   Ja
    Bei einer passiven Karte (HiSax):
         EURO/DSS1 oder 1TR6 anwählen
Filesystems
    fat fs support                                                   Ja
    msdos fs support                                                 Ja
    vfat fs support                                                  Ja
    ISO 9660 CDROM filesystem support                                Ja
    Microsoft Joliet filesystem extensions                           Ja
    /proc filesystem support                                         Ja
    Second Extended fs Support                                       Ja
    Network File Systems                          alle Unterpunkte abwählen
```

Zum besseren Verständnis sollte ich vielleicht noch einmal darauf eingehen, warum bestimmte Punkte nicht angewählt werden sollten:

kernel module loader support Hinter diesem Punkt verbirgt sich das schon erwähnte automatische Laden von Modulen durch den Kernel. Es ist vermutlich eine läßliche Sünde, hier »Ja« zu sagen. Wer aber nicht möchte, daß sein Kernel hinter seinem Rücken Module lädt, sollte diese Option deaktivieren.

IP: kernel level autoconfiguration Diese Einstellung bewirkt, daß der Kernel seine Netzwerkadressen beim Booten mittels DHCP erfragt. Abgesehen davon, daß nicht immer ein DHCP-Server zur Verfügung steht, ist es sinnlos, einem Server die Netzwerkadresse dynamisch zuzuweisen. Seine Klienten wüßten dann ja nicht, unter welcher Adresse sie ihn erreichen können. Schließlich ist das Protokoll unsicher, und es sind viele Angriffe denkbar, die damit beginnen könnten, einen falschen DHCP-

Server aufzusetzen und dafür zu sorgen, daß der echte kurzzeitig nicht antworten kann.

IP: ICMP masquerading Auch in diesem Fall handelt es sich mehr um eine Frage der Einstellung als um eine Todsünde. ICMP-Pakete der Rechner im lokalen Netz brauchen normalerweise nicht weitergeleitet zu werden, es sei denn, die Benutzer sehen es als unabdingbar an, die Erreichbarkeit von Rechnern im Internet mittels `ping` zu überprüfen. Tatsache ist aber, daß diese Funktion normalerweise nicht benötigt wird und deshalb gar nicht erst eingerichtet werden sollte. Damit verhindert man z. B., daß bestimmte Trojaner mit dem Internet kommunizieren können. Auch auf ICMP basierende DoS-Angriffe[10] werden so an der Firewall blockiert.

Fast Switching Ist diese Einstellung aktiv, so können bestimmte Pakete, die nicht für den Rechner selbst bestimmt sind, direkt zwischen den Netzwerkkarten ausgetauscht werden. Das Firewalling wird dabei umgangen, was in unserem Fall nicht wünschenswert wäre.

Network File Systems Netzwerkdateisysteme erlauben es, Verzeichnisse und Dateien für den Zugriff von anderen Rechnern aus freizugeben. Diese Funktionalität ist mit den hohen Sicherheitsanforderungen, die an eine Firewall zu stellen sind, unvereinbar. Weder sollte sie einem Rechner soweit vertrauen, ihm Zugriff auf die eigenen Dateien zu gewähren, noch sollten die lokalen Server ein derartiges Vertrauen in die Firewall setzen.

Es bleibt die Frage, ob Netzwerk- und ISDN-Karten als Module kompiliert werden sollten. Ist der »Loader für Kernelmodule« deaktiviert, bedeutet die Benutzung von Modulen, daß ein eigenes Skript nötig ist, um die Module beim Bootvorgang zu laden. Andererseits existieren bestimmte Treiber für ISDN-Karten, die nur als Modul funktionieren. Welche dies sind, erfährt man in der entsprechenden Kernel-Dokumentation unter */usr/src/linux/Documentation/isdn/README.<Treiber>*.

Der `yast` unter SuSE-Linux erwartet, daß Module bei der Konfiguration automatisch geladen werden. Dies soll nicht heißen, daß er nicht benutzt werden kann, wenn die Treiber in den Kernel kompiliert wurden, es ist aber mit kleineren Anpassungsschwierigkeiten zu rechnen.

Schließlich kann man mit Modulen bestimmen, in welcher Reihenfolge Netzwerkkarten eingebunden werden. Tauscht man eine Netzwerkkarte eines Herstellers gegen ein Produkt eines anderen Herstellers aus und befindet sich im Rechner eine zweite Netzwerkkarte, die von einem dritten Hersteller stammt, so kann sich die Reihenfolge, in der die Netzwerkkarten vom Kernel erkannt werden, nach dem Umbau ändern. Dies führt dazu, daß aus dem Netzwerk-Interface `eth0` mit einem Male `eth1` wird (und umgekehrt). In der Folge müßten alle Skripte, die die Interface-Bezeichnungen explizit benutzten (insbesondere das Firewalling), geändert werden. Lädt man dagegen die Module manuell, hat man es selbst in der Hand, die Reihenfolge zu bestimmen.

10 Denial-of-Service-Angriffe (DoS-Angriffe) zielen nicht darauf ab, einen Rechner unter Kontrolle zu bringen, sondern seine Funktionalität zu stören. So kann ein Server durch eine große Anzahl von ICMP-Paketen u. U. überlastet und damit außer Betrieb gesetzt werden.

Abschließend noch ein Hinweis zum HiSax-Treiber für diverse passive ISDN-Karten. Hier muß nicht nur die gewünschte Karte ausgewählt werden, man muß auch sicherstellen, daß das benutzte D-Kanal-Protokoll unterstützt wird. Früher wurde in Deutschland 1TR6 benutzt, heutzutage ist in ganz Europa DSS1 (auch *Euro-ISDN* genannt) gebräuchlich. Im Normalfall sollte es daher genügen, nur DSS1 zu konfigurieren. Wenn Sie allerdings auf Nummer Sicher gehen wollen, können Sie auch beide Protokolle einkompilieren und dann ausprobieren, mit welchem Protokoll Ihnen der Verbindungsaufbau gelingt.

Konfiguration eines Kernels der Serie 2.4

Um nun einen Kernel zu erzeugen, sollten wir als erstes seine gewünschten Eigenschaften festlegen. Dazu wechseln wir mit

```
# cd /usr/src/linux
```

in das Verzeichnis mit den Kernelquellen. *linux* ist dabei ein symbolischer Link auf das eigentliche Verzeichnis mit den aktuellen Quellen. Dieses hat dann normalerweise einen Namen der Art *linux-<Versionsnr.>*. Eine Ausnahme bildet hierbei Debian. Dort heißt das Verzeichnis *kernel-source-<Versionsnummer>*, und ein symbolischer Link fehlt.

Haben Sie schon einmal eine Version des Kernels kompiliert und wollen nun wieder zu den Standardeinstellungen zurückkehren, so können Sie dies mit dem folgenden Befehl erreichen:

```
# make mrproper
```

Nun können Sie den Kernel an Ihre Bedürfnisse anpassen. Dazu existiert ein menügesteuertes Konfigurationsprogramm, das Sie mit

```
# make menuconfig
```

aufrufen können.

Es empfiehlt sich, die Menüs Punkt für Punkt durchzugehen und gezielt nur die Unterstützung für diejenigen Komponenten auszuwählen, die auch tatsächlich benötigt werden. Ist dies bei einer Einstellung unklar, so kann mit »?« ein Hilfetext angezeigt werden. Dort wird dann in der Regel eine kurze Erklärung der Option und eine Empfehlung der Art »If unsure say ‚Yes' « gegeben.

Als erstes können wir unter »Code maturity level options« einstellen, ob wir auch Funktionalitäten im Kernel wünschen, die noch nicht ausgereift sind. Normalerweise ist dies nicht notwendig:

```
Code maturity level options
    Prompt for development [...] drivers                                    Nein
```

Allerdings wird die ISDN-Karte Fritz!Card PCI in moderneren Fassungen ab Version 2 nur unterstützt, wenn wir diese Funktion aktivieren:

```
Code maturity level options
    Prompt for development [...] drivers                                    Ja
```

Als nächstes werden wir gefragt, ob wir Funktionen als Module kompilieren wollen. Grundsätzlich würde ich davon abraten, da diese Funktion es einem lokalen Angreifer ermöglicht, spezielle Rootkits zu installieren, die kaum festzustellen sind.

Allerdings existieren bestimmte Treiber für ISDN-Karten, die nur als Modul funktionieren. Welche dies sind, erfährt man in der entsprechenden Kernel-Dokumentation unter

/usr/src/linux/Documentation/isdn/README.<Treiber>.

Für eine Fritz!Card PCI benötigen Sie z. B. keine Module, wenn Sie den neuen experimentellen Treiber benutzen.

Ein anderes Problem stellt der `yast` unter SuSE-Linux dar. Er erwartet, daß Module bei der Konfiguration automatisch geladen werden. Dies soll nicht heißen, daß er nicht benutzt werden kann, wenn die Treiber in den Kernel kompiliert wurden, es ist aber mit kleineren Anpassungsschwierigkeiten zu rechnen. Wenn Sie aber, wie hier beschrieben, eigene Skripte zur Konfiguration des Netzwerkes benutzen, so sollte Sie dies nicht betreffen.

```
Loadable module support
    Enable loadable module support                                        Nein
```

Als nächstes folgen einige notwendige Grundeinstellungen:

```
General setup
    Networking Support                                                      Ja
    BSD Process Accounting                                                  Ja
    Sysctl Support                                                          Ja
Blockorientierte Geräte
    Loopback device support                                                 Ja
Networking options
    Packet Socket                                                           Ja
    Kernel/User netlink socket                                              Ja
    Routing messages                                                        Ja
    Netlink device emulation                                                Ja
    Network packet filtering                                                Ja
    Unix domain sockets                                                     Ja
    TCP/IP networking                                                       Ja
    IP: advanced router                                                     Ja
    IP: verbose route monitoring                                            Ja
    IP: kernel level autokonfiguration                                   NEIN!
    IP: TCP syncookie support                                               Ja
    IP: Netfilter Configuration --->
        Connection Tracking [...]                                           Ja
            FTP protocol support                                            Ja
        IPtables support[...]                                               Ja
            limit match support                                             Ja
            MAC address support                                             Ja
            netfilter MARK match support                                    Ja
            TOS match support                                               Ja
            Connection state match support                                  Ja
```

```
        Packet filtering                                        Ja
            REJECT target support                               Ja
        Full NAT                                                Ja
            MASQUERADE target support                           Ja
            REDIRECT target support                             Ja
        Packet mangeling                                        Ja
            TOS target support                                  Ja
            MARK target support                                 Ja
        LOG target support                                      Ja
    The IPX protocol                                            Nein
    Appletalk protocol Support                                  Nein
    DECnet support                                              Nein
    802.1d Ethernet Bridging                                    Nein
```

Bevor wir weitermachen, noch ein paar Worte zu den Einstellungen, von denen hier abgeraten wird:

IP: kernel level autoconfiguration Diese Einstellung bewirkt, daß der Kernel seine Netzwerkadressen beim Booten mittels DHCP erfragt. Abgesehen davon, daß nicht immer ein DHCP-Server zur Verfügung steht, ist es sinnlos, einem Server die Netzwerkadresse dynamisch zuzuweisen. Seine Klienten wüßten dann ja nicht, unter welcher Adresse sie ihn erreichen können. Schließlich ist das Protokoll unsicher, und es sind eine Menge Angriffe denkbar, die damit beginnen könnten, einen falschen DHCP-Server aufzusetzen und dafür zu sorgen, daß der echte kurzzeitig nicht antworten kann.

IPX, Appletalk, DECnet support Hier handelt es sich um diverse Netzwerkprotokolle. Diese werden aber vom Firewalling nicht unterstützt, das nur IP, ICMP, UDP und TCP kennt. Normalerweise besteht auch kein Grund, diese Protokolle in das Internet zu routen. Sie sind nur für den Betrieb im LAN vorgesehen.

802.1d Ethernet Bridging Beim Bridging tritt ein Rechner gar nicht in Erscheinung, er wirkt vielmehr wie ein Verstärker zwischen zwei Netzwerksegmenten, der Pakete aus einem Segment transparent in das andere leitet und umgekehrt. Hier wollen wir dagegen einen Router aufsetzen, der von Rechnern im LAN gezielt angesprochen wird, wenn Pakete in das Internet geroutet werden sollen.

Kommen wir nun zu den Netzwerkkarten:

