Linux Kernel HOWTO

Brian Ward (bri@blah.math.tu-graz.ac.at) und Peter Sütterlin (pit@uni-sw.gwdg.de) v0.80-3, 1. Juli 1998

Dieser Text gibt eine detaillierte Anleitung zu Konfiguration, Kompilation und Upgrades des Linux-Kernels für ix86-basierte Systeme.

Inhaltsverzeichnis

1 Einleitung

Dies ist die Version 0.80 des *Kernel HOWTO*. Diese deutsche Version basiert auf der englischen Version 0.80 vom 26. Mai 1997 von Brian Ward (`bri@blah.math.tu-graz.ac.at`).

1.1 Für wen ist dieser Text?

Für alle, die einen neuen Kernel kompilieren und installieren wollen und/oder müssen, weil sie

- Hardware besitzen, die von ihrem derzeitigen Kernel nicht unterstützt wird, wohl aber von neueren;
- ein bestimmtes Softwarepaket installieren wollen, das eine höhere Kernelversion voraussetzt,
- neugierig sind auf die Fähigkeiten eines neuen Kernels
- oder einfach nur wissen wollen, wie das alles im Prinzip funktioniert.

1.2 Copyright

1.3 Einige wichtige Vorbemerkungen

Einige der in diesem Text aufgeführten Beispiele gehen davon aus, daß die GNU-Versionen einiger Hilfsprogramme wie `tar`, `find` und `xargs` installiert sind. Dies ist zwar unter Linux Standard, sollte aber dennoch nicht unerwähnt bleiben.

Der Leser sollte weiterhin auch über die Struktur des Dateisystems auf dem Rechner Bescheid wissen. Wer sich darüber nicht ganz sicher ist, sollte auf jeden Fall die Ausgabe des `mount` Kommandos irgendwo aufschreiben, oder auch den Inhalt der Datei `/etc/fstab`. Dies kann einem mancherlei Unannehmlichkeiten ersparen, wenn man das System durch eine Unachtsamkeit in einen nicht bootfähigen Zustand versetzt hat.

Der gegenwärtig aktuelle stabile Kernel hat die Versionsnummer 2.0.30. Die angegebenen Beispiele beziehen sich auf diese Version. Obwohl dieser Text soweit wie möglich versionsunabhängig geschrieben wurde, kann es bei der schnellen Entwicklung von Linux nicht ausbleiben, daß Theorie (dieser Text) und Wirklichkeit (der allerneueste Kernel) etwas auseinanderwandern. Dies sollte keine größeren Probleme verursachen, kann aber den einen oder anderen vielleicht etwas verwirren.

Es gibt zwei unterschiedliche Versionen der Linux-Kernels, die *stabilen* und die *Entwickler*-Versionen. Man kann sie anhand ihrer Versionsnummer voneinander unterscheiden: Die stabilen Kernels haben eine *gerade* Versionsnummer; der erste stabile Kernel war 1.0.x, danach kam 1.2.x und die derzeit aktuelle stabile Kernelversion ist 2.0.x. Diese Versionen werden auch gerne als »Production Release« angesehen. Sie sind im Normalfall extrem stabil und fehlerfrei. Oder wie soll man eine Laufzeit von mehreren hundert Tagen ohne Absturz sonst nennen?

Die dazwischenliegenden, ungeraden Versionen wie 1.1.x, 1.3.x und der sicher bald kommende 2.1.x sind Testversionen, in denen neue Treiber in den Kernel aufgenommen werden, neue Konzepte oder Protokolle integriert werden. Kurzum, eine Spielwiese für die Kernel-Hacker. Kernels dieser Versionen können durchaus manchmal instabil sein und zu Abstürzen führen. Wer also auf einen absturzfreien Rechner angewiesen ist, sollte immer bei den stabilen Versionen bleiben. Wer aber gerne ein wenig experimentiert, sollte durchaus auch die Entwickler-Kernels ausprobieren, denn nur so können Fehler in diesen Versionen entdeckt werden, und nur so kann der nächste stabile Kernel auch wirklich stabil werden.

Dies war wohl ein Problem mit den derzeitigen 2.0.x Kernels, deren Vorversionen nicht intensiv genug getestet wurden, so daß einige dieser Kernels ein paar unentdeckte Fehler enthielten.

2 Wichtige Fragen und ihre Antworten

2.1 Was macht der Kernel überhaupt?

Der Unix-Kernel stellt eine Art Vermittler zwischen den Anwenderprogrammen und der Hardware des Computers dar. Er verwaltet den Arbeitsspeicher des Rechners und sorgt dafür, daß jedes laufende Programm (Prozeß) angemessene Anteile der Prozessor-Arbeitszyklen zugewiesen bekommt. Der Kernel stellt eine von der speziellen Hardware unabhängige Schnittstelle zum Zugriff auf diese Hardware zur Verfügung.

Es gibt zwar noch eine Menge weiterer Dinge, für die der Kernel zuständig ist, doch sind dies die wichtigsten, über die *jeder* Bescheid wissen sollte.

2.2 Warum sollte ich auf eine neuere Kernelversion umsteigen?

Zunächst unterstützen neuere Kernels praktisch immer mehr Hardware als die älteren, d.h. sie besitzen zusätzliche Treiber. Sie haben auch oft eine bessere Prozeßverwaltung und arbeiten dadurch manchmal deutlich schneller. Sie können einfach stabiler sein als die alten Versionen und dumme Fehler beheben, die sich in diesen noch versteckt hatten. Der häufigste Grund für einen Kernel-Upgrade sind wohl die Treiber und die korrigierten Fehler.

2.3 Welche Hardware wird von den neuen Kernels unterstützt?

Die Anzahl der unterstützten Hardware ist inzwischen sehr groß und wächst laufend weiter. Das *Hardware HOWTO* befaßt sich speziell mit diesem Thema. Alternativ kann man sich auch die Datei `/usr/src/linux/config.in` der Kernel-Quellen ansehen oder einfach bei einem `make config` schauen, was alles angeboten wird. Dabei wird allerdings nur aufgeführt, was die Standard-Kerneldistribution unterstützt. Darüber hinaus gibt es aber noch eine Unmenge an zusätzlichen Treibern, die unabhängig vom eigentlichen Kernel entwickelt und verwaltet werden. Die PCMCIA-Treiber, die als Module in den Kernel eingebunden werden, sind hierfür ein Beispiel.

2.4 Welche Versionen von gcc und libc benötige ich?

Linus empfiehlt im `README` der Kernel-Quellen eine Version, mit der dieser Kernel kompiliert werden sollte. Ist diese Version des `gcc` auf dem eigenen System noch nicht vorhanden, so sollte man sie installieren. Welche Version der libc man für eine bestimmte Version des `gcc`-Kompilers benötigt, kann man der Dokumentation des Kompilers entnehmen. Dies ist keine sehr schwierige Prozedur, man muß sich nur *genau* an die Anweisungen halten.

2.5 Was ist ein (ladbares) Modul?

Das sind Teile des Kernels, die nicht direkt in den Kernel eingebunden (*linked*) sind. Man kompiliert sie separat und kann sie nach Belieben in den laufenden Kernel einbinden und wieder entfernen. Aufgrund dieser extremen Flexibilität ist das inzwischen der bevorzugte Weg, um bestimmte Dinge im Kernel zu programmieren. Viele verbreitete Treiber, wie z.B. die PCMCIA- oder QIC-80/40-Treiber, sind solche ladbaren Module.

2.6 Wieviel Platz auf der Festplatte wird benötigt?

Das hängt etwas von der jeweiligen Systemkonfiguration ab. Die komprimierten Quellen des Kernels der Version 2.0.10 sind bereits fast 6 Megabytes groß, unkomprimiert belegen sie dann etwa 24 MB. Doch das ist noch nicht alles - man will den Kernel ja schließlich auch kompilieren. Für ein »typisches« System (Netzwerk, SCSI, drei oder vier verschiedene Dateisysteme, Unterstützung der seriellen und parallelen Schnittstellen) muß man etwa 30 MB einkalkulieren. Will man die eigentlichen Kernelquellen auch noch in ihrer komprimierten Form aufbewahren, kommen so 36 MB zusammen. Für Systeme, die weit mehr Treiber benötigen, kann es auch mehr sein. Weiterhin sollte man bedenken, daß neuere Kernels mit Sicherheit noch mehr Platz verbrauchen werden. Man sollte also mit Blick in die Zukunft nicht zu knapp kalkulieren. Andererseits ist es bei den heutigen Preisen für Festplatten auch kein allzu großes Problem mehr, bei Platzmangel einfach eine weitere Platte zu kaufen.

2.7 Wie lange dauert der »Kernelbau«?

Für die meisten Leute lautet die Antwort: »Ziemlich lang«. Die Leistungsfähigkeit des jeweiligen Systems sowie der zur Verfügung stehende Arbeitsspeicher geben hier den Ausschlag, ein wenig kann man diese Zeit auch durch die Anzahl der Treiber beeinflussen, die man einbinden will. Auf einem 486DX4/100 mit 16 MB RAM dauert die Kompilierung eines Kernels der Version 1.2 mit fünf Dateisystemen, Netzwerk- und Soundkartenunterstützung etwa 20 Minuten. Ein 386DX/40 mit 8 MB benötigt für dieselbe Konfiguration etwa 1,5 Stunden. Es empfiehlt sich also je nach Ausstattung, in der Zwischenzeit einen Kaffee zu kochen oder zu lesen. Eine andere Möglichkeit besteht darin, den Kernel von einem Bekannten mit einem schnelleren Rechner kompilieren zu lassen. Auf einem Pentium Pro Multiprozessorrechner dauert es nur ein paar Minuten :-).

