

15. Prozessverwaltung



Dieses Kapitel beschreibt, wie Linux mit Prozessen umgeht. Im Verlauf dieses Kapitels lernen Sie,

- » welche Möglichkeiten es gibt, Programme zu starten und (zur Not auch gewaltsam) wieder zu beenden,
- » wie Sie ein Programm als gewöhnlicher Benutzer ausführen, als wären Sie root,
- » was Dämonen sind und
- » wie Sie Programme zu bestimmten Zeiten automatisch starten können.

15.1 Prozesse starten, verwalten und stoppen

In diesem Kapitel ist überwiegend von Prozessen die Rede. Aber fast immer könnten Sie das Wort »Prozess« durch »Programm« oder »Kommando« oder auch durch »Task« ersetzen. (Linux-intern gibt es übrigens keine Unterscheidung zwischen einem Programm oder einem Kommando. Umgangssprachlich werden textorientierte Programme wie `ls` aber oft als Kommandos bezeichnet.)

Programme,
Kommandos,
Prozesse, Tasks

Genau genommen handelt es sich bei einem Programm bzw. einem Kommando um eine ausführbare Datei. Eine Programmdatei unterscheidet sich von anderen Dateien nur dadurch, dass das Zugriffsbit `x` gesetzt ist (siehe auch Seite 344).

Bei den beiden folgenden Dateien handelt es sich bei `server.tex` um eine Datendatei und bei `si chere` um ein Programm. (Genau genommen handelt es sich um ein einfaches Shell-Script, das ein Backup durchführt. Die Programmierung von Shell-Scripts wird in Kapitel 20.8 beschrieben.) In diesem Fall sind beides Textdateien, aber nur `si chere` ist ausführbar, weil dort die Zugriffsbits `x` gesetzt sind.

```
user$ ls -l s*
-rw-r--r--  1 kofler  users      180383 Apr 24 10:20 server.tex
-rwxr-xr-x  1 kofler  users         222 Jun  6 10:58 si chere
```

Erst durch den Start einer gleichsam leblosen Programmdatei wird diese zu einem lebendigen Prozess (Synonym »Task«), der vom Linux-Kernel verwaltet wird. So gesehen müsste die Überschrift dieses Abschnitts eigentlich lauten: *Programme und Kommandos starten, Prozesse verwalten und stoppen.*

***.exe-Dateien** Hin und wieder taucht die Frage auf, wo denn unter Linux die *.exe-Dateien sind. Bis vor einigen Jahren hieß die richtige Antwort: Es gibt keine *.exe-Dateien. Ausführbare Programme werden unter Linux dadurch gekennzeichnet, dass das Zugriffsbit x gesetzt ist. Damit wird die Datei als ausführbar (*executable*) gekennzeichnet. Die von Windows bekannte Dateikennung *.exe ist somit überflüssig.

Mittlerweile ist diese Antwort insofern nicht mehr ganz richtig, als es auf vielen Linux-Systemen tatsächlich vereinzelte *.exe-Dateien gibt. Dabei handelt es sich um Programme, die in der Programmiersprache C# entwickelt wurden und die zur Ausführung auf die Mono-Bibliothek zurückgreifen. Die Mono-Bibliothek ist wiederum eine Open-Source-Implementierung des .NET-Frameworks von Microsoft. Weitere Informationen zu Mono finden Sie auf Seite 571.

Programme starten

Programmstart unter X Unter X starten Sie Programme im Regelfall über ein Menü oder durch das Anklicken eines Icons. KDE und Gnome bieten mit dem Tastenkürzel `Alt+F2` eine weitere Möglichkeit, Programme rasch zu starten.

Textkonsole, Shell-Fenster Alternativ können Sie Programme auch in einem Shell-Fenster (z. B. xterm, konsole etc.) oder in einer Textkonsole starten. Dazu geben Sie einfach den Namen des Programms ein und drücken `↵`. Gerade Linux-Profis wählen oft diesen Weg, weil das Eintippen der paar Buchstaben meist schneller geht als die Suche des Programms in verzweigten Menüs.

Normalerweise reicht es aus, wenn Sie einfach den Namen des Programms angeben. Der Shell-Interpreter sucht das Programm dann in allen Verzeichnissen, die in der Umgebungsvariable PATH angegeben sind. Die folgenden Zeilen zeigen eine typische Einstellung dieser Variable:

```
user$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Wenn Sie ein Programm starten möchten, das sich in keinem dieser Verzeichnisse befindet, müssen Sie den vollständigen Pfad angeben. Das gilt auch für Programme im gerade aktuellen Verzeichnis! Hier wird der Pfad einfach durch einen Punkt angegeben (also beispielsweise `./meinprogramm`).

Vordergrund- und Hintergrundprozesse

Wenn Sie Programme unter X per Menü starten, laufen diese selbstverständlich als sogenannte Hintergrundprozesse, ohne sich gegenseitig zu behindern. Sie können also weitere Programme starten, ohne auf das Ende der bisher gestarteten Programme warten zu müssen.

Ganz anders ist das Verhalten, wenn Sie ein Programm in einer Textkonsole bzw. einem Shell-Fenster ausführen. Das Programm wird als Vordergrundprozess gestartet. Bevor Sie das nächste Kommando eingeben können, müssen Sie auf das Ende des zuletzt gestarteten Programms warten.

Aber auch in Textkonsolen oder Shell-Fenstern können Sie Programme im Hintergrund starten. Dazu geben Sie einfach am Ende des Kommandos das Zeichen & an:

```
user$ emacs &
```

Wenn Sie & vergessen haben, können Sie das Programm auch nachträglich in einen Hintergrundprozess umwandeln. Unterbrechen Sie die Programmausführung mit `[Strg]+[Z]`, und setzen Sie das Programm mit `bg` fort:

```
user$ emacs
<Strg>+<Z>
[1]+ Stopped emacs
user$ bg
[1]+ emacs &
```

Wenn Sie statt `bg` das Kommando `fg` verwenden, wird das Programm als Vordergrundprozess fortgesetzt.

Bei manchen Kommandos stören bei der Hintergrundauführung diverse Textausgaben. Diese können Sie aber leicht unterdrücken, indem Sie sie nach `/dev/null` umleiten. Beispielsweise wird durch das folgende Kommando ein USB-Stick im Hintergrund formatiert:

```
root# mkfs.ext3 /dev/sdc > /dev/null &
```

Liste aller laufenden Prozesse (`ps`, `top`)

Eine Liste der zurzeit laufenden Prozesse können Sie sehr einfach mit `ps` erzeugen. Ohne Optionen zeigt `ps` nur Ihre eigenen Prozesse an – und nur solche, die aus Textkonsolen bzw. Shell-Fenstern gestartet wurden. `ps` kann durch zahlreiche Optionen gesteuert werden, von denen die allerwichtigsten auf Seite 1151 beschrieben sind. Im folgenden Beispiel wurde die Liste aller Prozesse aus Platzgründen stark gekürzt:

```
user$ ps ax
  PID TTY          STAT       TIME COMMAND
    1 ?            S           0:00 init [5]
    2 ?            SN          0:00 [ksoftirqd/0]
    3 ?            S           0:00 [watchdog/0]
    4 ?            S<          0:00 [events/0]
  ...
3064 pts/2      S           0:39 emacs command.tex
3151 pts/2      S+          1:23 /bin/sh ./lvauto
3735 pts/4      S           0:00 su -l
3740 pts/4      S+          0:00 -bash
```

Praktischer als `ps` ist meist `top`: Dieses Kommando ordnet die Prozesse danach, wie sehr sie die CPU belasten, und zeigt die gerade aktiven Prozesse zuerst an. Das Programm gibt auch einen Überblick über den aktuellen Speicherbedarf etc. Die Prozessliste wird alle paar Sekunden aktualisiert, bis das Programm mit `[Q]` beendet wird. Die folgenden Zeilen zeigen ein System im Leerlauf:

```
top - 14:45:38 up 6:56, 3 users, load average: 0.73, 0.55, 0.32
Tasks: 187 total, 1 running, 186 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.7%us, 1.1%sy, 0.2%ni, 95.3%id, 0.6%wa, 0.1%hi, 0.1%si, 0.0%st
Mem: 6006012k total, 3043980k used, 2962032k free, 213476k buffers
Swap: 3903480k total, 0k used, 3903480k free, 1269644k cached
```

