4. Linux-Grundlagen: Dateisysteme, Prozesse & Co.

Nach der Lektüre von *Kapitel 2 Linux-Praxis: Anmelden und Loslegen* können Sie zwar bereits mit dem System arbeiten, doch einige Dinge sind im Unklaren geblieben, so etwa die Hinweise zum Erzeugen eines symbolischen Links.

Für die tägliche Arbeit mit Linux ist es sehr hilfreich zu wissen, wie und wo Linux Dateien ablegt, was Zugriffsrechte sind und wie Programmprozesse von Linux verwaltet und von Ihnen beeinflusst werden können.

Dieses Kapitel ist wie *Kapitel 7 Arbeiten auf der Kommandozeile* für den ersten Kontakt mit Linux nicht obligatorisch. Es sei Ihnen dennoch anempfohlen.

Dateisysteme

Wenn Sie bisher nur mit Microsoft Windows und evtl. mit dessen Vorgänger MS-DOS gearbeitet haben, wird Ihnen der Begriff "Dateisystem" noch nicht sehr oft begegnet sein. Windows NT, 2000 und XP weisen bei ihrer Installation darauf hin, dass sie mit dem moderneren Dateisystem NTFS (New Technology File System) arbeiten, während MS-DOS und alle übrigen Windows-Versionen (3.11, 95, 98, Me) das alte Dateisystem FAT (File Allocation Table) verwenden.

Was ist nun aber ein Dateisystem? Auf PCs werden Festplatten in verschiedene Partitionen unterteilt. Diese Partitionen können dann verwendet werden, um eine Verzeichnishierarchie sowie Dateien anzulegen. Von Windows ist Ihnen dieses Konzept vielleicht schon bekannt: Haben Sie auf einer Festplatte mehrere "Laufwerke", also z. B. *C:* und *D:*, so ist die Festplatte in mehrere Windows-Partitionen unterteilt, die über diese "Laufwerksbuchstaben" ansprechbar sind.

Um nun die komplexen Daten überhaupt speichern zu können, muss das Betriebssystem die Partition auf der Platte vorbereiten: Sie wird nach einem Schema formatiert, nach dem z. B. der Bereich für das Inhaltsverzeichnis des obersten Verzeichnisses (unter Windows *C:* oder *D:*) angelegt wird. Das Inhaltsverzeichnis und auch die entsprechenden Bereiche für Unterverzeichnisse enthalten nicht nur den Dateinamen und den Speicherort für jede Datei in diesem Verzeichnis, sondern es werden auch Zusatzdaten, unter DOS/Windows die so genannten *Attribute*, gespeichert: Die klassischen Attribute auf einem Windows-System sind *Read-only* (nur

lesbar), *Hidden* (versteckt), *Archive* und *System*. Versteckte Dateien werden nicht angezeigt, und System-Dateien lassen sich nicht löschen. Auf der DOS-Kommandozeile – unter Windows "Eingabeaufforderung" genannt – können diese Attribute mit dem DOS-Befehl *attrib* modifiziert werden.

Zu alten DOS/Windows-Zeiten gab es nur ein Schema, nach dem die Formatierung der Partitionen erfolgte: MS-DOS verwendete dafür das FAT-System. FAT ist ein recht einfach aufgebautes Dateisystem, das von allen gängigen Betriebssystemen gelesen werden kann: Natürlich können alle Windows-Versionen damit umgehen, aber auch Linux und OS/2 können FAT-Partitionen lesen. Das mit Windows NT eingeführte NTFS ist moderner und leistungsfähiger – alte Windows-Versionen oder MS-DOS können aber mit einer NTFS-Partition nichts anfangen, die entsprechenden Laufwerksbuchstaben fehlen beim Booten von DOS auf einem NT-System.

Linux kann als UNIX-Variante nicht in eine FAT- oder NTFS-Partition installiert werden: Die UNIX-typischen Attribute, die sich von den DOS-Attributen unterscheiden, werden von diesen Dateisystemen nicht unterstützt. So kennt etwa jede Linux-Datei einen Besitzer und eine zur Datei gehörende Benutzergruppe, ferner gibt es für jede Datei separate Lese-, Schreib- und Ausführrechte für den Besitzer der Datei, ihre Gruppe und beliebige Anwender. All dies ist nötig, da Linux (wie alle UNIX-Systeme und im Gegensatz zu Windows-Installationen, die das FAT-Dateisystem verwenden) ein Mehrbenutzersystem ist: Schließlich sollen private Dateien des einen Anwenders nicht für jeden anderen Systembenutzer lesbar sein.

Klassische Linux-Dateisysteme

Um UNIX-Attribute und einige Besonderheiten von UNIX/Linux wie etwa symbolische Links zu unterstützen, muss ein "richtiges" UNIX-Dateisystem verwendet werden. Das klassische von Linux eingesetzte System ist *ext2*, das *Second Extended Filesystem*. Dieses wird von allen aktuellen Linux-Distributionen unterstützt. Sein Vorgänger hieß einfach *ext* (*Extended Filesystem*), wird aber heute nicht mehr benutzt.

Ein weiteres Dateisystem, das unter Linux gelegentlich Einsatz findet, ist Minix: Es handelt sich um ein altes Dateisystem, das vom ausschließlich für Lehr- und Forschungszwecke konzipierten UNIX-System Minix verwendet wird. Da Minix wesentlich weniger Verwaltungsinformationen auf einem Datenträger anlegt als ext2, wird es gerne zum Formatieren von Disketten verwendet.

Journaling

Die klassischen Linux-Dateisysteme haben einen entscheidenden Nachteil: Wenn der Computer aufgrund eines Stromausfalls, Hardwaredefekts oder sonstigen Problems nicht korrekt heruntergefahren wurde, wird beim nächsten Systemstart ein Platten-

check durchgeführt: Hierbei werden alle Partitionen auf mögliche Fehler untersucht, die das plötzliche Abschalten des Rechners hervorgerufen haben könnten. Stellen Sie sich etwa vor, dass eine Datei kurz vor dem Abschalten des Rechners zum Schreiben geöffnet war und Daten in diese Datei geschrieben wurden, der Dateieintrag im Verzeichnis aber noch nicht aktualisiert wurde. Die Datei ist dann länger als im Verzeichnis angegeben. So ein Eintrag ist problematisch und muss vom Programm, das den Dateisystemcheck durchführt, korrigiert werden.

Festplatten haben heute eine Größe erreicht, bei der eine komplette Überprüfung aller Partitionen sehr lange dauern kann – für einen Server, der nach einem Ausfall möglichst schnell wieder ans Netz muss, ist das nicht akzeptabel. Somit war lange Jahre eine der Hauptforderungen an die Linux-Entwickler, Dateisysteme zu schaffen, die ohne diese langen Dateisystemprüfungen auskommen.

Genau dies leisten Journaling-Dateisysteme: Sie führen ein Journal, in das geplante Änderungen am Dateisystem vor der eigentlichen Änderung eingetragen werden. Wurde die Änderung dann erfolgreich ausgeführt, kann der entsprechende Eintrag aus dem Journal entfernt werden. Wird der Rechner nun mitten in der Änderung ausgeschaltet, ist noch der Journal-Eintrag vorhanden. Beim Rechnerneustart reicht also ein Blick in das Journal, um alle Fehler im Dateisystem zu finden.

Für Linux gibt es heute eine Reihe von Journal-Dateisystemen – die wichtigsten sind *ReiserFS* (Reiser Filesystem) und *ext3* (Third Extended Filesystem). Wie der Name ext3 schon andeutet, handelt es sich hier um den Nachfolger des ext2-Systems: Es ist zu ext2 kompatibel, so dass eine ext2-Partition recht einfach in eine ext3-Partition umgewandelt werden kann. ReiserFS verwendet völlig andere Datenstrukturen, so dass eine Umwandlung von ext2 in ReiserFS nicht direkt möglich ist – stattdessen müssen die Daten gesichert, die Partition neu als ReiserFS formatiert und dann das Backup wieder eingespielt werden.

ReiserFS ist besonders effizient bei Verwendung vieler kleiner Dateien – das liegt daran, dass ReiserFS keine Blockstruktur verwendet. "Gewöhnliche" Dateisysteme arbeiten mit diesen Blöcken, welche die kleinsten Zuordnungseinheiten innerhalb der Partition darstellen. Ist die Blockgröße etwa 4 KByte, so nimmt auch eine Datei, die nur aus 20 Bytes besteht, diese 4 KByte in Anspruch – der restliche Platz innerhalb des Blocks bleibt ungenutzt. ReiserFS verfolgt hier einen anderen Ansatz, durch den bei besonders kleinen Dateien weniger Platz auf der Festplatte verschwendet wird.

