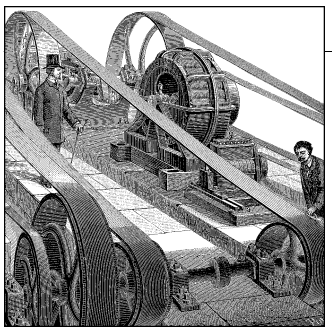

KAPITEL 9

Der Kernel

Der Name *Linux* wird für sehr verschiedene Dinge benutzt. *Linux* wird einerseits im weitesten Sinne für das gesamte Betriebssystems mit allen seinen Anwendungen verwendet, dann wird mit *Linux* in einem engeren Sinne das eigentliche Betriebssystem bezeichnet, zuletzt ist *Linux* im engsten Sinne der Name des eigentlichen Kernels (Kern). Diese Kapitel verwendet *Linux* im engsten Sinne und widmet sich dem Kernel.

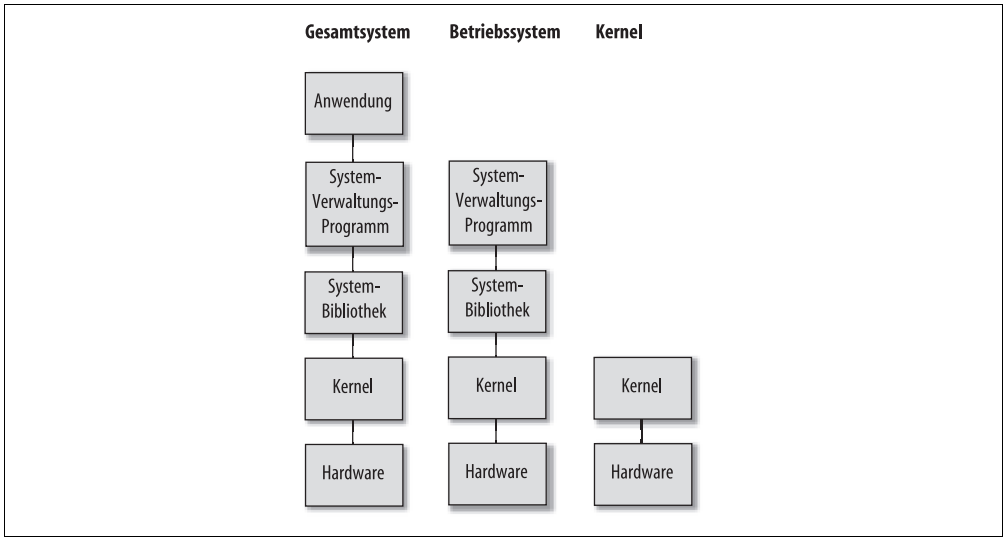

Abbildung 9-1: Gesamtsystem – Betriebssystem – Kernel

Die Rolle des Kernels

Die ersten Programme wurden vor mehr als 50 Jahren mit Kippschaltern an einer Konsole eingegeben. Seitdem wurde die Bedienung von Computern deutlich verbessert und viele Annehmlichkeiten wurden eingeführt.

Linux hat von *Unix* die Idee übernommen, Geräte, Daten und Dateien im Wesentlichen gleich zu behandeln. Dateien sind gewissermaßen eingefrorene Datenströme, Geräte sind spezielle Dateien. Damit müssen keine speziellen Vorgehensweisen für Ein- und Ausgabe definiert werden, alles ist dateiorientiert.

Die Unabhängigkeit der Programme von der Ausführung eines speziellen Computers – insbesondere den Hardware-Eigentümlichkeiten dieser Maschine – wurde systematisch weiterentwickelt. Die Idee war nun nahe liegend, eine überall lauffähige Version eines Programms zu entwickeln, anstatt für jede spezielle Ausführung eines Computers eine spezifische Version dieses Programms bereitstellen zu müssen.

Linux verwendet ein Schichtenmodell. Der Kernel des Betriebssystems verdeckt größtenteils die spezifischen Eigenschaften der Hardware. Er stellt eine Anzahl *System Calls* zur Verfügung, die primitive Funktionen wie das Öffnen und Schließen von Dateien und das Stellen der Rechneruhr erlauben. Einige dieser *System Calls* sind nach wie vor plattformabhängig, aber die Verwendung von *System Calls* erlaubt es, die Auswirkungen der Unterschiede zwischen den verschiedenen Hardware-Plattformen stark einzugrenzen und gut handhabbar zu machen.