3 Die Kernelkonfiguration

3.1 Download der Quellen

Die Quelldateien des Linux-Kernels können über Anonymous-FTP von `ftp.funet.fi` bezogen werden; sie befinden sich dort unterhalb des Verzeichnisses `/pub/Linux/PEOPLE/Linus`. Dieser Server wird an vielen Stellen gespiegelt, es lohnt sich also, zunächst mal auf dem lokalen Server nachzusehen. Einige nationale und internationale Mirrors sind:

USA

- `metalab.unc.edu:/pub/Linux/kernel`
- `tsx-11.mit.edu:/pub/linux/sources/system`

UK

- `sunsite.doc.ic.ac.uk:/pub/unix/Linux/sunsite.unc-mirror/kernel`

Österreich

- `ftp.univie.ac.at:/systems/linux/sunsite/kernel`

Deutschland

- `ftp.Germany.EU.net:/pub/os/Linux/Local.EUnet/Kernel/Linus`
- `sunsite.informatik.rwth-aachen.de:/pub/Linux/PEOPLE/Linus`

Frankreich

- `ftp.ibp.fr:/pub/linux/sources/system/patches`

Australien

- `sunsite.anu.edu.au:/pub/linux/kernel`

Generell ist auch eine Spiegelung von `metalab.unc.edu` ein guter Startpunkt. Die Datei `/pub/Linux/MIRRORS` enthält eine Liste mit den bekannten Spiegelungen.

Im angegebenen Verzeichnis befinden sich Unterverzeichnisse mit Namen wie `v1.3`, `v2.0` usw. Die höchste Versionsnummer stellt den aktuellsten Kernel dar, wobei ungerade Versionsnummern wie `v1.3` die *Beta-Versionen* sind. Wer also auf einen stabilen Kernel Wert legt, sollte bei den geraden Versionsnummern bleiben.

In den jeweiligen Verzeichnissen stehen dann die Linux-Quellen in Dateien mit den Namen `linux-x.y.z.tar.gz`; `x.y.z` ist dabei die Versionsnummer. Die Datei mit der höchsten Versionsnummer stellt den neuesten Kernel dar. In der Regel sollte dieses auch die beste Version sein.

Ebenfalls in diesem Verzeichnis befinden sich Dateien mit den Namen `patch-x.y.z.gz`. Dabei handelt es sich um sogenannte *Patch-Dateien*, mit denen man bereits vorhandene Kernel-Quellen einer älteren Version auf den neuesten Stand bringen kann. Näheres dazu steht im Abschnitt 5.

Eine weitere Möglichkeit, die Kernel-Quellen zu bekommen, stellen Mailbox-Systeme dar. Eine Liste von Mailboxen, die die Linux-Quellen anbieten, wird regelmäßig in der Newsgruppe `comp.os.linux.announce` gepostet.

Wenn Sie auf der Suche nach allgemeinen Informationen zu Linux und unterschiedlichen Distributionen sind, werfen Sie einen Blick auf diesen Server:

```
http://www.linux.org
```

3.2 Auspacken der Quellen

Die Installation der Quellen verlangt `root`-Rechte, so daß man sich entweder als `root` einloggen oder durch `su` die Identität wechseln muß. Der Kernel-Verzeichnisbaum residiert normalerweise im Verzeichnis `/usr/src`:

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

Wer - wie die meisten - bereits bei der ersten Installation seines Linux-Systems die Kernel-Quellen mitinstalliert hat, wird hier ein Verzeichnis mit dem Namen `linux` finden. Wer genug Platz auf der Festplatte hat und auf Nummer Sicher gehen will, kann dieses Verzeichnis beibehalten. Eine gute Idee ist es, die Versionsnummer des Kernels in diesem Verzeichnisbaum herauszufinden und das gesamte Verzeichnis umzubenennen. Hierfür gibt es zwei Möglichkeiten:

1. Es läuft noch der ursprüngliche Kernel, dann sind die Versionsnummern von Quellen und laufendem Kernel identisch. Die Versionsnummer des laufenden Kernels erfährt man durch den Befehl:

   ```
   # uname -r
   1.2.13
   ```

2. Wer sich nicht sicher ist: Die Versionsnummer für den Verzeichnisbaum steht im obersten Makefile `/usr/src/linux/Makefile` ganz am Anfang:

   ```
   VERSION = 1
   PATCHLEVEL = 2
   SUBLEVEL = 13
   ```

   Hier handelt es sich also um einen Kernel 1.2.13.

Der gesamte Verzeichnisbaum wird nun einfach umbenannt:

```
# mv linux linux-1.2.13
```

Eine weitere, sehr viel platzsparendere Möglichkeit ist es, den alten Verzeichnisbaum als gepacktes TAR-Archiv aufzuheben:

```
# tar cvfz linux-1.2.13.tgz linux
# rm -rf linux
```

Wer sicher ist, daß er die alten Quellen nicht mehr benötigt, oder einfach nicht genug Platz für zwei Versionen hat, kann auch einfach den vorhandenen Verzeichnisbaum löschen:

```
# rm -rf linux
```

In jedem Fall muß aber sichergestellt sein, daß es in `/usr/src` kein Verzeichnis mit dem Namen `linux` gibt, bevor man die neuen Quellen auspackt!

Dieses Auspacken geschieht mit dem Befehl

```
# tar xpvfz linux-x.y.z.tar.gz
```

oder, wenn die Datei bereits entkomprimiert ist, mit

```
# tar xpvf linux-x.y.z.tar
```

Wer das TAR-Archiv nicht im aktuellen Verzeichnis stehen hat, muß natürlich den vollen Pfad zu dieser Datei angeben, also etwa:

```
# tar xpvfz /home/ftp/incoming/linux-x.y.z.tar.gz
```

In jedem Fall wird während des Auspackens der Inhalt des Archives am Bildschirm angezeigt (durch die Option `v` bei tar), und am Ende steht dann ein neues Verzeichnis `linux` in `/usr/src`. In dieses sollte man nun wechseln und als erstes die Datei `README` lesen. Darin gibt es einen Abschnitt »INSTALLING the kernel«, in dem insbesondere einige symbolische Links aufgeführt sind, deren Existenz überprüft werden muß.

3.3 Konfiguration des Kernels

Hinweis: Der folgende Abschnitt wiederholt in Teilen den entsprechenden Abschnitt der Datei `README`.

Im Verzeichnis `/usr/src/linux` startet der Befehl

```
# make config
```

ein Konfigurationsskript, das eine ganze Menge Fragen stellt. Da dieses Skript zu seiner Abarbeitung zwingend die Shell `bash` benötigt, muß sichergestellt sein, daß diese Shell zur Verfügung steht: Entweder als `/bin/bash`, als `/bin/sh` oder aber unter dem in der Variablen `$BASH` angegebenen Pfad.

Mittlerweile gibt es einige Alternativen zu `make config`, die von den meisten als komfortabler angesehen werden. Wer mit dem X Window System arbeitet, sollte

```
# make xconfig
```

ausprobieren, sofern auch Tk installiert ist. Wer (n)curses installiert hat, und ein textbasiertes Menü bevorzugt, sollte

```
# make menuconfig
```

verwenden. Der große Vorteil beider Varianten ist, daß man bei einer Fehleingabe problemlos zum entspechenden Punkt zurückgehen und den Fehler korrigieren kann.

Die Fragen können im Normalfall mit `y` (Ja) oder `n` (Nein) beantwortet werden. Viele Treiber bieten außerdem die Option `m` an, dies steht für *Modul*. Dadurch wird der Treiber zwar kompiliert, aber nicht direkt in den endgültigen Kernel eingebunden, sondern als ladbares Modul bereitgestellt.

Seit der Version 2.0 des Kernels gibt es weiterhin zu fast allen Fragen die Antwortmöglichkeit `?`. Dadurch wird ein kurzer Hilfetext der jeweiligen Konfigurationsmöglichkeit angezeigt. Die dort gegebene Information ist in jedem Fall die aktuellste.

Im folgenden werden einige der wichtigeren Optionen genauer erläutert. Weniger wichtige oder offensichtliche Optionen sind weiter unten im Abschnitt 3.3.12 zusammengestellt.

3.3.1 Kernel math emulation

Wer keinen mathematischen Koprozessor hat, weil er noch einen einfachen 386 oder einen 486SX benutzt, *muß* hier mit `y` antworten. Wer einen Koprozessor hat, kann trotzdem mit `y` antworten, denn der Kernel erkennt selbständig, wenn ein Koprozessor vorhanden ist und ignoriert dann den Emulator. Lediglich der Kernel wird dadurch etwa 45 kB größer und verbraucht unnötig RAM.

Angeblich ist die Emulation des Koprozessors relativ langsam. Dies mag mit ein Grund sein, warum das X Window System auf Rechnern ohne Koprozessor extrem langsam ist. Aber das gehört eigentlich nicht hierher.