```
Network device support
    Network device support?                                     Ja
    Dummy net driver support                                    Ja
    Treiber für eingebaute                          Ja (u.U. als Modul)
    Netzwerkkarten
```

Wenn Sie ein Modem oder eine DSL-Anbindung benutzen wollen, so müssen Sie das Point-to-Point Protocol aktivieren:

```
    PPP (point-to-point) support                                Ja
        PPP support for async serial ports                      Ja
        PPP support for sync tty ports                          Ja
        PPP Deflate compression                                 Ja
        PPP BSD-Compress compression                            Ja
```

Wenn Sie sich über ISDN in das Internet einwählen wollen, so benötigen Sie die folgenden Einstellungen. Andernfalls können Sie »ISDN support« abwählen:

```
ISDN Subsystem
    ISDN support                                          Ja (u. U. als Modul)
    Support Synchronous PPP                                                Ja
    Use VJ compression with synchronous PPP
    Support generic MP [...]                                               Ja

    Nötige Treiber                                                         Ja
```

Beim HiSax-Treiber für diverse passive ISDN-Karten muß nicht nur die gewünschte Karte ausgewählt werden, es gilt auch sicherzustellen, daß das benutzte D-Kanal-Protokoll unterstützt wird. Früher wurde in Deutschland 1TR6 benutzt, heutzutage ist in ganz Europa DSS1 (auch *Euro-ISDN* genannt) gebräuchlich. Im Normalfall sollte es daher genügen, nur DSS1 zu konfigurieren. Wenn Sie allerdings auf Nummer Sicher gehen wollen, können Sie auch beide Protokolle einkompilieren:

```
    Passive ISDN cards
        HiSax Support for EURO/DSS1                                        Ja
        HiSax Support for german 1TR6                                      Ja
```

Bitte beachten Sie, daß für die Fritz!Card PCI zwei Treiber existieren. Das liegt daran, daß es mehrere Versionen dieser Karte gibt. Karten, die heutzutage verkauft werden, tragen die Versionsnummer 2 oder höher. Diese werden von dem alten Treiber (`AVM PNP/PCI (Fritz!PnP/PCI)`) nicht unterstützt. Es gibt einen neuen experimentellen Treiber, den Sie aktivieren, wenn Sie folgende Einstellung treffen:

```
        AVM Fritz!Card PCI/PCIv2/PNP support                               Ja
```

Als letztes können wir noch einstellen, welche Dateisystem-Typen wir unterstützen wollen. Zwar werden wir auf dem Rechner nicht zusätzlich ein Windows NT installiert haben und daher auf Unterstützung für NTFS verzichten können, aber eine Unterstützung für Disketten, die unter einem Microsoft-Betriebssystem formatiert wurden, ist manchmal praktisch.

Wir sollten aber keine Netzwerkdateisysteme wie z. B. SMB (Windows-Freigaben) oder NFS (das Unix-Äquivalent) in den Kernel kompilieren. Diese haben auf einer Firewall nichts zu suchen, da die von ihnen benutzten Protokolle inhärent unsicher sind.

```
Filesystems
    fat fs support                                                         Ja
        msdos fs support                                                   Ja
        VFAT (Windows 95) fs support                                       Ja
    ISO 9660 CDROM filesystem support                                      Ja
        Microsoft Joliet filesystem extensions                             Ja
    /proc filesystem support                                               Ja
    Second Extended fs Support                                             Ja
    Network File Systems                              alle Unterpunkte abwählen
```

Konfiguration eines Kernels der Serie 2.6

Zuerst einmal sollten wir in das Verzeichnis mit den Kernelquellen wechseln. Üblicherweise heißt es

/usr/src/linux-<*Versionsnr.*>

Unter Debian folgt man allerdings einer eigenen Konvention. Dort heißt es

/usr/src/kernel-source-<*Versionsnummer*>

Teilweise kommt es auch vor, daß die Quellen als gepacktes Tar-Archiv unter */usr/src* abgelegt sind. In diesem Fall müssen Sie sie noch mit

```
# tar xvzf <Name>-<Versionsnr.>.tar.gz
```

bzw.

```
# bunzip2 <Name>-<Versionsnr.>.tar.bz2
# tar xvf <Name>-<Versionsnr.>.tar
```

entpacken.

Bringt Ihre Distribution die Kernelquellen des 2.6er Kernels nicht von Haus aus mit, so besteht die Gefahr, daß auch die Systemprogramme noch nicht aktuell genug sind. Haben Sie also die Quellen direkt von *ftp://ftp.kernel.org* oder *ftp://ftp.de.kernel.org* heruntergeladen, so vergewissern Sie sich, daß alle Programme auf dem aktuellen Stand sind. Eine Aufstellung der benötigten Programmversionen finden Sie in der Datei

<*Inst.-Verzeichnis*>*/Documentation/Changes*

Sehen Sie dabei genau hin, was `modprobe -V` anzeigt. Die Ausgabe sollte die Bezeichnung `module-init-tools` enthalten. Kernel der Serien 2.2 und 2.4 benutzen statt dessen die `modutils`. Diese können aber keine Module eines Kernels der Serie 2.6 laden. Darüber hinaus haben die `module-init-tools` derzeit eine sehr niedrige Versionsnummer, die deutlich unter der der aktuellen `modutils` liegt. Wenn man nicht genau hinsieht und nur die Versionsnummer wahrnimmt, dann kann man leicht davon ausgehen, daß man die richtige Version hat, obwohl dies nicht der Fall ist.

Haben Sie bereits vorher versucht, einen Kernel zu konfigurieren, dann bietet es sich an, als erstes einen definierten Ausgangszustand herzustellen. Mit

```
# make clean
```

werden eventuelle Reste einer früheren Kompilation entfernt. Benutzen Sie dagegen

```
# make mrproper
```

so wird zusätzlich auch eine eventuell vorhandene Konfiguration aus einem früheren Versuch gelöscht.[11]

11 Eine Übersicht, welche Targets sie `make` noch geben können, erhalten Sie, wenn Sie »`make help`« eingeben.

Danach können Sie mit folgendem Aufruf eine Standardkonfiguration generieren:

```
# make defconfig
```

Rufen Sie den Befehl ruhig auch bei der allerersten Installation auf, um sicherzugehen, das Sie auch wirklich von demselben Grundzustand ausgehen, den ich im folgenden beschreibe. Auf meinem Debiansystem wurden ansonsten die Vorgaben des installierten 2.2er Kernels übernommen.

Nach diesen Vorbereitungen können wir nun beginnen, den neuen Kernel zu konfigurieren. Dies geschieht mit dem folgenden Befehl:

```
# make menuconfig
```

Mit ihm starten Sie eine menügesteuerte Anwendung, in der Sie jetzt gezielt auswählen können, welche Eigenschaften Ihr zukünftiger Kernel aufweisen soll.

Wenn Sie dies das erste Mal machen oder vorher nur 2.4er Kernel kompiliert haben, dann empfehle ich Ihnen dringend, jeden einzelnen Punkt durchzugehen. Es ist durchaus möglich, sich einen Kernel ohne jede Unterstützung für Tastatur und Monitor zu kompilieren.

Sie sollten sicherstellen, daß alle benötigten Komponenten vom Kernel unterstützt werden. Andererseits sollten Sie aber keine unnötigen Features einkompilieren. Wenn Sie im Einzelfall nicht wissen, wofür eine Option gut ist, können Sie sich mit »?« einen Hilfetext anzeigen lassen. Dort wird dann in der Regel eine kurze Erklärung der Option und eine Empfehlung der Art »If unsure say ‚Yes' « gegeben.

Als erstes können wir unter »Code maturity level options« einstellen, ob wir auch Funktionalitäten im Kernel wünschen, die noch nicht ausgereift sind. Diese Funktionalitäten sind teilweise im Alpha-Stadium und können durchaus das System instabil machen.

Früher war dies nötig, da das Firewalling teilweise als »Experimentell« gekennzeichnet war. Heute ist es allerdings ausgereift, so daß wir hier »Nein« sagen können, wenn wir nicht Unterstützung für spezielle neue Hardware wie z. B. eine aktuelle Fritz!Card PCI benötigen.

```
Code maturity level options
    Prompt for development [...] drivers                                    Nein
```

Unter »General Setup« finden wir die Funktion »BSD Process Accounting«. Wenn wir diese aktivieren, können wir später mit dem Befehl `accton` eine Datei festlegen, in der alle Programmaufrufe protokolliert werden. Mit `lastcomm` kann man sich diese dann später anzeigen lassen. Im Normalfall ist dies übertrieben, aber es kann nicht schaden, diese Möglichkeit vorzubereiten.

Des weiteren sind dort zwei Optionen, die dafür sorgen, daß die Konfigurationseinstellungen des Kernels als gepackte Datei unter */proc* ausgelesen werden können. Ich halte dies für sehr sinnvoll, da man so später diese Einstellungen nachlesen kann, auch wenn man vergessen hat, alle Einstellungen bei der Konfiguration zu dokumentieren und sauber zu archivieren.

```
General Setup
    BSD Process Accounting                                              Ja
    Kernel .config support                                              Ja
    Enable access [...] through /proc/config.gz                         Ja
```

Als nächstes werden wir gefragt, ob wir Funktionen als Module kompilieren wollen. Im 2.6er Kernel ist dies normalerweise nicht mehr nötig. Daher würde ich Ihnen davon abraten, um so eine mögliche Schwachstelle gleich von Anfang an auszuräumen.

```
Loadable module support
    Enable loadable module support                                      Nein
```

Allerdings existieren bestimmte Treiber, die nur als Modul funktionieren. Auch mISDN, die neue Unterstützung für passive ISDN-Karten, muß als Modul kompiliert werden.

Benötigen Sie so einen Treiber, dann müssen Sie natürlich Modulsupport anwählen. Allerdings sollte neue Hardware nicht automatisch beim ersten Zugriff eingebunden werden, sondern nur dann, wenn Sie diese explizit aktivieren, indem Sie das Modul laden:

```
Loadable module support
    Enable loadable module support                                      Ja
    Automatic kernel module loading                                     Nein
```

Auch der yast unter SuSE-Linux erwartet, daß Module bei der Konfiguration automatisch geladen werden. Dies soll nicht heißen, daß er nicht benutzt werden kann, wenn die Treiber in den Kernel kompiliert wurden, es ist aber mit kleineren Anpassungsschwierigkeiten zu rechnen. Wenn Sie aber, wie hier beschrieben, eigene Skripte zur Konfiguration des Netzwerkes benutzen, so sollte Sie dies nicht betreffen.

Als nächstes sollten wir den richtigen Prozessor einstellen. Standardmäßig ist hier ein Pentium 4 eingestellt:

```
Processor type and Features
    passenden Prozessor anwählen
```

Kommen wir nun zu den Treibern für verschiedene Geräte. Als erstes finden wir Optionen für Speichermedien.

Hierbei ist zu beachten, daß sich die Unterstützung für CD-Brenner geändert hat. War es früher nötig, »SCSI emulation support« und »SCSI generic support« auszuwählen, so reicht es beim Kernel 2.6, nur »Include IDE/ATAPI CDROM support« zu aktivieren. Das Gerät kann dann als ganz normales IDE-Device */dev/hd<X>* angesprochen werden, während es vorher als SCSI-Device unter */dev/sr<N>* firmierte.

Wir sollten ferner die Unterstützung für den DMA-Modus und den Chipsatz des von uns verwendeten IDE-Controllers einkompilieren. Nur so können wir die Festplatten im schnelleren DMA-Modus betreiben.

