

Jochen Hein

Linux-Systemadministration

Einrichtung, Verwaltung, Netzwerkbetrieb



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

3 Ablauf eines Systemstarts

It's not so hard to lift yourself by your bootstraps once you're off the ground.

Daniel B. Luten

3.1 Überblick über einen Systemstart

Dem Start eines Computers ist etwas Magisches eigen. Üblicherweise wird der Vorgang auch als »boot strapping« bezeichnet. Dabei kommt einem im deutschen Sprachraum das Bild eines sich selbst an den Stiefelschäften aus dem Sumpf ziehenden Baron von Münchhausen in den Sinn. Aber es finden sich (in diesem Fall) logische Erklärungen, die zeigen, dass jede Stufe des Vorgangs ganz bestimmte Ergebnisse der vorausgegangenen Phasen voraussetzt und keinesfalls irgendwie in der Luft hängt. Der Start eines Linux-PCs findet in den im Folgenden aufgeführten Phasen statt:

- Ausführen des Power-On-Self-Tests (POST) des Basic-Input/Output-Systems (BIOS). Dieser ist oft in einem ROM oder Flash-EPROM gespeichert.
- Ausführen eines Boot-Laders von einer Diskette, Festplatte oder von einem CD-ROM-Laufwerk aus. Viele Unix-Workstations können auch von einem angeschlossenen Band gebootet werden, die PC-Architektur gibt das aber nicht her.
- Start des Kernels durch den Boot-Lader.
- Abarbeiten der individuellen Systemkonfiguration via `init`.

In den folgenden Abschnitten konzentriere ich mich auf die Initialisierung der PC-Hardware durch das BIOS. Andere Systeme führen ähnliche Funktionen aus, allerdings möglicherweise in anderer Reihenfolge oder mit anderen Verfahren.

3.2 Das Basic-Input/Output-System (BIOS)

An erster Stelle im Bootvorgang steht das BIOS der Hauptplatine. Diese zumeist in EPROMS (neuerdings auch Flash-EPROMS) befindliche Software enthält eine Reihe von Funktionen, die BIOS-Interrupts, so z. B. `0x10` für die Bildschirmsteuerung oder `0x13` für Disketten- und Festplattenzugriffe. Damit war es möglich, kleinere Abweichungen in der Hardware-Ansteuerung durch entsprechende Anpassungen im BIOS auszugleichen – das war einer der Gründe für den Erfolg der IBM-kompatiblen PCs.

Jede Erweiterungskarte bzw. Komponente, die IBM-kompatibel sein will, muss auf entsprechende Aufrufe, z. B. zur Initialisierung durch das BIOS in definierter Art und Weise, reagieren. Hierdurch wurde eine gewisse Unabhängigkeit von

spezifischen Peripheriegeräten geschaffen, was wesentlich zur Herausbildung des Marktes für IBM-kompatible Komponenten beigetragen hat. Eigentlich handelt es sich aber um eine Intel-Kompatibilität, genauer um eine Kompatibilität zum Intel-8088/86, da die im BIOS fest eingebrannte Software aus Instruktionen für diesen Prozessor besteht.

Es gibt eine Vielzahl unterschiedlicher BIOS, nämlich für jeden Typ von Motherboard eine eigene BIOS-Version. Wie einzelne Funktionen realisiert wurden und in welcher Reihenfolge sie abgearbeitet werden, liegt zu einem gewissen Maß im Ermessen des jeweiligen Herstellers. Die folgende Übersicht über den Ablauf eines BIOS-Starts hat insofern exemplarischen Charakter.

Nach dem Einschalten der Stromversorgung wird das BIOS hardwaremäßig in den Speicherbereich von `0xF000:E000` bis `0xF000:FFFF` eingeblendet und der Programmzähler auf die Adresse `0xF000:FFF0` gesetzt, womit die Ausführung des BIOS-Codes beginnt. Der Prozessor startet im so genannten Real-Mode, daher steht nur der Speicherbereich unterhalb von 1 MB zur Verfügung. Üblicherweise enthält das BIOS auch nur Code für diesen Modus, so dass es von Protected-Mode-Systemen wie Linux nicht verwendet werden kann (einige Funktionen des modernen PCI-BIOS können von Linux verwendet werden). Neue Linux-Versionen sind teilweise so groß, dass sie nicht mehr in das erste Megabyte Speicher geladen werden können. Ein Ausweg sind hier Kernel-Module oder die Erzeugung eines `bzImage`, das bereits im Protected Mode geladen wird.

