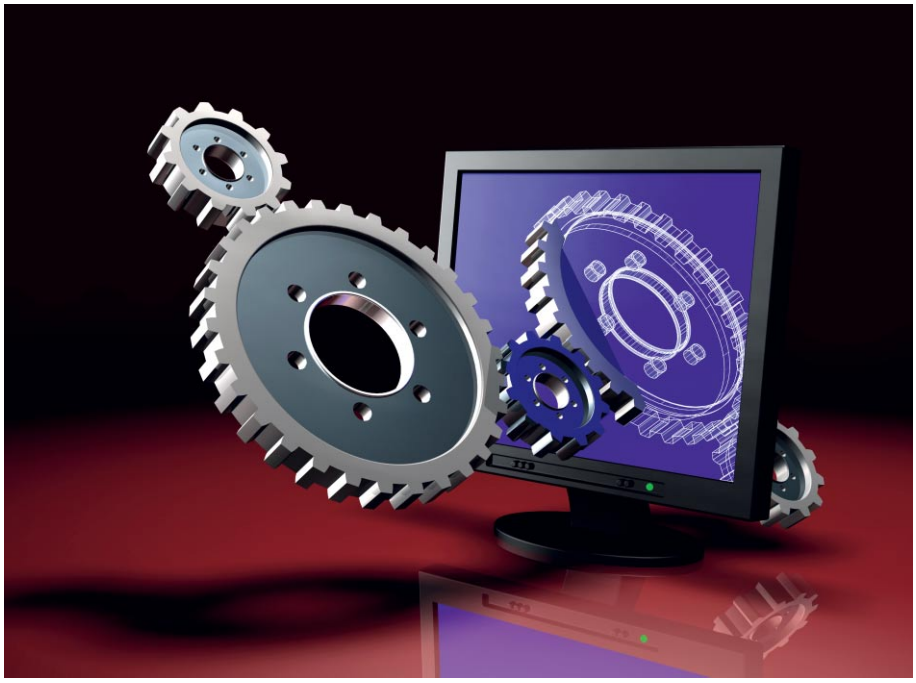
KERNEL TUNEN UND KOMPILIEREN

Kern-Kompetenz

Wer neue Hardware unter Linux nutzen möchte oder die idealen Einstellungen für einen Gameserver sucht, profitiert von einem selbst gebauten Kernel. Wir zeigen die Kniffe, mit denen der neue Kern garantiert bootet.

VON **Mattias Schlenker**



Neben Hardware- und Dateisystem-Treibern enthält der Kernel Komponenten für die Zuteilung von Arbeitsspeicher und Rechenzeit. Wer neue Treiber benötigt, die der alte Kernel nicht mitbringt, oder das Verhalten des Schedulers beeinflussen möchte, kommt deshalb um die Neuinstallation des Kernels nicht herum.

Welchen Kernel nehmen?

Prinzipiell haben Sie die Wahl zwischen den so genannten Vanilla-Kerneln von **www.kernel.org** und einem fast immer gepatchten Kernel Ihres Distributors. Wollen Sie einen einzelnen Treiber per Patch einfügen oder nur eine Feinabstimmung am Scheduler vornehmen, ist es am sinnvollsten, über das Paketmanagementsystem die Quellen des aktuellen Kernels zu installieren. Wird Unterstützung für neuere Hardware benötigt, kann auch ein Kernel auf Basis neuerer Distributionsquellcodes erstellt werden: Unter openSUSE 10.1 konnten wir problemlos den Kernel von openSUSE 10.3 (Alpha) bauen und installieren.