Auf dieser Ebene setzen Systembibliotheken, insbesondere die libc und in der GNU-Version die *glibc* auf, die die primitiven Funktionen des Kernels in bequem handhabbare Funktionen abbilden, die dann oft mehrere *System Calls* zusammenfassen. Diejenigen *System Calls*, die auf einer Hardware-Plattform nicht zur Verfügung stehen, werden auf der Ebene der *glibc* durch Aufruf anderer *System Calls* simuliert. Auf der Ebene der *glibc* sind dann Programme zwischen höchst unterschiedlichen *Linux*-Computern vollständig übertragbar. Dies heißt nicht, dass Programme zwischen unterschiedlichen Hardwareplattformen einfach hin- und herkopiert werden können.

Der Quelltext eines Programms, dessen Binary auf dem einen *Linux*-Computer läuft und auf einem anderen laufen soll, kann auf dem anderen *Linux*-Computer durch einen einfachen Compilerlauf in eine direkt ausführbare Binärdatei übersetzt werden. Dieses ist dann ohne Einschränkungen lauffähig, sofern die Regeln eingehalten wurden. Bei der meistverbreiteten Hardwareplattform, die auf *Intel-* oder *AMD 80X86*-Prozessoren aufbaut, ist jedoch die Übereinstimmung so groß, dass sogar ein einfaches Hin- und Herkopieren der Binärdatei eines Programms dann möglich ist, wenn die gleiche oder eine hinreichend ähnliche Version der *glibc* eingesetzt wird. Ähnliches gilt auch innerhalb anderer Hardwarearchitekturen beziehungsweise Prozessorfamilien, insbesondere auf den Mainframeversionen von *Linux*.

Der Aufbau des Kernels

Der Kernel ist ebenfalls – zu einem großen Teil – in einer Schichtenstruktur aufgebaut. Um die vollständige Umsetzung des Aufbaus in Schichten für den Kernel sind mehrere verbissene Auseinandersetzungen geführt worden, legendär sind die zwischen *Linus*

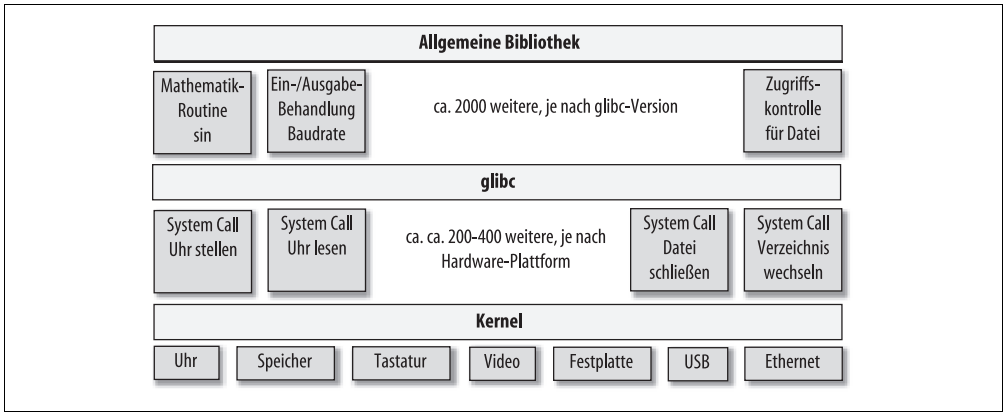

Abbildung 9-2: Rolle des Kernels

Thorvalds und *Andrew Tanenbaum* ausgetauschten E-Mails[1]. Eine offizielle Umsetzung des Micro-Kernel-Ansatzes gibt es noch nicht, aber an der TU Dresden[2] ist eine inoffizielle Umsetzung erarbeitet worden.

Ein strikter Schichtaufbau (Micro-Kernel) verbessert gegenüber einem monolithischen Kernel die Wartbarkeit (es können fast alle Teile einfach – sogar zur Laufzeit – ausgetauscht werden) und vermindert die Wahrscheinlichkeit des Auftretens von Fehlern, begrenzt die Auswirkungen von Fehlern und erlaubt den dynamamischen Austausch von Komponenten.

Ein monolithischer Aufbau verbessert gegenüber einem Mikro-Kernel die Leistung (heutzutage liegt der Unterschied zwischen monolithischem Kerneln und Mikro-Kerneln unter 5%). Untersuchungen über Leistungseinbußen nahm der leider viel zu früh verstorben *Jochen Liedtke* vor[3]. Rein monolithische Kernel sind in der Systemwartung sehr bequem, da nur eine einzige Binärdatei behandelt werden muss.