3.3.2 Normal (MFM/RLL) disk and IDE disk/cdrom support

Die meisten werden hier mit `y` antworten, denn dadurch wird die Unterstützung für die normalen Festplatten der PCs aktiviert. Lediglich Besitzer von reinen SCSI-Systemen können hier `n` antworten.

Bei einer positiven Antwort kann man weiterhin zwischen dem alten »disk-only« und dem neueren IDE-Treiber auswählen. Der alte Treiber unterstützt maximal zwei Festplatten an einem einzigen Host-Adapter, der neue erlaubt eine weitere Schnittstelle und unterstützt auch IDE-/ATAPI-CD-ROMs. Der neue Treiber ist etwa 4 kB größer und sicherlich auch verbessert, d.h. außer einer anderen Anzahl an Fehlern kann er auch die Leistung der Festplatte erhöhen. Dies gilt vor allem für die neueren EIDE-Geräte.

3.3.3 Networking support

Normalerweise wird man hier nur dann mit `y` antworten, wenn der Rechner an ein Netzwerk wie das Internet angeschlossen ist, oder wenn man SLIP, PPP, TERM oder etwas ähnliches verwenden will, um sich über ein Modem in ein Netzwerk einzuklinken. Da aber viele Pakete wie z.B. das X Window System Netzwerkunterstützung benötigen, selbst wenn der Rechner gar nicht wirklich an ein Netz angeschlossen ist, sollte man hier `y` antworten. Gleiches gilt für die etwas später kommende Frage nach »TCP/IP networking«. Wer sich nicht ganz sicher ist, sollte `y` sagen.

3.3.4 Limit memory to low 16MB

Es gibt einige fehlerhafte 386er DMA-Controller, die RAM-Bereiche oberhalb von 16 MB nicht fehlerfrei adressieren können. In den seltenen Fällen, daß man einen solchen besitzt, sollte man hier mit `y` antworten.

3.3.5 System V IPC

Eine der besten Definitionen, was IPC (Inter Process Communication) ist, findet sich im Glossar von Büchern über Perl. Kein Wunder, denn gerade Perl-Programmierer verwenden es, um verschiedenen Prozesse miteinander kommunizieren zu lassen. Auch viele andere Programmpakete (z.B. DOOM) verwenden IPC, es ist also keine gute Idee, es zu deaktivieren, außer man weiß genau was man tut.

3.3.6 Processor type (386, 486, Pentium, PPro)

In älteren Kernels lautete die Frage: »Use -m486 flag for 486-specific optimizations«. Früher wurden dadurch bestimmte Optimierungen für einen speziellen Prozessortyp aktiviert. Die Kernels wurden dadurch etwas in der Größe verändert, liefen aber auch auf anderen Prozessortypen. Dies gilt aber inzwischen nicht mehr, man sollte unbedingt den Prozessortyp angeben, für den der Kernel gedacht ist. Lediglich ein 386er Kernel arbeitet auf allen Rechnern.

3.3.7 SCSI support

Wer einen SCSI-Adapter und -Geräte besitzt, antwortet hier mit `y`. Es werden dann weitere Fragen zu Unterstützung unterschiedlicher Hardware (CD-ROM, Bandlaufwerk) gestellt. Das *SCSI HOWTO* gibt hier nähere Hilfestellungen.

3.3.8 Network device support

Wer eine Netzwerk-Karte besitzt, oder SLIP/PPP verwenden will, sagt hier `y`. Das Konfigurationsskript fragt dann nach der genauen Art der Hardware und welche Protokolle verwendet werden sollen.

3.3.9 Dateisysteme

Das Konfigurationsskript erfragt die Unterstützung für folgende Dateisysteme:

Standard (minix)

Die neueren Distributionen legen kaum noch minix-Dateisysteme an, und nur noch wenige Leute benutzen es. Dennoch kann es nicht schaden, dieses Dateisystem zu unterstützen. Einige Rettungsdisketten verwenden es, und auch sehr viele Floppies, die unter Unix verwendet werden, verwenden es, da es auf diesen viel einfacher zu benutzen ist.

Extended fs

Dies war die erste Version des erweiterten Dateisystem *extfs*, sie ist aber kaum noch in Gebrauch. Wer es benötigt, weiß das; alle anderen können ohne Gefahr mit `n` antworten.

Second extended

Dies ist das am weitesten verbreitete Dateisystem, praktisch alle neueren Distributionen verwenden es. Ziemlich sicher wird man hier also mit `y` antworten.

xiafs filesystem

Dieses Dateisystem - eine Alternative zu extfs - war einmal recht verbreitet, heute arbeitet aber kaum noch jemand damit.

msdos

Wer auf seine DOS-Partition unter Linux zugreifen will oder Disketten im DOS-Format mounten will, antwortet hier `y`.

vfat

Das Dateisystem von Win95, das endlich auch lange Dateinamen erlaubt. Wer Zugriff auf seine Win95-Partition haben will, sagt `y`.

umsdos

Dieses Dateisystem benutzt das einfache DOS System, um darauf ein Unix-Dateisystem mit allen Vorzügen (lange Dateinamen etc.) anzulegen. Ganz nett, wenn man Linux nur mal probieren will und seine DOS-Partition nicht neu formatieren will. Aber es ist relativ langsam und nicht sehr sinnvoll für diejenigen, die kein DOS verwenden.

/proc

Eine der größten Ideen seit der Erfindung des Milchpulvers; die Idee ist wohl von den Bell Labs abgeschaut. Man legt nicht wirklich ein Verzeichnis `/proc` auf der Festplatte an, dies ist vielmehr eine Schnittstelle zum Kernel und den laufenden Prozessen in Form eines Dateisystems. Sehr viele Hilfsprogramme wie z.B. `ps` benutzen es; auch einige Shells, insbesondere `rc`, verwenden `/proc/self/fd`, das auf anderen Systemen als `/dev/fd` bekannt ist, für den I/O. Diese Frage sollte *unbedingt* mit `y` beantwortet werden, da sehr viele wichtige Dinge und Programme unter Linux darauf angewiesen sind.

NFS

Wer seinen Rechner an ein Netzwerk angeschlossen hat und Verzeichnisse von anderen Rechnern in das eigene Dateisystem einklinken will, sagt hier `y`.

ISO9660

Das ist das Dateisystem der meisten CD-ROMs. Wer ein CD-Laufwerk besitzt und es unter Linux benutzen will, sagt auch hier `y`.

HPFS

Das ist das Dateisystem von OS/2. Im Moment kann es nur gelesen, aber nicht geschrieben werden.

System V und Coherent

System V und Coherent sind andere Varianten von Unix für PCs. Sie verwenden ein eigenes Dateisystem, das Linux natürlich auch unterstützt.

UFS

UFS wird von Sun und FreeBSD verwendet. Wer entsprechende Systeme hat, wird hier mit `y` antworten und kann dann - bislang allerdings nur lesend - auf deren Partitionen zugreifen.

Amiga FFS

Dies ist ein experimenteller Treiber für das Amiga Dateisystem.

Welche Dateisysteme brauche ich denn? Auf jeden Fall diejenigen, die im alten System auch benötigt werden. Dies erfährt man, indem man den `mount` Befehl ohne Parameter benutzt:

```
$ mount
/dev/hda1 on / type ext2 (defaults)
/dev/hda3 on /usr type ext2 (defaults)
none on /proc type proc (defaults)
/dev/fd0 on /mnt type msdos (defaults)
```

In jeder einzelnen Zeile gibt das Wort nach `type` das jeweilige Dateisystem an. In diesem Beispiel sind `/` und `/usr` vom Typ ext2, also *Second Extended*. Das `/proc` System wird unterstützt, außerdem ist eine Diskette unter `/dev/fd0` mit dem DOS-FAT gemounted.

Eine andere Möglichkeit, die derzeit unterstützten Dateisysteme herauszubekommen, stellt das `/proc`-Verzeichnis dar, vorausgesetzt natürlich, es ist im laufenden Kernel aktiviert. Hier ein Beispiel:

```
$ cat /proc/filesystems
        ext2
nodev   proc
        msdos
nodev   nfs
```

Nur sehr selten genutzte Dateisysteme in den Kernel einzubinden führt schnell zu einem unnötig großen Kernel; gleiches gilt natürlich auch für selten genutzte Hardware-Treiber. Eine sehr elegante Methode, dies zu umgehen, stellen die *Module* dar. Dies wird in einem späteren Abschnitt beschrieben. Warum ein großer Kernel generell von Nachteil ist, wird im Abschnitt 7 erläutert.

3.3.10 Character Devices

Hier kann man die Treiber für den Drucker bzw. den Parallelport, Bus-Mäuse, PS/2-Mäuse (das PS/2 Protokoll wird bei vielen Notebooks für den Trackball verwendet) sowie einige Bandlaufwerke aktivieren.

3.3.11 Sound card

Wer sein `biff` unbedingt bellen lassen will `;-)`, sollte hier mit `y` antworten. Dann wir etwas später ein weiteres Konfigurationsprogramm kompiliert und ausgeführt, das alles über die Konfiguration der Soundkarte abfragt. Dazu ein Hinweis: Bei der Frage, ob der komplette Sound-Treiber installiert werden soll, kann man mit `n` antworten und im darauffolgenden Dialog nur die gewünschten Komponenten aktivieren, das spart etwas Platz im Kernel. Das *Sound HOWTO* gibt hier weitere Informationen.