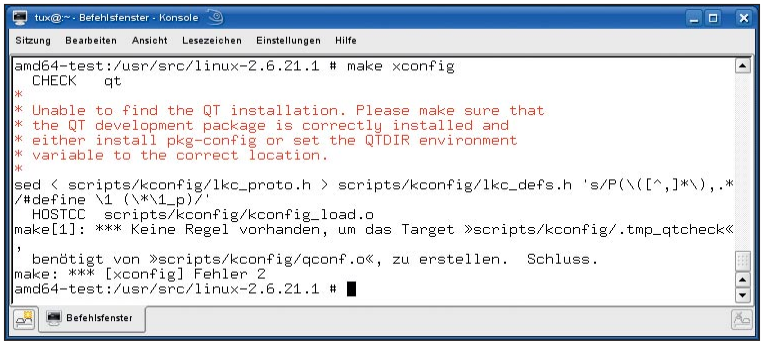
Aktueller als die Distributionskernel, aber ohne die distributionsspezifischen Patches für zumeist besseres Handling von Wechsellaufwerken und Multimedia-Geräten sind die Vanilla-Kernel. Der frischeste stabile Kernel hatte bei Redaktionsschluss die Versionsnummer 2.6.21.1. Wenn keine aktuellere Hardware unterstützt werden soll und bislang ein älterer Kern der 2.6er-Serie zum Einsatz kommt, empfehlen wir den etwas älteren, aber weitergepflegten Kernel der 2.6.16er-Reihe. Bei den neueren Kernel-Versionen wurden nämlich einige Schnittstellen geändert, und die neueren Treiber erfordern manchmal umfangreiche Eingriffe in die Konfiguration. Die Hardware-Unterstützung dieses Kernels ist weitestgehend auf dem Stand von März 2006. Vor allem für Bastler interessant sind Release-Candidates und Entwickler-Kernel: Als dieser Artikel entstand, lockte 2.6.21-git4 mit neuen Features. Release Candidates und Entwickler-Kernel können mitunter Abstürze und Datenverluste verursachen. Wer den Mut besitzt, einen solchen Kernel zu fahren, sollte deshalb wenigstens regelmäßig Backups machen und einen stabilen Kernel als „Boot-Alternative“ für den Notfall bereithalten. Kaum noch eine Rolle spielen die Kernel der 2.4er-Serie, weshalb wir auf deren Konfiguration und Kompilierung nicht näher eingehen wollen. 2.4er Kernel erhalten kaum noch neue Hardware-Treiber. Wer seinen Server auf ein neues Mainboard migriert und für dieses neue Chipsatz-Treiber benötigt, wird deshalb häufig dazu gezwungen, einen 2.6er-Kernel zu verwenden. Neue Kernel auf alten Systemen können jedoch zu gelegentlichen Problemen auf Grund geänderter Schnittstellen führen - schlimmstenfalls läuft das System instabil. Innerhalb der 2.6er-Serie erzwingen gelegentlich kleinere Änderungen am Kernel Updates von System-naher Software wie den *Wireless Tools* oder *udev*.

Werkzeugsammlung

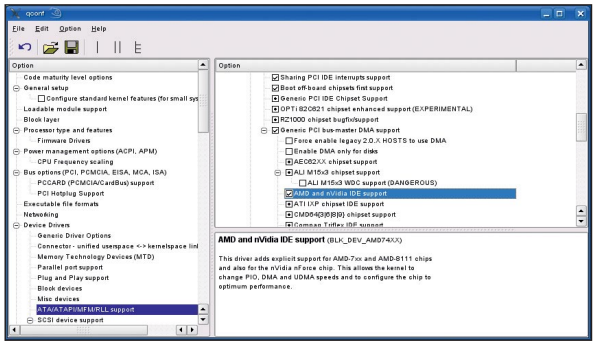
Um den Kernel zu kompilieren, braucht man zunächst einen Compiler. Der alleine reicht

Vorsicht, Falle!

Wer einen eigenen Kernel installiert oder den Distributions-Kernel stark modifiziert, steht oft nach dem Reboot vor einem schwarzen Bildschirm: Die proprietären binären Grafiktreiber-Module des alten Kernels lassen sich nicht in den neuen laden. Laden Sie deshalb bereits vor dem Neustart die ausführbaren Archivdateien von ATI oder nVidia herunter. Sie können dann nach dem Neustart auf der Textkonsole den Grafiktreiber passend zum neuen Kernel kompilieren. Gleiches gilt für proprietäre WLAN-Treiber, Treiber für einige ISDN- und DSL-Karten sowie die von VMware benötigten Kernel-Module.