```
Device Drivers
    Block Devices
        Loopback device support                                    Ja
        Network block device support                               Nein
    ATA/ATAPI/MFM/RLL support
        Include IDE/ATAPI CDROM support                            Ja
        oder
        SCSI emulation support                                     Ja
        Generic PCI bus-master DMA Support                         Ja
        Treiber für Ihren IDE-Controller                           Ja
    SCSI device support
        SCSI generic support                                       Ja
                                                        (bei SCSI emulation)
```

Unter »Networking support« finden wir neben einigen Standardeinstellungen auch die Einstellungen zur Paketfilterung:

```
Networking support
    Networking options
    TCP/IP networking                                              Ja
    IP: advanced router                                            Ja
    IP: verbose route monitoring                                   Ja
    IP: kernel level autoconfiguration                          NEIN!
    IP: TCP Expl. Cong. Not. Support                               Nein
    IP: TCP syncookie support                                      Ja
    DECnet support                                                 Nein
    802.1d Ethernet Bridging                                       Nein
    Network packet filtering
        IP: Netfilter Configuration --->
            Connection Tracking [...]                              Ja
                FTP protocol support                               Ja
                TFTP [...] support                              NEIN!
                Amanda [...] support                            NEIN!
            IPtables support[...]                                  Ja
                Alle anwählen, insbesondere:
                limit match support                                Ja
                IP range match support                             Ja
                MAC address support                                Ja
                TOS match support                                  Ja
                Connection state [...]                             Ja
                Owner match support                                Ja
                Packet filtering                                   Ja
                        REJECT [...]                               Ja
                Full NAT                                           Ja
                        MASQUERADE [...]                           Ja
                    REDIRECT [...]                                 Ja
                LOG target support                                 Ja
    The IPX protocol                                               Nein
    Appletalk protocol Support                                     Nein
```

Bevor wir weitermachen, noch ein paar Worte zu den Einstellungen, von denen hier abgeraten wird:

IP: kernel level autoconfiguration Diese Einstellung bewirkt, daß der Kernel seine Netzwerkadressen beim Booten mittels DHCP erfragt. Abgesehen davon, daß nicht immer ein DHCP-Server zur Verfügung steht, ist es sinnlos, einem Server die Netzwerkadresse dynamisch zuzuweisen. Seine Klienten wüßten dann ja nicht, unter welcher Adresse sie ihn erreichen können. Schließlich ist das Protokoll unsicher, und es sind eine Menge Angriffe denkbar, die damit beginnen könnten, einen falschen DHCP-Server aufzusetzen und dafür zu sorgen, daß der echte kurzzeitig nicht antworten kann.

802.1d Ethernet Bridging Beim Bridging tritt ein Rechner gar nicht in Erscheinung, er wirkt vielmehr wie ein Verstärker zwischen zwei Netzwerksegmenten, der Pakete aus einem Segment transparent in das andere leitet und umgekehrt. Hier wollen wir dagegen einen Router aufsetzen, der von Rechnern im LAN gezielt angesprochen wird, wenn Pakete in das Internet geroutet werden sollen.

TFTP, Amanda Backup Protocol TFTP wird von Rechnern ohne Festplatte verwendet, um alle benötigten Dateien herunterzuladen. Das Protokoll verfügt über keine nennenswerten Authentisierungsmechanismen und sollte auf keinen Fall durch die Firewall gelassen werden. In beide Richtungen wäre dies katastrophal. Weder ist es sinnvoll, daß die Rechner im lokalen Netz Systemdateien aus dem Internet herunterladen, die möglicherweise manipuliert sind, noch wollen Sie es einem Rechner im Internet erlauben, sich eine Paßwortdatei von einem Ihrer Server herunterzuladen.

Für das Amanda Backup Protocol gelten ähnliche Überlegungen. Im Normalfall wollen Sie doch Ihre Backups nicht auf einem Rechner im Internet durchführen.

IPX, Appletalk, DECnet support Hier handelt es sich um diverse Netzwerkprotokolle. Diese werden aber vom Firewalling nicht unterstützt, das nur IP, ICMP, UDP und TCP kennt. Normalerweise besteht auch kein Grund, diese Protokolle in das Internet zu routen. Sie sind nur für den Betrieb im LAN vorgesehen.

Kommen wir nun zur Unterstützung für Netzwerkkarten:

```
Network device support
    Network device support?                                    Ja
    Dummy net driver support                                   Ja
    Treiber für eingebaute                          Ja (u. U. als Modul)
    Netzwerkkarten
```

Für die Internet-Anbindung über Modem oder DSL brauchen Sie auch noch das Point-to-Point Protocol:

```
    PPP (point-to-point) support                               Ja
        PPP filtering                                          Ja
        PPP support for async serial ports                     Ja
        PPP support for sync tty ports                         Ja
        PPP Deflate compression                                Ja
        PPP BSD-Compress compression                           Ja
```

Wenn Sie sich über ISDN in das Internet einwählen wollen, so benötigen Sie die folgenden Einstellungen. Andernfalls können Sie »ISDN support« abwählen:

```
ISDN Subsystem
    ISDN support                                          Ja (u. U. als Modul)
```

In Kernel 2.6 hat man begonnen, die Unterstützung für ISDN-Karten auf ein neues Treibermodell namens CAPI v2.0 umzustellen. Auf diese Weise werden derzeit aber nur einige wenige Karten unterstützt. Für die Mehrzahl benötigen Sie immer noch die alten ISDN4Linux-Treiber, wie sie bereits im Kernel 2.4 enthalten waren:

```
        Old ISDN4Linux
            Support Synchronous PPP                                          Ja
            Use VJ compression with synchronous PPP                          Ja
            Support generic MP [...]                                         Ja
```

Beim HiSax-Treiber für diverse passive ISDN-Karten muß nicht nur die gewünschte Karte ausgewählt werden, es gilt auch sicherzustellen, daß das benutzte D-Kanal-Protokoll unterstützt wird. Früher wurde in Deutschland 1TR6 benutzt, heutzutage ist in ganz Europa DSS1 (auch *Euro-ISDN* genannt) gebräuchlich. Im Normalfall sollte es daher genügen, nur DSS1 zu konfigurieren. Wenn Sie allerdings auf Nummer Sicher gehen wollen, können Sie auch beide Protokolle einkompilieren und dann ausprobieren, mit welchem Protokoll Ihnen der Verbindungsaufbau gelingt:

```
            Passive Cards
                HiSax Support for EURO/DSS1                                  Ja
                HiSax Support for german 1TR6                                Ja
```

Nun müssen Sie den geeigneten Treiber für Ihre Karte auswählen. Wenn Sie eine Fritz!Card PCI benutzen, so müssen Sie wissen, daß AVM vor geraumer Zeit eine neue Version mit der Versionsnummer 2 herausgebracht hat. Diese Karte unterscheidet sich von ihrer Vorgängerin deutlich, so daß der klassische Treiber für die Fritz!Card PCI mit der Fritz!Card PCI v2 nicht mehr funktioniert. Es existiert aber ein neuer experimenteller Treiber, mit dem diese Karte problemlos funktioniert. Dieser Treiber hat auch den Vorteil, daß er im Gegensatz zu seinem Vorgänger nicht als Modul kompiliert zu werden braucht:

```
            AVM Fritz!Card PCI/PCIv2/PNP                                     Ja
            support (EXPERIMENTAL)
```

Unter »Input device support« werden die Eingabegeräte konfiguriert. Hier sollten wir darauf achten, daß unsere Tastatur und gegebenenfalls Maus unterstützt wird.

```
    Input device support
        serial port line discipline                                          Ja
        Mice                                                                 Ja
        PS/2 mouse                                                           Ja
        Serial Mouse                                          Ja (falls vorhanden)
        Misc                                                                 Ja
        PC Speaker support                                                   Ja
```

Nachdem wir die Gerätetreiber ausgewählt haben, können wir nun einstellen, welche Dateisystem-Typen wir unterstützen wollen. Zumindest das Second Extended File System sollte als unser Standarddateisystem auch vom Kernel unterstützt werden.

Zwar werden wir auf dem Rechner nicht zusätzlich ein Windows NT installiert haben und daher auf Unterstützung für NTFS verzichten können, aber eine Unterstützung für Disketten und CDs, die unter einem Microsoft-Betriebssystem formatiert wurden, ist manchmal praktisch. Allerdings war sowohl VFAT- als auch Joliet-Unterstützung standardmäßig angewählt.

Wir sollten aber keine Netzwerkdateisysteme wie z. B. SMB (Windows-Freigaben) oder NFS (das Unix-Äquivalent) in den Kernel kompilieren. Diese haben auf einer Firewall nichts zu suchen, da die von ihnen benutzten Protokolle inhärent unsicher sind.

```
Filesystems
    Second Extended fs Support                                    Ja
    Kernel automounter version 4 support                          Nein
    Network File Systems                          alle Unterpunkte abwählen
```

Kernelkompilation

Nachdem wir festgelegt haben, wie der zukünftige Kernel aussehen soll, müssen wir ihn nun noch kompilieren. Mit den Kerneln der Serie 2.6 wurde auch die Kernelkompilation überarbeitet. Daher wird sie im folgenden in einem eigenen Abschnitt besprochen. Die Kernel der Serie 2.2 und 2.4 unterscheiden sich hingegen nicht im Vorgehen und werden daher hier zusammen beschrieben.

Kernel der Serien 2.2. und 2.4

Zuerst werden mit

```
# make dep
```

Abhängigkeiten überprüft, worauf dann mit

```
# make clean
```

die Überreste früherer Kompilationen gelöscht werden.

Nun folgt die eigentliche Kompilation des Kernels. Dieser kann in zwei Formen vorkommen: als zImage und als bzImage. In beiden Fällen darf er eine bestimmte Größe nicht überschreiten. Diese Grenze liegt im ersten Fall so niedrig, daß leicht Fehlermeldungen der Art »Kernel too big« auftreten können. Im zweiten Fall ist die Maximalgröße des Kernels dagegen deutlich höher angesetzt und wird daher in der Regel nicht erreicht. Der kompilierte und komprimierte Kernel kann unter */usr/src/linux/arch/i386/boot/* gefunden werden.

Je nach Situation wird einer der folgenden Aufrufe zu dem gewünschten Resultat führen[12]:

`make zImage`	Kompilation des Kernels als zImage und Ablage im oben genannten Verzeichnis
`make bzImage`	wie zuvor beschrieben als bzImage
`make zlilo`	Kompilation des Kernels und Installation für LiLo. Dabei wird der bisher installierte Kernel überschrieben. Wurde bei der Konfiguration ein Fehler begangen, so ist das System nur noch mit einer speziellen Bootdiskette zu starten.
`make bzlilo`	s. o. als bzImage
`make zdisk`	Kompilation des Kernels und Installation auf eine Diskette (Bootdisk). Damit können wir erst einmal ausprobieren, ob der Kernel die in ihn gesetzten Erwartungen erfüllt, bevor man ihn permanent im System installiert.
`make bzdisk`	s. o. als bzImage

Für den ersten Versuch ist es sinnvoll, mit `make zImage` oder `make bzImage` einen Kernel zu kompilieren[13], um dann erst einmal ein Backup des alten Kernels zu erzeugen und den Bootloader zu konfigurieren (siehe Kapitel 8, Unterabschnitt *Installation des Kernels*, ab Seite 154). So kann man das System immer noch mit dem alten Kernel booten, wenn bei der Konfiguration des neuen Kernels ein Fehler begangen wurde.

Alternativ besteht auch die Möglichkeit, mit `make zdisk` oder `make bzdisk` den Kernel »roh« auf eine Diskette zu schreiben und dann von dieser zu booten. Hierbei wird der Bootloader nicht benutzt. Dies bietet die Möglichkeit, »mal eben« eine neue Kernelkonfiguration auszuprobieren, ohne bleibende Schäden am System zu hinterlassen. Allerdings können dem Kernel bei dieser Variante keine Parameter mitgegeben werden.

Haben Sie schon einen Kernel kompiliert und besitzen eine Backup-Konfiguration, so können Sie nach kleineren Änderungen auch `make zlilo` oder `make bzlilo` benutzen. Sie besitzen ja im Zweifelsfall den Backup-Kernel, falls der neukompilierte nicht booten will. Kompilieren Sie allerdings einen neuen Kernel, weil sich die Hardware grundlegend geändert hat, so sollten Sie vorher sicherstellen, daß der Backup-Kernel immer noch in der Lage ist, das System soweit hochzufahren, daß ein neuer Kernel kompiliert werden kann.

Bei der Benutzung von `make [b]zlilo` ist auch zu beachten, daß der Automatismus so eingerichtet ist, daß der neue Kernel unter dem Namen *vmlinuz* nach */$INSTALLPATH* kopiert wird. Existiert dort bereits eine Datei *vmlinuz*, so wird diese vorher in *vmlinuz.old* umbenannt.

12 Es braucht nur einer der Aufrufe eingegeben zu werden. Für eine normale LiLo-Installation z. B. `make [b]zlilo`.

13 In der Regel wird man die zweite Form benötigen.

Damit nun der Kernel wie gewohnt nach */boot* installiert wird, muß allerdings in */usr/src/linux/Makefile* die Zeile