3.3.12 Weitere Konfigurationsmöglichkeiten

Hier sind bei weitem nicht alle auftretenden Optionen aufgeführt. Zum einen ändert sich das Konfigurationsskript mit jedem neu hinzugekommenen Treiber (was recht häufig geschieht), außerdem sind viele der Fragen selbsterklärend: Unterstützung für die Netzwerk-Karte 3Com 3C509 braucht man genau dann, wenn man diese Karte auch besitzt, usw. Im Zweifel gibt die Online-Hilfe durch Eingabe von `?` einen kurzen Hinweis.

3.3.13 Kernel hacking

Hierzu ein Zitat aus dem `README` von Linus:

> Die Aktivierung der »Kernel hacking« Option führt im Normalfall zu einem größeren oder langsameren Kernel (oder sogar beides). Dadurch kann der Kernel sogar definitiv instabiler werden, da manche Routinen dann etwas unterschiedlich kompiliert werden, um gezielt schlechte Routinen zum Absturz zu bringen und dadurch Fehler im Kernel aufzudecken (`kmalloc()`). Soll der Kernel ganz normal verwendet werden, sollte man deshalb hier mit `n` antworten.

3.4 Das Makefile

Wird `make config` ordnungsgemäß beendet, teilt eine kurze Meldung mit, daß der Kernel nun konfiguriert ist und man das »Top-Level Makefile« auf zusätzliche Konfigurationsoptionen hin untersuchen soll.

Derzeit ist die einzige Option, die im Makefile direkt eingestellt wird, die Unterstützung von SMP (Symmetric Multi-Processing) für Mainboards mit mehr als einem Prozessor. Wer ein solches Board besitzt, sollte unbedingt auch die Datei `/usr/src/linux/Documentation/SMP.txt` lesen.

4 Die Kompilierung

4.1 clean und depend

Am Ende der Konfiguration weist das Skript ebenfalls darauf hin, man solle ein `make dep` sowie ein `make clean` durchführen. Dies sollte man in jedem Fall tun, damit die wechselseitigen Abhängigkeiten der Quell- und Include-Dateien richtig zusammengestellt werden. Dies dauert nicht sehr lange; auf meinem DX/2-80 aber fast länger als die komplette Kompilierung des Kernels. Danach löscht man mit `make clean` alle alten Objekt-Dateien und stellt so sicher, das sie wirklich neu übersetzt werden. Dieser Schritt ist wirklich wichtig, und einige fehlgeschlagene Kompilierungsversuche beruhten nur auf einem vergessenen:

```
# make dep
# make clean
```

4.2 make

Nun kommt der zeitraubende Teil.

```
# make zImage
```

kompiliert den gesamten Kernel und hinterläßt die Datei `zImage` im Verzeichnis `arch/i386/boot`. Dies ist der neue, komprimierte Kernel.

```
# make zdisk
```

macht dasselbe, installiert aber diesen neuen Kernel gleich auf eine Diskette, die man hoffentlich rechtzeitig in das Laufwerk A: geschoben hat. Die letztere Methode ist ziemlich praktisch, um neue Kernels relativ gefahrlos zu testen. Wenn er aus irgendeinem Grund nicht richtig funktioniert oder gar abstürzt, nimmt man einfach die Diskette aus dem Laufwerk und bootet den alten Kernel. Generell ist es immer eine gute Idee, eine solche bootfähige Diskette mit einem funktionierenden Kernel zur Hand zu haben. Denn irgendwann kommt immer der Tag, an dem man aus Versehen den Kernel von der Festplatte löscht oder eine ähnliche Dummheit begeht.

Alle halbwegs aktuellen Kernels sind komprimiert, daher das `z` am Anfang der Namen. Ein solcher komprimierter Kernel entpackt sich automatisch selber, wenn er bootet.

4.3 Andere Optionen (»Targets«) für make

`make mrproper` macht etwas ähnliches wie `clean`, aber sehr viel umfassender. Manchmal ist das notwendig, um ein wirklich »sauberen« Verzeichnisbaum zu generieren. Dabei werden aber auch die alten Einstellungen der Konfiguration gelöscht; eventuell sollte man sich deshalb eine Sicherungskopie der Datei `.config` aufheben, um bei Bedarf die alten Einstellungen nachsehen zu können.

`make oldconfig` versucht, die Kernel-Konfiguration automatisch anhand einer alten Konfigurationsdatei durchzuführen. Wer noch nie einen Kernel kompiliert hat, sollte diese Option besser nicht benutzen, da sicherlich die eine oder andere Einstellung verändert werden muß.

`make modules` wird in einem eigenen Abschnitt beschrieben.

4.4 Installation des neuen Kernels

Jetzt, nachdem der Kernel erfolgreich den eigenen Wünschen entsprechend kompiliert wurde, ist es an der Zeit, ihn zu installieren. Die meisten Leute benutzen LILO, den **Li**nux **Lo**ader, um Linux und eventuell auch einige weitere Betriebssysteme zu booten. Für diesen Fall genügt meist ein einfaches `make zlilo`. Dabei wird der Kernel kompiliert, installiert und `lilo` aufgerufen. Danach sollte alles für einen Reboot des neuen Kernels bereit sein.

Das funktioniert aber nur dann, wenn `lilo` folgendermaßen eingestellt und installiert ist: Der Kernel ist `/vmlinuz`, `lilo` befindet sich in `/sbin` und die Konfigurationsdatei für `lilo` (`/etc/lilo.conf`) stimmt mit dieser Einstellung überein. Ist dies nicht der Fall, muß man `lilo` selber aufrufen, nachdem der neue Kernel an die richtige Stelle kopiert wurde.

Eigentlich ist `lilo` ein Paket, das sehr einfach zu installieren und auch zu benutzen ist. Dennoch lassen sich manche von der Konfigurationsdatei (`/etc/lilo.conf` oder, bei älteren Versionen, `/etc/lilo/config`) verwirren. Ein typischer Eintrag in dieser Datei sieht so aus:

```
image = /vmlinuz
    label = Linux
    root = /dev/hda1
    ...
```

Der Eintrag `image =` gibt den vollen Pfad des gegenwärtig installierten Kernels an; die meisten verwenden `/vmlinuz`. `label` gibt einen Namen, unter dem man diesen Eintrag, wenn mehrere Einträge vorhanden sind, ansprechen kann, und `root` gibt diejenige Partition der Festplatte an, die als `/` gemountet werden soll. Um für das hier beschriebene System den neuen Kernel zu installieren, sollte man also vom alten Kernel eine Sicherheitskopie machen, den neuen Kernel an die angegebene Stelle kopieren und `lilo` aufrufen:

```
# mv /vmlinuz /Old_Kernel
```

```
# mv /usr/src/linux/arch/i386/boot/zImage /vmlinuz
# lilo
```

Bei älteren Versionen von `lilo` muß die letzte Zeile eventuell

```
# /etc/lilo/install
```

oder sogar

```
# /etc/lilo/lilo -C /etc/lilo/config
```

lauten.

4.4.1 Beispiel für eine LILO-Konfiguration

LILO kann im Prinzip beliebig viele verschiedene Systeme booten, deshalb kann man es auch sehr gut dazu verwenden, neuen und alten Kernel gleichzeitig bootfähig zu machen. Hierzu ein Beispiel, wobei Zeilen, die mit einem # beginnen, als Kommentare verstanden werden:

```
# LILO Konfigurationsdatei
#            von Peter Sütterlin, September 1996
#
# Start des globalen Abschnittes

boot = /dev/hda
message=/etc/lilo.bootmenue
compact
prompt
timeout = 100
image = /vmlinuz
        label = 1
        root = /dev/hda2
image = /vmlinuz_old
        label = 2
        root = /dev/hda2
other = /dev/hda1
        label = 3
        table = /dev/hda
```

Der erste Eintrag gibt an, wo LILO installiert werden soll; hier auf der ersten Festplatte. In der zweiten Zeile wird eine Datei angegeben, deren Inhalt LILO beim Laden am Bildschirm ausgibt. In dieser Datei steht etwa folgendes:

```
^LBitte eine der angegebenen Konfigurationen auswaehlen:

Def. --> 1  Linux (neuer Kernel)

         2  Linux (alter Kernel)

         3  MSDOS 6.0
```

Das `^L` (`CONTROL-L`) am Anfang bewirkt dabei, daß der Bildschirm gelöscht wird. Der dritte Eintrag (`compact`) optimiert den Ladevorgang. Das `prompt` in der nächsten Zeile bewirkt, daß LILO auf eine Eingabe des Benutzers wartet, der nun auswählen kann, welche der drei Konfigurationen er starten will. Auf diese Eingabe wartet LILO

10 Sekunden (`timeout = 100`) und lädt dann automatisch den ersten Eintrag der folgenden Liste. Diese Liste enthält hier drei Einträge. Die ersten beiden beginnen mit `image =` und weisen damit auf Linux-Systeme hin. Der dritte Eintrag (`other = /dev/hda1`) betrifft ein nicht näher spezifiziertes Betriebssystem, dessen Boot-Partition `/dev/hda1` ist; in diesem Fall ist es eine DOS-Partition.