```

PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM   TIME+  COMMAND
6645 kofler    20   0  935m  628m  51m  S   4  10.7   5:06.99 VirtualBox
4118 kofler    20   0  364m   47m  18m  S   2   0.8   3:31.16 compiz.real
12602 kofler    20   0  391m   19m  13m  S   2   0.3   0:39.50 mplayer
   1 root      20   0  4104   924   632  S   0   0.0   0:00.83 init
   2 root      15  -5     0     0     0  S   0   0.0   0:00.00 kthreadd
...

```

Der Wert in der PID-Spalte gibt die Prozessnummer an. Wenn Sie diese Nummer kennen, können Sie außer Kontrolle geratene Programme oder Hintergrundprozesse mit dem Kommando `kill` gewaltsam stoppen.

Prozesse können verschiedene Zustände annehmen. Die zwei häufigsten Zustände sind R (*running*) und S (*sleeping*, das Programm hat also gerade nichts zu tun und wartet auf Eingaben). Programme können auch vorübergehend unterbrochen werden und weisen dann den Zustand T (*stopped*) auf.

`top` nimmt auch interaktiv Kommandos entgegen. Damit können Sie Prozesse stoppen (`[K]`, `kill`) oder ihre Priorität verändern (`[R]`, `renice`).

Grafische Varianten zu `top`

Zum textbasierten Kommando `top` gibt es natürlich auch grafische Alternativen, z. B. `ksysguard` (KDE) oder `gnome-system-monitor` (Gnome).

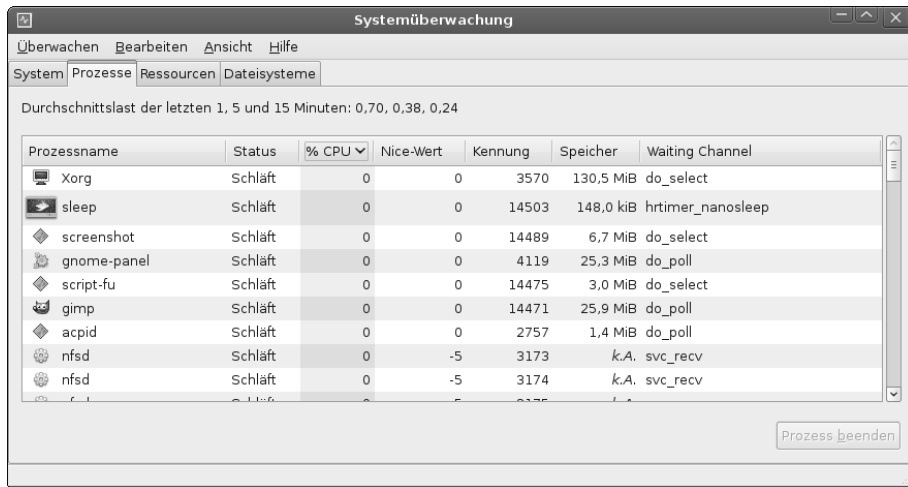


Abbildung 15.1: Prozessübersicht mit `gnome-system-monitor`

Prozessnummer ermitteln

Wenn Sie den Programmnamen wissen und die dazugehörige Prozessnummer (PID) ermitteln möchten, hilft `pidof`. Wenn es mehrere Prozesse mit dem gleichen Namen gibt, liefert `pidof` eine ganze Liste von Nummern:

```

root# pidof nscd
1777 1776 1775 1774 1765 1763 1753

```

Manchmal ist es auch nützlich festzustellen, welche Programme auf eine bestimmte Datei oder ein Verzeichnis zugreifen. Die entsprechenden Prozessnummern können Sie mit `fuser` feststellen. Ein Verzeichnis gilt auch dann als benutzt, wenn darin ein Programm gestartet wurde. Das folgende Kommando zeigt, dass die Shell `bash` das Verzeichnis `/media/dvd` nutzt:

```

root# fuser -v /media/dvd
          USER      PID ACCESS COMMAND
/media/dvd  kofler    2183 ..c..  bash
          root      Kernel mount  /media/dvd

```

Beachten Sie, dass ein Dateizugriff nur festgestellt werden kann, wenn das Programm die Datei wirklich geöffnet hat. Das ist bei einem Texteditor beispielsweise nicht der Fall. (Der Editor hat die Datei zum Laden geöffnet, dann aber wieder geschlossen. Zum Speichern öffnet er die Datei wiederum nur für kurze Zeit.)

Manche Hintergrundprozesse speichern im Verzeichnis `/var/run` eine PID-Datei (z. B. `/var/run/httpd.pid`). Diese Datei enthält in der ersten Zeile die Prozessnummer; weitere Zeilen können Zusatzinformationen wie die Netzwerkschnittstelle enthalten. PID-Dateien ermöglichen das gezielte Beenden eines bestimmten Prozesses durch ein Init-V-Script (siehe Seite 746), und zwar auch dann, wenn es mehrere gleichnamige Prozesse gibt.

PID-Dateien

Prozeshierarchie

Intern wird mit jedem Prozess auch die PID-Nummer des Elternprozesses gespeichert. Diese Information ermöglicht die Darstellung eines Prozessbaums, an dessen Spitze immer der Prozess `init` steht. `init` ist das erste Programm, das unmittelbar nach dem Laden des Kernels gestartet wird (siehe Seite 741).

Die Darstellung der Prozesshierarchie gelingt am einfachsten mit dem Kommando `pstree`. Mit der Option `-h` werden die Elternprozesse zum gerade laufenden Prozess fett hervorgehoben. In Abbildung 15.2 wurde `pstree` von einer `bash`-Shell in einem Konsole-Fenster ausgeführt.

Prozesse gewaltsam beenden (`kill`, `xkill`)

Normalerweise endet ein Prozess mit dem Programmende. Aber leider kommt es auch unter Linux vor, dass Programme Fehler enthalten, sich nicht mehr stoppen lassen und womöglich immer mehr Speicher und CPU-Kapazität beanspruchen. In solchen Fällen muss der Prozess gewaltsam beendet werden.

Bei textorientierten Kommandos hilft in den meisten Fällen einfach `Strg+C`. Damit wird das Programm sofort beendet.

Das Kommando `kill` versendet Signale an einen laufenden Prozess, der durch die PID-Nummer spezifiziert wird. (Diese Nummer können Sie mit `top` oder `ps` ermitteln.) Um ein Programm »höflich« zu beenden, wird das Signal 15 verwendet. (`kill` verwendet dieses Signal per Default.) Hilft das nicht, muss das Signal 9 eingesetzt werden (hier für den Prozess 2725):

kill

```
user$ kill -9 2725
```

`kill` kann nur für eigene Prozesse verwendet werden. Nur `root` darf auch fremde Prozesse beenden.

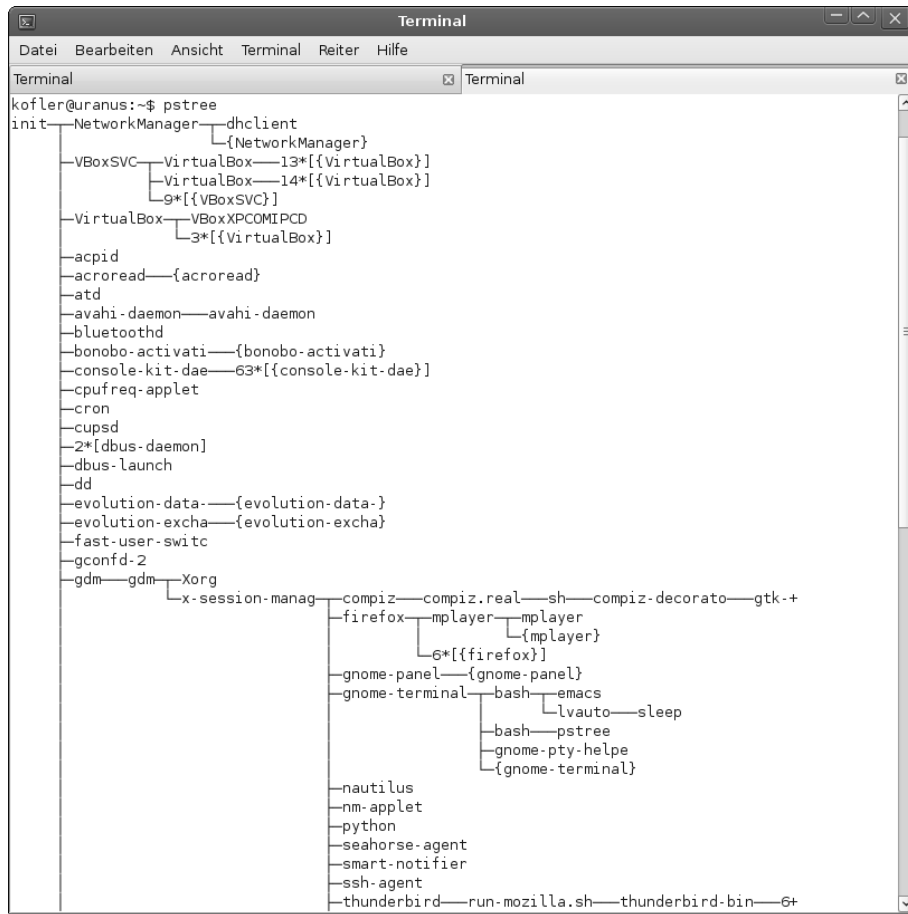


Abbildung 15.2:
Prozessüber-
sicht mit pstree

- top** Auch mit top können Sie Prozesse beenden: Geben Sie einfach **[K]** und anschließend die Prozessnummer und das gewünschte Signal ein!
- killall** killall ist insofern bequemer, als keine Prozessnummer, sondern der Programmname angegeben werden kann. Allerdings werden nun *alle* Prozesse dieses Namens beendet.