Bevor das BIOS damit beginnt, sich selbst und die Peripherie des Computers zu testen, untersucht es den Speicherbereich zwischen dem Grafikadapter an der Adresse `0xA000` und seiner eigenen Startadresse nach BIOS-Erweiterungen anderer Steckkarten, z. B. von Netzwerkkarten. Karten mit eigenem BIOS werden an der BIOS-Signatur `0xAA55` bzw. `0x55AA` am Ende eines 32K-Blocks erkannt. Bereits vor dem endgültigen Booten des Systems werden hier Initialisierungen durchgeführt.

Danach führt das BIOS den Power-On-Self-Test (POST) durch: Die Funktionen des Prozessors selbst, die Register und einige Befehle werden überprüft. Falls der Prozessor defekt ist, wird der PC, logischerweise ohne Angabe von Gründen, angehalten. Bei manchen Fehlern geben einige BIOS-Versionen noch akustische Signale aus, die je nach Anzahl der Piepser Aufschluss über die Art des Fehlers geben. Danach werden Prüfsummen über das ROM selbst gebildet und mit den darin festgelegten Werten verglichen, um mögliche Defekte im ROM-BIOS oder, wenn es in den Hauptspeicher kopiert wurde, des betreffenden RAM-Bausteins zu finden.

Anschließend werden weitere Bauteile auf der Hauptplatine, wie Interrupt- und DMA-Controller, sowie der Hauptspeicher getestet und initialisiert. Fortgesetzt wird der POST mit dem Überprüfen von Tastatur, Disketten- und Plattenlaufwerken. Die Ergebnisse dieser Tests werden in BIOS-Variablen im Hauptspei-

cher abgelegt und die Interrupt-Vektor-Tabelle wird initialisiert. Mittels des BIOS-Interrupts `0x19` wird das eigentliche Booten eingeleitet bzw. der »bootstrap loader« aufgerufen.

Durch das frühe Initialisieren der Erweiterungskarten ist es möglich, dass eine Netzwerkkarte im Laufe des weiteren Boot-Vorgangs die Funktionen des Boot-Laders übernehmen kann. Diese Funktion wird verwendet, wenn man einen Rechner über das Netz bootet. Dies hat einige Vorteile:

- Auch wenn der Rechner ausgeschaltet ist, kann man auf dem Boot-Server Anpassungen in der Konfiguration vornehmen.
- Der Rechner braucht keine oder nur eine kleine Festplatte für eine Swap-Partition. Er ist damit ausgesprochen leise.
- Man kann einen Rechner relativ einfach umkonfigurieren, z. B. auf eine neue IP-Adresse oder einen neuen Kernel.

Natürlich hat diese Funktion auch Nachteile: Die Netzlast wird höher und Dateizugriffe sind via NFS langsamer als auf eine lokale Festplatte. Wenn Sie diese Funktion nutzen möchten, dann finden Sie weitere Informationen in der Datei `Documentation/nfs-root.txt` in den Kernel-Quellen und in den Mini-HowTos `NFS-Root` und `NFS-Root-Client`.

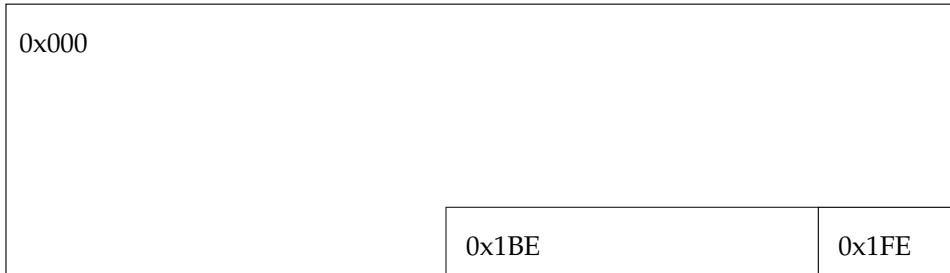
Bei Systemen, die nicht auf Intel-Prozessoren basieren, nennt man das BIOS und die entsprechenden Initialisierungs- und Laderoutinen häufig PROM, Bootmonitor oder Console. Vielfach sind hier auch interaktive Funktionen zur Hardware-Prüfung oder Initialisierung untergebracht, die auch ohne jegliches Betriebssystem verwendet werden können. Außerdem sind in vielen Fällen leistungsfähige Boot-Lader implementiert, mit denen man beliebige Kernel oder via Netzwerk booten kann.