Die der Außenwelt am nahesten liegenden Bestandteile des Kernels sind die Gerätetreiber. Sie sind vollständig modularisiert und können, müssen aber nicht als Module ausgeführt sein. Einige Gerätetreiber – wie das SCSI-Subsystem – sind ihrerseits in Schichten aufgebaut, man spricht dann – zum Beispiel – vom SCSI-Subsystem.

Die darüber liegende Geräteverwaltung und der Scheduler sind beim Standardkernel monolithisch.

Für die Netzwerkprotokolle ist ein modularer Aufbau üblich.

1 *http://www.educ.umu.se/~bjorn/mhonarc-files/obsolete/*, *http://people.fluidsignal.com/~luferbu/misc/Linus_vs_Tanenbaum.html*

2 *http://os.inf.tu-dresden.de/L4/LinuxOnL4/*

3 *http://os.inf.tu-dresden.de/pubs/sosp97/*

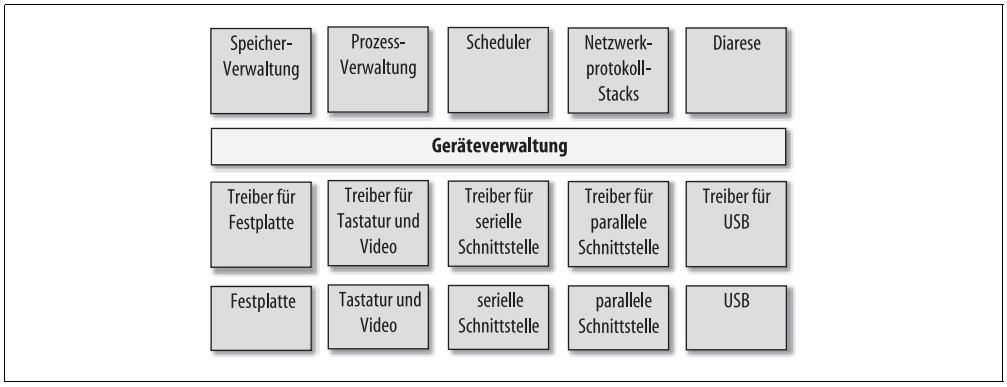

Abbildung 9-3: Aufbau des Kernels

Vorbereitungen zur Erneuerung des Kernels

Die erste und wichtigste Frage, die zu beantworten ist lautet: *Ist die Erneuerung des Kernels wirklich notwendig?* Wenn diese Antwort nicht ein klares *Ja* ist, sollte der vorhandene Kernel behalten werden.

Der Befehl `uname -a` zeigt Ihnen an, mit welcher Version des Kernels Sie arbeiten. Die Ausgabe sollte etwa so aussehen:

```
tigger% uname -a
Linux tigger 2.4.19-64GB-SMP #2 SMP Fri Aug 9 12:44:16 CEST 2002 i686
```

In diesem Fall handelt es sich um einen Rechner mit der Version 2.4.19 des Kernels (konfiguriert für einen Rechner mit mehr als einem Prozessor [SMP] und maximal 64 GByte RAM), der zuletzt am 9. August 2002 kompiliert wurde. Es werden noch weitere Informationen angezeigt, zum Beispiel der Name des Rechners, wie oft dieser Kernel kompiliert wurde (zweimal) sowie die Angabe, dass es sich um einen Pentium Pro- oder einen äquivalenten Prozessor handelt (wie an `i686` zu sehen ist). In der Manpage zu `uname`*(1)* erfahren Sie mehr hierzu.

Notwendig ist die Erneuerung eines Kernels dann, wenn

- Sicherheitsrisiken[4] bekannt geworden sind, die mit dem neuen Kernel behoben werden können,
- die vorhandene Hardware[5] nur von dem neuen Kernel unterstützt wird,
- der neue Kernel eine unverzichtbare Leistungsverbesserung[6] erlaubt.

Ein so weit gehender Eingriff in das System wie der Austausch des Kernels ist immer mit einem Risiko behaftet. Der Vorgang ist komplex, ein einziger Fehler kann fatal sein und

4 *http://www.kernel-security.org*

5 *www.kernel.org*

6 *www.kernel.org*

zum Totalverlust der Daten führen. Daher ist es sehr wichtig, vor einer Erneuerung des Kernels eine Sicherung aller persönlichen Daten vorzunehmen.

Danach wird die vollständige Konfiguration des Systems dokumentiert. Hierzu gibt es in der Distribution spezifische Möglichkeiten, beispielsweise in der *SuSE*-Version das Paket `hwinfo`. Die Ergebnisse werden ausgedruckt.