```
root# killall -9 firefox
```
- xkill** Unter X geht es noch bequemer. Starten Sie in einem Shell-Fenster xkill, und klicken Sie einfach das Fenster des Programms an, das Sie beenden wollen. An den Prozess wird wiederum das Signal 9 gesandt.

Unter KDE können Sie xkill auch mit **[Strg] + [Alt] + [Esc]** starten. Wenn das irrtümlich passiert, können Sie xkill mit **[Esc]** abbrechen.

Manchmal wird durch `xkill` zwar das Fenster geschlossen, der Prozess oder Teile davon laufen aber weiter. Vergewissern Sie sich mit `top` bzw. mit `ps`, dass das Programm wirklich beendet ist. Zur Not müssen Sie mit `kill -9 n` nachhelfen.

Hartnäckige Fälle

Wirklich unangenehm wird es, wenn ein X-Programm nicht nur hängen bleibt, sondern dabei auch den Tastatur- und Maus-Focus an sich reißt oder X sonstwie blockiert. Der Rechner reagiert dann auf (fast) keine Eingaben mehr.

Blockierte Tastatur oder Maus

In solchen Fällen hilft manchmal die magische Tastenkombination `Strg+Alt+F1` weiter, mit der der Wechsel in die erste Textkonsole erfolgt. Dort können Sie sich einloggen und das betreffende Programm mit `top` suchen und beenden.

Wenn die Tastatur vollständig blockiert ist, besteht immer noch die Möglichkeit, sich über ein Netzwerk via `ssh` einzuloggen und `kill` auf diese Weise auszuführen. Diese Variante ist natürlich nur möglich, wenn Sie in einem lokalen Netz arbeiten und auf dem lokalen Rechner `sshd` läuft.

Sollte X selbst nach dem Ende des Programms blockiert bleiben, können Sie versuchen, auch X gewaltsam zu beenden bzw. schließlich `shutdown` auszuführen. All diese Varianten sind besser als das Drücken der Reset-Taste, was zu Datenverlusten führen kann!

Bei Programmen, die über eine Shell gestartet werden (etwa bei allen Kommandos, die in einem Shell-Fenster ausgeführt werden), können Sie mit dem Shell-Kommando `ulimit` den maximalen Speicherverbrauch, die maximale Größe erzeugter Dateien etc. begrenzen. `ulimit` wird üblicherweise in `/etc/profile` eingestellt.

Prozessgröße beschränken

Verteilung der Rechenzeit (`nice`, `renice`)

Im alltäglichen Betrieb von Linux ist die Rechenkapazität meist mehr als ausreichend, um alle laufenden Prozesse ohne Verzögerungen auszuführen. Wenn Linux gerade mit rechenaufwendigen Prozessen beschäftigt ist – z. B. während des Kompilierens eines umfangreichen Programms –, versucht es, die zur Verfügung stehende Rechenzeit gerecht an alle Prozesse zu verteilen.

In manchen Fällen kann es sinnvoll sein, einem Prozess bewusst mehr oder weniger Rechenzeit zuzuteilen. Dazu dient das Kommando `nice`, mit dem Programme mit reduzierter oder erhöhter Priorität gestartet werden können. Dazu wird an `nice` die gewünschte Priorität übergeben, die von 19 (ganz niedrig) bis -20 (ganz hoch) reicht. Per Default werden Prozesse mit der Priorität 0 gestartet. Im folgenden Beispiel wird ein Backup-Programm mit niedrigerer Priorität gestartet, damit es keine anderen Prozesse beeinträchtigt. (Es ist ja egal, ob das Backup zwei oder drei Sekunden länger dauert.)

```
user$ nice -n 10 sichere
```

Mit `renice` kann auch die Priorität von bereits laufenden Prozessen geändert werden. Als Parameter muss die Prozess-ID angegeben werden, die vorher mit `top` oder `ps` ermittelt wurde. Details zu `renice` finden Sie auf der `man`-Seite. Auch `top` (Kommando `R`) ist in der Lage, interaktiv die Priorität eines Prozesses zu verändern. Allerdings kann nur `root` Programme mit einer höheren Priorität als 0 starten bzw. die Priorität eines bereits laufenden Prozesses erhöhen.

**Threading
(NPTL)**

Linux kann nicht nur mehrere Prozesse parallel ausführen, sondern unterscheidet auch innerhalb eines Prozesses zwischen Teilprozessen (Threads). Vor allem Server-Anwendungen verteilen ihre Funktionen oft auf mehrere Threads. Das steigert die Performance insbesondere dann, wenn mehrere CPUs oder mehrere CPU-Cores zur Verfügung stehen. Bei der Verwaltung und Kommunikation der Threads helfen Kernelfunktionen. Seit Kernel 2.6 unterstützt die *Native POSIX Thread Library* Threads gemäß dem POSIX-Standard.

Ein- und Ausgabeumleitung, Pipe

Fast alle textorientierten Programme (Kommandos) erwarten Eingaben über den sogenannten Standardeingabekanal (per Default die Tastatur) und senden Ausgaben an den Standardausgabekanal (der Text wird in der Konsole bzw. im Shell-Fenster angezeigt). Sowohl die Ein- als auch die Ausgabe lassen sich umleiten, wodurch sich viele Möglichkeiten ergeben. Beispielsweise speichert das folgende Kommando die Liste aller Dateien des Verzeichnisses `xy` in der Datei `z`:

```
user$ ls xy > z
```

Durch sogenannte Pipes kann die Ausgabe eines Kommandos als Eingabe für das nächste Kommando verwendet werden. Beim folgenden Beispiel filtert `grep` aus der Liste aller installierten Pakete die heraus, die die Zeichenkette »mysql« in beliebiger Groß- und Kleinschreibung enthalten. `sort` sortiert diese Liste schließlich.

Mit anderen Worten: Die Ausgaben des Kommandos `rpm` werden dank des Zeichens `|` an das zweite Kommando `grep` weitergeleitet, dessen Ausgaben mit dem zweiten Zeichen `|` an `sort`. Mehr Details und Beispiele zur Ein- und Ausgabeumleitung sowie zur Verwendung von Pipes finden Sie in Kapitel 20 ab Seite 456.