Aus diesen drei Möglichkeiten kann man nun, wie in der Meldung angegeben, durch Drücken der Tasten `1`, `2` oder `3`, entsprechend den Einträgen `label=` in den einzelnen Abschnitten, eine auswählen.

Dies beschreibt nur äußerst knapp die unzähligen Möglichkeiten, die LILO bietet. Wer sich näher darüber informieren will, sollte die sehr ausführliche Dokumentation, die mit LILO mitgeliefert wird, studieren.

5 Patchen des Kernels

Die Quelldateien des Linux-Kernels sind mittlerweile mit immerhin 6 MB sehr umfangreich und es wäre unbequem, sich mit jeder neuen Kernelversion die kompletten Dateien erneut zu besorgen. Stattdessen werden nur diejenigen Teile, die sich verändert haben, in Form eines inkrementellen Patches verbreitet. Wer also z.B. die vollständigen Kernel-Quellen der Version 2.0.0 besitzt und sich die Datei `patch-2.0.1.gz` besorgt, kann damit seinen Verzeichnisbaum auf die Version 2.0.1 upgraden, indem er diese Patch-Datei einspielt.

5.1 Einspielen eines Patches

Wer der Sache noch nicht so recht traut, kann zunächst mit

```
# cd /usr/src
# tar cvfz linux_old.tgz linux
```

eine Sicherungskopie der alten Version anlegen, bevor er den neuen Patch einspielt. Vorher empfiehlt es sich aber in jedem Fall, ein `make clean` durchzuführen.

Das eigentliche »Patchen« geschieht nun durch folgende Befehle:

```
# cd /usr/src
# zcat patch-2.0.1.gz | patch -p0 2>&1 | tee patch.out
```

Jetzt werden jede Menge Meldungen über den Bildschirm rasen über »hunks«, die eingespielt werden, und ob das erfolgreich war. Da es recht schwierig ist, in diesem vorbeirasenden Wirrwarr Fehlermeldungen zu entdecken, wurden im Beispiel die gesamten Meldungen zusätzlich in der Datei `patch.out` mitprotokolliert. Diese kann man nun nach der Zeichenkette `fail` durchsuchen, um festzustellen, ob alles glatt gegangen ist. Eine noch einfachere Möglichkeit ist es, `patch` mit der Option `-s` zu starten. Dadurch wird `patch` veranlaßt, nur noch bei Fehlern Meldungen am Bildschirm auszugeben.

Außer durch die Ausgabe von `patch` lassen sich gescheiterte Patch-Versuche auch folgendermaßen auffinden: Schlägt ein Patch-Versuch fehl, so werden die beanstandeten Abschnitte der zu patchenden Datei mit der Endung `.rej` versehen abgespeichert. Um diese »Reject«-Dateien (zurückgewiesenen Dateien) zu finden, kann man sich des Programmes `find` bedienen:

```
# find . -name '*.rej' -print
```

Dieses listet alle Dateien mit der Endung `.rej` auf, die sich im aktuellen Verzeichnis und dessen Unterverzeichnissen befinden.

Sind keine Fehler aufgetreten, kann man die neue Kernelversion wie gewohnt kompilieren.

Wem das zu kompliziert ist - bitte, Linux wäre nicht Linux, wenn es dafür nicht auch eine Lösung gäbe. Im Verzeichnis `/usr/src/linux/scripts` steht ein Shell-Skript mit dem Namen `patch-kernel`, welches einem fast alle Arbeit abnimmt. Es erkennt automatisch die Versionsnummer der momentan installierten Kernel-Quellen und sucht dann im aktuellen Verzeichnis nach Patch-Dateien mit höheren Versionsnummern als der installierte Kernel. Diese werden dann automatisch eine nach der anderen eingespielt. Tritt ein Fehler auf, bricht das Skript selbstverständlich ab.

5.2 Was tun bei Fehlern?

Wer niemals selber in den Kernel-Quellen herumeditiert, für den sollten die Patches eigentlich immer ohne Probleme einspielbar sein. Lediglich in alten Kernel-Versionen gab es ein Problem, wenn eine Datei mit dem Namen `config.in` gepatcht werden sollte. Wurde diese aber verändert, um der eigenen Rechnerkonfiguration Rechnung zu tragen, so fand `patch` sich in dieser Datei nicht mehr zurecht und konnte den Patch nicht einspielen. Diese Probleme sind aber inzwischen berücksichtigt und sollten nicht mehr auftreten.

Kommt es dennoch einmal soweit, sollte man als erstes die »Reject«-Datei ansehen und sie dann mit der zu patchenden Datei vergleichen. Oft weicht die Originaldatei nur geringfügig von der Form ab, die die Patch-Datei erwartet. Da `patch` jeweils zeichenweise vergleicht, kann bereits ein fehlendes Leerzeichen oder eine Leerzeile zuviel ein erfolgreiches Einspielen des Patches verhindern.

Eine letzte mögliche Fehlerquelle besteht darin, daß man einen Patch außerhalb der Reihe einspielen will, also z.B. einen mit einer geringeren Versionsnummer als die bereits vorhandenen Kernel-Quellen. Dann hält `patch` zumeist mit folgender Meldung an:

```
previously applied patch detected:  Assume -R?
```

Antwortet man darauf mit `y`, so versucht `patch`, die vorhandenen Quellen wieder auf den alten Stand zurückzusetzen, wird dabei aber ziemlich sicher scheitern. Dann wird man kaum umhinkommen, sich die vollständigen Kernel-Quellen neu zu besorgen. Ein Grund mehr also, das Skript `patch-kernel` zu verwenden, denn dieser Fehler tritt damit sicherlich nicht auf.

Hat man einen Patch versehentlich eingespielt, so kann er mit dem Befehl `patch -R` wieder rückgängig gemacht werden.

5.3 Diese lästigen .orig Dateien...

`patch` legt von allen veränderten Dateien Sicherungskopien mit der Endung `.orig` an. Diese nehmen bereits nach einigen eingespielten Patches einen nicht unerheblichen Platz ein; von 1.1.48 bis 1.1.51 waren es über ein halbes MB.

Auch hier hilft der `find`-Befehl, all diese Dateien auf einmal zu löschen:

```
# find .  -name '*.orig' -exec rm -f {} ';'
```

Alternativ kann auch das GNU-Programm `xargs` verwendet werden:

```
# find .  -name '*.orig' | xargs rm
```

Die Benutzer von `patch-kernel` sind wiederum im Vorteil, denn dieses Skript löscht automatisch alle `.orig`-Dateien.

5.4 Andere Patches

Es gibt außer den von Linus verwalteten Patches auch noch andere, nicht zur Standard-Kerneldistribution zugehörige Patches. Diese sind meist für spezielle Hardware und befinden sich oft noch im experimentellen Stadium. Viele dieser Patches werden vielleicht in späteren Kernel-Versionen in die Standard-Distribution aufgenommen. Quota und Unterstützung für den Iomega ZIP-Drive sind diesen Weg gegangen. Solange sie aber noch »exotisch« sind, können diese Patches dazu führen, daß die Standard-Patches von Linus nicht mehr fehlerfrei eingespielt werden können. In diesem Fall hat man dann nur die Möglichkeit, den Patch vor dem Kernel-Upgrade rückgängig zu machen, sich komplett neue Kernel-Quellen zu besorgen oder zu versuchen, die fehlgeschlagenen Patches von Hand zu korrigieren. Dies kann auf Dauer recht frustrierend sein. Wer nicht mit jedem neuen Kernel in den Quellen herumsuchen will, der sollte den externen Patch rückgängig machen (`patch -R`) bevor er den offiziellen Patch einspielt, oder gleich die volle Kernel-Version neu installieren. Erst danach kann man sehen, ob sich der inoffizielle Patch in die neuen Kernel-Quellen einspielen läßt. Ist dies nicht der Fall, kann man entweder mit dem alten Kernel weiterarbeiten und auf ein Upgrade verzichten, bis jemand anders diesen Patch an die neue Kernelversion angepaßt hat, oder aber selber in den Quellen und dem Patch herumsuchen und selber versuchen, ihn zum Laufen zu bekommen.

Wie viele dieser inoffiziellen Patches gibt es? Nun, es tauchen immer wieder welche auf, und wer die Newsgroups verfolgt, wird bald über sie stolpern. Auf der anderen Seite bieten die neuen Kernels durch die ladbaren Module eine viel elegantere Methode, um neue Treiber und ähnliches in den Kernel einzubinden, ohne daß dabei die Originalquellen verändert werden müßten. Aus diesem Grund wird die Anzahl der wirklichen Patches mit der Zeit zurückgehen.

6 Zusätzliche Pakete

Der Linux-Kernel besitzt eine Vielzahl an Merkmalen, die in den eigentlichen Quelldateien kaum erwähnt werden. Sie werden typischerweise durch externe Hilfsprogramme angesprochen und ausgenutzt. Einige der am weitesten verbreiteten sind hier aufgeführt.

6.1 kbd

Die Linux Console besitzt eine Unmenge an Besonderheiten, vielleicht sogar zu viele. Darunter sind die Möglichkeiten den Zeichensatz zu wechseln, die Tastatur umzudefinieren, den Videomodus umzuschalten (in neueren Kernelversionen) usw. Das `kbd` Paket stellt Programme zur Verfügung, mit denen der Benutzer all dies machen kann, zusätzlich jede Menge Zeichensätze und Tastaturlayouts für fast jede denkbare Tastatur. Dieses Paket findet man auf denselben Servern, auf denen auch die Kernel-Quellen liegen. Die derzeit aktuelle Version ist `kbd-0.91.tar.gz`.