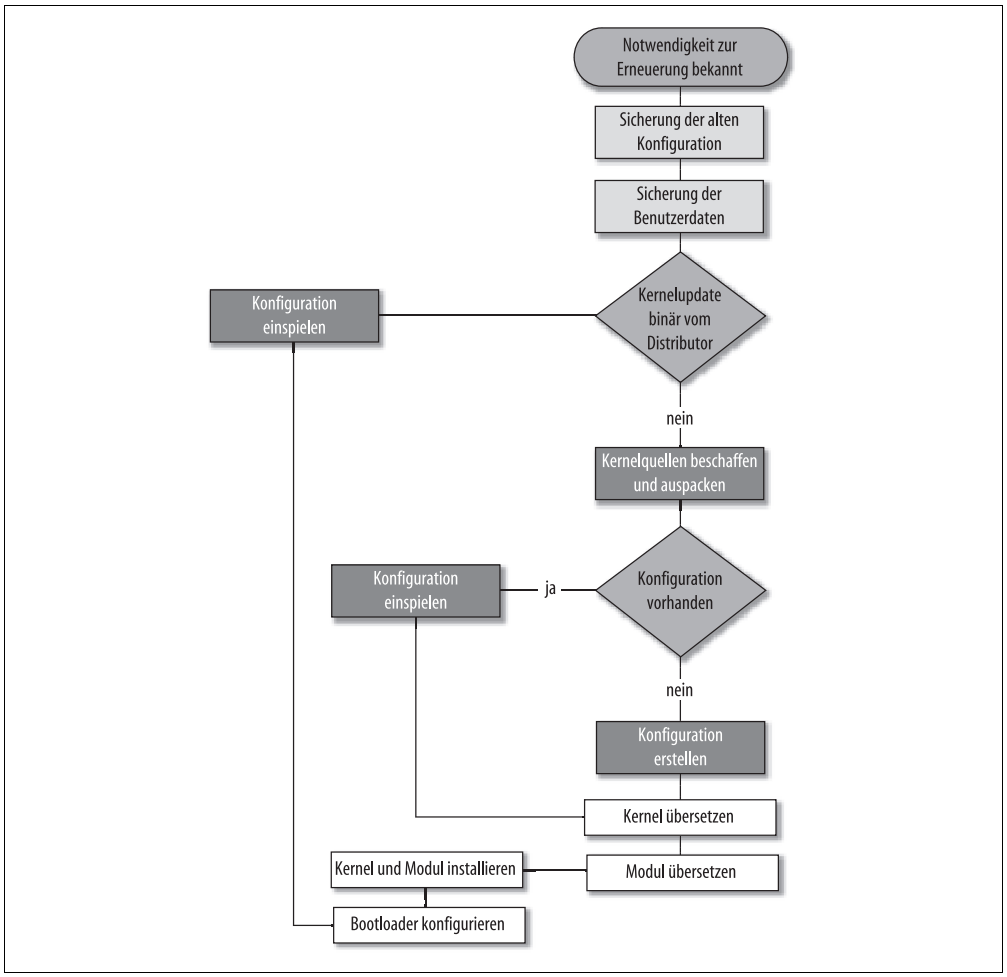

Abbildung 9-4: Vorgehen zur Erneuerung des Kernels

Auslesen der Konfiguration eines aktiven Kernels

Im Idealfall stellen die aktuellen Versionen des Kernels im */proc*-Verzeichnis eine Pseudo-Datei bereit, die die aktuellen Konfigurationsdaten enthält. Diese Daten sind komprimiert. Diese komprimierten Daten werden wie folgt in eine Datei geschrieben:

```
zcat /proc/config.gz > ./root/config.aktuell
```

Wenn dies nicht möglich ist, müssen die Dokumentation des Distributors oder die bisher genutzten Quellen des Kernels herangezogen werden. Über die geladenen Module gibt der Befehl `lsmod` Auskunft. Häufig stellt der Distributor als Dokumentation vorkonfigurierte Quelltexte bereit, die dann so wie *bisher genutzten Quellen* behandelt werden. Der Kernel wird immer im Verzeichnis */usr/src* installiert und heißt *linux*. Meistens ist */usr/src/linux* nur ein Link auf das tatsächliche Verzeichnis der Quellen namens */usr/src/linux-X.X.X* mit den tatsächlich benutzten Versionsnummern. Im Verzeichnis */usr/src/linux* liegt eine Datei, die je nach Einstellung des Programms ls nicht angezeigt wird, also wird

```
ls -a
```

aufgerufen. Die Datei heißt */usr/src/linux/.config* . Diese Datei wird gesichert

```
cp /usr/src/linux/.config /root/config.aktuell
```

Beschaffen des Quelltextes eines Kernels

Wenn eine Sicherheitswarnung der Grund für die Aktualisierung des Kernels ist, ist im Allgemeinen auch eine besondere Bezugsquelle für den verbesserten Kernel angegeben.