```
user$ rpm -qa | grep -i mysql | sort
mysql-5.1.32-1.fc11.i586
mysql-connector-java-3.1.12-7.fc11.i586
...
```

**15.2 Prozesse unter einer anderen Identität ausführen
(su)**

Bei der Programmausführung durch gewöhnliche Benutzer gibt es zwei Einschränkungen:

- » Gewöhnliche Benutzer dürfen nur die Prozesse ausführen, bei denen die Zugriffsrechte (Besitzer, Gruppe, r- und x-Zugriffsbits) dies zulassen. Bei gewöhnlichen Programmen ist das keine Einschränkung. Es gibt aber beispielsweise im Verzeichnis `/usr/sbin` einige Kommandos zur Systemadministration, die nur von `root` gestartet werden können.
- » Prozesse gehören gleichsam dem Benutzer, der sie gestartet hat. Das bedeutet, dass der Prozess auf die gleichen Dateien zugreifen darf wie der Benutzer. (Umgekehrt formuliert: Dateien, die Sie als Benutzer nicht verändern dürfen, dürfen auch die von Ihnen gestarteten Programme nicht verändern.) Vom Prozess neu erzeugte Dateien gehören ebenfalls dem Benutzer, der das Programm gestartet hat (siehe auch Seite 348).

Als gewöhnlicher Benutzer können Sie aus diesen Gründen viele administrative Arbeiten nicht durchführen. Die offensichtlich einfachste Lösung besteht darin, sich als root einzuloggen. Es wurde in diesem Buch aber schon mehrfach darauf hingewiesen, dass es keine gute Idee ist, ständig als root zu arbeiten: Die Gefahr ist einfach zu groß, dass Sie irrtümlich Schaden anrichten. Aus diesem Grund sperren manche Distributionen den root-Login vollständig. So ist es unter Ubuntu unmöglich, sich als root anzumelden.

Dieser Abschnitt beschreibt, wie Sie mit su bzw. ssh dennoch administrative Tätigkeiten durchführen können, ohne sich als gewöhnlicher Benutzer auszuloggen. Der nächste Abschnitt zeigt eine alternative Vorgehensweise mit sudo, die sich vor allem unter Ubuntu bewährt hat. Abschnitt 15.4 präsentiert schließlich das Programm PolicyKit, das ganz neue Wege bei der Ausführung von root-Aufgaben geht.

Wie so oft gäbe es mehr zu schreiben, als hier Platz hat. Weitere Informationsquellen sind das Security-HOWTO und das Remote-X-Mini-HOWTO. Beide Dokumente sind zwar schon ziemlich alt, die darin enthaltenen Tipps sind aber noch immer gültig:

<http://www.tldp.org/HOWTO/Security-HOWTO/index.html>

<http://www.tldp.org/HOWTO/Remote-X-Apps.html>

In vielen Fällen geht es nur darum, rasch ein Kommando als root auszuführen – da wäre ein Verlassen von X sehr unkomfortabel. Die einfachste Möglichkeit, innerhalb eines X-Shell-Fensters den Benutzer zu ändern, bietet das Kommando su *name*. Wenn Sie das Kommando nicht als root ausführen, werden Sie nach dem Passwort des jeweiligen Anwenders gefragt. Innerhalb des Shell-Fensters (xterm, konsole) können Sie jetzt Kommandos unter dem geänderten Namen ausführen, bis Sie durch exit oder `[Strg]+[D]` zurück in den Normalmodus wechseln.

Die folgenden Zeilen zeigen, wie ein gewöhnlicher Benutzer sich kurz als root anmeldet, als root eine Festplattenpartition in den Verzeichnisbaum einbindet und sich dann als root wieder ausloggt und normal weiterarbeitet:

```
user$ su -l root
Password: xxx
root# mount -t ext2 /test /dev/hdc7
root# <Strg>+<D>
logout
user$ ls /test
```

Damit su ein vollwertiger Ersatz für einen root-Login ist, müssen Sie die Option -l verwenden! Damit erreichen Sie, dass alle Login-Startdateien (etwa zur korrekten Definition von PATH) eingelesen werden.

Unter KDE verwenden Sie am besten kdesu zum Start von X-Programmen mit Administratorrechten. Das Programm zeigt einen ansprechenden Dialog zur Eingabe des root-Passworts an.

```
user$ kdesu kate /etc/fstab
```

su

kdesu

kdesu funktioniert nur, wenn der Dämon kdesud läuft. Dieser wird üblicherweise beim KDE-Start gestartet. Eine Zusammenfassung der zahlreichen kdesu-Optionen erhalten Sie mit `kdesu -help-all`. Bei einigen Distributionen ist kdesu direkt in das KDE-Menü integriert. Wenn Sie also ein Programm starten, das Administratorrechte beansprucht, erscheint automatisch die kdesu-Login-Box.

gksu Das Gnome-Gegenstück zu kdesu heißt gksu. Es funktioniert ohne einen zusätzlichen Hintergrundprozess. Eine Zusammenfassung der Optionen gibt man `gksu`.

su-to-root Debian (aber nicht Ubuntu!) verwendet zum Start von Administratorwerkzeugen aus dem Gnome- oder KDE-Menü das desktopunabhängige Script `su-to-root`. Das Script ist Teil des `menu`-Pakets. `su-to-root` fasst die wenigen Optionen zusammen.

consolehelper Als Variante zu gksu kommt unter Fedora und Red Hat `consolehelper` zum Einsatz. Dieses Programm bietet ebenfalls einen ansprechenden Passwortdialog, ist aber ganz anders implementiert. Die Grundidee besteht darin, dass die betreffenden Administratorwerkzeuge in das `/usr/sbin` Verzeichnis installiert werden, in dem sie nur für `root` zugänglich sind. Für gewöhnliche Benutzer befindet sich in `/usr/bin` ein symbolischer Link auf `consolehelper`. Das folgende Kommando zeigt eine derartige Konfiguration für `system-config-network` (Netzwerkkonfiguration für Red Hat oder Fedora).

```
user$ ls -l /usr/sbin/system-config-network /usr/bin/system-config-network
-rwxr-xr-x ... root root /usr/sbin/system-config-network
lrwxrwxrwx ... root root /usr/bin/system-config-network -> consolehelper
```

Führt nun `root` das Kommando `system-config-network` aus, wird direkt `/usr/sbin/system-config-network` gestartet. Führt dagegen ein gewöhnlicher Benutzer `system-config-network` aus, wird `consolehelper` gestartet. Wenn der Benutzer das korrekte `root`-Passwort angibt, startet `consolehelper` das gewünschte Programm `system-config-network`.

ssh Bei den meisten Distributionen funktioniert `su` nur für Textkommandos. Das kann mehrere Gründe haben: Erstens ist zum Start eines X-Programms die Umgebungsvariable `DISPLAY` erforderlich. Diese Variable muss den Namen des Rechners enthalten, auf dem das Programm angezeigt werden soll (`export DISPLAY=localhost:0`). Zweitens kann X aus Sicherheitsgründen verbieten, dass fremde Benutzer Programme starten können. (Abhilfe schafft das Kommando `xhost`.) Drittens kann der Netzwerk-Port für die Kommunikation zum X-Server gesperrt sein.

Wenn `kdesu`, `gksu` oder `consolehelper` nicht zur Verfügung stehen, bietet `ssh` die einfachste Lösung all dieser Probleme: Führen Sie das gewünschte Kommando einfach in der folgenden Form aus:

```
user$ ssh -X -l benutzer localhost
```

suid- und guid-Zugriffsbits

Die `suid`- und `guid`-Zugriffsbits stellen eine weitere Möglichkeit dar, bestimmte Programme so zu kennzeichnen, dass jeder sie ausführen kann, als wäre er bzw. sie `root`. Der wesentliche Unterschied zu `sudo` besteht darin, dass die `suid`- und `guid`-Zugriffsbits für *alle* Benutzer gelten (während die Benutzer bei `sudo` in `/etc/sudoers` explizit aufgezählt werden müssen). Weitere Informationen zu den `suid`- und `guid`-Zugriffsbits finden Sie auf Seite 346.

15.3 Prozesse unter einer anderen Identität ausführen (sudo)

sudo verfolgt einen ganz anderen Ansatz als die oben beschriebenen su-Varianten. Das Programm ermöglicht nach entsprechender Konfiguration bestimmten Benutzern die Ausführung bestimmter Programme mit root-Rechten. Zur Sicherheit muss nochmals das *eigene* Passwort (nicht das root-Passwort!) angegeben werden.

sudo führt diese Programme dann so aus, als wären sie von einem anderen Benutzer gestartet worden (Default: root). Damit können einzelne Benutzer administrative Aufgaben übernehmen bzw. systemkritische Kommandos ausführen, ohne dazu das root-Passwort kennen zu müssen. sudo protokolliert alle ausgeführten Kommandos (auch gescheiterte Versuche) üblicherweise in `/var/log/messages`.

sudo merkt sich das Passwort für 15 Minuten. Wenn Sie innerhalb dieser Zeit ein weiteres Kommando mit sudo ausführen, werden Sie nicht neuerlich nach dem Passwort gefragt. (Diese Zeit kann in `/etc/sudoers` mit dem Schlüsselwort `timestamp_timeout` verändert werden.)

Die Konfiguration von sudo erfolgt durch die Datei `/etc/sudoers`. Vereinfacht ausgedrückt, beschreibt die Datei in drei Spalten, welche Benutzer von welchem Rechner aus welche Programme ausführen dürfen. Die folgende Zeile bedeutet, dass die Benutzerin kathrin am Rechner uranus das Kommando `/sbin/fdisk` ausführen darf. (Das Schlüsselwort ALL bedeutet, dass kathrin das Kommando unter jedem beliebigen Account ausführen darf, also als root, als news, als lp etc.)

Konfiguration