Der Aufruf von *make xconfig* schlägt fehl, wenn die Entwicklerversion der Qt-Bibliotheken fehlt.



Mit make *xconfig* können alle Kernel-Optionen in einer übersichtlichen Baumstruktur angepasst werden.

nicht. Eine funktionierende „Toolchain" beinhaltet auch noch die *binutils*, mit denen einzelne Binärdateien zum funktionierenden Ganzen zusammengefügt werden. Daneben ist *make* notwendig. Debian-basierte Distributionen bringen alle diese Tools im Meta-Paket *build-essential* mit. Unter openSUSE müssen Sie via *YaST* einzelne Pakete auswählen. Für das komfortable grafische Konfigurations-Frontend sind zudem die Entwicklerversionen der Qt-Bibliothek (*qt3-dev* oder *-devel*) sowie der C++-Compiler *g++* notwendig. Für das einfachere textbasierte Frontend genügen *libncurses5-dev* und der normale C-Compiler. Beachten Sie, dass die Bibliotheken für die grafische Konfiguration einen Rattenschwanz an Abhängigkeiten nach sich ziehen. Auf Servern, die nur die notwendigste Software erhalten sollen, oder auf langsam ans Internet angebundenen Rechnern sollten Sie deshalb nur das textbasierte Konfigurations-Frontend verwenden.

Konfigurationsrecycling

Der schnellste Weg zum eigenen Kernel führt über die Konfiguration des aktuell laufenden Kernels. Sie finden diese entweder als Textdatei *config-version* im Verzeichnis */boot* (der Name entspricht dabei der Ausgabe von uname -r) oder als gepackte Pseudodatei */proc/config.gz* im Prozessdateisystem. Kopieren Sie die Konfigurationdatei nach *.config* in den frisch entpackten Kernel-Source-Tree:

```
gunzip -c /proc/config.gz > \
/usr/src/linux-2.6.20.4/.config
```

beziehungsweise:

```
cp /boot/config-2.6.18.2-020 \
/usr/src/linux-2.6.20.4/.config
```

Nun muss diese Konfiguration in eine zum neuen Kernel passende überführt werden. Das erledigen Sie mit dem Befehl

```
make oldconfig
```

Gestellte Fragen können Sie mit dem vorgeschlagenen Default-Wert beantworten - einfach *Enter* drücken. Der so konfigurierte Kernel entspricht weitestgehend dem alten, verzichtet also auf viele neue Treiber und Features.

Flickwerk: Patches anbringen

Viele Kernel-Erweiterungen werden in Form von Patches ausgeliefert, die in der Regel innerhalb des Kernel-Trees angebracht werden müssen. Auch die Linux-Kernel-Entwickler verwenden Patches, damit bei kleinen Modifikationen nicht der ganze 40 MByte große Tarball heruntergeladen werden muss. Kernel-Patches werden gegen die letzte große Version erstellt. Um den Patch *2.6.20.4* anzubringen, benötigen Sie die Kernel-Sourcen von *2.6.20*. Entpacken Sie zunächst die Sourcen und wechseln Sie dann in das entstandene Verzeichnis:

```
tar xvjf linux-2.6.20.tar.bz2
cd linux-2.6.20
```

Hier wird nun der Patch angebracht. Handelt es sich um einen komprimierten Patch, muss dieser entpackt werden. Das erledigt die folgende Pipeline:

```
bunzip2 -c ../patch-2.6.20.4.bz2 |
➥patch -N -p1
```

Ungepackte Patches können direkt angebracht werden:

```
patch -N -p1 < ../patch-2.6.20.4
```