Gerätedateien

Unter Windows werden Partitionen über Laufwerksbuchstaben angesprochen. Dieser Begriff ist eigentlich ein wenig verwirrend, da zwei Partitionen auf einer Platte zwei Buchstaben (C und D) erhalten, obwohl es sich doch um ein einziges (Festplatten-)

Laufwerk handelt. Auch die häufig anzutreffenden Formulierungen "auf der C-Platte" oder "auf dem D-Laufwerk" sind eine Konsequenz dieser falschen Namensgebung. Der Ursprung dürfte bei den ersten PCs mit Festplatte zu suchen sein: 5-MByte-Platten möchte man nicht aufteilen, so dass damals jeder Festplatte ein Laufwerksbuchstabe zugeordnet wurde.

Windows durchsucht beim Systemstart die angeschlossenen Laufwerke und vergibt nach einem bestimmten Schema Laufwerksbuchstaben an FAT- und NTFS-Partitionen auf Festplatten und anderen Medien sowie CD-Laufwerke. Wer gleichzeitig mit einem DOS-basierten Windows und einer NT-Version (NT, 2000, XP) arbeitet, wird sich vielleicht auch über die unterschiedliche Benennung der Partitionen wundern: Die Windows-Versionen folgen keinem einheitlichen Standard für das Durchzählen.

Linux und die übrigen UNIX-Systeme machen das anders: Alle Festplattenpartitionen und sonstigen Datenträger werden hier über Gerätedateien angesprochen – das sind Dateien, die im Verzeichnis */dev* liegen und den direkten Zugriff auf die Hardware erlauben. So lässt sich beispielsweise die zweite Partition auf einer ersten IDE-Festplatte über */dev/hda2* ansprechen. Um nun auf die enthaltenen Daten einer Partition zugreifen zu können, muss diese in den Linux-Verzeichnisbaum eingebunden, neudeutsch: gemountet, werden. Wie dies geht, verrät der später folgende Abschnitt *Mountpoints*.

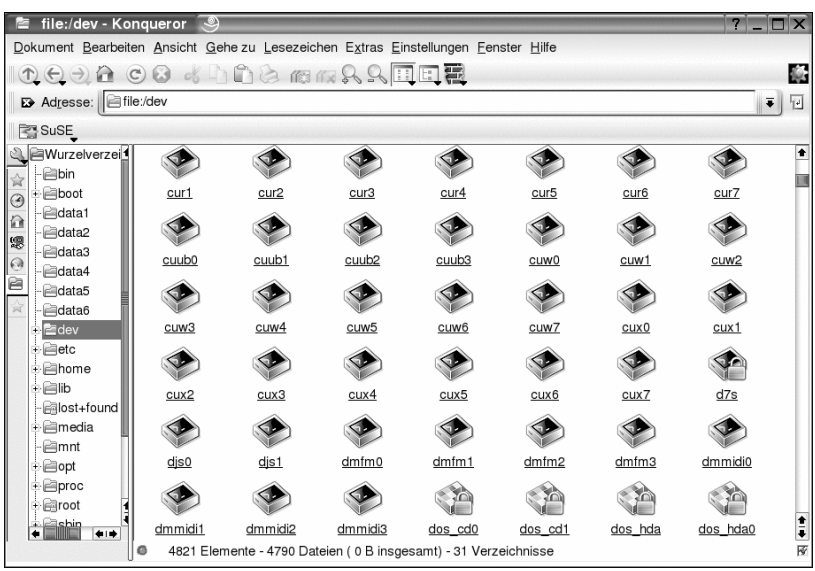

Abb. 4.1: Im Verzeichnis /dev *liegen die Gerätedateien: Diese sind keine gewöhnlichen Dateien, sondern erlauben den Zugriff auf fast alle Geräte Ihres PCs.*

Zunächst sollen aber die verschiedenen Gerätedateien behandelt werden.

Festplatten

Platten haben eine Bezeichnung, die sich aus der Plattenart (IDE oder SCSI) und ihrer Position in der Plattenreihenfolge ableitet. Dabei gelten für IDE und SCSI unterschiedliche Nummerierungsregeln.

IDE-Platten sind in der Regel an einem der beiden On-board-IDE-Controller angeschlossen. An jeden der zwei Controller können zwei Geräte, "Master" und "Slave" genannt, angeschlossen werden. Die beiden Controller werden auch "primärer" und "sekundärer" Controller genannt. Die Namen der Gerätedateien beginnen stets mit *hd*; eine Übersicht über alle möglichen Bezeichnungen gibt die folgende Tabelle.

Controller	Master/Slave	Gerätedatei
Primärer Controller	Master	*/dev/hda*
Primärer Controller	Slave	*/dev/hdb*
Sekundärer Controller	Master	*/dev/hdc*
Sekundärer Controller	Slave	*/dev/hdd*

Übersicht der Gerätedateinamen für IDE-Festplatten

Für die Vergabe dieser Gerätedateinamen ist es unerheblich, wie viele Geräte angeschlossen sind: Haben Sie beispielsweise nur einen Master am ersten Controller und einen Slave am zweiten Controller, dann sind die entsprechenden Gerätedateien */dev/hda* und */dev/hdd* – die "dazwischen liegenden" */dev/hdb* und */dev/hdc* können nicht verwendet werden, da keine solchen Geräte vorhanden sind.

Sind nicht alle angeschlossenen Geräte Festplatten, sondern beispielsweise CD-ROM- oder DVD-Laufwerke, so werden diese trotzdem nach obigem Schema einer Gerätedatei zugeordnet: Ein CD-ROM-Laufwerk als Master am zweiten Controller heißt dann beispielsweise */dev/hdc*, obwohl es keine Platte ist.

Bei SCSI-Geräten wurde ein anderer Weg für die Benennung der Geräte gewählt, da hier auch die Geräte anders angeschlossen werden. Alle SCSI-Geräte an einem SCSI-Controller haben eindeutige SCSI-IDs – das sind Nummern zwischen 0 und 15 bzw. zwischen 0 und 7 (abhängig vom Controller-Typ). Die ID 0 ist dabei meist für den Controller selbst reserviert, so dass für die Geräte die IDs 1–15 bzw. 1–7 übrig bleiben.

SCSI-Festplatten erhalten nun Gerätedateinamen, die mit *sd* (für *SCSI Disk*) beginnen; dabei wird entsprechend der SCSI-ID-Reihenfolge *sda*, *sdb*, *sdc* etc. durchgezählt. Es ist wichtig zu wissen, dass hier wirklich nur Platten gezählt werden – sonstige SCSI-Geräte erhalten andere Gerätedateinamen.

Wenn mehrere SCSI-Controller im Rechner stecken, werden alle Platten gemeinsam durchnummeriert, wobei die Reihenfolge der Controller auch eine Rolle spielt: Sind am ersten Controller beispielsweise zwei Platten angeschlossen, werden diese den Gerätedateien */dev/sda* und */dev/sdb* zugeordnet, während eine weitere Platte am zweiten SCSI-Controller */dev/sdc* heißt.

Partitionen

Über die Gerätedateien für die Platten kann nun auf diese zugegriffen werden – das nützt allerdings zunächst nicht viel, da nicht eine Platte selbst, sondern eine der darauf enthaltenen Partitionen gemountet werden muss: Schließlich sind es die Partitionen und nicht die Platten, auf denen sich die Dateisysteme befinden.

Bevor die Bezeichnungen für Partitionen eingeführt werden, soll zunächst das Partitionierungssystem erklärt werden, das auf PCs eingesetzt wird. Diese Informationen gelten gleichermaßen für IDE- und SCSI-Platten.

Jede Festplatte kann vier so genannte primäre Partitionen enthalten. Es gibt auf jeder Festplatte ein zentrales Inhaltsverzeichnis, das entsprechenden Speicherplatz für Informationen über diese vier Partitionen bietet. Die vier primären Partitionen werden von 1 bis 4 durchnummeriert. Diese Nummer wird an den Namen der Plattengerätedatei angehängt, um den Namen der Partitionsgerätedatei zu bilden: So heißt etwa die erste Partition auf der Platte */dev/hda* einfach */dev/hda1*.

Damit mehr als vier Partitionen auf einer Platte verwendet werden können, wurde zu MS-DOS-Zeiten das Konzept der erweiterten Partition eingeführt: Auf jeder Festplatte darf eine der vier primären Partitionen vom Typ "erweitert" sein – innerhalb dieser erweiterten Partition können dann weitere "logische" Partitionen angelegt werden.

In Microsoft-Sprache heißen diese dann logische Laufwerke. Linux kann mit diesen erweiterten Partitionen umgehen und innerhalb einer solchen prinzipiell beliebig viele logische Partitionen anlegen. Sie werden ab 5 durchnummeriert, so dass etwa die erste logische Partition auf der Platte */dev/hda /dev/hda5* heißt.

In einem laufenden System können Sie sich mit YaST einen Überblick über die aktuelle Partitionierung verschaffen; wählen Sie dazu das Modul *System > Partitionieren*.