Ansonsten wird der Kernel von einem der bekannten vertrauenswürdigen Server oder einem seiner Spiegel (Mirror-Server) heruntergeladen. Die Referenz für Quelltexte von den originalen Kernelquellen ist der Server des Kernelprojektes[7]. Von dort aus können sowohl die Quellen direkt heruntergeladen werden, oder es kann, was dringend zu empfehlen ist, ein passender Mirror-Server gesucht werden.

Die aktuellen Versionen der verschiedenen Entwicklungslinien des Kernels werden auf der Startseite angeboten. Die einfachste Methode zum Herunterladen des Quelltextarchives für einen Kernel ist die Verwendung eines Browsers. Durch direktes Anwählen des Quellarchives *mittels Mausklick* oder über die *Rechte-Maustaste*-Methode wird das Archiv auf die Platte übertragen. Der Standard für das Ziel ist */usr/src*.

Beschaffen eines vorkonfigurierten Kernels oder Quelltextes

Die kommerziellen, aber auch einige der freien Distributoren bieten angepasste Kernels an, die für ihre jeweilige Distribution optimiert sind. Ebenso ist es eine besondere Dienstleistung von Distributoren, Sicherheits- und Hardware-Patches (*Patch* = Flicken, begrenzte Änderung) einzupflegen, die noch nicht in die offiziellen Kernelquellen integriert sind. Zudem sind diese Quellen-Archive mit für die Distribution passenden Vorab-Konfigurationen versehen. Die Distributoren haben einen speziellen Bereich auf ihren Servern, von dem die Quellen-Archive heruntergeladen werden können. Manche Distributoren beschränken den Zugang zu bestimmten Teilen ihrer Server auf zahlende Kunden.

7 *http://www.kernel.org/*

Je nach Wunsch oder Politik des Distributors stehen entweder nur die Kernelquellen zur Verfügung oder es werden parallel Quellen und die passend übersetzten Binärdateien zur Verfügung gestellt. Das Risiko von Fehlern ist bei der Nutzung der Binärdateien deutlich geringer als bei der Erzeugung der Binärdateien durch die eigene Übersetzung der Quellen. Insbesondere wenn die Binärdateien für einen Installationsmanager wie *RPM* oder *DPKG* bereitgestellt werden, sind die Risiken stark verringert.

Bereitstellen einer Ausweichlösung

Das Austauschen des Kernels stellt einen schwer wiegenden Eingriff in ein *Linux*-System dar, der immer die Möglichkeit in sich trägt, dass sich das *Linux*-System nach dem Austausch des Kernels nicht wie gewünscht verhält. Um die Auswirkungen von Funktionseinschränkungen möglichst klein zu halten, wird die bislang erfolgreich arbeitende Version gesichert. Zu sichern sind

- die selbst erstellten Daten und Dateien in Form eines klassischen Backups
- die vorhandene Kernelkonfiguration

Das Vorgehen bei einem Backup wird an anderer Stelle behandelt. Dieses Backup ist eine reine Sicherheitsmaßnahme für den Fall, dass alle Rettungsversuche fehlschlagen. Das tritt zwar äußerst selten auf, aber leider kommen gerade in den besonders kritischen Momenten auch Hardwarefehler, Stromausfälle und sonstig völlig Unerwartetes vor. Der funktionierende Kernel wird dadurch gesichert, dass er zusätzlich unter einem anderen als dem bisher verwendeten Namen gespeichert wird. Das früher übliche Speichern des alten Kernels auf einer bootfähigen Diskette ist leider wegen der gewachsenen Größe des Kernels oft nicht mehr möglich. Die folgende Anleitung

```
cp –a /boot/vmlinuz /boot/vmlinuz.running
cp –a /boot/initrd.img /boot/initrd.running
```

geht von dem Ort für Kernel gemäß dem *FHS*-Standard vor. Danach muss der neue Eintrag in den Bootmanager eingetragen werden. Für den Bootmanager *Lilo* sieht die modifizierte */etc/lilo.conf* ohne Berücksichtigung etwaiger Zusatzeinträge so aus:

```
label = linux
     image = /boot/vmlinux
     initrd = /boot/initrd.img
label = running
     image = /boot/vmlinux.running
     initrd = /boot/initrd.running
```

Danach ist der Bootmanger *Lilo* aufzurufen:

```
lilo -v
```

Der Erfolg wird mit der Meldung *added* für jeden Eintrag bestätigt.