```
# in /etc/sudoers
kathrin uranus=(ALL) /sbin/fdisk
```

Wenn der ersten Spalte von sudoers das Zeichen % vorangestellt wird, gilt der Eintrag für alle Mitglieder der angegebenen Gruppe. Diverse weitere Syntaxvarianten beschreibt man `sudoers`.

Aus Sicherheitsgründen sollte `/etc/sudoers` ausschließlich mit `visudo` editiert werden (siehe auch dessen `man`-Seite)! `visudo` führt vor dem Speichern einen Syntaxtest durch und stellt so sicher, dass Sie sich nicht durch eine fehlerhafte `sudoers`-Datei selbst von weiteren Administrationsarbeiten ausschließen. Besonders wichtig ist das bei Distributionen wie Ubuntu, die keinen root-Login vorsehen.

Achtung

Kathrin kann nun `fdisk` folgendermaßen ausführen. Als Passwort muss das Passwort der Benutzerin kathrin angegeben werden. Bei `fdisk` muss der vollständige Pfad angegeben werden, falls sich `fdisk` nicht in einem der `PATH`-Verzeichnisse von kathrin befindet. `fdisk` wird automatisch im Account root ausgeführt. Ein anderer Account kann mit `sudo -u account` gewählt werden.

Anwendung

```
kathrin$ sudo /sbin/fdisk /dev/sda
Password: xxxxxx
```

Es besteht die Möglichkeit, einem bestimmten Benutzer das Ausführen von sudo ohne Passwortangabe zu erlauben. Dazu fügen Sie in `sudoers` eine Zeile nach dem folgenden Muster ein:

sudo ohne
Passwort

```
kofler ALL=(ALL) NOPASSWD: ALL
```

Das ist natürlich ein Sicherheitsrisiko, aber wer oft Administrationsaufgaben ausführen muss, wird die so gewonnene Bequemlichkeit schätzen. Beachten Sie, dass das NOPASSWD-Tag nur gültig ist, wenn es keine anderen sudoers-Zeilen gibt, die vom selben Benutzer ein Passwort verlangen. Das gilt auch für Gruppeneinträge, also z. B. %admin ...

Mehrere Kommandos mit sudo ausführen

Bei umfangreicheren Administrationsaufgaben wird es zunehmend lästig, jedem Kommando sudo voranzustellen. Eleganter ist es, mit sudo -s in den root-Modus zu wechseln. Alle weiteren Kommandos werden wie von root ausgeführt. Sie beenden diesen Modus mit `[Strg]+[D]`.

gksudo

Beim Start von Administrationsprogrammen unter X kommt bei vielen Distributionen gksudo zum Einsatz. Dieses Programm ermöglicht die Passwort-Eingabe in einem Dialog und startet dann das gewünschte Programm (siehe Abbildung 15.3).



Abbildung 15.3: Start eines Administrationsprogramms mit gksudo

Link Die Konfiguration von `/etc/sudoers` bietet viel mehr syntaktische Möglichkeiten, als hier angedeutet wurde. Lesen Sie die man-Seiten zu sudo und zu sudoers! Noch mehr Details sind auf der sudo-Homepage nachzulesen:

<http://www.courtesan.com/sudo/>

sudo bei Ubuntu

Bei Ubuntu und einigen anderen Distributionen ist root ohne gültiges Passwort eingerichtet. Ein root-Login ist damit unmöglich. Auch `su` oder `ssh -l root` funktionieren nicht. Die einzige Möglichkeit zur Ausführung administrativer Kommandos bietet hier sudo. Die Datei `/etc/sudoers` enthält nur drei Zeilen:

```
# Defaultkonfiguration in /etc/sudoers bei Ubuntu
Defaults        env_reset
root            ALL=(ALL) ALL
%admin          ALL=(ALL) ALL
```

Die erste Zeile bewirkt, dass beim Benutzerwechsel alle Umgebungsvariablen zurückgesetzt werden. Die zweite Zeile gibt root uneingeschränkten Zugriff auf alle Programme. Die Zeile ist unter Ubuntu eigentlich zwecklos, weil der root-Login gesperrt ist. Am wichtigsten ist die dritte Zeile: Sie erlaubt allen Mitgliedern der Gruppe admin den Aufruf sämtlicher Programme.

Per Default ist unter Ubuntu nur der erste Benutzer (also der, der während der Installation eingerichtet wurde) admin-Mitglied. Wenn Sie weitere Benutzer anlegen, müssen Sie diese der Gruppe ADMINISTRATOR zuordnen (SYSTEM|SYSTEMVERWALTUNG|BENUTZER, Button BENUTZER HINZUFÜGEN, Dialogblatt KOMPLEX).

Die folgende zusätzliche Zeile in `/etc/sudoers` erlaubt es dem Benutzer `kofler`, das Kommando `apt-get` und das Programm `Synaptic` ohne Passwort auszuführen. Damit können auch Updates ohne Passworтеingabe durchgeführt werden.

```
# Ergänzung in /etc/sudoers bei Ubuntu
kofler ALL=NOPASSWD: /usr/sbin/synaptic, /usr/sbin/synaptic
```

sudo bei SUSE

Bei SUSE spielt `sudo` eine viel kleinere Rolle als unter Ubuntu. Wenn Sie dennoch `sudo` einsetzen möchten, müssen Sie auf einige Besonderheiten in der Defaultkonfiguration achten (bzw. diese gegebenenfalls ändern).