6.2 util-linux

Rik Faith (`faith@cs.unc.edu`) hat eine Kollektion von Hilfsprogrammen zusammengestellt, die durch einen dummen Zufall den Namen *util-linux* bekommen habe. Inzwischen wird das Paket von Nicolai Langfeldt (`util-linux@math.uio.no`) betreut; man findet es z.B. auf:

```
metalab.unc.edu:/pub/Linux/system/misc
```

Darin enthalten sind Programme wie `setterm`, `rdev` oder `ctrlaltdel`, die für die Funktion des Kernels relevant sind. Darüber hinaus sind aber noch jede Menge andere Programme enthalten, die nicht unbedingt alle installiert werden sollten. Wie Rik im `README` angibt: Nie ohne zu überlegen installieren, es könnte sonst durchaus etwas nachhaltig durcheinander geraten.

6.3 hdparm

Dieses Programm erlaubt es, die Kommunikation mit der Festplatte zu beeinflussen und zu optimieren. Wie viele andere Programme war `hdparm` zunächst einer der inoffiziellen Kernel-Patches zusammen mit einigen Hilfsprogrammen. Die Patches wurden inzwischen in den Standard-Kernel übernommen, die Programme werden aber immer noch separat betreut und verteilt.

6.4 gpm

`gpm` steht als Abkürzung für »General Purpose Mouse«, also Vielzweck-Maus. Mit diesem Programm können Textstücke mit Cut-and-Paste zwischen den verschiedenen virtuellen Konsolen ausgetauscht werden.

7 Einige Fußangeln

7.1 make clean

Wenn sich der neu installierte Kernel seltsam verhält, stehen die Chancen recht hoch, daß vergessen wurde, vor der Kompilation ein `make clean` durchzuführen. Die typischen Symptome reichen von einem totalen Systemabsturz über unerklärliche I/O-Probleme bis hin zu einer Verlangsamung des Systems. `make dep` sollte man auch nie vergessen.

7.2 Sehr große und/oder langsame Kernel

Belegt der Kernel zu viel des wertvollen Hauptspeichers, oder dauert die Kompilierung trotz des nagelneuen 786DX/6-440 viel zu lange, sind möglicherweise einige gar nicht benötigte Dinge (Gerätetreiber, Dateisysteme usw.) in den Kernel integriert. Ein typisches Anzeichen für einen solchen überfrachteten Kernel ist eine überhöhte Swap-Aktivität. Dabei werden die jeweils gerade nicht benötigten Speicherbereiche auf die Festplatte ausgelagert und bei Bedarf wieder zurückgeladen. Ein zu großer Kernel läßt weniger physikalischen Hauptspeicher für die Anwendungen übrig, so daß das Swappen früher einsetzt. Erkennbar ist es an einer dauernden Festplattenaktivität.

Einen solchen Monster-Kernel kann man aber oft vermeiden. Generell gilt: Was man nicht benötigt, wird nicht konfiguriert; was man nur selten benötigt, sollte nach Möglichkeit als ladbares Modul kompiliert werden.

Den tatsächlich vom Kernel belegten Anteil des Hauptspeichers findet man folgendermaßen heraus: Durch den Befehl `cat /proc/meminfo` oder `free` erfährt man den Betrag des gesamt zur Verfügung stehenden Hauptspeichers (`Mem: total` bzw. `MemTotal`). Diesen Wert subtrahiert man einfach vom insgesamt installierten Speicher. Eine andere Möglichkeit bietet der Befehl `dmesg`, mit dem man sich die Boot-Meldungen des Kernels ansehen kann. Ziemlich am Anfang steht dort eine Zeile:

```
Memory: 15124k/16384k available (552k kernel code, 384k reserved, 324k data)
```

Wer nun aber dennoch auf einen großen Kernel angewiesen ist, kann folgendes versuchen:

```
# make bzimage
```

In diesem Fall ist es vermutlich ebenfalls notwendig, eine neuere Version des Linux Loaders LILO zu installieren.

7.3 Die Kernel-Kompilierung schlägt fehl

Eine mißlungene Kernel-Kompilierung kann vielerlei Ursachen haben. Die einfachste ist wieder ein vergessenes `make dep ; make clean`. Ebenfalls kann ein fehlgeschlagener Patch (man suche nach `.rej` Dateien) oder ein anderweitig durcheinandergeratener Quell-Verzeichnisbaum die Kompilierung verhindern. In diesem Fall ist es oft die schnellste Lösung, sich einen kompletten neuen Kernel-Quellcode zu besorgen und zu installieren.

Die neuen Kernels der 2.0.x Generation bieten die Möglichkeit, die Konfiguration menügesteuert entweder mit `make menuconfig` oder `make xconfig` durchzuführen. Diese sehr komfortablen Programme hatten zeitweise kleinere Probleme mit dem Soundtreiber. Wer eine dieser Konfigurationshilfen verwendet und den Kernel nicht ordnungsgemäß kompilieren kann, sollte mal versuchen, ein normales `make config` durchzuführen, das hat manchmal geholfen.

Weiterhin kann es sein, daß man versucht, einen Kernel der Version 1.2.x mit einem neueren ELF-Compiler (gcc 2.6.3 und höher) zu übersetzen. Typischerweise bekommt man dabei haufenweise Fehlermeldungen der Art `****** undefined`. Normalerweise ist dieser Fehler schnell behoben: Am Anfang der Datei `arch/i386/Makefile` müssen die folgenden Zeilen eingefügt werden:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Danach sollte sich der Kernel mit

```
# make dep
# make clean
# make zImage
```

übersetzen lassen.

Etwas schwieriger aufzudecken sind falsche oder falsch installierte Versionen des Compilers `gcc`. Das `README` von Linus gibt Hinweise dazu und auch auf einige symbolische Links, deren Korrektheit man überprüfen sollte.

In wirklich sehr seltenen Fällen kann `gcc` auch aufgrund von Hardware-Problemen abstürzen. Die Fehlermeldung lautet dann in etwa `xxx got fatal signal 11` und man hat keinerlei Ahnung, was das bedeutet. Insbesondere diese `Signal 11` Abstürze deuten immer auf Probleme mit der Speicher-Hardware hin. Dies können fehlerhafte SIMMs oder Cache-Bausteine sein, oder einfach nur zu knapp eingestellte Timings im BIOS des Mainboards. Oft hilft es z.B., in einem solchen Fall die Anzahl der Waitstates im »Advanced Chipset Setup« zu erhöhen.

Es gibt einen eigenen Hilfetext, der sich speziell mit dieser Art von Problemen beschäftigt:

```
http://www.bitwizard.nl/sig11/
```

7.4 Der neu installierte Kernel bootet nicht

`lilo` wurde nach der Installation nicht ausgeführt. Wurde z.B. der alte Kernel nur umbenannt, so bootet LILO in diesem Fall trotzdem noch die alte Version, da es den Kernel nur über seine Position auf der Festplatte lädt, nicht über seinen Namen. Erst bei einem erneuten Lauf von `lilo` wird dieser Positionszeiger auf den neu installierten Kernel gesetzt.

Eventuell ist auch die Konfigurationsdatei fehlerhaft. Ein oft auftretender Fehler, den man nur zu leicht übersieht, ist z.B. wenn man anstelle von `boot = /dev/hda` die Zeile `boot = /dev/hda1` eingetragen hat.

Ein weiterer Grund für Probleme mit LILO können große Festplatten mit mehr als 1024 Zylindern sein. Das *LILO mini-HOWTO* oder die Dokumentation von LILO selber geben dazu nähere Informationen.

7.5 LILO vergessen: Das System bootet nicht mehr

Wohl dem, der sich beizeiten eine Rettungsdiskette oder zumindest eine Bootdiskette (`make zdisk`) erstellt hat. Mit einer Bootdiskette kann man einfach das System von der Floppy starten und dann `lilo` aufrufen.

Besitzt man nur eine Rettungsdiskette, wird es etwas komplizierter; man muß den neuen Kernel von der Festplatte auf eine weitere Diskette übertragen. Dazu muß man einiges über sein System wissen:

- Auf welcher Partition befindet sich das Root-Verzeichnis (`/`) des Systems?
- Auf welcher Partition steht der kompilierte Kernel? Dies kann entweder ebenfalls das Root-Verzeichnis sein, wenn man den Kernel bereits dorthin kopiert hat, oder aber die Partition, auf der sich das Verzeichnis `/usr/src/linux` befindet.
- Den Typ des Dateisystems, auf dem sich der Kernel befindet, also z.B. `ext2` oder `minix`.

Im folgenden Beispiel ist `/` auf `/dev/hda1`, und das gesamte `/usr`-Verzeichnis - also auch `/usr/src/linux` - befindet sich auf der Partition `/dev/hda3` und ist vom Typ `ext2`.