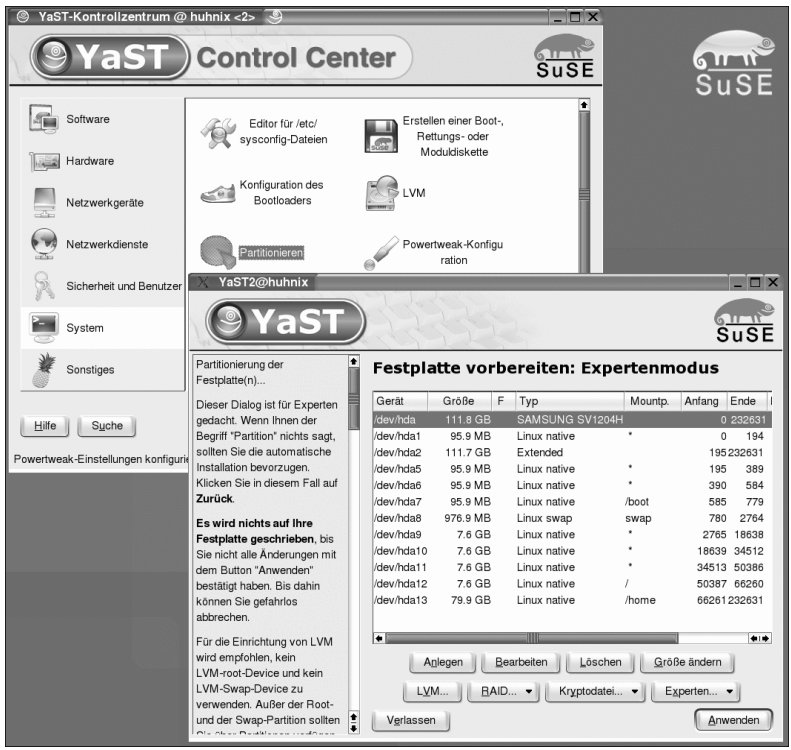

Abb. 4.2: YaST zeigt die Partitionen an.

CD-ROM, DVD, Disketten und ZIP

Auch CD-ROM- oder DVD- und Diskettenlaufwerke werden über Gerätedateien angesprochen. Wie oben bereits erwähnt wurde, haben ATAPI- (d. h. IDE-)Laufwerke den gleichen Namen, den eine Festplatte hätte, die an entsprechender Stelle angeschlossen wäre. Für SCSI-Laufwerke gibt es hingegen andere Bezeichnungen: Hier werden alle am SCSI-Controller erkannten CD- und DVD-Laufwerke als */dev/scd0*, */dev/scd1* etc. durchnummeriert. "scd" steht dabei für SCSI CD.

SuSE Linux erkennt CD- und DVD-Laufwerke bei der Installation in der Regel selbstständig und richtet für diese die Gerätedateien */dev/cdrom*, */dev/cdrecorder* oder */dev/dvd* ein, so dass Sie diese Dateinamen anstelle der tatsächlichen verwenden können. Das in heutigen Systemen meist einzige Diskettenlaufwerk wird über */dev/fd0* (Floppy Disk 0) angesprochen; ein zweites Laufwerk hieße entsprechend */dev/fd1*. An der Benennung ändert sich übrigens nichts, wenn ein altes 5,25-Zoll-Laufwerk verwendet wird.

ZIP-Drives sind keine Diskettenlaufwerke – für sie gelten dieselben Namenskonventionen wie für Festplatten, schließlich sind ZIP-Medien auch in der Regel partitioniert: Original-Iomega-Medien haben eine reguläre Partitionstabelle, auf der eine FAT-Partition mit der Partitionsnummer 4 angelegt ist – die entsprechende Gerätedatei heißt dann z. B. */dev/hdc4* oder */dev/sdb4*. Leider gibt es einige Hersteller von ZIP-Medien, die sich nicht an diese Konvention halten – so verwenden manche NoName-Medien Partition 1. Zudem gibt es Medien, die gar nicht partitioniert, sondern als Ganzes FAT-formatiert sind: In einem solchen Fall ist z. B. */dev/hdc* oder */dev/sdb* anzusprechen.

Mountpoints

Da Linux keine "Laufwerksbuchstaben" vergibt, müssen Datenträger anders in den Verzeichnisbaum eingebunden werden: Dies geschieht durch Angabe eines bestimmten Verzeichnisses, unterhalb dessen sich dann die Hierarchie des Datenträgers befinden soll. Bei der Erstinstallation sorgt Linux dafür, dass es auf seine eigenen Partitionen, aber auch auf "DOS-/Windows-Laufwerke" zugreifen kann: Es nimmt dazu entsprechende Einträge in der Mount-Konfigurationsdatei */etc/fstab* (File System Table) vor. Werfen Sie einen Blick in eine typische Datei dieser Art:

```
/dev/hda5       /                 reiserfs    defaults 1 2
/dev/cdrecorder /media/cdrecorder auto        ro,noauto,user,exec 0 0
/dev/cdrom      /media/cdrom      auto        ro,noauto,user,exec 0 0
/dev/hda8       /data1            auto        noauto,user 0 0
devpts          /dev/pts          devpts      defaults 0 0
/dev/dvd        /media/dvd        auto        ro,noauto,user,exec 0 0
/dev/fd0        /media/floppy     auto        noauto,user,sync 0 0
proc            /proc             proc        defaults 0 0
usbdevfs        /proc/bus/usb     usbdevfs    noauto 0 0
/dev/hda1       /windows/C        vfat        noauto,user 0 0
/dev/hda7       /windows/D        ntfs        ro,noauto,user,umask=022 0 0
/dev/hda9       /windows/E        vfat        noauto,user 0 0
/dev/hda6       swap              swap        pri=42 0 0
```

Betrachten Sie nur die Einträge, die mit */dev* beginnen; die übrigen dienen Spezialfunktionen des Kernels. Jede Zeile in dieser Datei legt einen Mountpoint fest, wobei eine Zeile aus sechs durch Leerzeichen oder Tabulator getrennten Feldern besteht. Interessant sind an dieser Stelle nur die ersten vier: Sie geben das zu mountende Gerät, also beispielsweise eine Partition oder eine Diskette, den Mountpoint, das verwendete Dateisystem, beispielsweise ReiserFS bei Linux-Partitionen oder VFAT für Windows-Datenträger, und die Mount-Optionen an.

Im Einzelnen haben die Spalten die folgenden Bedeutungen:

Spalte 1: Gerätedatei

In der ersten Spalte steht stets die Gerätedatei, über die das Medium angesprochen werden kann. Bei Festplattenpartitionen ist dies also immer eine Gerätedatei der Form */dev/hda1* bzw. */dev/sda1*. Geht es um Disketten, steht hier eine der Gerätedateien */dev/fd0* oder */dev/fd1*. Für CD-ROM-Laufwerke wird meist nicht der Dateiname der konkreten Gerätedatei eingetragen; stattdessen findet man */dev/cdrom*, was dann ein symbolischer Link auf die richtige Gerätedatei ist.

Anstelle eines Gerätes kann hier auch ein Dateisystem angegeben werden, das über das Netzwerk erreichbar ist. Wenn sich im Netz etwa ein NFS-Server namens *server* befindet, der die Home-Verzeichnisse für alle Workstations im Netzwerk speichert, könnte der Eintrag in der Gerätedateispalte auch die Form *server:/export/home* haben.

Spalte 2: Mountpoint

In der zweiten Spalte wird festgelegt, unter welchem Verzeichnisnamen das eingehängte Dateisystem erreichbar sein soll. Dieses Verzeichnis muss im System bereits vorhanden sein und sollte leer sein. Wenn es nicht leer ist, führt das zu keinem Fehler, aber die Dateien, die "ursprünglich" in diesem Verzeichnis lagen, sind so lange nicht erreichbar, wie das gemountete Dateisystem den Platz blockiert.

Typische Mountpoints für Plattenpartitionen sind */* (das Wurzeldateisystem), */usr*, */var*, */home* und */usr/local*. Wechselmedien werden unter Linux typischerweise in Verzeichnisse wie */mnt/cdrom* und */mnt/floppy* gemountet, SuSE Linux weicht hier allerdings vom Üblichen ab und verwendet stattdessen die Verzeichnisnamen */media/floppy*, */media/cdrom*, */media/dvd*, */media/cdrecorder* etc.

DOS-/Windows-Partitionen werden von SuSE Linux als */windows/C*, */windows/D* usw. in den Verzeichnisbaum eingehängt – die "Laufwerksbuchstaben" müssen aber nicht unbedingt mit denen unter Windows übereinstimmen.

Spalte 3: Dateisystemtyp

Hier wird angegeben, um welche Art von Dateisystem es sich handelt. Die Unterschiede zwischen den verschiedenen Dateisystemen wurden schon weiter vorne in diesem Kapitel behandelt; in dieser Spalte steht immer das Kürzel, unter dem das Dateisystem dem Linux-Kernel bekannt ist.