3.3 Laden von Diskette oder Festplatte

In einer für alle Betriebssysteme gleichen Reihenfolge wird versucht, das System zu starten. Die Reihenfolge, in der versucht wird, von den verschiedenen Laufwerken zu booten, kann normalerweise im BIOS-Setup eingestellt werden. Früher wurde versucht, zunächst von `/dev/fd0` (Laufwerk A:), danach von `/dev/hda` (Laufwerk C:), den ersten Sektor zu lesen. Heute hat man die Wahl, ob das Laufwerk A:, Laufwerk C:, ein SCSI- oder das CD-ROM-Laufwerk als Boot-Device verwendet werden soll. Bei Disketten wird dieser erste Sektor *Boot-Sektor*, bei Festplatten *Master Boot Record* (MBR) genannt. Er ist jeweils 512 Byte groß.

Viele Unix-Workstations können auch vom Band oder von CD gebootet werden, auch diese Funktionen sind nur in den neueren PC-BIOS-Versionen bzw. im BIOS des SCSI-Controllers implementiert. Wenn Ihr BIOS diese Funktion unter-

stützt, dann können Sie direkt von einer geeignet erstellten Linux-CD booten und das System installieren – ohne Disketten.



Offset: 0x000 Programmcode
 0x1BE Partitionstabelle
 0x1FE Magic Number (0xAA55)

Abbildung 3.1 Aufbau eines Master Boot Records

Im Master Boot Record (MBR, siehe Abbildung 3.1) einer Festplatte befinden sich vier jeweils 16 Byte große Einträge, jeweils einer für jede der vier möglichen (primären) Partitionen. Ist eine solche Partition weiter unterteilt, wird sie erweiterte Partition genannt. Pro (erweiterter) Partition können bis zu vier so genannte logische Laufwerke eingerichtet werden, d. h., es sind maximal 16 logische Laufwerke je Festplatte möglich. Der Aufbau des ersten Sektors einer erweiterten Partition entspricht aus Gründen der Einfachheit dem Aufbau des MBR.

Jeder der vier Partitionseinträge des MBR enthält neben Angaben zum Typ der Partition (z. B. Linux = 0x83, Linux Swap = 0x82) und Angaben zu Beginn, Ende und Größe der Partition auch ein Bootflag, das anzeigt, ob die Partition »aktiv« ist und damit von ihr gebootet wird. Es kann immer nur eine primäre Partition durch das Bootflag als aktiv markiert sein, die dann vom Standard-MBR gebootet wird.

Der vor der Partitionstabelle des MBR liegende Programmcode lädt den Boot-Sektor der als aktiv markierten Partition und wird deshalb als Boot-Lader bezeichnet. Unabhängig vom benutzten Betriebssystem hat ein Boot-Lader beim Starten von einer Festplatte die Aufgabe, die aktive Partition zu bestimmen und unter Zuhilfenahme der BIOS-Routinen den Boot-Sektor dieser aktiven Partition zu laden. Dies ist auch die Funktion des Standard-MS-DOS-Boot-Sektors.

Wann immer es möglich ist, sollten Sie den MBR unverändert lassen und stattdessen z. B. in den Boot-Sektor der Root-Partition installieren und diese aktivieren. Wenn Sie weitere Betriebssysteme installieren oder modernisieren wollen, dann können diese die Bootfähigkeit der Linux-Installation nicht gefährden.