```
# Defaultkonfiguration in /etc/sudoers bei SUSE ab Version 11.1
Defaults always_set_home # verhindert die Ausführung von X-Programmen
Defaults env_reset      # verhindert die Ausführung von X-Programmen
Defaults env_keep = "LANG LC_ADDRESS LC_CTYPE ..."
Defaults targetpw      # sudo fragt nach dem Passwort des Zielbenutzers
ALL ALL=(ALL) ALL      # mit dem richtigen Passwort darf jeder alles
root ALL=(ALL) ALL
```

Die ersten drei Zeilen verhindern aus Sicherheitsgründen die Ausführung von X-Programmen mit `sudo`. Wenn `sudo` den direkten Start solcher Programme unterstützen soll, müssen Sie diese Zeile löschen bzw. ein `#`-Zeichen voranstellen.

`Defaults targetpw` bedeutet, dass grundsätzlich das Passwort für den Account angegeben werden muss, in dem das Kommando ausgeführt werden soll (in der Regel also das `root`-Passwort). Die Zeile `ALL ALL=(ALL) ALL` erlaubt schließlich allen Benutzern, jedes Kommando auszuführen, sofern das richtige Passwort für den Ziel-Account bekannt ist.

15.4 Prozesse unter einer anderen Identität ausführen (PolicyKit)

Die Grundidee des PolicyKits besteht darin, Programme in zwei Komponenten zu zerlegen: Der eine Teil enthält die Benutzeroberfläche und läuft mit gewöhnlichen Benutzerrechten. Der zweite Teil des Programms, der in der Nomenklatur des PolicyKits als *mechanism* bezeichnet wird, ist für Systemeingriffe zuständig und läuft mit `root`-Rechten. Diese Trennung hat den fundamentalen Vorteil, dass nicht mehr ein riesiges Programm mit `root`-Rechten laufen muss, sondern nur noch kleine Teile. Das reduziert mögliche Sicherheitsrisiken. Außerdem besteht theoretisch die Möglichkeit, dass verschiedene Benutzeroberflächen (z. B. ein Gnome- und ein KDE-Programm) auf ein einheitliches Set von Mechanismen zurückgreift.

Konzept

Die Kommunikation zwischen den beiden Komponenten erfolgt durch das Bussystem (in der Regel über den D-Bus, siehe Seite 520). Ob ein bestimmter Mechanismus ausgeführt werden darf oder nicht, entscheiden Funktionen der PolicyKit-Bibliothek, die auf eine zentrale Rechtedatenbank zurückgreifen. Für die Entscheidung werden drei Kriterien berücksichtigt:

- » **Subjekt:** Wer bzw. welcher Benutzer will Systemänderungen durchführen?
- » **Objekt:** Welches Objekt soll verändert werden (z. B. eine Datei, eine Partition oder eine Netzwerkverbindung)?
- » **Aktion:** Was soll gemacht werden (z. B. eine Partition in das Dateisystem einbinden)?

Benutzersicht

In vielen Fällen bemerkt der Benutzer gar nichts vom PolicyKit. Beispielsweise erlaubt die Standardkonfiguration bei den meisten aktuellen Distributionen dem Dateimanager, externe Datenträger in das Dateisystem einzubinden. Dazu ist keine weitere Authentifizierung erforderlich, der Vorgang erfolgt automatisch, sobald der Datenträger angeschlossen wird.

Eine zweite Variante besteht darin, dass die PolicyKit-Regeln eine Autorisierung verlangen – beispielsweise zur Durchführung eines Updates mit dem PackageKit oder beim Zugriff auf eine Partition einer lokalen Festplatte. In diesem Fall erscheint ein Authentifizierungsdialog, der wie in Abbildung 15.4 aussieht. Bemerkenswert ist, dass sich das PolicyKit bei entsprechender Konfiguration die Authentifizierung merkt und in *.auths-Dateien in /var/lib/PolicyKit/ speichert. Wenn ein Benutzer sich also ein einziges Mal für einen bestimmten Vorgang authentifiziert hat, fragt PolicyKit in Zukunft nicht mehr nach.



Abbildung 15.4:
PolicyKit-
Authentifizierung

Auf eine dritte Variante stoßen Sie bei diversen Gnome-Administrationswerkzeugen, wie sie beispielsweise unter Ubuntu zum Einsatz kommen: Hier führt der Button ENTSPERREN zum Authentifizierungsdialog. Erst nach der Angabe des root- oder Benutzerpassworts können tatsächlich Systemveränderungen durchgeführt werden.

**Konfiguration
und Adminis-
tration**

Die Konfiguration des PolicyKits erfolgt an folgenden drei Orten:

| | |
|--------------------------------------|--------------------------------|
| /etc/PolicyKit/PolicyKit.conf | (globale Konfiguration) |
| /usr/share/PolicyKit/policy/*.policy | (Aktionen) |
| /var/lib/PolicyKit/*.auths | (explizit eingestellte Rechte) |

Bei der Grundkonfiguration gibt es distributionsspezifische Besonderheiten. Beispielsweise gibt PolicyKit.conf bei Ubuntu-Systemen allen Benutzern der admin-Gruppe root-Rechte. Das ist erforderlich, weil unter Ubuntu normalerweise kein root-Login möglich ist und es daher auch kein root-Passwort gibt.

Die PolicyKit-Benutzerrechte können auch zentral in einer komfortablen Benutzeroberfläche eingestellt werden. Dazu dient das Programm `polkit-gnome-authorization` (siehe Abbildung 15.5). Alternativ stehen auch diverse Kommandos zur Verfügung, um die Rechedatenbank zu verändern (`polkit-auth`, `polkit-action` etc.). Damit wird auch der zweite wesentliche Vorteil des PolicyKits im Vergleich zu anderen Authentifizierungsverfahren offensichtlich: Erstmals ist es auf ganz einfache Weise möglich, einzelnen Benutzern bestimmte Systemadministrationsaufgaben zu übertragen. Weder muss dazu das `root`-Passwort preisgegeben werden, noch ist ein unbequemes und fehleranfälliges Hantieren mit Gruppenzuordnungen erforderlich.

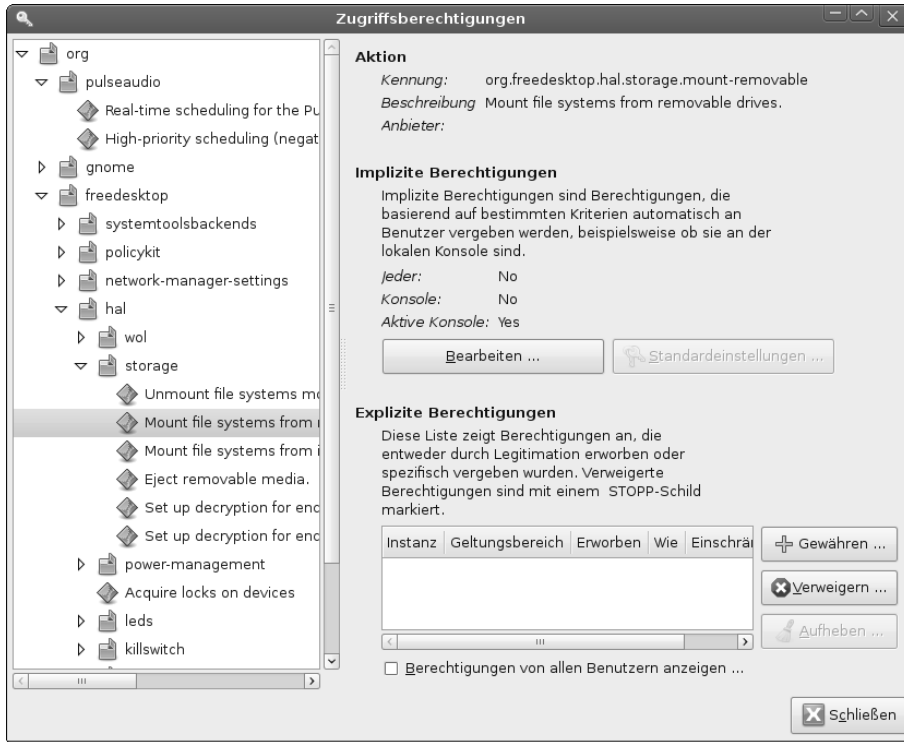


Abbildung 15.5: PolicyKit-Administration

15.5 Systemprozesse (Dämonen)

Als Dämonen (englisch *daemons*) werden Hintergrundprozesse zur Systemverwaltung bezeichnet. Diese Prozesse werden normalerweise während des Hochfahrens des Rechners im Rahmen des `Init-V`-Prozesses gestartet. Wenn Sie mit der Windows-Diktion vertraut sind, entsprechen Linux-Dämonen den Windows-Services. Tabelle 15.1 beschreibt ganz kurz die Aufgaben der wichtigsten Dämonen. Soweit die Programme in diesem Buch beschrieben werden, werden Seitenverweise angegeben. Beachten Sie, dass manche Dämonennamen je nach Distribution variieren (z. B. `httpd` oder `apache2` für den Webserver Apache).

| PROZESS | BEDEUTUNG |
|--------------------|---|
| atd | startet andere Programme zu vorgegebenen Zeiten (ähnlich wie cron) |
| avahi-daemon | automatische Netzwerkkonfiguration (ZeroConf, Rendezvous, Bonjour) |
| bluetoothd | Bluetooth-Verwaltung |
| cron | startet andere Programme zu vorgegebenen Zeiten (Seite 378) |
| cupsd | Drucker-Spooler (Seite 986) |
| dbus-daemon | D-BUS-Kommunikation (Seite 520) |
| dhclient | DHCP-Client (Seite 817) |
| dhcpd | weist anderen Rechnern die IP-Netzwerkadresse zu (Seite 855) |
| dhcpcd | liest die IP-Netzwerkadresse (Seite 816) |
| gdm | Gnome-Login-Manager (Seite 582) |
| gpm | Mausverwaltung für Textkonsolen (Seite 494) |
| haldaemon | Hardware-Verwaltung (Seite 520) |
| hpiod | HP-Druck- und -Scan-Funktionen (Seite 988) |
| httpd | Webserver (z. B. Apache) |
| kdm | KDE-Login-Manager (Seite 582) |
| klogd | protokolliert Kernelmeldungen (Seite 523) |
| lockd | NFS-Locking |
| lpd | herkömmlicher Drucker-Spooler auf der Basis von BSD-LPD |
| mdnsd | automatische Netzwerkkonfiguration (ZeroConf, Rendezvous, Bonjour) |
| mysqld | MySQL-Datenbank-Server (Seite 942) |
| named | Domain-Nameserver (Seite 865) |
| nm-system-settings | Network Manager (Seite 793) |
| nmbd | Nameserver für Windows/Samba (Seite 958) |
| nscd | Cache für Benutzer, Gruppen und Rechnernamen (Seite 508) |
| ntpd | Zeiteinstellung mit dem <i>Network Time Protocol</i> (Seite 999) |
| pppd | VPN-Client (Seite 841), Internetzugang (Seite 827) |
| pptpd | PPTP-Server für VPN (Seite 902) |
| portmap | Teil des NFS-Servers (Seite 951) |
| postfix | Mail-Server zum Versenden von E-Mails |
| rpc.* | RPC-Netzwerkdienste (<i>remote procedure call</i>), zumeist für NFS |
| sdpd | Bluetooth-Verwaltung |

Tabelle 15.1:
Wichtige
Systemprozesse

| PROZESS | BEDEUTUNG |
|-----------|--|
| sendmail | Mail-Server zum Versenden von E-Mails |
| smartd | SMART-Festplattenüberwachung (Seite 685) |
| smbd | Datei-Server für Windows/Samba (Seite 958) |
| squid | Web-Proxy und -Cache |
| sshd | Secure-Shell-Server (Seite 930) |
| syslogd | protokolliert Systemmeldungen (Seite 523) |
| syslog-ng | protokolliert Systemmeldungen (Seite 523) |
| udev | Device-Verwaltung (Seite 357) |
| vsftpd | FTP-Server (Seite 948) |
| xdm | X-Display Manager (Seite 579) |
| xinetd | startet andere Netzwerkdämonen (Seite 765) |

Tabelle 15.1:
Wichtige
Systemprozesse
(Forts.)

Kernel-Threads

Neben gewöhnlichen Server-Diensten wie `httpd` (Apache) gibt es eine Reihe von Hintergrundprozessen, bei denen es sich aber nicht um richtige Programme handelt, sondern um Teilprozesse (*threads*) des Kernels. Sie erkennen diese Prozesse daran, dass `ps` `axu` ihre Namen in eckige Klammern stellt. Manchen dieser Teilprozesse ist eine Nummer hintangestellt, die auf die CPU hinweist. `kerneld/0` verwaltet somit den Block-Device-Buffer für die erste CPU, `kerneld/1` für die zweite CPU etc.

Die meisten Kernel-Threads betreffen Low-Level-Aufgaben des Betriebssystems (Speicherverwaltung, Prozessverwaltung, CPU-Steuerung etc.). Sie werden überwiegend bereits während der Systeminitialisierung zu Beginn des Systemstarts gestartet. Für normale Anforderungen ist keine spezielle Konfiguration erforderlich. Die Funktion der wichtigsten Kernel-Threads ist in Tabelle 15.2 zusammengefasst.

| KERNEL-THREAD | BEDEUTUNG |
|-------------------------|---|
| <code>aio</code> | asynchrone IO-Verwaltung (z. B. für Netzwerkprozesse) |
| <code>events</code> | Ereignis- und Software-Interrupt-Verwaltung |
| <code>kacpid</code> | ACPI-Funktionen |
| <code>kerneld</code> | verwaltet den Block-Device-Buffer |
| <code>kernelperd</code> | lädt bzw. entfernt Kernelmodule für Benutzerprogramme |
| <code>khubd</code> | verwaltet das Ein- und Ausstecken von USB-Geräten |
| <code>kjournald</code> | führt das Journaling für ext3/4-Dateisysteme durch |

Tabelle 15.2:
Wichtige Kernel-
Threads

| KERNEL-THREAD | BEDEUTUNG |
|---------------|--|
| knfsd | NFS-Server |
| kthread | verwaltet Threads |
| nfsd | NFS-Server |
| kscand | Speicherverwaltung |
| kseriod | kommuniziert mit seriellen Geräten |
| ksoftirqd | Hardware-Interrupt-Verwaltung |
| kswapd | Swapping |
| lockd | NFS-Locking |
| migration | bestimmt, welche Prozesse auf welcher CPU laufen |
| pccardd | verwaltet PCMCIA-Karten |
| pdflush | speichert Dateiänderungen physikalisch |
| rpciod | NFS |
| scsi_eh | verwaltet SCSI-Fehler und -Timeouts |
| watchdog | überwacht, ob das System noch reagiert |

Tabelle 15.2:
Wichtige Kernel-
Threads (Forts.)

Dämonen starten und beenden

Bei den meisten Distributionen werden viele der oben aufgezählten Dämonen über das sogenannte Init-V-System gestartet. Grundlagen und Details des Init-V-Systems werden ausführlich ab Seite 741 beschrieben. Dort erfahren Sie auch, wie Sie selbst neue Scripts in das System integrieren.

Bei einigen neuen Distributionen kommt statt des Init-V-Systems Upstart zum Einsatz (siehe Seite 749). Allerdings besitzt Upstart einen Init-V-Kompatibilitätsmodus, der weiterhin für den Start der meisten Dämonen zuständig ist. Aus diesem Grund sind die aus dem Init-V-System vertrauten Vorgehensweisen weiterhin gültig. In den folgenden Tabellen sind teilweise auch Upstart-Kommandos angegeben – sie gelten aber nur, wenn ein Dämon direkt von Upstart gesteuert wird (und nicht über dessen Init-V-Kompatibilitätsschicht).

An dieser Stelle finden Sie lediglich eine kurze Zusammenfassung, wie Sie einen Dämon manuell starten bzw. stoppen, was Sie tun müssen, damit der Dämon beim Systemstart automatisch gestartet wird, bzw. wie Sie den automatischen Start vermeiden. Diese Informationen werden Sie insbesondere beim Einrichten und Konfigurieren von Netzwerkdiensten häufig benötigen. Beachten Sie, dass nicht nur das Kommando, sondern auch der Dienstname je nach Distribution variieren kann. Beispielsweise heißt das Script zum Starten des Webservers Apache bei Debian, Ubuntu und SUSE apache2, bei Fedora und Red Hat dagegen httpd.

Um einen Dämon bzw. Netzwerkdienst oder -Server zu starten, führen Sie das folgende Kommando aus. Beachten Sie, dass neue Netzwerkdienste bei manchen Distributionen sofort nach der Installation des Pakets gestartet werden (z. B. bei Debian und Ubuntu), während der Start bei anderen Distributionen manuell erfolgen muss!

Manuell starten