Eine unvollständige Übersicht möglicher Einträge gibt die folgende Tabelle:

Typ	Name	Beschreibung
ext2	Second Extended Filesystem	Das ehemalige Standarddateisystem unter Linux
ext3	Third Extended Filesystem	ext2-Nachfolger mit Journaling
reiserfs	Reiser Filesystem	Journaling-Dateisystem
iso9660	ISO-9660	Standard für CD-ROMs
vfat	MS-DOS VFAT	Dateisystem von DOS und Windows 95/98/Me
ntfs	NT Filesystem	Dateisystem von Windows NT, 2000, XP
hpfs	High Performance Filesystem	Dateisystem von IBM OS/2
minix	Minix-Dateisystem	
nfs	Network Filesystem	Netzwerk-Dateisystem, greift auf Server zu
auto	Auto-Dateisystem	Kernel versucht, Typ automatisch zu erkennen
swap	Swap	Auslagerung; wird nicht gemountet

Wenn Sie einen Blick in die Kernel-Konfiguration werfen, werden Sie noch etliche weitere Dateisystemtypen finden – die meisten werden aber eher selten verwendet.

Spalte 4: Optionen

In die vierte Spalte werden Optionen eingetragen, die dem *mount*-Befehl beim Einbinden des Dateisystems übergeben werden sollen. Eine Erklärung dieser Optionen finden Sie im folgenden Abschnitt über das Mounten.

Mounten: Manuell und automatisch

Dateisysteme werden normalerweise beim Betriebssystemstart eingebunden – das gilt in jedem Fall für diejenigen, die vom Linux-System benötigt werden. Einige will man aber vielleicht nicht ständig eingebunden haben, und insbesondere bei Wechseldatenträgern ergibt es sowieso keinen Sinn, schon beim Booten einen Mount-Versuch zu unternehmen.

Daher kann man Datenträger automatisch mounten lassen oder dies manuell vornehmen. Das automatische Mounten funktioniert über die Konfigurationsdatei */etc/fstab*, die bereits besprochen wurde – für das manuelle Mounten müssen Sie die Kommandozeile bemühen. Sind Sie mit dem Umgang in der Shell noch nicht vertraut, sollten Sie an dieser Stelle einen Sprung zu *Kapitel 7 Arbeiten auf der Kommandozeile* machen und dort die einführenden Abschnitte lesen.

Für das manuelle Mounten sind stets Root-Rechte erforderlich. Geben Sie in der Shell zunächst "su" und dann das Administratorpasswort ein; so werden Sie zum Benutzer *root*. Der Befehl zum Mounten heißt *mount*. Seine Syntax ist stets wie folgt:

```
mount -t dateisystem -o Optionen Gerätedatei Mountpoint
```

Um etwa eine Windows-FAT-Partition */dev/hda1* nach */mnt/windows* zu mounten, würden Sie den Befehl

```
mount -t vfat /dev/hda1 /mnt/windows
```

eingeben. Die Liste der Optionen ist hier leer, dadurch wird mit den Standardeinstellungen für das Dateisystem *vfat* gemountet. Da Linux in den meisten Fällen den Dateisystemtyp einer Partition selbstständig erkennen kann, würde auch die Eingabe

```
mount /dev/hda1 /mnt/windows
```

ausreichen. Das Verzeichnis */mnt/windows* muss dabei übrigens existieren – versuchen Sie, ein Dateisystem in ein nicht vorhandenes Verzeichnis zu mounten, erhalten Sie die Fehlermeldung

```
mount: Mountpunkt /mnt/falsch existiert nicht.
```

Die Dateisystemangabe hinter "-t" verwendet die gleichen Bezeichnungen, wie sie in der Datei */etc/fstab* benutzt werden (siehe obige Tabelle). Wollen Sie sehen, welche Typen von Ihrem Kernel unterstützt werden, werfen Sie einen Blick in die Datei */proc/filesystems*:

```
esser@pentium4:~> cat /proc/filesystems
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   tmpfs
nodev   shm
nodev   pipefs
        ext2
        minix
        iso9660
nodev   nfs
nodev   devpts
        reiserfs
nodev   usbdevfs
esser@pentium4:~> _
```

Dabei werden aber nur die Dateisysteme angezeigt, deren Unterstützung entweder fest in den Kernel eingebaut wurde oder deren Kernel-Modul gerade geladen ist. Wollen Sie zusätzlich noch die potenziell mit Ihrem Kernel nutzbaren Dateisysteme sehen, schauen Sie in das Modulverzeichnis:

```
ls -l /lib/modules/2.4.21-99-athlon/kernel/fs/
```

Mount-Rechte für alle

Bevor Sie sich dem Abschnitt über Mount-Optionen widmen, gibt es hier vorab ein Beispiel für eine besonders nützliche Option. Wie bereits erklärt, darf nur der Administrator *root* den *mount*-Befehl verwenden – schließlich wäre es auf einem Mehrbenutzersystem nicht wünschenswert, wenn ein anderer Anwender die Festplatten manipuliert. Dennoch ist es manchmal wünschenswert, allen Anwendern das Recht zum Mounten eines Datenträgers einzuräumen, vor allem bei Wechseldatenträgern wie Disketten, CDs und ZIPs.

Genau dafür gibt es die spezielle Option "user", die in eine Zeile der Mount-Konfigurationsdatei */etc/fstab* eingetragen und häufig mit "noauto" verwendet wird: "user" erlaubt Anwendern, das Dateisystem zu mounten, "noauto" verhindert ein automatisches Mounten beim Systemstart. Tatsächlich enthält die *fstab*-Datei bei SuSE Linux schon eine Reihe dieser Einträge:

```
esser@pentium4:~> cat /etc/fstab | grep user
/dev/cdrecorder /media/cdrecorder  auto   ro,noauto,user,exec 0 0
/dev/cdrom      /media/cdrom       auto   ro,noauto,user,exec 0 0
/dev/dvd        /media/dvd         auto   ro,noauto,user,exec 0 0
/dev/fd0        /media/floppy      auto   noauto,user,sync 0 0
/dev/hda1       /windows/C         vfat   noauto,user 0 0
esser@pentium4:~> _
```

Wollen Sie nun als Anwender einen der hier beschriebenen Datenträger mounten, geben Sie (z. B. für eine DVD) einfach

```
mount /media/dvd
```

ein. Auch alle vom Linux-Installer erkannten Windows-Partitionen lassen sich so mounten.

Nur dank dieser Einstellungen kann auch KDE automatisch CDs und Disketten einbinden: Wenn Sie auf eines der Laufwerkssymbole auf dem KDE-Desktop klicken, wird das *mount*-Kommando ausgeführt, weil in der Konfigurationsdatei das Mounten durch normale Anwender zugelassen wurde.

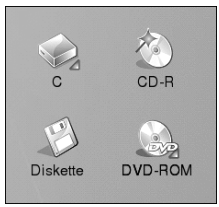

Abb. 4.3: Über diese Icons lassen sich Wechselmedien unter KDE bequem mounten – weil das System so konfiguriert wurde, dies zuzulassen.

Mount-Optionen

An dieser Stelle soll nun noch ein kurzer Überblick über die am häufigsten genutzten Mount-Optionen gegeben werden. Sollen mehrere Optionen gleichzeitig verwendet werden, so sind diese durch Kommata ohne Leerzeichen zu trennen. Ein Beispiel ist "noauto,user,exec". Die Vielzahl der Optionen ist in allgemeine Optionen und solche unterteilt, die nur für bestimmte Dateisysteme anwendbar sind.

Allgemeine Optionen

Einige Optionen können für jedes Dateisystem verwendet werden, dazu gehören unter anderem die folgenden:

- *user*: Wurde bereits oben besprochen; erlaubt es Anwendern, das Dateisystem ohne *root*-Rechte zu mounten.
- *nouser*: Erlaubt genau dies nicht (Standard).
- *ro*: Das Dateisystem wird nur mit Leserechten (ro: read-only) gemountet. Das ist sinnvoll, wenn die Daten einer Partition zwar gelesen, aber vor Veränderungen geschützt werden sollen.
- *rw*: Das Dateisystem wird mit Lese- und Schreibrechten gemountet.
- *auto*: Das Dateisystem wird beim Booten automatisch gemountet (Standard).
- *noauto*: Das Dateisystem wird beim Booten nicht gemountet.
- *exec*: Programme auf dem Dateisystem können ausgeführt werden.
- *noexec*: Programme dürfen nicht ausgeführt werden, selbst wenn sie die dafür nötigen Attribute haben.
- *default*: Standardeinstellungen. *default* entspricht *rw,suid,dev,exec,auto,nouser,async*.
- *remount*: Das Dateisystem ist bereits gemountet und wird erneut, aber mit veränderten Optionen gemountet – das ist nur sinnvoll, wenn der *mount*-Befehl auf der Kommandozeile ausgeführt wird. In der *fstab* wird man diese Option nie finden.

Eine Auflistung sämtlicher allgemeiner Optionen finden Sie in der Man-Page zu *mount*.

Spezialoptionen

Für die meisten Dateisysteme gibt es neben den allgemeinen Optionen noch zusätzliche, die speziell auf das jeweilige Dateisystem zugeschnitten sind. In diesem Abschnitt werden nur die Spezialoptionen für die Windows-Dateisysteme und CDs besprochen; wenn Sie andere Dateisysteme einsetzen wollen, hilft ein Blick in die Man-Page zu *mount*: Dort sind sämtliche Optionen aller von Linux unterstützten Dateisysteme aufgelistet.