Am Anfang gab es unter MS-DOS nur primäre Partitionen und der Standard-MBR kann nur von solchen Partitionen das Betriebssystem laden – das `MSDOS-FDISK` kann deshalb nur primäre Partitionen aktivieren. Die MS-DOS-Bootsequenz sieht vor, dass der MBR den Boot-Sektor der aktiven Partition startet. Von diesem werden die Dateien `IO.SYS` und `MSDOS.SYS` geladen, in denen DOS gespeichert ist. Die moderneren Boot-Lader unter Linux bieten demgegenüber viele weitere Möglichkeiten, es kann auch von logischen Laufwerken in erweiterten Partitionen gebootet werden.

3.4 Die Linux Boot-Lader

Gegenüber dem MS-DOS-Verfahren haben die moderneren Boot-Lader wie `lilo` unter Linux viele weitere Möglichkeiten, z. B. das Booten anderer Betriebssysteme, das Wechseln zwischen verschiedenen Versionen eines Betriebssystems und das Übergeben von Parametern beim Start des Kernels.

Die drei wichtigsten Boot-Lader für Linux auf der Intel-Plattform sind: `lilo`, `loadlin` und `syslinux`. Für die anderen Plattformen gibt es jeweils eigene Boot-Lader, z. B. `SILO` für Linux/Sparc, `MILO` für Linux/Alpha und `amiboot` für Linux/m68k (Amiga). In den folgenden Abschnitten werden wir uns nur mit der Intel-Plattform beschäftigen.

Allen Boot-Ladern ist die Fähigkeit gemeinsam, schon beim Aufruf oder zu einem bestimmten Zeitpunkt während des Ladens quasi »per Hand« übergebene Parameter oder Argumente zu akzeptieren. Diese Parameter beeinflussen entweder den Boot-Lader selbst, den Kernel, den `init`-Prozess oder werden als Umgebungsvariablen abgelegt und können so zur Konfiguration der zu startenden Prozesse benutzt werden. Diese Angaben werden häufig als Kommandozeilenparameter des Kernels bezeichnet. Die Bezeichnung Boot-Parameter wäre jedoch treffender, da sie vielen Systemadministratoren vorwiegend in speziellen Konfigurationsdateien begegnen.

Die Parameter für die Boot-Lader selbst enthalten Dinge wie den Namen und Pfad der zu ladenden Kernel-Datei und Angaben zur Partition, auf der diese zu finden ist. Diejenigen Parameter, die die Boot-Lader nicht verstehen, werden einfach an den Kernel weitergegeben; was dieser selbst bzw. ein Gerätetreiber auswertet, bezieht sich zumeist auf das Vorhandensein und ggf. die Konfiguration bestimmter Hardware. Der Kernel wiederum übergibt die Parameter, die ihm unbekannt sind, oder das, was erkannt, aber nicht behandelt wird, an `init`. Die Werte, die auch `init` nicht kennt, werden als Environmentvariablen im Format `variable=wert` abgelegt und können somit von einer Shell oder einem beliebigen anderen Programm ausgewertet werden.

Diese Flexibilität der verschiedenen beteiligten Programme in der Handhabung der Boot-Parameter hat aber auch zur Folge, dass Optionen nicht nur durchgereicht, sondern auch überschrieben werden können. Dies ist z. B. dann beachtlich, wenn in den Kernel fest einkompilierte Werte, etwa der Interrupt und der Port einer Netzwerkkarte, sich im Nachhinein als falsch herausstellen.

3.4.1 Der Linux-Lader lilo

`lilo` von Werner Almesberger ist der meistbenutzte und ein recht vielseitiger Boot-Lader für Linux. Die Abkürzung `lilo` steht für Linux-Loader, `lilo` kann aber auch andere Betriebssysteme laden. Bei vielen Linux-Benutzern hat sich `lilo` zum erstenmal nach der Installation einer Linux-Distribution durch das Anzeigen des Textes `LILO` nach einem Neustart des Systems bemerkbar gemacht. Nach dem Drücken einer der Tasten `[Umschalt]`, `[Strg]` oder `[Alt]` oder wenn entweder `[Umschalt-Fest]` oder `[Rollen]` eingeschaltet sind, zeigt `lilo` den `boot:-`Prompt und wartet auf eine Benutzeraktion, d. h. darauf, dass der Name einer zu ladenden Boot-Konfiguration eingegeben wird. Durch die Eingabe von `[Tab]` wird eine Liste der verfügbaren Konfigurationen auf den Bildschirm ausgegeben.