Benennen Sie anschließend den Source-Tree um:

```
cd ..
mv linux-2.6.20 linux-2.6.20.4
```

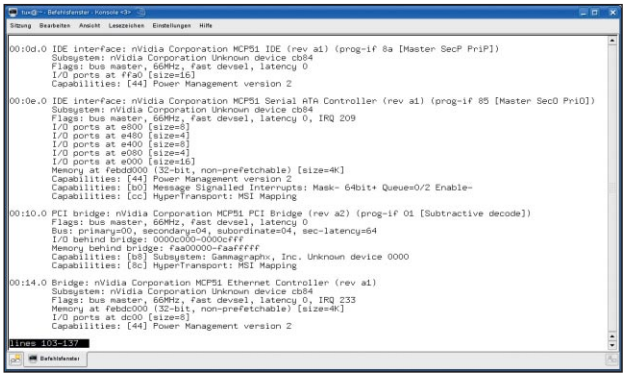
Auch viele Erweiterungen werden als Patch ausgeliefert. Wie eng diese an einen bestimmten Kernel gebunden sind, ist stark vom Typ der Erweiterung abhängig: Sicherheitspatches greifen häufig an vielen Stellen tief in die Quellen ein und erfordern deshalb die Anwendung auf einem im Detail spezifizierten Kernel, während zusätzliche Dateisysteme oft auf eine ganze Serie angewendet werden können. Ein schönes Beispiel ist das komprimierte Dateisystem *squashfs*, das von vielen Live-Distributionen verwendet wird, aber auch prima dazu benutzt werden kann, den Platz auf CDs und DVDs besser auszunutzen.
Statt einer ISO-Datei erstellt man eine so genannte Squashfs-Datei und brennt diese auf den optischen Datenträger. Um diesen mounten zu können, benötigt Ihr Kernel *squashfs*-Unterstützung, die als Patch im *squashfs*-Tarball beiliegt:

```
patch -N -p1 < \
../squashfs3.2-r2/kernel-patches/
➥linux-2.6.20/squashfs3.2-patch
```

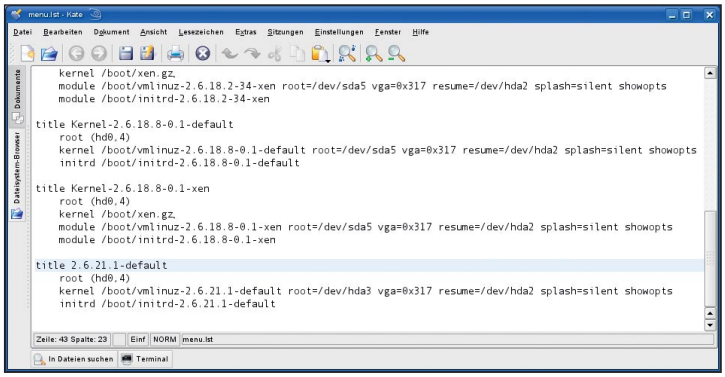
Wenn Sie anschließend

```
make menuconfig
```

aufrufen, finden Sie unter *Filesystems* den neuen Eintrag für *SquashFS*. Wie die anderen Dateisysteme kann die Unterstützung als Modul gebaut oder statisch integriert werden.



Der Aufruf von *lspci -v* gibt Auskunft über verwendete Chipsätze und hilft so bei der Auswahl benötigter Treiber.



Der Aufruf von *lspci -v* gibt Auskunft über verwendete Chipsätze und hilft so bei der Auswahl benötigter Treiber.