```
root# /etc/init.d/name start      (Debian, Fedora, Red Hat, SUSE, Ubuntu)
root# invoke.rc name start      (Debian, Ubuntu)
root# service name start       (Fedora, Red Hat)
root# rcname start              (SUSE)
root# start name                (nur für Upstart-Prozesse)
```

So sieht das Kommando aus, um einen Dienst wieder zu stoppen:

Manuell stoppen

```
root# /etc/init.d/name stop      (Debian, Fedora, Red Hat, SUSE, Ubuntu)
root# invoke.rc name stop      (Debian, Ubuntu)
root# service name stop       (Fedora, Red Hat)
root# rcname stop              (SUSE)
root# stop name                (nur für Upstart-Prozesse)
```

Viele Netzwerkdienste sind in der Lage, im laufenden Betrieb die Konfigurationsdateien neu einzulesen. Die reload-Anweisung ist notwendig, damit der Dienst Änderungen an der Konfigurationsdatei berücksichtigt. Dienste, die reload nicht unterstützen, müssen Sie durch restart vollständig neu starten.

Reload/Restart

```
root# /etc/init.d/name reload/restart (Debian, Fedora, Red Hat, SUSE, Ubuntu)
root# invoke.rc name reload/restart (Debian, Ubuntu)
root# service name reload/restart (Fedora, Red Hat)
root# rcname reload/restart (SUSE)
```

Sobald ein Netzwerkdienst einmal korrekt eingerichtet ist, soll er in der Regel beim Hochfahren des Rechners automatisch gestartet, beim Herunterfahren automatisch gestoppt werden. Bei manchen Distributionen (z. B. Debian, Ubuntu) werden die Dienste bereits bei der Paketinstallation entsprechend konfiguriert. Bei anderen Distributionen ist das aus Sicherheitsgründen nicht der Fall, und der automatische Start muss explizit aktiviert werden!

Automatischer Start beim Hochfahren

Bei Debian versieht update-rc.d die Init-V-Start-Links und -Stop-Links standardmäßig mit der durchlaufenden Nummer 20. Wenn Sie andere Nummern wünschen, geben Sie diese mit n1 n2 an (siehe auch Seite 752).

Bei Red Hat und Fedora reicht in vielen Fällen das erste chkconfig-Kommando. Das zweite Kommando ist nur erforderlich, wenn das Init-V-Script keine Angaben enthält, in welchem Runlevel der Dienst normalerweise gestartet werden soll (fast immer 3 und 5, siehe auch ab Seite 756).

```
root# update-rc.d name defaults [n1 n2] (Debian, Ubuntu)
root# chkconfig --add name              (Fedora, Red Hat)
root# chkconfig --level 35 name on      (Fedora, Red Hat)
root# inserv name                       (SUSE)
```

Automatischen Start verhindern

Die folgenden Kommandos verhindern in Zukunft den automatischen Start beim Hochfahren. Wenn der Dienst bereits läuft, wird er durch das folgende Kommando allerdings nicht gestoppt; dazu ist ein eigenes stop-Kommando erforderlich.