Das Paket `lilo` besteht aus zwei Teilen, die in gewisser Weise voneinander abhängen. Diese Teile sind:

- Das Programm `/sbin/lilo`. Es muss, sofern `lilo` als Boot-Lader verwendet wird, nach jeder Änderung an den Kernel-Dateien (meist `/vmlinuz`) aufgerufen werden. Dieses Programm bestimmt die Sektornummern, in denen das Kernel-Image gespeichert ist, und vermerkt diese Liste in der Map-Datei.
- Verschiedene Boot-Sektoren, die andere Betriebssysteme (auch von der zweiten Platte) starten können oder ein Kernel-Image laden. Da diese Boot-Sektoren keine Kenntnisse von der internen Struktur, z. B. des Extended-2-Dateisystems, haben, benötigen sie eine aktuelle Map-Datei.

Dies ist auch der größte Nachteil von `lilo`: Nach jeder Änderung am Kernel muss das Programm `lilo` aufgerufen werden, um die Map-Datei zu aktualisieren. Der Systemverwalter sollte außerdem darauf achten, dass immer ein Backup-Kernel zur Verfügung steht und in die `lilo`-Konfiguration eingebunden ist. Die von den *BSD-Systemen bekannte Funktion, einen beliebigen Kernel aus dem Dateisystem zu booten, ist leider nicht verfügbar.

`lilo` kann bis zu 16 verschiedene Konfigurationen booten und sowohl in den MBR als auch in den Boot-Sektor von primären Partitionen installiert werden. `lilo` kann

- auf einer Linux-Diskette als Boot-Sektor dienen,
- auf der ersten Festplatte als MBR fungieren und

- auf beliebigen primären Partitionen (außer Swap) der ersten Festplatte als Boot-Sektor installiert sein.

Demgegenüber kann es nicht

- auf einer MS-DOS-Diskette (oder irgendeiner anderen Nicht-Linux-Diskette) installiert werden,¹
- in eine logische Partition installiert werden oder
- auf eine andere als die erste Festplatte installiert werden, wenn nicht ein dazu fähiger Boot-Lader vor `lilo` verwendet wird.

Die zuletzt genannte Beschränkung stellt jedoch keine wirkliche Einschränkung beim Entwurf einer Boot-Konfiguration dar, da `lilo` ein Kernel-Image von anderen als der ersten Festplatte laden kann. Es kann auch beliebige Partitionen, d. h. auch logische Laufwerke, auf weiteren Festplatten als root-Dateisystem benutzen.

Die Installation von `lilo` in den MBR oder den Boot-Sektor erfordert logischerweise `root`-Rechte. `lilo` wird über die Datei `/etc/lilo.conf` konfiguriert. Anschließend sollte vor der endgültigen Installation mit `/sbin/lilo -t` zunächst getestet werden, ob die in der Konfigurationsdatei befindlichen Befehle bekannt und syntaktisch korrekt sind, ob die angegebenen Kernel-Dateien gefunden wurden und die spezifizierten Partitionen existieren.

lilo konfigurieren

Zwar akzeptiert `lilo` auch Parameter, die am Boot-Prompt eingegeben werden, aber im Regelfall eines fertig konfigurierten Systems werden permanent gewünschte Einstellungen über die Konfigurationsdatei `/etc/lilo.conf` übergeben. Eine solche Datei kann z. B. wie in Listing 3.1 aussehen.

```
boot = /dev/hda2
compact
image = /boot/vmlinuz
    label=linux
image = /boot/vmlinuz.old
    label=linux.old
other = /dev/hda1
    table = /dev/hda
    label = win2k
```

Listing 3.1 Eine sehr einfache lilo-Konfiguration

Bei der in Listing 3.1 dargestellten Konfiguration werden drei Möglichkeiten der Boot-Konfiguration installiert. Der erste `image`-Eintrag definiert den Start des aktuellen Linux-Kernels `/boot/vmlinuz` unter dem Namen `linux`, das ist der

1. Zu diesem Zweck kann `SYSLINUX` verwendet werden.

Default. Der zweite `image`-Eintrag sorgt dafür, dass der automatisch von `make zlilo` bzw. `make install` erzeugte Backup-Kernel ebenfalls gebootet werden kann. Mit dem Eintrag `other=` kann ein anderes System, hier Windows, gebootet werden. Die ersten beiden Zeilen im Listing 3.1 bestimmen, dass `lilo` im Boot-Sektor der Partition `/dev/hda2` installiert wird und versucht wird, mit möglichst wenigen Befehlen die Kernel von der Platte zu lesen.