NTFS und FAT

Windows-Dateisysteme verwenden andere Dateiattribute (*read-only*, *hidden*, *system*, *archive*) als Linux. Das FAT-Dateisystem kennt zudem keine Dateibesitzer oder Gruppen. Wird eines dieser Dateisysteme mit Standardeinstellungen gemountet, gehören alle dort liegenden Dateien dem Systemadministrator *root*. Über die Optionen *uid=nnn*, *gid=nnn* und *umask=nnn* kann der Zugriff auf die dort gespeicherten Daten vereinfacht werden.

- *uid=nnn*: Diese Option legt fest, wer aus Linux-Sicht Besitzer der Datei sein soll. Wollen Sie für Ihren normalen Benutzer-Account Lese- und (für FAT-Partitionen) Schreibzugriff einräumen, prüfen Sie zunächst als normaler Anwender mit dem Befehl *id*, welche Benutzer-ID Sie haben. Ändern Sie dann als *root* in der Datei */etc/fstab* die Optionen für die Windows-Partition von "noauto,user" auf "uid=500,gid=100". Dadurch wird zukünftig die Windows-Partition automatisch gemountet, und Sie erscheinen als Besitzer aller Dateien.
- *gid=nnn*: Diese Option leistet etwas Ähnliches wie die vorherige, nur wird hier die Gruppen-ID gesetzt. Alle Anwender gehören der Gruppe *users* mit der Gruppen-ID 100 an.
- *umask=nnn*: Über diese Option können pauschal bestimmte Rechte an allen Dateien des Dateisystems vergeben werden. In der Regel wird hier aber automatisch ein passender Wert gesetzt.

Ist für eine Windows-Partition die Option *user* gesetzt, werden die Dateirechte beim Mounten durch einen Anwender automatisch so gesetzt, dass er Inhaber dieser Dateien wird. Die *uid*- und *gid*-Optionen sind also nur dann sinnvoll, wenn die Partition schon beim Systemstart gemountet werden soll und der Anwender feststeht, der auf sie zugreifen wird.

ISO9660

Das CD-Dateisystem ISO9660 ist in seiner ursprünglichen Form nur eingeschränkt nutzbar, da es zum Beispiel keine langen Dateinamen und keine Unterscheidung von Groß- und Kleinbuchstaben kennt. Daher wurden sowohl in der UNIX- als auch in der Windows-Welt Erweiterungen zu ISO9660 geschaffen.

Unter UNIX/Linux sind dies die so genannten Rockridge Extensions: Diese erlauben lange Dateinamen, UNIX-typische Besitzer- und Gruppeninformationen sowie Rechte. So kann eine CD, die im ISO9660-Format mit Rockridge Extensions gebrannt wurde, unter Linux als vollwertiges UNIX-konformes Dateisystem angesprochen werden. In der Windows-Welt gibt es analog dazu die Joliet Extensions, die ebenfalls lange Dateinamen und einige Windows-spezifische Dinge unterstützen. Beide Erweiterungen lassen sich auch gemeinsam verwenden.

- *nojoliet*: Unterbindet die Verwendung der Joliet Extensions. Das kann sinnvoll sein, wenn eine CD nur mit Joliet Extensions (und ohne Rockridge Extensions) gebrannt wurde und Sie auf die aus Kompatibilitätsgründen stets vorhandenen "reinen" ISO9660-Daten zugreifen wollen. Die Dateinamen erinnern dann an alte DOS-Zeiten.
- *norock*: Unterbindet die Verwendung der Rockridge Extensions.
- *session=n*: Bei einer Multi-Session-CD kann über diese Option die n-te Session gemountet werden. Sind gegenüber der finalen Session Dateien von der CD "gelöscht" worden, kann das Mounten einer älteren Session diese wieder zugänglich machen.

Prozesse

Eine der wichtigsten Stärken von Linux ist die außergewöhnlich hohe Stabilität. Diese Eigenschaft eines Betriebssystems, das parallele Laufen mehrerer Programme zu ermöglichen und selbst bei einer Fehlfunktion eines Programms den Betrieb aufrechtzuerhalten, wird wesentlich von der Speicher- und Prozessverwaltung beeinflusst.

Prozesse kann man sich vereinfacht als laufende Programme vorstellen: Jedes Mal, wenn Sie ein Programm starten, wird ein Speicherbereich für das Programm reserviert, das Programm selbst wird in einen Teil dieses Speichers geladen, und schließlich wird es in die Prozesstabelle aufgenommen. Mehrere Programme gleichzeitig auszuführen, ist nicht möglich; dafür müsste der Rechner für jedes Programm einen eigenen Prozessor besitzen. Tatsächlich teilen sich also alle laufenden Programme die Rechenzeit eines einzigen Prozessors. Selbst auf Mehrprozessorsystemen laufen in der Regel deutlich mehr Programme als es Prozessoren gibt, so dass für diese das Gleiche gilt.

Die Kunst des Betriebssystems liegt nun darin, die Rechenzeit in kleinsten Scheiben auf die aktiven Programme zu verteilen. Dafür besitzt das System einen Mechanismus, mit dem es einen Prozess starten, anhalten und später fortsetzen kann. Auch wenn Sie nur auf den Bildschirm schauen und kein Programm irgendeiner sichtbaren Aktivität nachgeht, läuft dieser Vorgang des Unterbrechens und Weiterreichens der Prozessorzeit an den nächsten Prozess in der Warteliste ständig ab.

Präemptives Multitasking

Ältere Windows-Versionen (3.x) und auch MacOS (bis Version 9) arbeiten mit einem System, das sich "kooperatives Multitasking" nennt. Was hier sehr positiv und freundlich klingt, hat einen entscheidenden technischen Nachteil.

Mit kooperativ ist nicht das Betriebssystem gemeint, sondern es drückt die Hoffnung aus, dass alle laufenden Programme sich kooperativ verhalten: Dort kann nämlich nur dann ein anderer Prozess Rechenzeit erhalten, wenn der gerade laufende freiwillig auf eine Fortsetzung verzichtet.

Aktuelle Betriebssysteme setzen stattdessen auf präemptives Multitasking: Hier kontrolliert das System die Prozesse und lässt nicht zu, dass ein Programm alle Ressourcen an sich zieht und so andere Prozesse nicht mehr zur Ausführung gelangen.

Prozessverwaltung

Auf die Prozesse kann vielfältig Einfluss genommen werden. Die einfachsten Dinge sind banal: Sie können neue Prozesse starten – etwa durch Aufruf eines Programms aus dem Startmenü –, Sie können sie anhalten und fortsetzen oder komplett beenden. Außerdem lässt sich die so genannte Prozesspriorität verändern, so dass ein Prozess "wichtiger" als andere wird. Das ist beispielsweise interessant, wenn Sie auf einem stark ausgelasteten System eine CD brennen wollen: Sie können das Brennprogramm mit maximaler Priorität laufen lassen, um "verbrannte" CDs zu vermeiden.

Die Prozesse verwalten Sie mit der Konsole. Die Shell bietet eine Reihe von Befehlen, mit denen dies sehr effektiv ist. Wer es lieber grafisch will, kann das KDE-Programm *ksysguard* verwenden, das sich über *System > Überwachung > Systemüberwachung* aufrufen lässt.

Dieses bietet eine Ansicht aller laufenden Prozesse wahlweise nach Prozessnummer sortiert oder in einer Baumansicht, die darstellt, welcher Prozess von welchem gestartet wurde: Ruft ein Prozess einen zweiten auf, so heißt der erste *Vater* des zweiten, der zweite heißt *Sohn*. Im Englischen wird statt *Vater* die Bezeichnung *Parent* verwendet, was in einer etwas steifen Übersetzung *Elternteil* hieße.

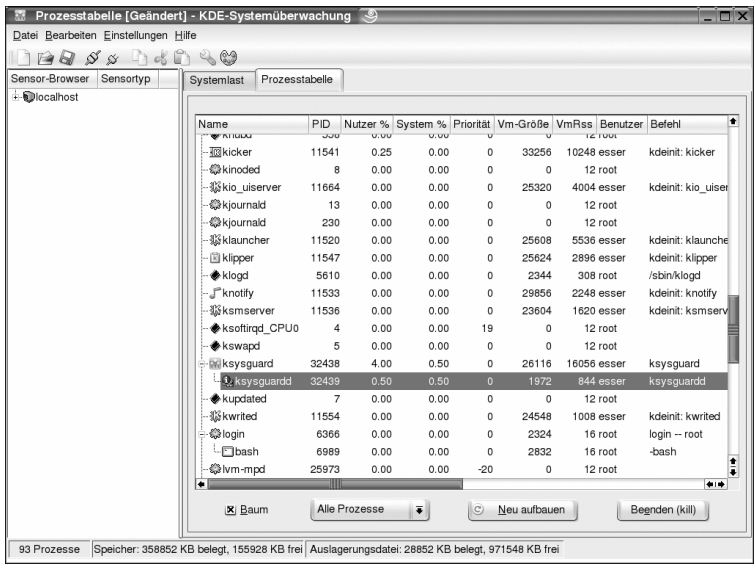

Abb. 4.4: ksysguard *zeigt im Tab* Prozesstabelle *die Baumstruktur der laufenden Prozesse an.*

Daneben bietet dieses Programm noch eine grafische Übersicht der CPU-Auslastung, die recht hübsch anzusehen ist: In vier Teilfenstern finden Sie laufend aktualisierte Informationen zu CPU-Last, Speicher- und Swap-Verwendung.