Zunächst bootet man also die Rettungsdiskette. Dann muß das Dateisystem, welches den Kernel enthält, gemounted werden. Hierfür benötigt man einen *Mount Point*, meist wird dafür `/mnt` verwendet. Falls dieses Verzeichnis auf der Rettungsdiskette bereits existiert - was sehr wahrscheinlich ist - wird der erste Befehl eine Fehlermeldung verursachen, die aber ignoriert werden kann:

```
# mkdir /mnt
# mount -t ext2 /dev/hda3 /mnt
```

Nun wechselt man in das Verzeichnis mit dem neuen Kernel. Dabei muß im angegebenen Fall berücksichtigt werden, daß das Verzeichnis nicht wie gewohnt unter `/usr` gemounted wurde. Für den Pfadnamen gilt also folgendes:

```
/mnt + /usr/src/linux/arch/i386/boot - /usr = /mnt/src/linux/arch/i386/boot
```

Man legt eine formatierte Diskette (*nicht* die Boot-/Root-Diskette der Rettungsdiskette!) in das erste Laufwerk (DOS A:), überträgt den Kernel auf diese Diskette und konfiguriert ihn für das richtige Root-Dateisystem:

```
# cd /mnt/src/linux/arch/i386/boot
# dd if=zImage of=/dev/fd0
# rdev /dev/fd0 /dev/hda1
```

Nachdem man in das Hauptverzeichnis zurückgewechselt ist und die Festplattenpartition wieder ent-mounted ist, kann das System mit der so erstellten Bootdiskette neu gestartet werden:

```
# cd /
# umount /mnt
# shutdown -r now
```

Nach dem Reboot sollte in jedem Fall die erste Aktion ein Lauf von `lilo` sein!

7.6 »warning: bdflush not running«

Dies ist nur noch für sehr alte Installationen ein Problem, dort aber ein schwerwiegendes. Dieses Programm schreibt in regelmäßigen Abständen die Dateisystem-Buffer auf die Festplatte zurück. Mit dem Release der Kernelversion 1.0 (April 1994) wurde das Programm durch eine neue Version ersetzt. Man muß sich die aktuelle Version von `bdflush` besorgen; man sollte sie von derselben Quelle bekommen, von der auch die Kernel-Quellen stammen. Diese Version installiert sich dann unter dem Namen `update`. Nach einem Reboot sollten die Fehlermeldungen verschwinden.

7.7 Mein IDE-/ATAPI-CD-ROM-Laufwerk wird nicht erkannt

Obwohl mit der Einführung des ATAPI-Standards der Anschluß von CD-ROM-Laufwerken stark vereinheitlicht wurde, treten noch recht häufig Probleme auf, da immer noch einige Dinge beachtet werden müssen.

Ist das CD-ROM-Laufwerk das einzige Gerät am jeweiligen IDE-Interface, so muß es als `master` oder `single` konfiguriert sein. Dies ist ein sehr oft vorkommender Fehler.

Viele Hersteller von Soundkarten (z.B. Creative Labs) haben heutzutage eine IDE-Schnittstelle auf der Karte integriert. Dieses kann unter Umständen zu Problemen führen. Denn auf einigen Mainboards sind bereits zwei IDE-Schnittstellen vorhanden (die zweite meist auf IRQ 15), so daß diejenige auf der Soundkarte als dritte IDE-Schnittstelle konfiguriert wird (oft über IRQ 11). Die 1.2.x Versionen des Linux-Kernels unterstützen jedoch nur maximal zwei IDE-Schnittstellen. Wer noch diesen alten Kernel verwendet, hat mehrere Möglichkeiten zur Auswahl:

Nur in den seltensten Fällen sind an beiden IDE-Schnittstellen auf dem Mainboard bereits zwei Geräte angeschlossen. In diesem Fall kann man das ATAPI-CD-ROM dort anschließen und die Schnittstelle auf der Soundkarte ausschalten. Dies hat außerdem den positiven Nebeneffekt, das ein IRQ frei wird.

Wer sowieso nur eine IDE-Schnittstelle auf dem Mainboard besitzt, kann die auf der Soundkarte als Nummer zwei (IRQ 15) umkonfigurieren; sie sollte dann von Linux korrekt erkannt werden.

Die Kernel der neuen 2.0 Generation (genauer bereits seit der frühen 1.3.x Phase) haben solche Probleme nicht mehr. Ihr neuer IDE-Treiber unterstützt bis zu vier IDE-Schnittstellen. Wer also wirklich drei IDE-Schnittstellen verwenden will oder muß, sollte auf diese Version umsteigen, was sowieso eine gute Idee ist.

7.8 »obsolete routing requests«

Wer nach einem Kernel-Upgrade derartige seltsame Meldungen bekommt, wenn er sein Netzwerk konfiguriert, hat eine zu alte Version des `route` Programmes und einiger damit zusammenhängender Routinen. Die Include-Datei `/usr/include/linux/route.h` hat sich in neueren Versionen des Kernels verändert. Man sollte eine neuere Version dieser Programme installieren. `route` ist Bestandteil des `NetKit-A` Paketes, das man ebenfalls von derselben Quelle wie den Kernel beziehen kann:

```
ftp.funet.fi:/pub/OS/Linux/PEOPLE/Linus/net-source/base
```

7.9 »Not a compressed kernel Image file«

Wer beim Booten diese Meldung bekommt, hat den falschen Kernel installiert. Der richtige Kernel ist *nicht* `vmlinux` in `/usr/src/linux` sondern `arch/i386/boot/zImage`.

7.10 Console-Probleme nach dem Upgrade auf 1.3.x oder später

Die neuen Versionen verwenden eine andere Kennung für die Konsole. In der Datei `/etc/termcap` sollte entweder im `termcap`-Eintrag für `console` das Wort `dumb` durch `linux` ersetzt werden oder aber ein kompletter neuer Eintrag für `linux` erstellt werden. Siehe dazu auch das *Keyboard HOWTO*.

7.11 Seit dem Kernel-Upgrade kann ich nichts mehr kompilieren

Manche Programme benötigen gewisse Informationen und Definitionen aus dem Linux-Kernel. Diese bekommen sie, indem in den *Include-Dateien*, das sind die Dateien mit der Endung `.h`, die entsprechenden Dateien aus den Kernel-Quellen eingebunden werden:

```
#include <linux/xyzzy.h>
```

Die Datei `xyzzy.h` wird dann in dem Verzeichnis `/usr/include/linux` gesucht. Dieses Verzeichnis ist aber nur ein symbolischer Link auf das `include`-Verzeichnis der Kernel-Quellen; normalerweise lautet der Name des Verzeichnisses `/usr/src/linux/include/linux`. Es gibt mehrere Möglichkeiten, warum der Compiler die gesuchten Dateien nicht findet:

1. Der Link existiert nicht. Dies läßt sich einfach beheben:

   ```
   # ln -s /usr/src/linux/include/linux /usr/include/linux
   ```

2. Der Link zeigt an eine falsche Stelle. Dies kann geschehen, wenn der Kernel-Verzeichnisbaum nicht an der Standard-Stelle befindet. In diesem Fall muß er entsprechend »umgebogen« werden, also z.B. für den Fall, daß der Kernel im Verzeichnis `/opt/Sources` ausgepackt wurde:

   ```
   # cd /usr/include
   # rm -f linux
   # ln -s /opt/Sources/linux/include/linux linux
   ```

3. Die Kernel-Quellen sind nicht installiert. Oft wird aus Platzmangel der gesamte Kernel-Verzeichnisbaum gelöscht. Hier hilft es nur, die `include`-Dateien wieder zu installieren:

   ```
   # tar zxvpf linux.x.y.z.tar.gz linux/include
   ```

4. Die Rechte der Dateien sind falsch gesetzt. Wer z.B. als `root` eine `umask` verwendet, die anderen Benutzern den Lesezugriff auf Dateien verwehrt, muß beim Auspacken des Kernels unbedingt die Option `p` für `tar` verwenden oder zumindest für das `include`-Verzeichnis den lesenden Zugriff für alle ermöglichen, da sonst nur `root` den Compiler benutzen kann. Dieses geschieht nachträglich mit dem Befehl:

   ```
   # cd /usr/src/linux
   # chmod -R go+r include
   ```

7.12 Erhöhung von Systembeschränkungen

Die folgenden *Beispiele* sind vielleicht ein Hinweis für all diejenigen, die sich fragen, wie man die diversen veränderbaren Schrankenwerte (»Soft Limits«) im Kernel verändern kann:

```
# echo 4096 > /proc/sys/kernel/file-max
# echo 12288 > /proc/sys/kernel/inode-max
# echo 300 400 500 > /proc/sys/vm/freepages
```

8 Hinweise zum Upgrade auf Version 2.0

Mit der Version 2.0 des Linux-Kernels wurden eine ganze Menge an Neuerungen und Veränderungen eingeführt. Die Datei `Documentation/Changes` im Verzeichnisbaum der 2.0.x Kernel-Quellen enthält alle Informationen, die man bei einem Upgrade auf diese Version benötigt. Insbesondere kann es notwendig sein, diverse andere Systemkomponenten auf den neuesten Stand zu bringen, so etwa `gcc`, libc oder SysVInit. Manche Systemdateien müssen eventuell ebenfalls editiert und dem neuen Kernel angepaßt werden. Aber keine Panik, das ist einfacher, als es klingt.