Vor der eigentlichen Konfiguration des Kernels sollten Sie sich selbst die Frage beantworten: „Mit oder ohne Initrd?“ Bei der „Initial ramdisk“ (Initrd) handelt es sich um ein mit *gzip* komprimiertes Festplatten-Image, das beim Systemstart vom Bootloader mitgeladen und vom Kernel automatisch gemountet wird. Neuere Kernel verwenden das etwas flexiblere *initramfs*, bei dem ein *cpio*-Archiv zum Einsatz kommt. Die *Initrd* enthält Treibermodule, die für das Einbinden der Wurzelpartition notwendig sind: in der Regel Dateisystemtreiber und Treiber für IDE-, ATA- oder SCSI-Chipsätze. Ein Script *linuxrc* lädt diese Treiber und mountet schließlich die Wurzelpartition. Der Vorteil bei Kerneln mit Initrd ist die erhöhte Flexibilität: Ein einmalig erstellter Standard-Kernel kann auf verschiedenen Rechnern laufen, lediglich die Initrd muss für jede Maschine neu erstellt werden, was wesentlich schneller geht als der Bau eines kompletten Kernels. Muss ein Kernel nur auf einem Rechner laufen, sollten Sie auf die Initrd - und damit eine potentielle Fehlerquelle - verzichten. Für den Systemstart notwendige Treiber müssen dann aber fix in den Kernel integriert sein: Statt *[M]* muss in der Konfiguration *[*]* angezeigt werden. Für nicht beim Systemstart benötigte Treiber gibt es keinen Grund für ein statisches Einbinden. Einzige Ausnahme: Sie bauen ein Mini-Linux, für das zusätzliche Module nur stören würden.

Feintuning und Kompilierung

Für die Kernelkonfiguration stehen drei (pseudo-) grafische Frontends bereit. Mit

```
make xconfig
```

starten Sie das auf der (auch von KDE genutzten) Bibliothek *Qt* basierende „offizielle“ Konfigurationswerkzeug. Es benötigt aller-

Tuning-Möglichkeiten

Ein optimal auf die Hardware angepasster Kernel wirkt sich positiv auf die Leistung des Systems aus. Allerdings sollten Sie die Wirkung nicht überbewerten. Jede der vorgestellten Maßnahmen ist im Schnitt für Leistungssteigerungen im einstelligen Prozentbereich gut, in der Summe sollte der Performance-Gewinn im niedrigen zweistelligen Bereich liegen. Alle folgenden Einstellungen finden Sie unter *Processor Type and Features*:

Multi Processor Support

Aktivieren Sie diese Option nur, wenn Ihr Rechner über mehrere Prozessoren oder Prozessorkerne verfügt oder Hyperthreading unterstützt.
Bis Kernel 2.6.17 war die Verwendung von Multiprozessorkerneln auf Einprozessorsystemen mit erheblichen Leistungseinbußen verbunden.

Seit 2.6.18 sorgt verbesserter SMP-Code für nur geringe Leistungseinbußen.

Processor Family

Wählen Sie hier Ihre exakte Prozessorfamilie. Der resultierende Kernel nutzt den zur Verfügung stehenden Befehlssatz aus.

SMT und Multi-Core Scheduler Support

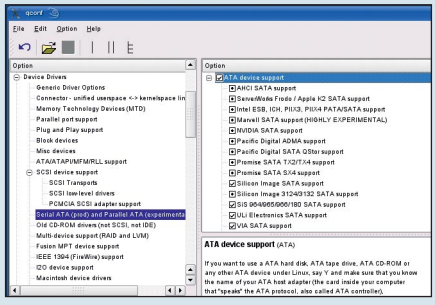
Hyperthreading (SMT) und Mehrkern-Prozessoren erfordern geringfügig andere Scheduling-Entscheidungen als echte Mehrprozessorsysteme. Insbesondere Prozessoren mit Hyperthreading, bei denen der zweite Kern nur virtuell vorhanden ist, profitieren massiv von dieser Option.

High Memory Support

Diese Option hilft bei 32 Bit-Systemen, Speicher jenseits der Adressierungsgrenze von 4 Gigabyte anzusprechen. Wählen Sie *Off*, wenn Sie über 2GByte oder weniger verfügen, ansonsten 4GByte oder 64GByte.