```
INSTALL_PATH=/boot
```

existieren, damit der Kernel als */boot/vmlinuz* abgelegt wird. Bei der SuSE-Version des Kernels ist dies in der Regel der Fall. Lädt man aber die originalen Kernelquellen z. B. von *ftp://ftp.de.kernel.org/*, so ist diese oft auskommentiert:

```
#INSTALL_PATH=/boot
```

Der Kernel wird dann direkt unter / abgelegt.

Kernel der Serie 2.6

In der Serie 2.6 entfallen die Aufrufe `make dep`, `make zImage`, `make zdisk` und `make zlilo`. Auch `make bzlilo` ist durch einen neuen Aufruf mit abweichender Semantik ersetzt worden.

Es reicht nun, `make` ohne Parameter aufzurufen, um den Kernel und die Module zu bauen. Um die Module zu installieren, muß auch weiterhin `make modules_install` aufgerufen werden. Aber dazu mehr im nächsten Abschnitt.

Hier die wichtigsten Befehle im Überblick. Eine vollständige Liste zeigt der Befehl `make help` an:

`make`	Kompilation des Kernels und der Module, ohne sie zu installieren
`make install`	Installation des Kernels. Dabei wird der bisher installierte Kernel überschrieben. Wurde bei der Konfiguration ein Fehler gemacht, so ist das System nur noch mit einer speziellen Bootdiskette zu starten.
`make bzdisk`	Kompilation des Kernels und Installation auf eine Diskette (Bootdisk). Wenn das Booten von der Diskette funktioniert, können wir den Kernel installieren. Andernfalls nehmen wir die Diskette heraus, booten normal von der Festplatte und starten einen neuen Versuch, einen passenden Kernel zu konfigurieren.
`make fdimage`	Es wird nicht direkt auf eine Diskette, sondern in eine Image-Datei geschrieben.

`make install` sollten Sie nur benutzen, wenn Sie bereits eine Backup-Konfiguration erstellt haben, zu der Sie zurückkehren können, falls Ihr neukompilierter Kernel nicht lauffähig ist. Für den ersten Versuch kompilieren Sie den Kernel besser mit `make`, machen eine Kopie des alten Kernels und kopieren den neuen dann manuell (siehe Kapitel 8, Unterabschnitt *Installation des Kernels*, ab Seite 154).

Technisch ist `make install` etwas komplizierter als in früheren Versionen `make bzlilo`. Es wird zuerst versucht, ein Programm `/bin/installkernel` aufzurufen. Wenn dieses nicht existiert, wird als zweites `/sbin/installkernel` versucht. Wenn dieses ebenfalls nicht gefunden wird, wird der Kernel in das Verzeichnis kopiert, das in der Variable *INSTALL-*

PATH vorgegeben ist. Anschließend wird `/sbin/lilo` bzw., wenn dieses Programm nicht existiert, `/etc/lilo/install` aufgerufen.

Der Kernel wird dabei in `vmlinuz` umbenannt. Existiert im Zielverzeichnis bereits eine Datei namens `vmlinuz`, wird selbige in `vmlinuz.old` umbenannt. Eine eventuell vorhandene Datei `vmlinuz.old` wird dabei gelöscht.[14]

Die Variable *INSTALLPATH* wird in der Datei *Makefile* im Hauptverzeichnis gesetzt. Bei den Originalkerneln von *ftp.kernel.org* oder *ftp.de.kernel.org* ist diese Zuweisung in der Regel auskommentiert, so daß der Kernel direkt im Rootverzeichnis als `/vmlinuz` abgelegt wird:

```
#INSTALL_PATH=/boot
```

Wollen wir, daß er statt dessen in */boot/* installiert wird, so müssen wir die Zeile aktivieren:

```
INSTALL_PATH=/boot
```

Ist unser Kernel klein genug[15], dann können wir statt dessen auch eine Diskette in unser erstes Diskettenlaufwerk[16] (*/dev/fd0*) einlegen und `make bzdisk` benutzen. Damit sollte es dann möglich sein, das System von der so erzeugten Diskette zu booten. Funktioniert der neue Kernel nicht wie gewünscht, braucht man nur die Diskette herauszunehmen und neu zu booten. Schon ist alles wie zuvor.

Kompilation und Installation der Module

Nun haben wir einen auf unsere Bedürfnisse zugeschnittenen Kernel. Was jetzt noch fehlt, sind die benötigten Module. Wenn bestimmte Komponenten des Kernels, die wir für unsere Zwecke benötigen, nur als Module verfügbar sind, kommen wir nicht darum herum, auch die gewählten Module zu kompilieren und zu installieren.

Dazu erfolgt bei Kerneln der Serien 2.2 und 2.4 mit

```
# make modules
```

der Bau der modularen Kernelkomponenten. Beim Kernel 2.6 entfällt dies, da die Module automatisch mitkompiliert wurden.

Anschließend wird mit

```
# make modules_install
```

die Installation der fertigen Module in die Wege geleitet.

14 Vgl. *arch/i386/boot/install.sh*

15 Dies ist keine geringe Einschränkung. Ein monolithischer Kernel, wie ich ihn hier beschreibe, erfüllt diese Bedingung wahrscheinlich nicht.

16 Der Rechner, auf dem ich dies schreibe, hat tatsächlich derer noch zwei.

Schließlich muß für die Module mit

```
# depmod -a
```

eine Liste mit Abhängigkeiten erzeugt werden. Nun können die Module jederzeit mit

```
# modprobe modul_name [Parameter...]
```

geladen werden.

```
# lsmod
```

zeigt die geladenen Module an, und

```
# rmmod [-s] [-r] modul_name
```

entfernt ein Modul wieder aus dem Speicher. Die Option -s gibt dabei an, daß Ausgaben mittels Syslog protokolliert und nicht direkt auf das Terminal ausgegeben werden sollen. Die Option -r bewirkt, daß auch versucht wird, Module zu entfernen, die von dem zu entfernenden Modul benutzt werden.

Etwas komplizierter wird es, wenn man eine neue Version des Kernels kompiliert hat. In diesem Fall wird der oben genannte Aufruf von `depmod` die Module nicht finden, da er nach Modulen für den gerade aktiven Kernel sucht. Obwohl man dem Befehl explizit eine Versionsnummer mitteilen kann (z. B. `depmod -a 2.2.13`), ist es besser, ihn erst nach dem nächsten Booten ausführen zu lassen. Oft ist ein entsprechender Aufruf schon in den Runlevel-Skripten eingebaut.

Unterbleibt der Aufruf von `depmod -a` ganz, so wird sich dies dadurch bemerkbar machen, daß all diejenigen Treiber, die als Module kompiliert wurden, nach dem nächsten Neustart nicht mehr zur Verfügung stehen.

Installation des Kernels

Nun da wir einen neuen Kernel kompiliert haben, müssen wir ihn nur noch an eine geeignete Stelle kopieren und den Bootloader so umkonfigurieren, daß er ihn auch benutzt.

Kopieren des Kernels

Bevor wir den neuen Kernel installieren, ist es sinnvoll, den Vorgänger aufzuheben. So besitzt man eine funktionierende Konfiguration, zu der man zurückkehren kann, falls der neue Kernel nicht lauffähig ist. Dies kann z. B. der Fall sein, wenn versehentlich zum Booten benötigte Hardware (z. B. SCSI-Adapter, IDE-Festplatten, Ext2-Dateisystem) nicht einkompiliert wird.

Hierzu benennt man den alten Kernel um:

```
# mv /boot/vmlinuz /boot/vmlinuz.prev
```

Bitte benutzen Sie nicht *vmlinuz.old*, da `make [b]zlilo` diese Bezeichnung ebenfalls für die jeweilige Vorgängerversion benutzt und damit die eigene Version überschreibt. Kompiliert man dann den Kernel versehentlich zweimal hintereinander, so ist die Sicherungskopie mit der aktuellen Kernelversion identisch und damit unbrauchbar.

Nun kann man den neuen Kernel mit

```
# cp /usr/src/linux/arch/i386/boot/zImage /boot/vmlinuz
```

oder

```
# cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz
```

umkopieren.

Bootoptionen

Nun kann es vorkommen, daß Treiber für eine bestimmte Hardware in den Kernel kompiliert wurden, das Gerät aber beim Booten nicht erkannt wird. Dies ist insbesondere dann der Fall, wenn mehr als eine Netzwerkkarte des gleichen Typs installiert wurde. Hier bricht der Kernel in der Regel die Suche ab, sobald er eine Karte gefunden hat. Auch lassen sich Karten oft auf diverse Ports und Interrupts konfigurieren, auf denen der Kernel unter Umständen nicht standardmäßig nach ihnen sucht.

Ein solcher Fall ist der Treiber für NE2000-kompatible ISA-Netzwerkkarten. Dieser findet Karten nur dann, wenn sie Port 0x300 benutzen. Die Ursache für dieses Verhalten liegt in der Tatsache, daß die Kommandos, die der Treiber benutzen muß, um festzustellen, ob ein bestimmter Port von einer solchen Netzwerkkarte benutzt wird, ältere SCSI-Hostadapter zum Absturz bringen können. Unglücklicherweise gibt es aber eine Reihe von Ports, auf die sowohl NE2000-Karten als auch besagte SCSI-Adapter eingestellt werden können. Der Treiber begnügt sich daher mit Port 0x300, auf den NE2000-Karten normalerweise voreingestellt sind.

Um dem Kernel nun abweichende Parameter mitzuteilen, existiert eine Reihe von Optionen, die man sich unter SuSE-Linux mit

```
> zcat /usr/share/doc/howto/en/txt/BootPrompt-HOWTO.gz | less
```

bzw. unter Debian mit

```
> zcat /usr/share/doc/HOWTO/en-txt/BootPrompt-HOWTO.gz | less
```

anzeigen lassen kann.

Auch die Kernelquellen enthalten eine Aufstellung von Parametern. Sie finden sie im Unterverzeichnis *Documentation/kernel-parameters.txt*.

Für ISDN-Karten kann es allerdings sinnvoll sein, sich die aktuelle Dokumentation unter

/usr/src/linux/Documentation/isdn/

anzusehen. Die meisten ISDN-Karten (z. B. USR Sportster International TA, Teles-, ELSA- und Fritz-Karten) werden dabei vom HiSax-Treiber unterstützt, dessen Parameter ausführlich in der Datei *README.HiSax* beschrieben sind. Dort wird auch ausgeführt, welche Kartentreiber nur als Modul geladen werden können und welche auch direkt in den Kernel kompiliert werden dürfen.

Für Netzwerkkarten lauten die Parameter unabhängig von der verwendeten Karte entweder

```
ether=<irq>,<iobase>,<name>
```

(Kernel 2.2 und 2.4) oder

```
netdev=<irq>,<io>,<mem_start>,<mem_ends>,<name>
```

(Kernel 2.6). Wobei bei `netdev` der Parameter *<mem_start>* teilweise auch benutzt wird, um dem Treiber andere Dinge mitzuteilen.

Dies könnte z. B. so aussehen:

```
ether=5,0x300,eth0 ether=9,0x340,eth1
```

Hiermit würden zwei Netzwerkkarten für die Interfaces `eth0` und `eth1` konfiguriert, welche die Ports 0x300 und 0x340 sowie die Interrupts 5 und 9 benutzen. Allerdings hat die Angabe des Interfaces nur geringen Einfluß auf die tatsächliche Vergabe der Bezeichner. Grundsätzlich werden Netzwerkkarten in der Reihenfolge gefunden, in der ihre Treiber in den Kernel kompiliert wurden. Die Numerierung erfolgt dann in der Reihenfolge, in der die Karten erkannt werden.

Bei ISDN-Karten ist das ganze schon komplizierter. Nicht nur hat jeder Treiber seine eigenen Parameter, auch für den gebräuchlichsten Treiber HiSax hängen die zu übergebenden Parameter von der verwendeten Karte ab. Generell folgen Sie aber hier dem folgenden Schema, wobei die Zeile der Übersichtlichkeit halber umbrochen wurde:

```
hisax=<typ1>,<dp1>,<pa1>,<pb1>,<pc1>
[,<typ2>,<dp2>,<pa2>,<pb2>,<pc2>...]
[,<id1>[%<id2>...]]
```

Für eine einzelne ISDN-Karte also:

```
hisax=<typ1>,<dp1>,<pa1>,<pb1>,<pc1>[,<id1>]
```

Im folgenden finden Sie kurze Erläuterungen der Parameter:

typ Gibt den Kartentyp an (Hersteller, Modell und Ausführung). So hat z. B. eine Fritz!PCI-Karte der Version 1 den Typ 27, während eine normale Fritz-Karte (ISA, nicht PnP) den Typ 5 hat.

dp Legt das Protokoll des D-Kanals fest. Früher benutzte man in Deutschland 1TR6 (dp = 1), heute benutzt man in ganz Europa Euro-ISDN EDSS1 (dp = 2).

pa, pb, pc Stellen kartenspezifische Parameter ein. Diese Parameter werden für jede Karte anders benutzt und müssen deshalb in der Dokumentation nachgeschlagen werden. Darüber hinaus brauchen einige (PCI-)Karten keine zusätzlichen Parameter, so daß diese dann entfallen.

id Dieser Parameter gibt der Karte einen Bezeichner, unter dem Konfigurationsprogramme gezielt eine bestimmte Karte ansprechen können. Dieser String kann willkürlich gewählt werden, zwei Karten dürfen aber nicht denselben Bezeichner haben. Aus technischen Gründen werden die Bezeichner durch Prozentzeichen getrennt.

Eine Ausnahme bildet die Fritz!Card PCI Version 2, wie sie momentan verkauft wird. Diese benötigt keinen Parameter. Ist der Treiber einkompiliert, so wird die Karte automatisch erkannt.

Wie man die Parameter konkret übergibt, hängt vom verwendeten Bootloader ab. Aus diesem Grund wollen wir uns im folgenden einmal die beiden gängigsten Bootloader ansehen.

Grub

Nachdem wir den Grub schon einmal in Kapitel 8, Unterabschnitt *Grub*, ab Seite 124 unter dem Gesichtspunkt betrachtet haben, wie wir kontrollieren, welche Konfigurationen der Benutzer booten darf, wollen wir hier einmal einen umfassenderen Blick auf die Konfigurationsdatei werfen.

Diese finden wir unter */boot/grub/menu.lst*. Sie könnte folgendermaßen aussehen:

```
# GRUB Konfiguration
#
# - Globale Optionen
password wm3TcQ7
default 0
timeout 30

# - Konfigurationen
title Linux
    kernel (hd0,0)/boot/vmlinuz root=/dev/hda3
#    initrd (hd0,0)/boot/initrd


title Floppy
    lock
    root (fd0)
    chainloader +1
```

Die Datei beginnt mit den globalen Optionen. Dabei legt `password` ein Paßwort fest, ohne das die Konfiguration nicht verändert werden kann und keine zusätzlichen Parameter beim Booten angegeben werden können.

`default` gibt an, welche Konfiguration automatisch gebootet wird, wenn die in `timeout` angegebene Anzahl von Sekunden verstrichen ist, ohne daß der Benutzer eine Konfiguration ausgewählt hat.

Nun folgen die einzelnen Konfigurationen. Die Parameter bedeuten dabei im einzelnen:

title leitet eine Konfiguration ein. Der Text hinter dem Schlüsselwort wird im Bootmenü angezeigt.

lock sollte immer direkt nach `title` angegeben werden. Dieser Parameter bewirkt, daß der Menüpunkt abgebrochen wird, wenn das Paßwort noch nicht eingegeben wurde.

kernel gibt bei Linux-Systemen an, welcher Kernel verwendet werden soll. Gefolgt wird er von der Angabe des zu verwendenden Kernels sowie den Parametern, die dem Kernel übergeben werden sollen. Der wichtigste Parameter ist hierbei `root=/dev/hda3`, der angibt, welche Partition vom Kernel als Wurzelverzeichnis gemountet werden soll. Aber auch die in Kapitel 8, Unterabschnitt *Bootoptionen*, ab Seite 155 erwähnten zusätzlichen Parameter können hier angehängt werden. Wollen wir also Parameter für eine Netzwerkkarte übergeben, so sieht dies folgendermaßen aus:

```
kernel (hd0,0)/boot/vmlinuz root=/dev/hda3
ether=5,0x300,eth0
```

Der Kernel wird in einem speziellen Format angegeben. Zuerst wird die Partition angegeben, danach angehängt der Pfad auf dieser Partition. Die Angabe (`hd0,0`) bedeutet dabei »die erste Partition der ersten erkannten Festplatte«. Dies ist nicht zwangsläufig die Partition mit dem Wurzelverzeichnis, sondern die Partition, auf der sich der Kernel tatsächlich befindet. Haben wir also eine eigene Partition für */boot* angelegt, so müssen wir diese Partition angeben.[17]

Für das Schema ist dabei unerheblich, ob es sich dabei um IDE- oder SCSI-Festplatten handelt. Haben Sie allerdings nur IDE-Platten, so kann man davon ausgehen, daß */dev/hdc2* der Angabe (`hd2,1`) entspricht.

initrd brauchen Sie normalerweise nicht, aber es gibt Systeme, auf denen bestimmte Module aus einer Ramdisk nachgeladen werden, bevor das Wurzelverzeichnis gemountet wird. Der Inhalt der Ramdisk wird aus der Datei erstellt, die hinter dem Parameter angegeben wird. Die Angabe der Datei erfolgt im bereits dargestellten Format.

root gibt eine Bootpartition vor, wenn das dort installierte Betriebssystem im Gegensatz zu Linux vom Grub nicht direkt unterstützt wird. Das Format ist das bereits bekannte. Allerdings wurde in der Beispielkonfiguration statt einer Festplattenpartition das erste Diskettenlaufwerk angegeben. Man hätte aber auch mit (`hd0,0`) eine Partition vorgeben können, auf der z. B. Windows installiert ist.

chainloader gibt einen weiteren Bootloader an, der nach Grub gestartet wird. Steht hier `+1`, so wird der erste Sektor der mit `root` angegebenen Partition verwendet.

17 In diesem Beispiel befindet sich der Kernel in einer anderen Partition als das Wurzelverzeichnis (/dev/hda statt /dev/hdc). Es wird also wohl eine Partition auf */boot* gemountet. Eine Datei im Wurzelverzeichnis dieser Partition müßte also als (`hd0,0`)`/vmlinuz` angegeben werden. Dagegen würde ein Kernel, der sich im Unterordner */boot* auf der Partition des Wurzelverzeichnisses befindet, mit (`hd0,0`)`/boot/vmlinuz` angegeben werden. SuSE umgeht dies mit einem symbolischen Link in */boot*, der auf das Verzeichnis verweist, in dem er sich befindet (`ln -s . boot`). Damit ist es immer möglich, die oben angegebene Form zu benutzen, ohne sich Gedanken zu machen, ob */boot* nun eine eigene Partition ist oder nicht.

Um nun beim Booten auch unseren Backup-Kernel auswählen zu können, müssen wir noch eine weitere Konfiguration einfügen. Hierzu reicht es, wenn wir die Standardkonfiguration kopieren und einfach den Dateinamen des verwendeten Kernels anpassen:

```
title Backup Kernel
    kernel (hd0,0)/boot/vmlinuz.prev root=/dev/hda3
#    initrd (hd0,0)/boot/initrd
```

Beim nächsten Bootvorgang werden Sie nun einen Eintrag Backup Kernel im Bootmenü vorfinden. Wählen Sie diesen aus, wenn Ihr selbstkompilierter Kernel nicht starten will.

LiLo

Den Bootloader LiLo haben wir schon in Kapitel 8, Unterabschnitt *LiLo*, ab Seite 125 kennengelernt. Haben wir dort nur die Möglichkeit betrachtet, die Konfiguration am Bootprompt zu ändern, so soll es jetzt um die permanente Konfiguration gehen. Dies geschieht in der Datei */etc/lilo.conf*. Diese könnte etwa folgendermaßen aussehen:

```
# LILO Konfigurationsdatei
# Start LILO global Section
# If you want to prevent console users to boot with init=/bin/bash,
#  restrict usage of boot params by setting a passwd and using the option
#  restricted.
password=wm3TcQ7
restricted
boot=/dev/hda
#compact       # faster, but won't work on all systems.
#lba32
vga=normal
read-only
prompt
timeout=100
# End LILO global Section
#
# Standardkonfiguration
image = /boot/vmlinuz
  root = /dev/hda3
  label = l
```

Grundsätzlich unterteilt sich die Datei in mehrere Bereiche. Die ersten Zeilen bilden die globalen Optionen, die Einstellungen beinhalten, die für alle Bootkonfigurationen gleich sind.

Danach folgen die einzelnen Abschnitte für die einzelnen Installationen. Diese beginnen mit `image=` für Linux-Konfigurationen oder `other=` für andere Systeme wie DOS, Windows usw. Auf einer Firewall sollten aber grundsätzlich keine anderen Systeme installiert sein. Es soll daher der Hinweis genügen, daß der Befehl `man lilo.conf` weitere Informationen zu den hier nicht behandelten Parametern liefert. Alternativ kann auch in der Datei */usr[/share]/doc/packages/lilo/README* nachgesehen werden.

Die globalen Optionen `password` und `restricted` kennen wir ja schon aus dem Kapitel 8, Unterabschnitt *LiLo*, ab Seite 125. Dabei möchte ich ausdrücklich darauf hinweisen,

daß das Paßwort zwar durchaus vernünftig gewählt ist, von Ihnen aber unter keinen Umständen benutzt werden sollte, da es hier in diesem Buch steht.[18]

Die anderen Parameter haben die folgenden Bedeutungen:

boot Dieser Parameter gibt an, wohin LiLo installiert werden soll. Übliche Möglichkeiten wären der Master Boot Record (MBR) der ersten Partition (s. o.), eine Floppy (`boot=/dev/fd0`) oder der Bootsektor einer beliebigen Partition. Die letzte Variante wird normalerweise genutzt, wenn schon ein Bootloader installiert ist (z. B. für Windows NT), der nicht durch LiLo ersetzt werden soll.

compact Ist diese Option vorhanden, so werden mehrere Sektoren gleichzeitig gelesen. Dies bringt einen Geschwindigkeitsvorteil, der vor allem bei Floppy-Installationen spürbar ist. Es wird aber nicht garantiert, daß dies auch auf allen Systemen funktioniert.

lba32 Ohne diese Option müssen sich alle Dateien, die LiLo zum Booten braucht, auf den ersten 1024 Zylindern der Festplatte befinden. Mit dieser Option versucht LiLo, spezielle Funktionen neuerer BIOS-Versionen[19] zu nutzen. Damit können sogar 2048 GB adressiert werden. Dieser Parameter sollte nicht zusammen mit `compact` verwendet werden.

vga Diese Einstellung legt den Graphikmodus beim Booten fest. `normal` bedeutet 80 × 25 Textmodus, `ext` oder `extended` 80 × 50 Textmodus, `ask` bewirkt, daß beim Booten gefragt wird, welcher Modus verwendet werden soll. Tatsächlich existieren noch mehr Modi, die gegebenenfalls am Bootprompt mit `[TAB]` erfragt werden können, falls `ask` eingestellt ist.

read-only Das Rootdateisystem wird schreibgeschützt gemountet. Dies ist für die Prüfung des Dateisystems beim Bootvorgang nötig.

prompt Diese Option bewirkt, daß ein Bootprompt angezeigt wird.

timeout Hier wird in Zehntelsekunden eingestellt, wie lange LiLo warten soll, bevor die Standardkonfiguration automatisch gestartet wird.

image Diese Option leitet eine Konfiguration ein. Alle Optionen, die vor dem ersten Auftreten von `image` angegeben wurden, gelten global für alle Konfigurationen. Alle anderen Optionen gelten nur für jeweils eine Konfiguration. Als Parameter erhält die Option den Dateinamen des Kernels, der für diese spezielle Konfiguration verwendet werden soll.

root Hier wird festgelegt, welche Partition als »/« gemountet wird.

label Dieser Parameter gibt der Konfiguration einen Namen, der dazu genutzt werden kann, sie am Bootprompt auszuwählen.

Um nun auch unseren gesicherten Vorgängerkernel benutzen zu können, fügen wir eine zusätzliche Konfiguration an die Datei an. In unserem Beispiel sähe das folgendermaßen aus:

18 Bevor Sie fragen: Nein, ich selbst benutze dieses Paßwort ebenfalls nicht.

19 Es handelt sich dabei um *Logical Block Addressing*. Hierbei werden die Datenblöcke auf der Festplatte einfach durchnumeriert, statt sie als Sektor/Lesekopf/Zylinder anzugeben.