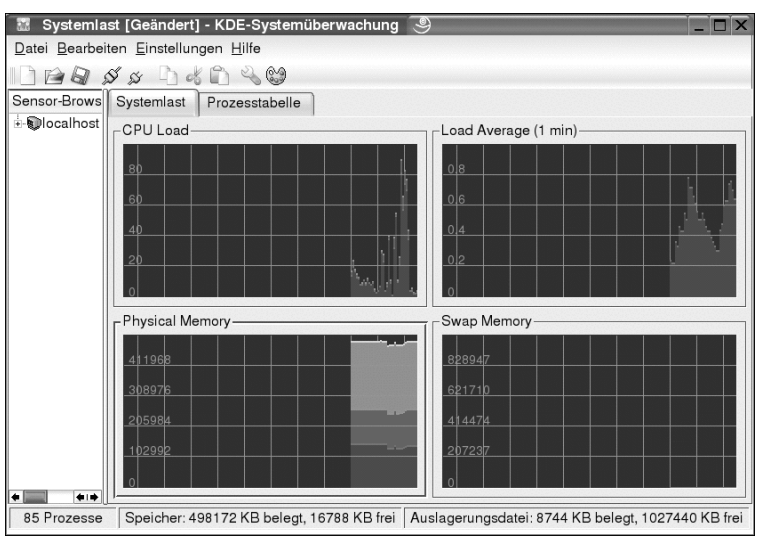

Abb. 4.5: Dazu gibt es noch eine grafische Übersicht der Auslastung.

Ein Fenster allein mit der Prozesstabelle, ohne Auslastungs-Tab, erhalten Sie übrigens durch Eingabe von "kpm" in das Schnellstartfenster (Alt+F2) oder in einer Konsole. Leider lässt sich mit den Prozessen an dieser Stelle nicht viel machen: Sie können markiert und per Klick auf *Beenden (kill)* beendet werden.

Falls auch GNOME installiert ist, können Sie dessen leistungsfähigeres Programm *System Monitor* verwenden: Dieses bietet zwar keine Baumansicht, kann dafür aber beliebige *Signale* an ausgewählte Prozesse schicken. Auf die Bedeutung dieser Signale, mit denen Sie jedes laufende Programm beispielsweise unterbrechen und wieder fortsetzen können, wird in einem der nächsten Abschnitte noch detailliert eingegangen. Es kann über *System > Weitere Programme > System-Monitor* aufgerufen werden.

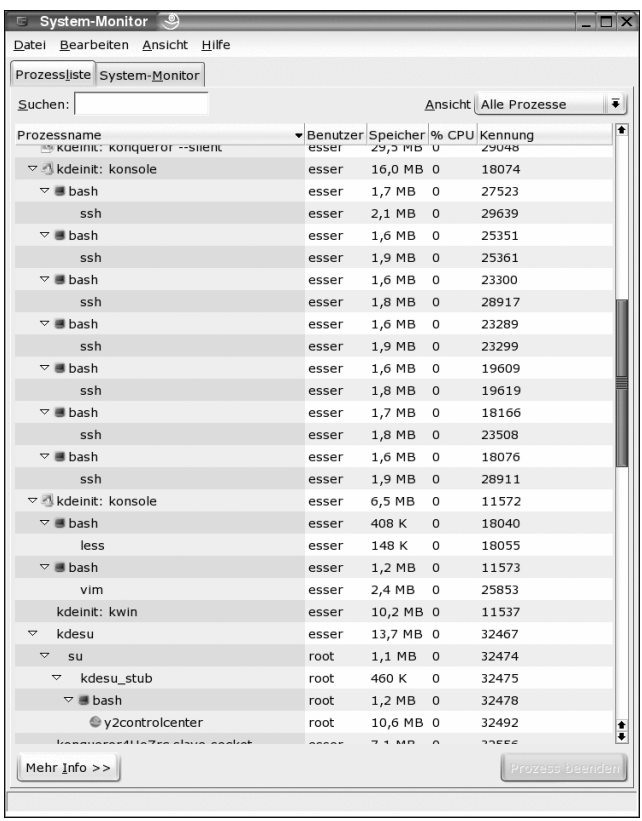

Abb. 4.6: Der GNOME-System-Monitor ist eine Alternative zu ksysguard*, die aber nur zur Verfügung steht, wenn GNOME installiert wurde.*

Prozesse in der Shell

Wollen Sie sich in der Shell einen Überblick über laufende Prozesse verschaffen, gibt es dafür zwei Möglichkeiten in Form der beiden Tools *ps* (process status) und *top*. *ps* ist eines der Standard-Tools, die es auf jeder UNIX-Maschine gibt. Wird es ohne Parameter aufgerufen, zeigt es vereinfacht gesagt alle Prozesse an, die vom aufrufenden Anwender gestartet wurden und in einem Terminal oder als X-Anwendung laufen:

```
esser@pentium4:~> ps
  PID TTY          TIME CMD
 2061 pts/0    00:00:00 bash
 2088 pts/0    00:01:44 Xvnc
 2094 pts/0    00:00:00 startkde
 2134 pts/0    00:00:00 kwrapper
 2146 pts/0    00:00:00 kinternet
 2159 pts/2    00:00:00 bash
 2161 pts/3    00:00:00 bash
 2167 pts/4    00:00:00 bash
 2970 pts/5    00:00:00 bash
 3001 pts/6    00:00:00 bash
 3139 pts/0    00:00:00 ps
esser@pentium4:~> _
```

Durch zusätzliche Optionen, die ohne das übliche "-" angegeben werden, kann die Ausgabe von *ps* erweitert werden. Die Option "a" (alle) ergänzt Prozesse aller Anwender, "u" (user) zeigt zu jedem Prozess den Benutzernamen an, "x" zeigt Prozesse, die zu keinem Terminal und nicht zu X gehören (darunter beispielsweise alle Daemon-Prozesse), und "w" (wide) verlängert die Ausgabezeile – prinzipiell versucht ps, die volle Kommandozeile anzuzeigen, mit der ein Prozess gestartet wurde, ab einer bestimmten Länge wird aber abgeschnitten. Mit "w" erhält man wieder mehr Informationen, für stärkere Wirkung kann "w" auch mehrfach verwendet werden. Die Optionen lassen sich kombinieren, etwa zum Aufruf "ps auxwww":

```
esser@pentium4:~> ps auxwww
USER  PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root    1  0.0  0.0   448  220 ?        S    09:46   0:05 init [5]
root    2  0.0  0.0     0    0 ?        SW   09:46   0:00 [keventd]
[…]
esser 2088  1.0  2.1 13288 11004 pts/0  S    10:01   1:54 Xvnc :1 -
desktop X -httpd /usr/share/vnc/classes -auth /home/esser/.Xauthority -geometry
1024x768 -depth 16 -rfbwait 120000 -rfbauth /home/esser/.vnc/passwd -rfbport 5901
esser 2094  0.0  0.2  2560  1208 pts/0  S    10:01   0:00 /bin/sh
/opt/kde3/bin/startkde
[…]
esser@pentium4:~> _
```

Wie Sie sehen, können solche Ausgabezeilen recht lang werden. Die letzte Ausgabe bestand aus 81 Zeilen und wurde hier nur verkürzt abgedruckt. Über die Spalte *COMMAND* lassen sich die laufenden Prozesse identifizieren. Sie können einem Prozess nun mit dem *kill*-Befehl ein Signal senden, um ihn zu unterbrechen, fortzusetzen oder ganz zu beenden. Welche Signale es gibt und wie Sie *kill* aufrufen, zeigt der folgende Abschnitt. Anders als *ps* gibt *top* eine ständig aktualisierte Übersicht der Prozesse aus, die am meisten CPU-Zeit verbrauchen. Wenn Sie die Auslastung eine Weile mit *top* beobachten, erhalten Sie eher einen Überblick über die Vorgänge als beim einmaligen Aufruf von *ps*.