9 Module

Ladbare Kernel-Module können Hauptspeicher sparen und vereinfachen meist die Konfiguration eines Systems. Die neue Red Hat-Distribution hat z.B. nur noch eine einzige Bootdiskette für alle Konfigurationen. Inzwischen können fast alle optionalen Kernel-Teile als Module konfiguriert werden: Dateisysteme, Netzwerk-Karten, serielle und parallele Schnittstellen, Bandlaufwerke, Drucker usw.

9.1 Installation der Hilfsprogramme

Das Hinzufügen und Entfernen der Module zum Kernel wird von einigen externen Programmen erledigt. Diese gehören nicht zur Standard-Kerneldistribution und müssen extra installiert werden. Man bekommt sie von denselben Servern wie auch den Kernel unter dem Namen `modules-x.y.z.tar.gz`. Man sollte dabei dasjenige Paket mit derselben oder, falls es das nicht gibt, mit der nächstkleineren Versionsnummer wie der verwendete Kernel benutzen. Nach dem Auspacken des Paketes mit

```
# tar zxvf modules-x.y.z.tar.gz
```

findet man in dem neu angelegten Verzeichnis `modules-x.y.z` eine `README`-Datei, die man aufmerksam lesen sollte. Darin werden auch Anweisungen zur Installation gegeben; dies beschränkt sich aber eigentlich immer auf ein einfaches `make install`. Dabei sollten die folgenden Programme im Verzeichnis `sbin` installiert werden: `insmod`, `rmmod`, `ksyms`, `lsmod`, `genksyms`, `modprobe` und `depmod`.

Im Unterverzeichnis `insmod` des Modul-Paketes ist auch ein kleines Beispiel für einen ladbaren Treiber enthalten (`/dev/hw`). Wer Lust hat, kann damit ein wenig herumspielen, die Datei `INSTALL` gibt dazu ein paar Hinweise.

`insmod` dient dazu, ein Modul in den laufenden Kernel einzufügen. Normalerweise handelt es sich bei den Modulen um Binärdateien mit der Endung `.o`. Der Beispieltreiber heißt z.B. `drv_hello.o`. Um diesen einzufügen, lautete der Befehl also:

```
# insmod drv_hello.o
```

Um zu sehen, welche Module gerade im Kernel geladen sind, dient der Befehl `lsmod`:

```
# lsmod
Module:         #pages:  Used by:
drv_hello          1
```

`drv_hello` ist der Name des Modules; es belegt eine »Page« (Seite, entspricht 4 kB) im Speicher. Keine weiteren Module des Kernels sind von ihm abhängig.

Um das Modul wieder zu entfernen, wird folgender Befehl verwendet:

```
# rmmod drv_hello
```

Wichtig ist hierbei, daß `rmmod` den *Namen des Modules*, so wie er von `lsmod` angezeigt wird, und *nicht* den Dateinamen als Argument benötigt.

Die Manual Pages der Modul-Programme geben weitere Informationen über deren Zweck und Optionen.

9.2 Die Module der Standard-Kerneldistribution

Zum gegenwärtigen Zeitpunkt (2.0.30) können folgende Bestandteile des Kernels als Modul kompiliert werden:

- alle Dateisysteme (außer `/proc`)
- die meisten Treiber für Ethernet sowie ISDN-Karten
- Unterstützung für SCSI-Geräte (Festplatten, CD-ROM, Tape)
- alle unterstützten SCSI-Karten außer AM53C974
- der Sound-Treiber
- alle unterstützten IDE-CD-ROMs
- Disketten, Drucker, Loopback und RAM-Disk
- serielle und parallele Schnittstelle (Drucker), BUS- und PS/2 Mäuse

Um diese zu benutzen, darf man sie nicht in den eigentlichen Kernel einbinden, d.h. während des `make config` darf man die entsprechenden Fragen nicht mit `y` beantworten, sondern mit `m` für Modul. Nachdem der Kernel wie bereits beschrieben kompiliert und installiert wurde, müssen die Module dann extra mit dem Befehl

```
# make modules
```

übersetzt werden. Mit

```
# make install
```

werden die übersetzten Module dann im Verzeichnis `/lib/modules/x.y.z` installiert, wobei `x.y.z` die verwendete Kernel-Version ist. Von dort können sie dann mit dem Befehl `lsmod` oder `modprobe` geladen werden. Der Vorteil von `modprobe` gegenüber `lsmod` ist dabei, daß der volle Pfad des Modules nicht angegeben werden muß. `modprobe` sucht automatisch in `/lib/modules/x.y.z`. Außerdem wird immer die richtige Version des Moduls, passend zum gerade laufenden Kernel, gelesen. Wer gerne mit verschiedenen Kernels experimentiert, wird das schnell zu schätzen wissen.

Ladbare Module sind ganz besonders für nur selten benutzte Dinge praktisch, ein gutes Beispiel sind Dateisysteme. Wer nur ab und zu mal eine Diskette mit dem MSDOS FAT-Dateisystem mounten muß, kann das entsprechende Modul (`msdos.o`) nur bei Bedarf laden und so unter Normalbedingungen etwa 50 kB an RAM einsparen. Noch komfortabler wird das Ganze, wenn man `kerneld` verwendet. Dies ist ein automatischer Modul-Lader, der benötigte Module selbsttätig in den Kernel lädt, sobald das entsprechende Gerät oder Protokoll benötigt wird, und es, wenn es nicht mehr in Benutzung ist, auch wieder entfernt.

Diese und viele weitere Feinheiten im Umgang mit den Modulen werden im *Module-HOWTO* beschrieben.

10 Andere Konfigurationsoptionen

In diesem Abschnitt werden einige weitere Optionen der Kernel-Konfiguration mit `make config` näher erläutert.

10.1 General setup

Normal floppy disk support

Modulfähig (spart auch 50 kB). Besitzer eines IBM Thinkpad sollten aber unbedingt `drivers/block/README.fd` lesen. Auch anderen kann das nicht schaden.

XT harddisk support

Hat noch jemand einen alten, verstaubten 8 Bit XT Festplattencontroller? Dann kann er hier mit `y` antworten und ihn auch unter Linux verwenden.

PCI bios support

Wer ein PCI-Mainboard hat, kann das mal ausprobieren. Aber Vorsicht, einige ältere Mainboards können dadurch abstürzen. Weitere Informationen enthält das *PCI HOWTO*.

Kernel support for ELF binaries

ELF (**E**xecutable **L**ink **F**ormat) ist das bevorzugte Binärformat der neueren Linux-Distributionen; man wird hier also wohl meist `y` antworten.

Kernel support for a.out binaries

Das alte Binärformat. Kann als Modul kompiliert werden. Da noch einige Programme im `a.out` Format im Umlauf sind, ist es eine gute Idee, dieses Format zu unterstützen.

Set version information on all symbols for modules

Bislang mußten Module explizit für jede Kernelversion neu kompiliert werden. Wenn man hier mit `y` antwortet, kann man auch Module von anderen Kernel-Versionen benutzen. In diesem Fall sollte man aber unbedingt die Datei `/usr/src/linux/Documentation/modules.txt` lesen.

Networking options

Siehe dazu das *NET-3 HOWTO*.

11 Tips und Tricks

11.1 Umlenken der Ausgabe von make oder patch

Gerade diese beiden Programme produzieren oft eine Unmenge von Meldungen, die über den Bildschirm huschen, ohne daß man sie lesen kann. Der Trick, diesen Text zusätzlich in eine Datei zu schreiben, wurde weiter oben bereits angewandt, man verwendet dazu das Programm `tee`. Die genaue Syntax hängt aber von der verwendeten Shell ab, deshalb sollte man zunächst feststellen, welche Shell man benutzt. Wer es nicht sowieso weiß, kann das leicht mit dem Befehl `finger` herausfinden:

```
% finger root
Login: root                              Name: root
Directory: /root                         Shell: /bin/bash
```

`finger` wertet übrigens die Datei `/etc/passwd` aus, in der die Login-Shell eingetragen ist; man kann also auch dort nachsehen. Der gesuchte Befehl lautet dann für

bash

⟨Befehl⟩ `2>&1 | tee` ⟨Datei⟩

csh/tcsh

⟨Befehl⟩ `|& tee` ⟨Datei⟩

rc

⟨Befehl⟩ `>[2=1] | tee` ⟨Datei⟩

11.2 Kernel updates

Die Änderungen des Linux-Kernels von Version zu Version werden von Michael Chastain (`mec@treflan.shout.net`) zusammengefaßt und werden von Russell Nelson (`nelson@crynwr.com`) archiviert:

```
http://www.crynwr.com/kchanges
```

.

12 Andere nützliche Dokumente

- *Sound HOWTO*: Soundkarten und Hilfsprogramme.
- *SCSI HOWTO*: Alles über SCSI, Controller und Geräte.
- *NET-3 HOWTO*: Netzwerk.
- *PPP HOWTO*: Netzwerkverbindungen über PPP.
- *PCMCIA HOWTO*: PCMCIA-Karten für Notebooks.
- *ELF HOWTO*: ELF: Was es ist; wie man umsteigen kann.
- *Hardware HOWTO*: Was wird von Linux unterstützt?
- *Module HOWTO*: mehr zu den Kernel-Modulen.
- *Kerneld mini-HOWTO*: über die Verwendung von `kerneld`.
- *BogoMips mini-HOWTO*: Falls Sie sich fragen, was das ist.