```
# Backup-Konfiguration
image = /boot/vmlinuz.prev
  root = /dev/hda3
  label = lold
```

Will das System später nicht mehr booten, so reicht am Bootprompt die folgende Eingabe, um wieder mit dem alten Kernel zu starten:

```
LILO: lold
```

Am Bootprompt können wir auch Kernel-Parameter eingeben. Das Einbinden einer Fritz!Card PCI-Karte der Version 1 sähe dabei folgendermaßen aus:

```
LILO: l hisax=27,2,HiSax
```

Für eine Fritz!Card PCI Version 2 ist dagegen kein spezieller Parameter nötig.

Nun wäre es sicherlich reichlich unbequem, dies bei jedem Bootvorgang einzugeben. Die *lilo.conf* kennt daher einen Eintrag `append`, mit dem man all diese Kernelparameter übergeben kann. In unserem Beispiel muß die entsprechende Konfiguration um den folgenden Eintrag erweitert werden:

```
append="hisax=27,2,HiSax"
```

Wollen wir zusätzlich zwei Netzwerkkarten verwenden, so sähe der Eintrag folgendermaßen aus:

```
append="hisax=27,2,HiSax ether=5,0x300,eth0 ether=9,0x340,eth1"
```

Ist die Konfiguration abgeschlossen, so muß ein neuer Bootloader erzeugt werden. Dies muß jedesmal wieder geschehen, wenn ein neuer Kernel erzeugt oder die Konfigurationsdatei verändert wurde. Dies veranlassen wir mit dem Aufruf

```
# /sbin/lilo
Added l *
Added lold
```

Die Ausgaben zeigen, daß LiLo nun die Konfigurationen `l` und `lold` kennt, wobei der Stern `l` als die Standardkonfiguration kennzeichnet, die automatisch gewählt wird, wenn der Benutzer nicht manuell eine andere Wahl trifft.

Eintrag in die modules.conf bzw. modprobe.conf

Werden Module oft manuell aufgerufen, kann es sinnvoll sein, benötigte Parameter in einer zentralen Konfigurationsdatei zu speichern. Wichtiger wird so eine Datei allerdings, wenn Module automatisch bei Bedarf geladen werden. Hier wird bei jedem Zugriff auf ein Stück Hardware, wie z. B. eine Karte oder ein angeschlossenes Gerät, der Befehl `modprobe` mit der Bezeichnung eines sogenannten *Devices* aufgerufen. Diese Device-Bezeichnung beschreibt nur, um was für eine Art Gerät es sich handelt (z. B. Maus, erste Netzwerkkarte oder etwas ähnliches), sie legt aber nicht fest, welches Modul hierzu zu laden ist und welche zusätzlichen Optionen (z. B. IRQ, Port) dafür nötig sind.

Aus diesen Gründen besitzt `modprobe` eine Konfigurationsdatei, in der die Zuordnungen von Device-Bezeichnungen zu Modulen sowie die zu verwendenden Optionen für das jeweilige Modul festgelegt sind. Hierbei handelt es sich bei Kerneln der Serie 2.2 und 2.4 um die Datei */etc/modules.conf*. Mit den Kerneln der Serie 2.6 wurde das ganze Modulkonzept gründlich überarbeitet. Neben anderen Dingen änderte sich auch der Name dieser Konfigurationsdatei. Sie heißt jetzt */etc/modprobe.conf* und enthält Zeilen der Art:

```
alias block-major-2 floppy
alias char-major-27 ftape
alias char-major-10-1 psaux
alias eth0 ne
alias block-major-43 hisax
alias block-major-44 hisax
alias block-major-45 hisax
options ne irq=5
options hisax id=HiSax typ=27 protocol=2
```

Hierbei definieren die `alias`-Befehle zusätzliche Namen, unter denen die Module angesprochen werden können. Dies nutzt man im Falle des Loaders für Kernel-Module, um die beschriebene Umsetzung einer Device-Bezeichnung in den Namen eines Treibermoduls zu realisieren. Man könnte dies allerdings auch dazu benutzen, einen Treiber je nach Bedarf mit verschiedenen Parametern aufzurufen, indem man ihm für jede Konfiguration einen eigenen Namen gibt, dem dann eigene Optionen zugeordnet werden.

Diese Optionen werden mit den `options`-Befehlen übergeben. So wird im oben gezeigten Beispiel dem Modul `ne` mitgeteilt, daß die von ihm unterstützte Netzwerkkarte den IRQ 5 benutzt. Der Hisax-Treiber wird dagegen dahingehend konfiguriert, daß der alte Treiber für die Fritz!Card PCI Version 1 mit DSS1 verwendet werden soll.

Hier wird zu diesem Zweck der Modulname benutzt. Prinzipiell kann man aber auch einen definierten alias-Namen benutzen. In diesem Fall gelten Optionen nur, wenn das Modul unter diesem alias-Namen aufgerufen wird. Hierbei gilt, daß Optionen, die `modprobe` auf der Kommandozeile übergeben werden, Vorrang vor Optionen haben, die für einen `alias`-Namen definiert wurden. Optionen, die für einen `alias`-Namen definiert wurden, haben wiederum Vorrang vor Optionen, die für den tatsächlichen Namen eines Moduls definiert wurden.

Um den Mechanismus besser zu verstehen, ist ein kleiner Exkurs in das Device-Konzept von Linux nötig. Geräte werden über spezielle Dateien im Verzeichnis */dev/* repräsentiert. Jeder Datei ist eine *Major-* und eine *Minor-Nummer* zugeordnet, die das Device definieren. Darüber hinaus werden Devices in *Block-* und *Character-Devices* eingeteilt. Bei Block-Devices handelt es sich um Devices, auf die immer nur in festen Blöcken zugegriffen werden kann. Hierbei handelt es sich in der Regel um Festplatten und verwandte Speichermedien, die nicht direkt ausgelesen, sondern in das Dateisystem gemountet werden.

Im Gegensatz dazu stehen die Character-Devices, aus denen einzelne Zeichen gelesen bzw. in die Zeichen geschrieben werden können. Bei diesen handelt es sich z. B. um Terminals, Netzwerkkarten und Mäuse.

Listen wir den Inhalt des Verzeichnisses */dev* auf, so werden wir Einträge folgender Art finden:

```
brw-rw-rw-   1 root     disk       2,   0 Dec 11 18:20 fd0
crw-rw-rw-   1 root     tty        3,  48 Dec 11 18:20 ttyS0
```

Bei dem ersten Eintrag handelt es sich um das erste Diskettenlaufwerk. Es ist ein Block-Device mit der Major-Nummer 2 und der Minor-Nummer 0. Der zweite Eintrag steht für die erste serielle Schnittstelle. Sie ist ein Character-Device mit der Major-Nummer 3 und der Minor-Nummer 48.

Auf dieselbe Weise funktionieren auch die `alias`-Einträge. Ein Eintrag der Art

```
alias block-major-2 floppy
```

bewirkt, daß für den Zugriff auf alle Devices mit der Major-Nummer 2 das Modul `floppy` geladen wird. Der Eintrag

```
alias char-major-10-1 psaux
```

gilt dagegen nur, wenn auch die Minor-Nummer (hier 1) übereinstimmt. Das ist in diesem Fall auch sinnvoll, da die Major-Nummer 10 diverse Mäuse zusammenfaßt, die von der normalen PS/2-Maus bis zur Atari-Maus reichen. Die unterschiedlichen Typen werden dabei anhand der Minor-Nummer unterschieden. Es ist also naheliegend, den PS/2-Treiber nicht für Atari-Mäuse zu laden.

Ethernet-Devices (`eth...`) stellen dabei einen Sonderfall dar. Für sie existieren keine Einträge der Art `/dev/eth0`. Hier wird statt der Angabe von Major- und Minor-Nummer der Device-Name (z. B. `eth0`) angegeben.

Sollen dem Modul noch spezielle Parameter übergeben werden, so dienen dazu die `options`-Zeilen. Im obigen Beispiel wird mit der Zeile

```
options ne irq=5
```

dem Modul `ne` (für NE2000-kompatible Karten) der Interrupt 5 als Parameter mitgegeben. Besitzt man mehrere NE2000-Karten, so kann man auch die Parameter für die einzelnen Karten durch Komma getrennt angeben:

```
options ne io=0x300,0x320 irq=5,7
```

Weitere Beispiele für Optionen finden sich

- im Handbuch zu SuSE-Linux,
- in der mit der Distribution mitgelieferten */etc/modules.conf* bzw. */etc/modprobe.conf*,
- in der Datei */usr/src/linux/Documentation/modules*,
- in den Dateien */usr/src/Linux/Documentation/isdn/README.<Treiber>*
  und
- in der Datei */usr/src/linux/Documentation/networking/net-modules*.

Möchte man eine Auflistung aller momentan geltenden Zuordnungen erhalten, so kann dies mit

```
# modprobe -c
```

erreicht werden.

Schließlich existiert noch eine Reihe von weiteren Befehlen, von denen insbesondere die folgenden interessant sind, weil sie sowohl in der Datei */etc/modules.conf* als auch in der */etc/modprobe.conf* gültig sind:

include <Datei> Der Inhalt der Datei wird ebenfalls eingelesen, als ob er in dieser Datei stünde. Zumindest in der */etc/modprobe.conf* kann hier auch ein Verzeichnis angegeben werden. In diesem Fall werden alle Dateien in diesem Verzeichnis eingebunden.

install <module> <command> Wenn `modprobe` angewiesen wird, das angegebene Modul zu laden, führt es statt dessen den Befehl *<command>* aus. Das kann grundsätzlich jeder Befehl sein, den Sie auf der Kommandozeile ausführen können.

remove <module> <command> Dies ist das Gegenstück zu `install`. Hier wird der Befehl ausgeführt, wenn versucht wird, das Modul wieder zu entladen (z. B. mit `rmmod` oder mit `modprobe -r`).

In der Datei */etc/modules.conf* sind noch eine Reihe weiterer Befehle möglich, diese wurden aber kaum genutzt und sind beim Umstieg auf die neue Modulverwaltung aufgegeben worden, um eine klarere und einfacher zu verstehende Konfiguration zu schaffen.

Laden der Module durch ein Runlevel-Skript

Haben wir den »Loader für Kernelmodule« deaktiviert, so müssen wir eventuell benötigte Module von Hand laden. Dies kann geschehen, indem entsprechende `modeprobe`-Befehle in ein entsprechendes Runlevel-Skript eingebunden werden. Für die Einbindung einer NE2000-Karte und einer Fritz!Card PCI Version 2 könnte es folgendermaßen aussehen:

```
#!/bin/sh
# ========================================================================
# modload {start|stop}
#
#    Ein kleines Skript, das die n"otigen Module l"adt
#
#    start: Laden
#    stop:  Entladen aller unbenutzten Module
#
# Copyright (C) 2003 Andreas G. Lessig
#
# Lizenz: GPL v2 oder h"ohere Version
#
# ========================================================================
```

```
### BEGIN INIT INFO
# Provides:          modload
# Required-Start:
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:     1 2 3 5
# Default-Stop:      0 6
# Short-Description: Laden von Modulen
# Description:       Laden von Modulen
### END INIT INFO


case "$1" in
   start)

      echo "Lade Module ..."