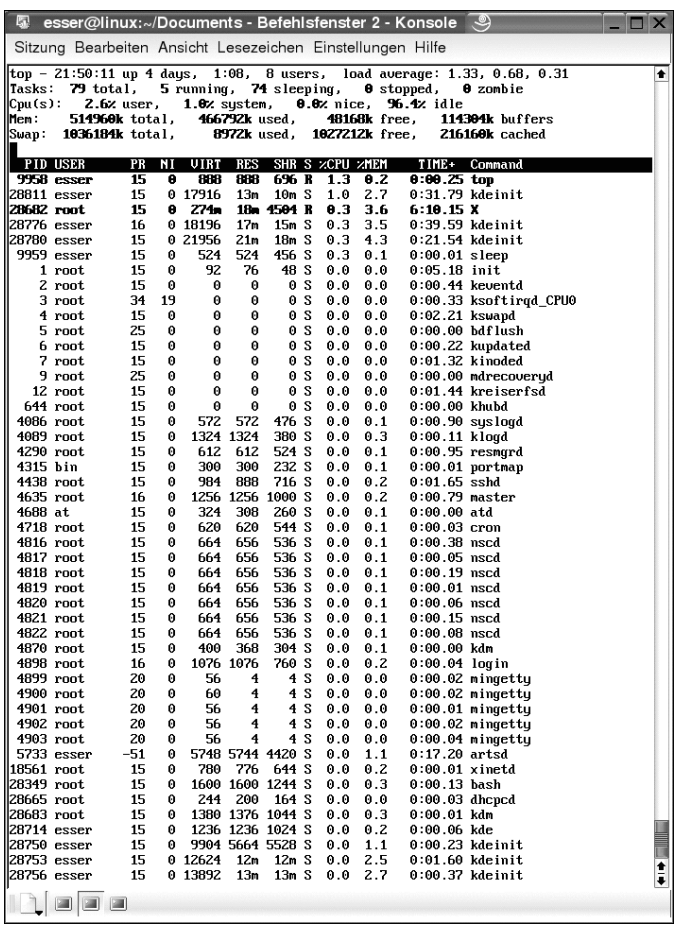

Abb. 4.7: Top *aktualisiert alle paar Sekunden die Anzeige der Prozesse, sortiert nach erzeugter CPU-Last.*

Signale

Prozesse können miteinander über Signale kommunizieren. Einige der Signale haben dabei eine besondere Bedeutung:

- SIGHUP (1): Das "Hangup"-Signal wird von einem Prozess an alle Kinderprozesse geschickt, wenn er beendet wird: Die übliche Wirkung ist, dass auch alle Kinderprozesse sofort beendet werden. Klassisch ist das Beenden einer Shell durch Schließen des Konsolenfensters: Wurden aus dieser Shell heraus andere X-Window-Anwendungen gestartet, so werden diese beim Schließen der Konsole ebenfalls geschlossen. Einige Programme wehren sich gegen diesen Zwangstod und ignorieren das Hangup-Signal. Das kann man auch selbst bewirken, indem man ein Programm über "nohup programmname &" startet: *Nohup* (No hangup) fängt SIGHUP ab und ignoriert es.
- SIGKILL (9): Das Kill-Signal wird verwendet, um einen Prozess abrupt und sofort zu beenden. Im Gegensatz zu SIGTERM (15) hat er keine Chance, noch Dateien zu schließen oder den Speicherinhalt zu sichern. Mit dem Kill-Signal werden typischerweise Prozesse abgeschlossen, die abgestürzt sind und auf nichts mehr reagieren.
- SIGSEGV (11): Das Segmentation-Violation-Signal (Speicherverletzung) wird vom System an einen Prozess geschickt, der versucht, außerhalb des für ihn freigegebenen Speicherbereichs zu arbeiten. Linux lässt so etwas nicht zu und beendet daher auf diese Weise jeden Prozess, der diesen Fehler macht.
- SIGTERM (15): Das Terminate-Signal fordert einen Prozess auf, sich zu beenden. Er wird dann in der Regel dafür sorgen, dass der Programmabbruch geregelt vor sich geht, also etwa offene Dateien geschlossen werden. Einige grafische Anwendungen reagieren auf SIGTERM sogar mit einem Hinweis-Fenster, nach dessen Bestätigung sie sich beenden. Soll ein Programm von der Konsole aus abgeschossen werden, ist SIGTERM immer das Signal, mit dem dies zuerst versucht werden sollte. Nur wenn der Prozess auf SIGTERM nicht reagiert, wird SIGKILL (9) eingesetzt.
- SIGCONT (18): Über das Continue-Signal wird ein Prozess aufgefordert, mit seiner Arbeit fortzufahren, nachdem er mit dem Signal SIGSTOP (19) vorübergehend angehalten wurde.
- SIGSTOP (19): Dieses Signal hält einen Prozess vorübergehend an. Er kann später mit SIGCONT (18) fortgesetzt werden.

Eine kurze Übersicht aller unter Linux eingesetzten Signale hält die Man-Page zu *signal(7)* bereit, die mit dem Befehl

➠ man 7 signal

aufgerufen werden kann. Die Handbuchnummer 7 ist wichtig, da es noch einen anderen Eintrag mit Namen "signal" gibt.

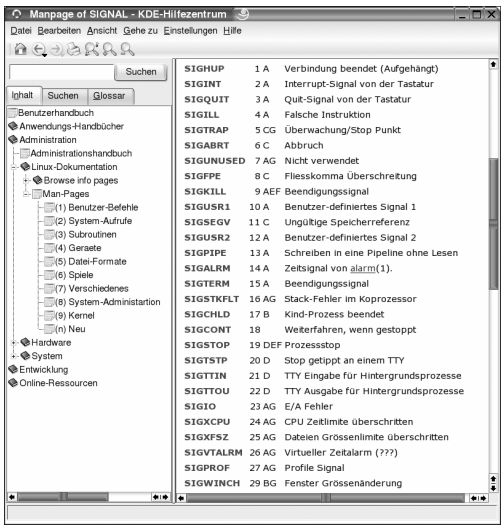

Abb. 4.8: Die Man-Page zu signal(7) *enthält eine Auflistung aller Signale mit Kurzbeschreibung (hier ein im KDE-Hilfezentrum angezeigter Ausschnitt).*

Will man ein Signal an einen Prozess schicken, muss man seine Prozessnummer (englisch: *Process ID*, PID) kennen. Diese steht in der Spalte *PID* der *ps*-Ausgabe. Der einfachste Aufruf von *kill* hat die Form "kill PID" und sendet ihm das Signal SIGTERM (15), also beispielsweise

- kill 3443

um einen Prozess mit der PID 3443 zu beenden. Soll ein anderes Signal verschickt werden, etwa SIGKILL (9), um den Prozess sofort zu beenden, so wird das Signal mit einem vorangestellten Minus als Option angegeben, zum Beispiel:

- kill -9 3443

Auch alle oben erwähnten Signale können so verwendet werden. Testen Sie das doch einmal mit den Signalen SIGSTOP und SIGCONT: Starten Sie probeweise den KDE-Editor Kate und finden Sie auf einer Konsole dessen Prozess-ID heraus. Natürlich können Sie, wie oben beschrieben, die Ausgabe von "ps auxw" untersuchen, einfacher geht es aber mit dem Befehl

- pidof kate

Ist die gefundene Prozess-ID beispielsweise 4321, schicken Sie ihm das Signal SIGSTOP (19):

- kill -19 4321

Versuchen Sie nun, im Editor-Fenster etwas einzugeben: Es wird Ihnen nicht gelingen. Ziehen Sie ein anderes Fenster über das Kate-Fenster hinweg, entstehen hässliche Artefakte (siehe nachfolgende Abbildung): Selbst die Aktualisierung des Fensterinhalts funktioniert nicht mehr, denn der Prozess ist deaktiviert.

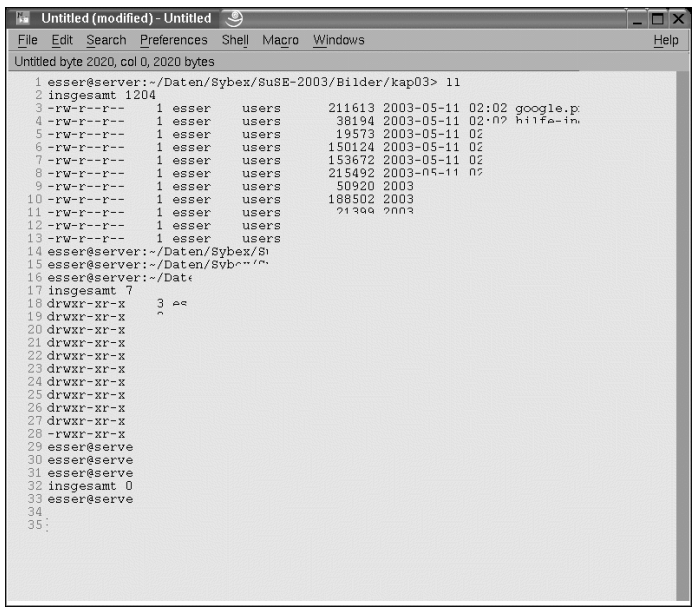

Abb. 4.9: Zieht man ein Fenster über ein anderes hinweg, dessen Prozess deaktiviert wurde, kann auch der Fensterinhalt nicht aktualisiert werden: Es entstehen Artefakte.