Bei den `lilo`-Optionen unterscheidet man in globale und `image`-spezifische Optionen. Globale Optionen beziehen sich auf alle Kernel-Dateien und wirken unabhängig von der ausgewählten Konfiguration. Die anderen Optionen beziehen sich jeweils nur auf eine von mehreren bootbaren Konfigurationen. Im Beispiel sind die globalen Optionen linksbündig dargestellt, die `image`-bezogenen sind zwei Zeichen eingerückt. Ein Block `image`-bezogener Optionen wird entweder durch `image=` oder `other=` eingeleitet. Bis zu 16 verschiedene solcher Blöcke sind möglich – auch mehrere `other=`-Blöcke. `lilo` kennt die folgenden globalen Optionen:

`backup=backup_file`

Vom ursprünglichen Boot-Sektor wird eine Sicherungskopie mit dem Namen `backup_file` erstellt. Sofern die Datei `/boot/boot.Device` nicht existiert, wird dort standardmäßig der alte Boot-Sektor gesichert.

`boot=boot_device`

Gibt die Festplatte bzw. die Partition an, die den Boot-Sektor von `lilo` enthält. Wenn Sie den NT-Loader verwenden, können Sie auch eine 512 Byte große Datei anlegen und diese in die Datei `boot.ini` aufnehmen. Genauere Informationen dazu finden Sie im Abschnitt »`lilo` und der Windows-Boot-Manager«.

`compact`

Versucht, Leseanforderungen an benachbarte Sektoren auf dem Laufwerk zu bündeln. Damit wird die Ladezeit, insbesondere beim Laden von Disketten, deutlich reduziert.

`default=name`

Der angegebene `image`- oder `other`-Eintrag wird als Standard zum Booten eingetragen. Wenn keine Angabe erfolgt, wird der erste `image`-Eintrag als Default-Wert verwendet.

`delay=tsecs`

Wartezeit in Zehntelsekunden, bevor `lilo` das Default-Image lädt. Wenn kein `delay` angegeben ist, dann wird unmittelbar gebootet.

`disk=device_name`

Mit diesem Eintrag beginnt ein Abschnitt in der Datei `/etc/lilo.conf`, in dem mit den Schlüsselwörtern `sectors=`, `heads=` und `cylinders=` die Geometrie des Boot-Device bestimmt werden kann. Diese Funktion sollten Sie nur einsetzen, wenn Sie die Platte mit anderen Parametern partitioniert ha-

ben, als diese selbst dem BIOS und damit `lilo` meldet. In der Regel sollten Sie diesen Eintrag nicht benötigen. Lesen Sie hierzu unbedingt die entsprechenden Abschnitte in der `lilo`-Dokumentation.

`fix-table`

Ermöglicht es `lilo`, dreidimensionale Sektor-, Kopf- und Zylinderadressen in Partitionstabellen zu korrigieren. Beachten Sie, dass Sie damit möglicherweise unangenehme Interaktionen mit anderen Betriebssystemen auslösen können.

`force-backup=backup_file`

Wie die Option `backup`, aber eine bereits existierende Backup-Datei wird überschrieben. Wenn `force-backup` zusammen mit `backup=` angegeben wird, ist nur die Option `force-backup` wirksam.

`ignore-table`

Instruiert `lilo`, defekte Partitionstabellen zu ignorieren.

`install=boot_sector`

Installiert die angegebene Datei als neuen Boot-Sektor. Wenn dieser Parameter nicht angegeben wird, verwendet `lilo` die Datei `/boot/boot.b` als `lilo`-Boot-Sektor.