      /sbin/modprobe ne
      /sbin/modprobe isdn
      /sbin/modprobe hisax_fcpcipnp

#     Weitere Module hier einf"ugen:

      ;;
   stop)

      echo "Entferne ungenutzte Module ..."

      /sbin/rmmod -r ne
      /sbin/rmmod -r hisax_fcpcipnp
      /sbin/rmmod isdn

#     Weitere Module hier einf"ugen:

      ;;
   *)
      echo "Usage: $0 {start|stop}"
      ;;
esac
```

Den Modulen können dabei auch Parameter übergeben werden wie die, die wir schon in Kapitel 8, Unterabschnitt *Eintrag in die modules.conf bzw. modprobe.conf*, ab Seite 161 kennengelernt haben. Dies könnte z. B. folgendermaßen aussehen:

```
/sbin/modprobe ne io=0x300,0x320 irq=5,7
```

In diesem Fall brauchen sie nicht in die Datei */etc/modules.conf* eingetragen zu werden.

Beim Einrichten der nötigen Links ist darauf zu achten, daß das Skript möglichst früh ausgeführt wird. Wird z. B. ein Modul für eine Netzwerkkarte geladen, so sollte das Ladeskript vor dem Skript ausgeführt werden, das die Netzwerk-Interfaces konfiguriert. Wurden auch elementare Treiber wie z. B. die serielle Schnittstelle als Modul kompiliert, so muß das Skript sogar noch früher geladen werden. Hier bietet sich die Verlinkung unter `S01aaamodload` bzw. `K99zzzmodload` an, um sicherzustellen, daß es beim Eintritt in den Runlevel als allererstes bzw. beim Verlassen des Runlevels als letztes ausgeführt wird.

Unter SuSE können wir uns derartig komplizierte Überlegungen sparen. Hier reicht ein

```
# insserv modload
```

Die Skripte in diesem Buch haben entsprechende Kommentare, so daß ihre Abhängigkeiten untereinander abgebildet sind. Nur beim letzten Skript dieser Kette müssen wir darauf achten, daß wir es in die Kommentare des SuSE-Skripts `network` eintragen. Aber dazu später mehr.

Konfiguration des /proc-Dateisystems

Während der Kernel früher (bis 2.0) bei der Kompilation konfiguriert wurde, bestimmen seit der Version 2.2 Variablen sein Verhalten, die von `root` im laufenden Betrieb gesetzt werden können. Dazu dient das virtuelle Verzeichnis */proc/sys*. In ihm befinden sich in diversen Unterverzeichnissen Dateien, deren Inhalt die Arbeitsweise des Kernels maßgeblich beeinflußt.

Obwohl es sich bei ihnen nicht um reale Dateien, sondern nur um ein Abbild bestimmter Kernelvariablen handelt, kann der Zugriff auf sie mit normalen Programmen zur Dateibearbeitung erfolgen. Es ist also kein Problem, z. B. das Forwarding von IP-Paketen mit dem Editor des Midnight Commanders anzustellen.

Die Dateien, die uns im Rahmen des Firewalling interessieren, finden sich unter */proc/sys/net/ipv4*:

ip_forward Nur wenn in dieser Datei eine »1« steht, routet der Rechner Pakete zwischen anderen Rechnern. Für eine Firewall ist diese Einstellung unabdingbar. Da sie alle anderen Einstellungen auf ihre Standardwerte zurücksetzt, sollte sie als erstes vorgenommen werden.

ip_always_defrag (nur Kernel 2.2) Wenn in dieser Datei eine »1« steht, wird jedes ankommende Paket defragmentiert, bevor die Firewallregeln darauf angewendet werden. Dies ist notwendig, um Angriffe durch Fragmentierung zu verhindern. Diese Angriffe wurden in Kapitel 4, Unterabschnitt *Fragment-Angriffe*, ab Seite 37 beschrieben.

Seit Kernel 2.4 geschieht die Defragmentierung automatisch, und die Datei entfällt.

ip_dynaddr Wenn ein Rechner seine IP-Adresse dynamisch beim Verbindungsaufbau zugewiesen bekommt, wird das erste Paket, das den Verbindungsaufbau in Gang gesetzt hat, in der Regel verlorengehen. Dies liegt daran, daß zu dem Zeitpunkt, als es abgeschickt wurde, noch nicht bekannt war, welche IP-Adresse der Rechner bekommen würde. Ohne gültige Absenderadresse kann ein Paket aber nicht beantwortet werden.

Diese Einstellung behebt nun den Mangel. Mit ihr schreibt der Kernel nachträglich die Absenderadresse um. Gültige Werte sind 0 (aus), 1 (an), 2 (Vermerken der Aktionen im Kernel-Log).

tcp_syncookies Aktiviert die Funktionalität, die wir mit »IP: SYN flood protection« (2.2.x) oder »IP: TCP syncookie support« (2.4.x) in den Kernel kompiliert haben.

Wenn der Kernel feststellt, daß in der Tabelle für halboffene Verbindungen kein Eintrag für eine neue Verbindungsanfrage vorhanden ist, so sendet er als eigene Folgenummer einen speziellen Wert (*Syncookie*), der aus der Adresse und dem Port von Absender und Server, einem internen Zähler und der Folgenummer der Anfrage gewonnen wurde. Wenn nun das dritte Paket des Anfragenden eintrifft, so kann anhand der Bestätigungsnummer geprüft werden, ob dieses Paket wirklich die Antwort auf ein Syncookie darstellt oder ob ein Angreifer versucht, eine Folgenummer zu erraten [7]. Auf diese Weise können weitere Verbindungen entgegengenommen werden, obwohl die Tabelle für den Aufbau von Verbindungen voll ist.

icmp_echo_ignore_broadcasts Wird diese Funktion aktiviert (1), so antwortet der Rechner nicht mehr auf ICMP-Requests, die nicht an ihn, sondern an alle gerichtet sind, die sich im selben Subnetz befinden. Dies verhindert, daß der Rechner als »Verstärker« für ein Flooding mit Ping-Paketen dient.

Neben diesen generellen Einstellungen können auch noch gezielt Einstellungen für die einzelnen Interfaces getroffen werden. Dazu befindet sich unter */proc/sys/net/ipv4/conf* jeweils ein Verzeichnis für jedes Interface, dem schon eine Adresse mit `ipconfig` zugewiesen wurde. Neu eingerichteten Interfaces werden dabei jeweils die Einstellungen zugewiesen, die unter dem Verzeichnis *defaults* eingetragen sind. Darüber hinaus existiert ein Verzeichnis *all*. Hier vorgenommene Einstellungen wirken sich auf alle Interfaces aus. Es existieren u. a. diese Einstellungen:

accept_redirects Bestimmt, ob ICMP-Redirect-Nachrichten angenommen werden oder nicht. Wir werden später derartige Pakete sowieso ausfiltern. Der Standardwert ist »1« für einen normalen Rechner, »0« für einen Router (»IP: optimize as router not as host«). Da wir den Kernel als Router konfiguriert haben, sollte die Standardeinstellung unseren Wünschen entsprechen.

accept_source_route Legt fest, ob Pakete mit einer Source Route angenommen werden oder nicht. Der Standardwert ist »0« für einen normalen Rechner, »1« für einen Router. Für unsere Firewall ist dies eher unerwünscht. Es empfiehlt sich daher, diese Einstellung zu deaktivieren (0).

rp_filter Ist diese Einstellung aktiviert, wird überprüft, ob ein Paket tatsächlich über das Interface empfangen wurde, das den Routing-Regeln entsprechend zu erwarten gewesen wäre. Dies dient dem Schutz vor IP-Spoofing, ist aber normalerweise nicht aktiv (0). Lediglich für lokale und Broadcast-Adressen wird diese Überprüfung immer durchgeführt. Es ist sehr sinnvoll, diese Funktion auf einer Firewall zu aktivieren (1).

Benutzen wir einen Kernel der Serie 2.6 und haben wir auch die Unterstützung für IPv6 einkompiliert, so sollten wir auch die Interface-spezifischen Einstellungen unter */proc/sys/net/ipv6/conf* anpassen. Auch hier existieren die Unterverzeichnisse *all*, *defaults* sowie Verzeichnisse für die einzelnen Interfaces. Hier finden wir insbesondere die folgenden Dateien:

forwarding Dieser Parameter sollte in allen Dateien gleich gesetzt sein. Er bestimmt, ob sich der Rechner eher wie ein Arbeitsplatzrechner oder wie ein Router verhält. Außer-

dem definiert */proc/sys/net/ipv6/all/forwarding*, ob eingehende IPv6-Pakete über andere Netzwerk-Interfaces weitergeschickt werden. Es ist im Gegensatz zu IPv4 nicht möglich, das Weiterleiten von Paketen für einige Interfaces zu aktivieren, für andere aber zu unterbinden.

Wenn wir IPv6-Pakete weitervermitteln wollen, so sollten wir in diese Datei »1« schreiben, andernfalls »0«.

Achtung: Das Schreiben in diese Datei setzt die im folgenden beschriebenen Dateien auf Standardwerte zurück. Wir müssen also darauf achten, in welcher Reihenfolge wir in die Dateien schreiben.

accept_ra Enthält die Datei eine »1«, so werden Router Advertisements akzeptiert. Das heißt, ein Protokoll wird genutzt, mit dem automatisch Router im Netzwerk gefunden und genutzt werden können. Dies würde unsere Firewall unnötigen Angriffen öffnen (siehe Kapitel 4, Unterabschnitt *Angriffe mittels ICMP*, ab Seite 30).

accept_redirects Enthält die Datei eine »1«, so werden ICMP-Redirect-Pakete akzeptiert und das Routing wird entsprechend angepaßt. Das sollten wir unbedingt abschalten (siehe Kapitel 4, Unterabschnitt *Angriffe mittels ICMP*, ab Seite 30).

autoconf Diese Datei enthält standardmäßig denselben Wert wie *accept_ra*. Handelt es sich dabei um eine »1«, so werden Router-Advertisements ausgewertet, um die eigenen Netzwerkadressen zu konfigurieren.[20] In dieser Datei sollte unbedingt eine »0« stehen.

Schließlich kann es noch passieren, daß jede Protokollmeldung der Paketfilter auch auf der Konsole erfolgt. Dies kann manchmal doch recht irritierend sein, wenn man z. B. gerade eine Konfigurationsdatei editiert.

Falls man den `syslogd` nicht falsch konfiguriert hat (siehe Kapitel 9, Abschnitt *Das Systemprotokoll*, ab Seite 205), so liegt die Ursache vermutlich in einer falschen Einstellung von

/proc/sys/kernel/printk

Diese Datei enthält vier Zahlen, die bestimmen, wann Meldungen nicht nur an den `syslogd` geschickt, sondern auch direkt auf die Konsole ausgegeben werden:

console_log_level Jede Mitteilung mit einer höheren Priorität wird direkt auf die Konsole ausgegeben. Dabei gilt, daß die Priorität um so höher ist, je niedriger die Zahl ist, die diese angibt. Die höchste Priorität ist 0 (emerg), die niedrigste 7 (debug).

Bei manchen Kerneln ist hier 6 (info) vorgegeben, wodurch fast jede Meldung des Kernels auch auf die Konsole ausgegeben wird.

default_message_level Dieser Wert wird immer dann als Priorität einer Nachricht benutzt, wenn eine Funktion des Kernels *printk()* aufruft, ohne eine Priorität vorzugeben. Es ist meistens auf 4 (warning) gesetzt.

20 Die Adressen basieren neben den Informationen in den Router-Advertisements zusätzlich auf der eigenen MAC-Adresse, so daß Adressen generiert werden, die garantiert eindeutig sind.

minimum_console_loglevel Gibt den niedrigsten Wert an, auf den *console_log_level* gesetzt werden kann. Normal ist hier 1 (alert).

default_console_loglevel Gibt den Wert an, auf den *console_log_level* beim Booten gesetzt wird. Standardmäßig ist dieser Wert 7 (debug).

Nach dem, was oben beschrieben wurde, liegt es nahe, ein Skript zu schreiben, das die nötigen Einstellungen für uns vornimmt. Dieses könnte z. B. kurz vor oder kurz nach einem Firewalling-Skript aufgerufen werden. Beide sollten aber auf jeden Fall ausgeführt werden, bevor die Netzwerk-Interfaces und eventuelle Serverdienste aktiviert werden, da ansonsten eine zeitliche Lücke entstünde, in der auf den Rechner schon über das Netz zugegriffen werden könnte, ohne daß schon alle Schutzmaßnahmen aktiv wären.

Die Weiterleitung von IPv6-Paketen ist deaktiviert. Dies kann aber durch Setzen der Variable *FORWARDV6* auf »1« geändert werden.