Wenn Sie in die Prozesstabelle schauen, finden Sie als Statusinformation ein "T":

```
esser@pentium4:~> ps auxw|grep kate
esser    4321  0.3  3.0  25256  15636 pts/6    T    13:59   0:01 kate
esser    4383  0.0  0.1   1764   688   pts/0    S    14:06   0:00 grep kate
esser@pentium4:~> _
```

Dieser Buchstabe zeigt an, dass der Prozess angehalten wurde. Wecken Sie den Prozess nun wieder auf; das geht mit dem Signal SIGCONT (18):

```
kill -18 4321
```

Sofort wird der Fensterinhalt wiederhergestellt, und es erscheinen sogar die Zeichen, die Sie eingegeben haben, während der Prozess inaktiv war: Es geht nichts verloren. Ein erneuter Blick in die Prozesstabelle bestätigt, dass Kate nun wieder aktiv ist:

```
esser@pentium4:~> ps auxw|grep kate
esser    4321  0.2  3.0   25308  15692   pts/6   S    13:59   0:01 kate
esser    4387  0.0  0.1    1764    688     pts/0   S    14:10   0:00 grep kate
esser@pentium4:~> _
```

Anstelle des "T" erscheint nun ein "S" in der Statusspalte – "S" steht für *sleep* (schlafend), was heißt, dass Kate im Augenblick des *ps*-Aufrufs nicht beschäftigt war – kein Wunder, schließlich arbeiten Sie ja gerade in der Konsole. Wie Sie gesehen haben, kann man das Verhalten eines Prozesses beeinflussen, indem man zunächst mit *ps* oder *pidof* nach der Prozess-ID sucht und dann *kill* aufruft. Schneller geht es mit dem Befehl *killall*, der die Funktion von *pidof* und *kill* integriert: Seine Syntax ist "killall-Signal Prozessname"; so würde beispielsweise

```
killall -9 kate
```

das Signal SIGKILL (9) an jeden Prozess senden, der *kate* heißt. Doch Vorsicht: Sie sollten diesen Befehl nur dann einsetzen, wenn Sie sicher sind, dass nicht versehentlich weitere Prozesse getroffen werden. Übrigens kann jeder Anwender nur seine eigenen Prozesse auf diese Weise bearbeiten: Sind Sie unter Ihrem normalen Account angemeldet, können Sie beispielsweise keine Prozesse beenden, die von *root* oder einem anderen Benutzer auf der gleichen Maschine gestartet wurden. Nur der Administrator *root* hat das Recht, jeden Prozess zu jedem Zeitpunkt zu beenden, anzuhalten etc.

Prozesspriorität

Im Grunde sind alle Prozesse gleich. Das bedeutet: Wenn Sie zwei Prozesse starten, können Sie davon ausgehen, dass diese ähnlich viel Rechenzeit erhalten werden. Führen beide die gleiche Aufgabe durch, sollten sie also ungefähr zur selben Zeit fertig werden. Durch Vergabe eines Prioritätswertes (bei Linux *Nice-Level* genannt) können Sie aber dem Betriebssystem mitteilen, dass einige Prozesse nicht so wichtig sind: Dazu wird der Nice-Level erhöht. Die Bedeutung des Namens wird schnell klar: *Nice* heißt freundlich, nett – und ein Programm, das freiwillig auf Rechenzeit zugunsten anderer Programme verzichtet, kann sicher als nett angesehen werden. Der Nice-Level eines Prozesses wird beim Programmstart festgelegt. Jedes Programm, das ohne besondere Vorkehrungen aufgerufen wird, hat einen Nice-Wert von 0 – das ist Standard. Wollen Sie nun Ihr Programm zu Freundlichkeit überreden, stellen Sie dem Befehl einfach "nice -19" voran. Aus einem Befehl

```
tar cf /dev/tape /home
```

der ein Backup auf ein Streamer-Band schreibt, würde dann der Aufruf

```
nice -19 tar cf /dev/tape /home
```

Dabei ist 19 schon die Grenze der Freundlichkeit; ein höherer Wert als 19 ist nicht möglich. Eine alternative Schreibweise des Befehls ist

- nice -n 19 tar cf /dev/tape /home

– hier wird der Nice-Wert über die Option "-n" übergeben. Dass es zwei Schreibweisen gibt, hat einen guten Grund: Für den Administrator *root* gelten wieder einmal besondere Regeln, er darf sogar negative Nice-Levels vergeben und damit einen Prozess besonders unfreundlich machen, was sein Verlangen nach Rechenzeit angeht. Das kann zum Beispiel dann sinnvoll sein, wenn Sie auf einem stark belasteten System eine CD brennen und Buffer Underruns verhindern wollen. In diesem Fall würden Sie als *root* den Brennbefehl

- cdrecord -v dev=0,4,0 speed=8 /tmp/suse9.0-1.iso

in der Form

- nice -n -20 cdrecord -v dev=0,4,0 speed=8 /tmp/suse9.0-1.iso

aufrufen: Dadurch erhält er den Nice-Level -20, was die unterste Grenze für diesen Wert darstellt. Probieren Sie das ruhig einmal aus; im Ergebnis kann sich das Gesamtsystem dann etwas ruckelig verhalten, denn oberstes Ziel des Kernels ist jetzt, den *cdrecord*-Prozess optimal mit Rechenzeit zu versorgen.

Nice-Level ändern

Auch wenn ein Prozess bereits läuft, können Sie seinen Nice-Level noch ändern – dazu wird das Programm *renice* verwendet. Es wird einfach mit dem neuen Nice-Level und der Prozess-ID aufgerufen. Dabei dürfen normale Anwender den Nice-Level stets nur heraufsetzen; *root* darf auch in die andere Richtung gehen. Selbst ein Prozess, der von einem Anwender mit hohem Nice-Level gestartet wurde, kann nicht wieder auf 0 zurückgesetzt werden. Das folgende Beispiel zeigt den Aufruf eines Programms, das Hochsetzen des Nice-Levels auf 10, den scheiternden Versuch, dies rückgängig zu machen und schließlich den Erfolg derselben Aktion mit *root*-Rechten. Der umständliche *ps*-Aufruf mit angehängtem *cut* dient nur dazu, einen Teil der *ps*-Ausgabe abzuschneiden, damit es im Druck dieses Buches ordentlich aussieht. Wenn Sie das nachvollziehen möchten, lassen Sie den Teil ab dem senkrechten Balken einfach weg. Durch die *ps*-Option "-l" wird übrigens der Nice-Level angezeigt, der nicht zum Standardumfang der Ausgabe gehört.

- esser@pentium4:~> korn &
- [1] 9032
- esser@pentium4:~> ps | grep korn
- 9032 pts/6 00:00:00 korn

```
esser@pentium4:~> ps -l -p 9032 | cut -c 14-35,68-
 PID  PPID  C PRI  NI  CMD
 9032     1    0   80    0    korn
esser@pentium4:~> renice 10 9032
9032: Alte Priorität: 0, neue Priorität: 10
esser@pentium4:~> ps -l -p 9032 | cut -c 14-35,68-
 PID  PPID  C PRI  NI  CMD
9032     1    0   80   10   korn
korn
esser@pentium4:~> renice 0 9032
renice: 9032: setpriority: Keine Berechtigung
esser@pentium4:~> su
Password: ....
pentium4:/home/esser # renice 0 9032
9032: Alte Priorität: 10, neue Priorität: 0
pentium4:/home/esser # ps -l -p 9032 | cut -c 14-35,68-
 PID  PPID  C   PRI   NI   CMD
9032     1    0    80     0    korn
pentium4:/home/esser # _
```

Den Befehl *reniceall*, der bei mehreren Prozessen gleichen Namens den Nice-Faktor ändert und damit analog zu *kill* / *killall* eine Erweiterung von *renice* darstellt, müssen Sie selbst schreiben. Wenn Sie an einem *reniceall*-Befehl interessiert sind, legen Sie einfach (als *root*) die folgende Datei (mit Namen *reniceall*) im Verzeichnis */usr/bin/* an:

```
#!/bin/bash
for pid in `pidof $2`; do renice $1 $pid; done
```

und geben Sie ihr mit "chmod a+x /usr/bin/reniceall" Ausführrechte – schon haben Sie ein eigenes reniceall, wie der folgende Aufruf zeigt:

```
pentium4:/home/esser # reniceall 1 bash
9128: Alte Priorität: 0, neue Priorität: 1
9040: Alte Priorität: 0, neue Priorität: 1
3001: Alte Priorität: 0, neue Priorität: 1
2970: Alte Priorität: 0, neue Priorität: 1
2188: Alte Priorität: 0, neue Priorität: 1
2167: Alte Priorität: 0, neue Priorität: 1
2161: Alte Priorität: 0, neue Priorität: 1
2159: Alte Priorität: 0, neue Priorität: 1
pentium4:/home/esser # _
```

Wer bei Google nach "reniceall" sucht, findet die *reniceall*-Projektseite des Autors, die eine leicht verbesserte Version dieses Programms enthält.