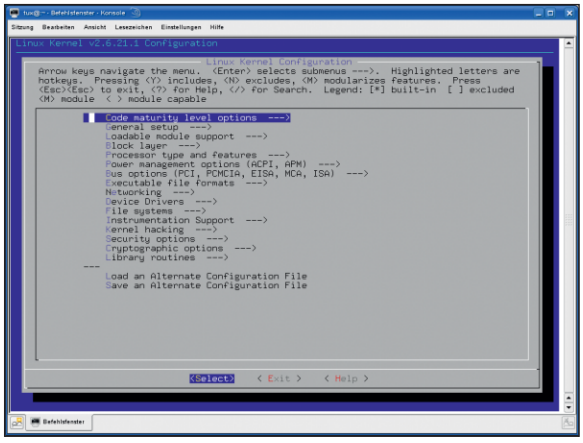
Timer Frequency

Hier bestimmen Sie die Frequenz, mit welcher der Scheduler arbeitet.

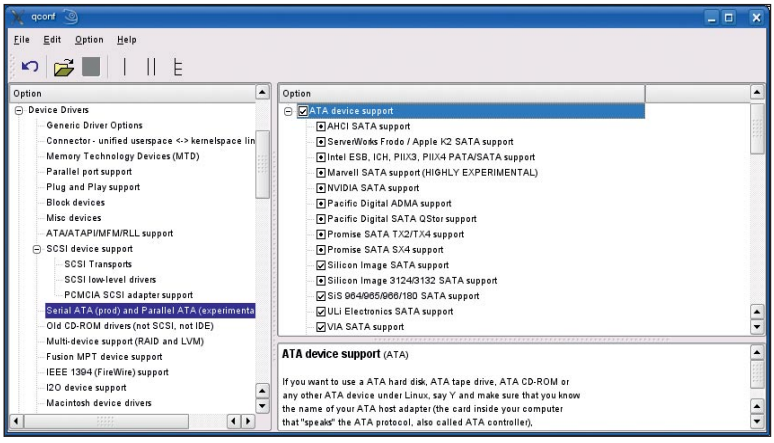


Damit der SATA-Treibner statisch eingebunden werden kann, müssen in der Hierarchie darüber liegende Treiber ebenfalls statisch eingebunden werden.

Für Server und gewöhnliche Workstations sind meist 100 Hz optimal, jeder Prozess kann dann wenigstens 10ms laufen, bevor ein anderer an die Reihe kommt. Gameserver oder Multimedia-Systeme benötigen häufig kürzere Reaktionszeiten, für sie sind 250Hz meist besser geeignet. Da hier nach 4ms Prozesse abgelöst werden und dieses Umschalten Prozessorzyklen kostet, ist die Gesamtleistung des schneller getimten Systems etwas geringer.



Auch wenn *make install* die */boot/grub/menu.lst* selbsttätig ergänzt: Nachkontrolle ist notwendig!



Mit weniger Abhängigkeiten als *xconfig* eignet sich *menuconfig* für Server und schwächere Rechner.

dings einen C++-Compiler und die Entwicklerversionen von Qt, die nicht auf jedem Rechner installiert sind. Die *gtk*-Alternative starten Sie mit

```
make gconfig
```

Funktionieren diese Tools wegen fehlender Bibliotheken oder fehlenden Zugriffsrechten auf den X-Server nicht, sollten Sie ein Terminalfenster etwas größer ziehen und mit

```
make menuconfig
```

das pseudografische Konfigurationstool starten. Die Navigation mit Tab- und Richtungstasten ist zwar nicht so komfortabel, dafür funktioniert diese Konfigurationsmethode auch per SSH auf dem Rootserver.

Ist die Konfiguration abgeschlossen und gespeichert, folgt das Bauen und die Installation des Kernels mit:

```
make
make modules_install
make install
```

Je nach Prozessorgeschwindigkeit und Auswahl der zu bauenden Treiber wird dieser Vorgang wenige Minuten bis zu einer guten Stunde dauern. Statt des letzten Schritts können Sie den Kernel *arch/i386/boot/bzImage* nach */boot* kopieren. Bei der automatischen Installation finden Sie unter */boot/vmlinuz-2.6.20.4* den Kernel und unter */lib/modules/2.6.20.4* die passenden Treibermodule.