```
#!/bin/sh
# ========================================================================
# procconf {start|stop}
#
#    Ein kleines Skript, das die IP-Einstellungen in
#    /proc den Bed"urfnissen einer Firewall anpa"st.
#
#    start: Forwarding an
#    stop:  Forwarding aus
#
# 2005-05-14    procset() "Uberpr"uft erst, ob eine Datei existiert,
#               bevor versucht wird hineinzuschreiben.
#
#               Router Advertisements und Redirects werden f"ur ipv6
#               nicht angenommen. Autoconf f"ur ipv4 und ipv6 verboten.
#
# 2005-05-17    Variable FORWARDV6 regelt, ob IPv6-Pakete geroutet
#               werden oder nicht.
#
# Copyright (C) 2003 Andreas G. Lessig
#
# Lizenz: GPL v2 oder h"ohere Version
#
# ========================================================================

### BEGIN INIT INFO
# Provides:          procconf
# Required-Start:    $local_fs
# Should-Start:      pf-ipt pf-ipc dmz-ipt dmz-ipc
# Required-Stop:     $network
# Should-Stop:
# Default-Start:     2 3 5
# Default-Stop:      0 1 6
# Short-Description: Konfiguration von /proc
# Description:       Konfiguration von /proc
### END INIT INFO


# Hier eine 1 eintragen, wenn IPv6-Pakete geroutet werden sollen

FORWARDV6="0"
```

```
procset()
{
        if [ -f "$2" ]
        then
                echo "$1" > "$2"
#               echo "Setting $2 to $1"
        else
                echo "WARNING: $2 does not exist"
        fi
}

case "$1" in
   start)

      echo "Aktiviere Forwarding ..."

      procset 1  /proc/sys/net/ipv4/ip_forward
      procset 1  /proc/sys/net/ipv4/ip_always_defrag
      procset 1  /proc/sys/net/ipv4/ip_dynaddr
      procset 1  /proc/sys/net/ipv4/tcp_syncookies
      procset 1  /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

      for f in /proc/sys/net/ipv4/conf/*
      do
         procset 1 $f/rp_filter
         procset 0 $f/accept_redirects
         procset 0 $f/accept_source_route
      done

      for f in /proc/sys/net/ipv6/conf/*
      do
         procset "$FORWARDV6" $f/forwarding
         procset 0 $f/accept_ra
         procset 0 $f/accept_redirects
         procset 0 $f/autoconf
      done

      procset "1 4 1 7" /proc/sys/kernel/printk
      ;;
   stop)

      echo "Stoppe Forwarding ..."

      procset 0 /proc/sys/net/ipv4/ip_forward
      procset 1 /proc/sys/net/ipv4/ip_always_defrag
      procset 1 /proc/sys/net/ipv4/ip_dynaddr
      procset 1 /proc/sys/net/ipv4/tcp_syncookies
      procset 1 /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

      for f in /proc/sys/net/ipv4/conf/*
      do
         procset 1 $f/rp_filter
         procset 0 $f/accept_redirects
         procset 0 $f/accept_source_route
      done
```

```
        for f in /proc/sys/net/ipv6/conf/*
        do
           procset 0 $f/forwarding
           procset 0 $f/accept_ra
           procset 0 $f/accept_redirects
           procset 0 $f/autoconf
        done

        procset "1 4 1 7" /proc/sys/kernel/printk
        ;;
     *)
        echo "Usage: $0 {start|stop}"
        ;;
esac
```

Bevor wir das Skript nun allerdings ausführen können, müssen wir es noch mit folgendem Aufruf ausführbar machen.

```
# chmod 700 procconf
```

Mit `procconf start` kann nun die Weiterleitung von Paketen durch die Firewall erlaubt werden, während der Rechner mit `procconf stop` zu einem normalen Klienten wird, der zwar selbst durchaus Verbindungen in angeschlossene Netze öffnen kann, der aber Verbindungen anderer Rechner nicht mehr weitervermittelt.

Das Skript wurde bewußt so geschrieben, daß es als normales Runlevel-Skript dienen kann (siehe Kapitel 8, Unterabschnitt *Init*, ab Seite 125). Wenn wir die dafür nötigen Links erzeugen, sollten wir darauf achten, daß das Skript vor dem gestartet wird, welches das Netzwerk startet[21].

Unter SuSE können wir das Skript wie schon `procconf` mit `insserv` installieren.

Konfiguration der /etc/fstab

Die Datei */etc/fstab* beschreibt, welche Partitionen und Medien auf welche Verzeichnisse gemountet werden sollen. Die Datei enthält Zeilen, die folgende Form haben:

<Dev> <Mountp.> <Fs> <Opt.> <Dump> <Pass>

Die einzelnen Felder haben folgende Bedeutung:

Dev gibt das Device an, das gemountet werden soll.

Mountp. beschreibt das Verzeichnis, auf das das Device gemountet werden soll.

Fs legt das verwendete Dateisystem fest. Neben den gängigen Linux-Dateisystemen `ext2`, `ext3`, `reiserfs` und `xfs` existieren u. a. noch folgende Dateisysteme:

auto weist den Kernel an, selber herauszufinden, welches Dateisystem sich auf dem Datenträger befindet.

iso9660 wird auf CDs verwendet.

21 Debian: verlinkt unter `/etc/rcS.d/S40networking`

proc wird nur für den Eintrag für */proc* verwendet.

vfat Das normale FAT-Dateisystem mit Unterstützung für lange Dateinamen.

Opt. enthält Optionen für den `mount`-Befehl. Davon existieren eine ganze Reihe, die in der Manpage zu `mount` beschrieben sind. Hier sollen nur die wichtigsten erwähnt werden:

auto,noauto gibt an, ob die Partition beim Booten gemountet werden soll oder nicht.

nodev verhindert, daß eventuelle Devices auf dem Datenträger als solche behandelt werden.

noexec verbietet das Ausführen von Dateien auf dem Datenträger.

nosuid sorgt dafür, daß das SUID-Bit ignoriert wird.

ro bewirkt, daß das Dateisystem nur zum Lesen gemountet wird.

user erlaubt, daß normale Benutzer das Dateisystem mounten dürfen. Auf einer Firewall ist dies nicht so sinnvoll. Auf normalen Systemen macht dies vor allem bei Wechselmedien Sinn. Diese Option impliziert `nodev`, `noexec` und `nosuid`. Das Mounten durch normale Benutzer setzt auch voraus, daß der `mount`-Befehl mit gesetztem SUID-Bit installiert ist.

Dump ist normalerweise 0. Wenn Sie Backups mit `dump` machen, steuert diese Zahl, wie oft die Daten auf diesem Dateisystem gesichert werden.

Pass steuert die Überprüfung des Dateisystems mit `fsck`. Das Wurzelverzeichnis sollte hier eine 1 enthalten, andere Dateisysteme eine 2. Für Wechselmedien ist eine Überprüfung mit `fsck` in der Regel nicht sinnvoll. Hier sollte man eine 0 eintragen.

Normalerweise brauchen wir uns um die Datei */etc/fstab* nicht zu kümmern, da diese bei der Installation automatisch angelegt wird. Nachdem wir jetzt aber einen eigenen Kernel kompiliert haben, kann es sein, daß einige Einträge nicht mehr funktionieren. Dies gilt insbesondere für Wechselmedien, die vom Betriebssystem automatisch beim Zugriff gemountet werden. Unter SuSE 9.2 sind z. B. die folgenden Einträge vorhanden:

```
/dev/cdrom       /media/cdrom       subfs fs=cdfss,ro,procuid,nosuid,
                                          nodev,exec,iocharset=utf8 0 0
/dev/dvdrecorder /media/dvdrecorder subfs fs=cdfss,ro,procuid,nosuid,
                                          nodev,exec,iocharset=utf8 0 0
/dev/fd0         /media/floppy      subfs fs=floppyfss,procuid,
                                          nodev,nosuid,sync 0 0
```

Die Umbrüche sind dabei durch den Druck bedingt. Es sind eigentlich nur drei Zeilen.

In allen wird ein Dateisystem `subfs` verwendet, das zumindest derzeit in einem Standardkernel gar nicht enthalten ist, sondern von SuSE in ihre Kernel zusätzlich eingebaut wurde. Wir sollten das Dateisystem daher durch die tatsächlichen Dateisysteme oder `auto` ersetzen:

```
/dev/cdrom       /media/cdrom       auto noauto,ro,nodev,nosuid 0 0
/dev/dvdrecorder /media/dvdrecorder auto noauto,ro,nosuid,nodev 0 0
/dev/fd0         /media/floppy      auto noauto,nodev,nosuid,sync 0 0
```

Neustart

Nachdem wir den innersten Kern unseres Betriebssystems umgekrempelt haben, ist es tatsächlich nötig, neu zu booten. Dies geschieht mit dem Befehl

```
# sync;reboot
```

Während des Bootvorgangs sehen wir dann Meldungen der Art:

```
scsi : 0 hosts.
scsi : detected total.
PPP: version 2.3.7 (demand dialling)
TCP compression code copyright 1989 Regents of the University of California
PPP line discipline registered.
ne.c:v1.10 9/23/94 Donald Becker (becker@cesdis.gsfc.nasa.gov)
NE*000 ethercard probe at 0x300: 00 80 ad 18 9c 97
eth0: NE2000 found at 0x300, using IRQ 9.
```

In diesem Beispiel wurde SCSI-Unterstützung in den Kernel kompiliert, es wurden aber keine SCSI-Hostadapter gefunden. Des weiteren ist das PPP-Protokoll (für den Modembetrieb) sowie eine NE2000-kompatible Netzwerkkarte vorhanden.

Diese Meldungen liefern einen guten Anhaltspunkt, ob die einkompilierten Treiber auch alle Devices erkannt haben. Allerdings ziehen sie beim Booten doch recht schnell vorbei. Die folgenden Tatstaturbefehle helfen, dieses Problem in den Griff zu bekommen:

<Shift><PgUp> nach oben scrollen
<Shift><PgDown> nach unten scrollen

Alternativ kann mit dem Befehl `dmesg | less` ein spezieller Speicherbereich im Kernel abgefragt werden, in dem die Bootmeldungen gespeichert werden. Unter SuSE-Linux wird das Ergebnis eines solchen Aufrufs direkt nach dem Systemstart in die Datei */var/log/boot.msg* geschrieben.