Im Falle des Bootmanager *Grub* wird der Eintrag in der Datei */boot/grub/menu.lst* so vorgenommen:

```
# For booting GNU/Linux
title  GNU/Linux-running
root (hd0,0)
```

```
kernel /boot/vmlinuz.running root=/dev/hda1
initrd /boot/initrd.img

# For booting GNU/Linux
title  GNU/Linux
root (hd0,0)
kernel /boot/vmlinuz root=/dev/hda1
initrd /boot/initrd.running
```

Der Bootmanager G*rub* benötigt keinen zusätzlichen Aufruf eines Programms.

Bereitstellen des Kernels

Die heruntergeladenen Kernelquellen werden im Verzeichnis */usr/src* entpackt. Dazu wird in dieses Verzeichnis gewechselt, */usr/src* wird damit zum Arbeitsverzeichnis:

```
cd /usr/src
```

Danach muss geprüft werden, ob schon Quelltexte oder Links bereitliegen, die mit den vorhandenen in Konflikt geraten können

```
ls -F linux*
```

Wenn beispielsweise ein älterer Kernel vorliegt, gibt es so eine Ausgabe:

```
linux@
linux-2.6.21/
```

Die vorhandenen Kernelquellen haben im vorliegenden Fall eine andere Versionsnummer als die zu installierenden Quellen, stören also nicht. Es muss aber der Link gelöscht werden

```
rm linux
```

Wenn eine Anzeige wie diese

```
linux/
```

erfolgt, ein echtes Verzeichnis anstelle eines Links vorhanden ist, muss das Verzeichnis umbenannt werden:

```
mv linux linux.old
```

Nochmalige Kontrolle mit `ls -F` *linux** bestätigt die Vollständigkeit der Vorbereitungen.

Wenn das Archiv mit dem Programm `bzip2` komprimiert wurde, ist der Aufruf zum Entpacken

```
tar xvjf linux-2-6-22.tar.bz2
```

Wenn das Archiv nicht im Verzeichnis */usr/src* liegt, muss das vollständige Verzeichnis für das Archiv angegeben werden, wenn also beispielsweise das Archiv im Verzeichnis */downloads/kernel* liegt:

```
tar xvjf /downloads/kernel/linux-2-6-22.tar.bz2
```

Das Kommando tar muss aber nach wie vor im Verzeichnis */usr/src* ausgeführt werden. Wenn das Programm gzip verwendet werden muss, lautet der Aufruf

```
tar xvzf linux-2-6.22.tar.gz
```

Danach wird mit `ls -F linux*` geprüft, unter welchem Namen der Kernel entpackt wurde. Wenn der Name nicht *linux/* ist, wird ein symbolischer Link erzeugt:

```
ln -s linux-2.6.22 linux
```

Nochmalige Prüfung mit `ls -F linux*` bestätigt die Vollständigkeit aller notwendigen Vorbereitungen.

Verwendung einer vorhandenen Konfigurationsdatei

Wenn eine Datei *.config* im Verzeichnis *linux* existiert (*/usr/src* ist immer noch das Arbeitsverzeichnis!),

```
ls -a linux
```

so wird diese umbenannt, um gegebenenfalls noch zur Verfügung zu stellen:

```
mv linux/.config linux/.config.shipped
```

Dann wird die vorhandene Konfigurationsdatei, oder eine vom Provider gelieferte, gleichwertige Datei verwendet. Dazu wird diese in das Verzeichnis */usr/src/linux* kopiert:

```
cp /root/config linux/.config
```

Wichtig ist der Punkt am Anfang von *.config*! Er darf auf keinen Fall fortgelassen werden, damit diese als Standard-Konfigurationsdatei akzeptiert wird. Damit sind die Vorbereitungen abgeschlossen.

Erzeugen einer Konfigurationsdatei von Grund auf

Wenn eine Datei *.config* im Verzeichnis linux existiert (*/usr/src* ist immer noch das Arbeitsverzeichnis!), so wird diese kopiert,

```
ls -a linux
```

um später immer noch zur Verfügung zu stellen:

```
cp -a linux/.config linux/.config.shipped
```

Es wird in das Verzeichnis der Kernelquellen gewechselt:

```
cd linux
```

Die passende Konfigurationsmethode wird aufgerufen

```
make xconfig
```

wenn das X-Window-System richtig funktioniert,

```
make menuconfig
```

wenn die Skript-Sprache perl und *Curses*-Terminal-Steuerung zur Verfügung stehen oder

```
make config
```

wenn nur ein Minimalsystem zur Verfügung steht. Diese letzte Methode ist nur ein absoluter Notbehelf, aber wichtig als letzte Ressource, wenn andere scheitern.