Im Idealfall verpackt und installiert *make install* auch die *initrd* und trägt den neuen Kernel in der GRUB-Konfiguration ein. Ist dies nicht der Fall, müssen Sie diese nun mit *mkinitrd*, *mkinitramfs* oder *yaird* erstellen. Die Syntax dieser Tools unterscheidet sich in Nuancen, in der Regel muss nur die Kernel-Version angegeben werden, für welche die *initrd* erstellt wird, und der Name der resultierenden *initrd*. Für die einzubindenden Module existiert meist unter */etc* eine Konfigurationsdatei. Kontrollieren Sie abschließend die Konfigurationsdatei */boot/grub/menu.lst* und fügen Sie gegebenenfalls einen Eintrag für den neuen Kernel hinzu. Am einfachsten ist es, den Eintrag des aktuell gebooteten Kernels zu kopieren, unten anzuhängen und den Dateinamen des gebooteten Kernels abzuändern. Wenn Sie auf die Initrd verzichtet haben und benötigte Module statisch einbinden, müssen Sie die entsprechende Zeile aus dem neuen Bootloader-Eintrag entfernen.

Im Gegensatz zur altehrwürdigen *Lilo* muss *Grub* nach einer Konfigurationsänderung nicht neu geschrieben werden. Rebooten Sie den Rechner, und wählen Sie im Bootmenü den Eintrag für den neuen Kernel aus. Verliefen alle Tests mit dem neuen Kern erfolgreich, können Sie den alten Kernel in Ihrer Grub-Konfiguration auskommentieren und den neuen Kernel zum Default machen (im oberen Abschnitt der *menu.lst*). **jkn**

Ohne Initrd: Diese Treiber benötigen Sie

Um mit einem Kernel ohne initiale Ramdisk booten zu können, müssen einige Treiber statisch im Kernel enthalten sein:

Dateisystemtreiber

Ermitteln Sie mit *cat /proc/mounts*, welchen Dateisystemtyp Ihre Wurzelpartition verwendet, und wählen Sie diesen unter *Filesystems* aus. Die Unterpunkte für erweiterte Attribute und Quota können Sie auf Desktop-Systemen meist ignorieren, auf Servern können sie jedoch sinnvoll sein.

IDE- oder SATA-Treiber

Die Ausgaben der Befehle *lsmod* und *lspci -v* sowie *dmesg* geben Auskunft über IDE- und SATA-Chipsätze. Wenigstens den beim Booten benutzten Typ sollten Sie aktivieren.

SATA-Treiber finden Sie unter *SCSI*. Vergessen Sie nicht, auch die SCSI- oder IDE-Festplattenunterstützung in den Kernel zu integrieren!

RAID-Treiber

Wenn Ihre Root-Partition auf einem Software-RAID liegt, müssen Sie auch unter *Device Drivers Multi-Device Support* die Unterstützung für die verwendeten RAID-Level (meist *1* und *5*) aktivieren. Statisch integrierte RAID-Treiber erkennen auf den Festplatten eingerichtete RAID-Laufwerke automatisch beim Systemstart.

Weitere Treiber

Falls erwünscht, können Sie weitere ständig geladene Module (Netzwerktreiber etc.) ebenfalls statisch in den Kernel integrieren. Die Boot-Zeit sinkt dadurch minimal, und in einigen Fällen funktioniert die Initialisierung der Hardware etwas besser. Wenn Sie alle ständig geladenen Treiber statisch integrieren, können Sie auf Module ganz verzichten, was vor allem bei Servern beliebt ist, weil diese Maßnahme das Nachladen bösartiger Codes (Rootkits) verhindert. Vergessen Sie nicht, dass jede Modifikation eine Neukompilierung des Kernels erfordert.