```
root# update-rc.d -f name remove      (Debian, Ubuntu)
root# chkconfig --del name           (Fedora, Red Hat)
root# insserv -r name                 (SUSE)
```

15.6 Prozesse automatisch starten (crontab)

Wenn Ihr Rechner plötzlich – scheinbar unvermittelt – damit beginnt, die Festplatte zu durchsuchen, Ihnen E-Mails zusendet etc., dann ist die Ursache fast immer der automatische Start von Prozessen durch den Dämon cron. Dieses Programm wird beim Rechnerstart durch den Init-V-Prozess automatisch gestartet. Es wird einmal pro Minute aktiv, analysiert alle crontab-Dateien und startet die dort angegebenen Programme. cron wird in erster Linie für Wartungsarbeiten verwendet – um Logging-Dateien abzuschneiden, um temporäre Dateien zu löschen, um Verzeichnisse zu aktualisieren etc.

Die globale Konfiguration von cron erfolgt durch die Datei `/etc/crontab`. Darüber hinaus dürfen Benutzer Ihre eigenen cron-Jobs in den benutzerspezifischen Dateien `/var/spool/cron/tabs/user` definieren.

Das Recht der benutzerspezifischen cron-Steuerung kann mit den beiden Dateien `/var/spool/cron/allow` und `/deny` eingestellt werden. Wenn `allow` existiert, dürfen nur die hier eingetragenen Benutzer cron-Kommandos ausführen. Wenn `deny` existiert, sind die hier eingetragenen Benutzer ausgeschlossen. Existiert keine dieser Dateien, hängt es von der Kompilation von cron ab, ob irgendwelche Benutzer außer root cron verwenden dürfen.

crontab Die Datei `/etc/crontab` bzw. die Dateien in `/etc/cron.d` enthalten zeilenweise Einträge für die auszuführenden Programme. Die Syntax sieht so aus:

```
# in /etc/crontab
min hour day month weekday user command
```

| SPALTE | BEDEUTUNG |
|---------|--|
| min | gibt an, in welcher Minute (0–59) das Programm ausgeführt werden soll |
| hour | gibt die Stunde an (0–23) |
| day | gibt den Tag im Monat an (1–31) |
| month | gibt den Monat an (1–12) |
| weekday | gibt den Tag der Woche an (0–7, 0 und 7 bedeuten jeweils Sonntag) |
| user | gibt an, für welchen Benutzer das Kommando ausgeführt werden soll (meist root) |
| command | enthält schließlich das auszuführende Kommando |

Tabelle 15.3:
crontab-Spalten

Wenn in den ersten fünf Feldern statt einer Zahl ein * angegeben wird, wird dieses Feld ignoriert. `15 * * * *` bedeutet beispielsweise, dass das Kommando immer 15 Minuten nach der ganzen Stunde ausgeführt werden soll, in jeder Stunde, an jedem Tag, in jedem Monat, unabhängig vom Wochentag. `29 0 * * 6` bedeutet, dass das Kommando an jedem Samstag um 0:29 Uhr ausgeführt wird.

Für die Zeitfelder ist auch die Schreibweise `*/n` erlaubt. Das bedeutet, dass das Kommando jede *n*-te Minute/Stunde etc. ausgeführt wird. `*/15 * * * *` würde also bedeuten, dass das Kommando viertelstündlich (*n*:00, *n*:15, *n*:30 und *n*:45) ausgeführt wird.

Was 0 für den Tag im Monat bzw. den Monat bedeutet, geht aus der sonst recht klaren Dokumentation leider nicht hervor (siehe man 5 crontab).

Die Dateien `/var/spool/cron/tabs/user` haben dasselbe Format wie crontab. Der einzige Unterschied besteht darin, dass die user-Spalte fehlt.

Um die globale cron-Konfiguration zu verändern, können Sie `/etc/crontab` bzw. die Dateien in `/etc/cron*` direkt mit einem Editor bearbeiten. Wenn Sie dagegen benutzerspezifische cron-Einträge vornehmen möchten, sollten Sie dazu das Kommando `crontab -e` einsetzen. (Führen Sie vorher `export EDITOR=emacs` aus, wenn Sie nicht mit dem vi arbeiten möchten. Die Manual-Seiten zu cron und crontab geben weitere Informationen.)

Bei den meisten Distributionen sieht die Defaultkonfiguration so aus, dass `/etc/crontab` lediglich einige wenige Einträge enthält, die bewirken, dass einmal pro Stunde alle Script-Dateien in `/etc/cron.hourly/*` ausgeführt werden, einmal pro Tag die Script-Dateien in `/etc/cron.daily/*` etc. Bei Ubuntu sieht `/etc/crontab` so aus:

`cron.hourly,`
`.daily, .weekly,`
`.monthly`

```
# /etc/crontab
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * * root    test -x /usr/sbin/anacron || \
    ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 * root    test -x /usr/sbin/anacron || \
    ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * * root    test -x /usr/sbin/anacron || \
    ( cd / && run-parts --report /etc/cron.monthly )
```

Im Klartext bedeutet das, dass Cron

- » 17 Minuten nach jeder vollen Stunde alle Script-Dateien des Verzeichnisses `/etc/cron.hourly` ausführt,
- » täglich um 6:25 alle Script-Dateien des Verzeichnisses `/etc/cron.daily` ausführt,
- » wöchentlich am Sonntag um 6:47 alle Script-Dateien des Verzeichnisses `/etc/cron.weekly` ausführt und
- » an jedem Ersten des Monats um 6:52 alle Script-Dateien des Verzeichnisses `/etc/cron.monthly` ausführt.

Die Scripts aus `/etc/cron.daily`, `-.weekly` und `-.monthly` werden nicht ausgeführt, wenn Anacron installiert ist (siehe unten).

Um selbst regelmäßig ein Backup-, Update- oder sonstiges Script auszuführen, fügen Sie die entsprechende Script-Datei einfach in eines der Verzeichnisse `/etc/cron.hourly`, `-.daily`, `-.weekly` oder `-.monthly` ein. Vergessen Sie nicht, das `execute`-Bit zu setzen (`chmod a+x datei`)!

Wenn Sie mit der durch Crontab vorgesehenen Zeit für die `/etc/cron.xxx`-Verzeichnisse nicht einverstanden sind, können Sie natürlich auch `/etc/crontab` um eine Zeile erweitern und den gewünschten Ausführungszeitpunkt und Ihr Script dort eintragen. Noch übersichtlicher ist es, in `/etc/cron.d` eine entsprechende neue Datei anzulegen. Achten Sie darauf, dass alle Cron-Konfigurationsdateien mit einem Zeilenumbruch enden müssen – andernfalls wird die letzte Zeile ignoriert!

Wenn Sie verhindern möchten, dass cron-Jobs einen zeitkritischen Vorgang – z. B. das Brennen einer DVD – beeinträchtigen, deaktivieren Sie cron einfach vorübergehend durch `/etc/init.d/cron stop`.

Anacron Neben Cron ist bei den meisten Distributionen auch Anacron installiert. Anacron kümmert sich darum, dass täglich, wöchentlich oder monatlich auszuführende Aufgaben auch dann erledigt werden, wenn ein Rechner nur unregelmäßig in Betrieb ist und beispielsweise über Nacht oder am Wochenende ausgeschaltet wird.

Anacron führt einmalig (soweit erforderlich) die Scripts der Verzeichnisse `/etc/cron.daily`, `-.weekly` und `-.monthly` aus. Anders als Cron endet Anacron nach der Ausführung aller Scripts und läuft nicht im Hintergrund weiter.

Anacron speichert den Ausführungszeitpunkt in Dateien des Verzeichnisses `/var/spool/anacron`. Damit wird ausgeschlossen, dass ein für die tägliche Ausführung bestimmtes Script am selben Tag zweimal ausgeführt wird. Anacron wird durch `/etc/anacrontab` gesteuert, wobei die Standardkonfiguration in der Regel ausreichend ist.

Achtung

Auf einem Server, der ja zumeist wochenlang ohne Unterbrechung läuft, ist die Verwendung von Anacron nicht zweckmäßig! Anacron führt die Cron-Jobs der Verzeichnisse `/etc/cron.daily`, `-.weekly` und `-.monthly` nur einmalig aus. Cron wiederum ignoriert diese Jobs, wenn Anacron installiert ist. Das führt dazu, dass sämtliche Jobs maximal ein einziges Mal ausgeführt werden, ganz egal, wie lange der Server läuft!