Bei der Verwendung von `make xconfig` können die Optionen mit der Maus angewählt werden, bei `make menuconfig` kann mit Tabulator und Tastenkombinationen wie `ALT-K`, `ALT-S` gearbeitet werden, im Falle von `make config` muss mit den Antworten `y` für *Yes*, `n` für *No*, `m` für *module* und `?` für *Erklärungen* gearbeitet werden. Für einige Fälle müssen Pfade oder Zahlen eingegeben werden, das Ende der Eingabe wird mit der Eingabetaste abgeschlossen. Der Aufruf `make config` erlaubt es nicht, zu schon beantworteten Fragen zurückzugehen. Es muss entweder der gesamte Verlauf abgeschlossen werden, die Änderungen werden behalten, oder es muss nach Abbruch durch `STRG-C` vollständig von neuem begonnen werden. Die beiden anderen Aufrufe `make menuconfig` und `make xconfig` verlangen, dass das Speichern der *.config*-Datei ausdrücklich gewünscht wird. `make config` tut dies ohne Abfrage.

Grundsätzlich sollte zuerst so wenig wie möglich geändert werden, insbesondere sind die Standardvorgaben für Prozessor- und Hardwarearchitektur sehr kritisch. Sie können einerseits die Leistung des *Linux*-Computers wesentlich verbessern, aber auch verhindern, dass der Kernel überhaupt laufen kann. Sinnvollerweise wird schrittchenweise vorgegangen, also zuerst nur das Unumgängliche geändert, weswegen überhaupt ein vollständig neuer Kernel erzeugt wurde. Dann kann schrittweise der Kernel optimiert[8] werden. Hierbei wird immer zumindest der letzte funktionierende Kernel behalten und wie oben gesichert.

Übersetzen des Kernels

Für Kernel der Version 2.4 und Vorgänger müssen jetzt noch die Abhängigkeiten der unterschiedlichen Konfigurationen ausdrücklich aufgelöst werden, dazu wird im Verzeichnis */usr/src/linux*

```
make dep
```

aufgerufen. Die Abhängigkeiten werden behandelt, es werden alle Quelltexte behandelt, das kann etwas dauern. Bei diesem Aufruf darf keine Laufzeitoptimierung mit `make -j` vorgenommen werden! Bei den Versionen des Kernels ab 2.6 kann dann direkt der eigentliche Aufruf zum Bau der Kernels erfolgen:

```
make bzImage
```

Über den Bildschirm laufen jetzt die Meldungen des jeweils behandelten Teils des Quelltextes. Es kann geschehen, dass es zu einem Übersetzungsfehler kommt, dann wird der betreffende Teil, in dem ein Fehler auftrat, mit der im vorigen Abschnitt besprochenen Konfiguration abgeschaltet, bei 2.4-er und älteren Kerneln noch einmal `make dep` aufgerufen und der Aufruf von `make bzImage` wiederholt.

8 Understanding the Linux Kernel, Daniel P. Bovet, Marco Cesati, O'Reilly, Sebastopol, 2002

Nach fehlerlosem Übersetzen des Kernels werden die Module übersetzt,

```
make modules
```

hier können auch Übersetzungsfehler auftreten, dann muss wieder konfiguriert (`make config`), die Abhängigkeiten aufgelöst (`make dep`) und der eigentliche Kernel erzeugt werden (`make bzImage`). Wenn dies fehlerlos erfolgt ist, dann werden die Module an ihren Bestimmungsort verbracht:

```
make modules_install
```

Damit sind die wesentlichen Schritte zur Bereitstellung der Binärdateien abgeschlossen.

Ort und Nutzung der Kerneldatei

Durch den Aufruf `make bzImage` wird als eine Binärdatei namens *bzImage* erzeugt, sie liegt für *Intel-80x86*-Prozessoren im Verzeichnis */usr/src/linux/arch/i386/boot*. Wenn eine andere Hardwarearchitektur als die des *Intel-80X86*-Prozessors verwendet wird, ist anstelle von *i386* die verwendete Architektur einzusetzen, beispielsweise *ia64* für den *Itannium* oder *ppc64* für den *PowerPC*. Diese Datei ist im Wesentlichen komprimiert und damit nicht direkt ausführbar. Am Anfang der Datei steht aber eine Routine, die direkt ausgeführt werden kann und zur Bootzeit den Rest der Datei entpackt und damit den eigentlichen Kernel im Speicher ablegt. Diese selbstentpackende Datei kann, sofern der Kernel keine Module braucht, sondern alle Hardware-Treiber fest eingebaut hat und klein genug ist – das ist leider keineswegs mehr selbstverständlich – auf eine Diskette mit `make bzdisk` geschrieben und dann direkt von Diskette gestartet werden

Installation des Kernels auf Festplatte

Die Datei *bzImage* wird in das Bootverzeichnis */boot* kopiert und bekommt dabei einen anderen Namen

```
cp arch/i386/boot/bzImage /boot/vmlinuz
```

Damit wird – so vorhanden – eine im Verzeichnis */boot* vorhandene Datei namens *vmlinuz* überschrieben. Daher wurde im Anfang dieses Kapitels das Vorgehen zum Sichern der bisherigen Kerneldatei beschrieben. Im Bootmanager existiert ein Eintrag für *vmlinux*. Es muss jetzt noch die Ramdisk erzeugt werden, die für den neuen Kernel benötigt wird. Dies wird beispielsweise durch das Shell-Skript `mk_initrd` bewerkstelligt:

```
mk_initrd -k /boot/vmlinux -i /boot/initrd.img
```

Bei der Verwendung des Bootmanagers *Grub* ist jetzt alles getan, für den Bootmanager *Lilo* müssen jetzt noch die notwendigen Information in den Bootsektor übertragen werden:

```
lilo -v
```

Ort für die Kerneldatei

Ursprünglich lagen die Kerneldatei oder -Dateien im Ursprungsverzeichnis / , *root* genannt. Dies ist jedoch – abhängig vom Bootmanager – fehlerträchtig. Zudem wird das Dateisystem belastet oder auch unübersichtlich. Daher wird in aktuellen Partitionierungen eine eigene Partition gewählt, die unterhalb der technischen Grenze von 1024 Plattenzylindern liegt. Damit ist ein Auswechseln des Bootmanagers jederzeit problemlos möglich.

Einstellungen am Bootprogramm

Die von *Linux* verwendeten Bootmanager starten ohne besondere Maßnahmen den Kernel, der an der ersten Stelle eingetragen ist. Dies sollte immer ein erprobter Kernel sein. Wenn der neue Kernel sich bewährt hat, wird er entweder an die erste Stelle geschoben oder – was einfacher und zuverlässiger ist – es wird ein Eintrag *default* gemacht, der auf den Namen des Kernels mit dem entsprechenden Tabelleneintrag verweißt. Bei *Grub* sieht das so aus:

```
default GNU/Linux-running
```

Der Bootmanager *Grub* benötigt keine weiteren Maßnahmen. Im Falle von *Lilo* wird der Eintrag so vorgenommen:

```
default = linux
```

Danach muss die geänderte Information in den Bootsektor übertragen werden:

```
lilo -v
```

Lilo bestätigt den geänderten Standardeintrag durch ein Sternchen hinter dem Namen bei der Ausgabe der bearbeiteten Einträge.

Test und Betrieb

Der erste Test besteht darin, dass der Kernel das Linux-System erfolgreich booten kann. Wenn der Bootvorgang scheitert, hat das meist eine der folgenden Ursachen:

- Die Unterstützung für den Controller, meist SCSI oder RAID, der die Platte mit dem Ursprungsverzeichnis / *root* ansteuert, fehlt. Abhilfe ist hier eine Neuübersetzung mit Einbindung des Treibers oder auch die korrekte Erzeugung der Ramdisk *initrd*.
- Die Unterstützung für das Dateisystem mit dem die Partition für das Ursprungsverzeichnis / *root* formatiert wurde fehlt. Abhilfe ist hier eine Neuübersetzung mit Einbindung des Treibers oder auch die korrekte Erzeugung der Ramdisk *initrd*.
- Die Hardware wurde zwischenzeitlich geändert, so dass der Bootmanager auf die falsche Platte oder die falsche Partition verweist. Abhilfe ist hier das Anpassen der Einträge im Bootmanager und im Falle von *Lilo* der wiederholte Aufruf von `lilo -v`.

Wenn die schwierigste Hürde überwunden ist, das *Linux*-System ordentlich bootet, müssen sämtliche Funktionen wie Zugriffe über das Netz, Zugriff auf Peripherie wie Scanner, Soundkarten und Grafikfunktionen einmal dem *Linux*-Computer abgefordert werden. Danach kann der *Linux*-Computer laufen – hoffentlich lange Zeit ohne einen weiteres Kernel-Update.