`linear`

Es sollen die linearen Sektoradressen anstelle von Zylinder-/Kopf-/Sektor-Adressen benutzt werden. Lineare Adressen werden zur Laufzeit umgesetzt und sind nicht von der Festplattegeometrie abhängig. Boot-Disketten, bei deren Erstellung `linear` benutzt wurde, funktionieren möglicherweise nicht korrekt, weil die BIOS-Funktionen zur Bestimmung der Festplattenparameter für Diskettenlaufwerke nicht verlässlich arbeiten.

`map=map_file`

Gibt den absoluten Pfad und Namen der zu benutzenden `map`-Datei an; Standardwert ist `/boot/map`.

`message=message_file`

Gibt den Namen der Datei an, in der am Bildschirm auszugebende Meldungen stehen. Diese Meldungen werden vor dem Boot-Prompt ausgegeben. Ein `0x0C`-Zeichen (`(Strg)-[]`, Form-Feed) in dieser Datei löscht den Bildschirm, z. B. eine eventuell dort befindliche BIOS-Meldung. Die maximale Größe der Datei beträgt 64 Kbyte. Wenn die `Message`-Datei geändert wird, muss die `map`-Datei durch den Aufruf von `lilo` neu erzeugt werden.

`nowarn`

Warnungen bezüglich möglicher Probleme werden nicht ausgegeben.

`optional`

Macht alle Images optional, so dass keine Prüfungen durchgeführt werden.

`password=password`

Setzt ein Passwort, das für alle Images oder `other`-Einträge abgefragt wird. Achten Sie darauf, dass normale Benutzer keine Leserechte auf die Datei `/etc/lilo.conf` erlangen und dieses Passwort ausspähen können. `lilo` gibt eine entsprechende Warnung aus.

`prompt`

Zeigt den Bootprompt an, ohne dass zuvor eine Taste gedrückt werden muss. Wenn zusätzlich `timeout` nicht gesetzt ist, bleibt der Rechner an dieser Stelle stehen und wartet auf eine Eingabe.

`restricted`

Hebt einige durch `password` gesetzte Begrenzungen wieder auf. Es kann eine beliebige Konfiguration gebootet werden, aber die Eingabe von Parametern ist nicht erlaubt. Mit dieser Option können Sie z. B. den Aufruf des Single-User-Modus durch Unbefugte verhindern.

`serial=parameter_string`

Ermöglicht die Systemkontrolle über eine serielle Leitung, d. h. `lilo` akzeptiert Eingaben sowohl über das Keyboard als auch von dem Terminal, das an der seriellen Leitung angeschlossen ist. Das Schicken eines `Break` über die Leitung entspricht dem Drücken der Taste `Umschalt` an der Konsole und bringt den `boot:-`Prompt auf den Bildschirm.

Wenn der Zugang über die serielle Leitung eingeschaltet ist, sollten alle Konfigurationen mittels `password` gesichert sein, sofern der Zugang über diese serielle Leitung weniger sicher ist als der zur Konsole selbst, z. B. wenn an der seriellen Schnittstelle ein Modem angeschlossen ist. Das Format des `parameter_string` hat die folgende Syntax: `port, bpsParityBits`. Die letzten drei Angaben können weggelassen werden. Wenn eine dieser drei Angaben nicht gemacht wird, müssen die nachfolgenden auch entfallen. Wenn nur der `port` angegeben wird, muss das Komma ebenfalls entfallen. Auf meinem System wird `serial=1,9600n8` für ein 9600-Baud-Terminal an `/dev/ttyS1` verwendet.

Wenn `serial=` gesetzt ist, wird der Wert für `delay` automatisch auf 20 Zehntelsekunden gesetzt. Mit der Einstellung `serial=0,2400n8` wird die erste serielle Schnittstelle mit den Standardwerten initialisiert. Damit kann ein serielles Terminal als einfache Konsole verwendet werden. Für ältere Kernel-Versionen existieren entsprechende Patches.

`timeout=tsecs`

bestimmt, wie lange auf Tastatureingaben gewartet wird. Wenn in der eingestellten Wartezeit keine Taste gedrückt wird, wird das Default-Image geladen. Sofern bei `default=` nichts angegeben wurde, ist dies das erste aufgeführte Image. Dementsprechend wird auch die Passwortabfrage abgebrochen, wenn innerhalb dieser Zeit keine Eingabe gemacht wurde. Der Standardwert für `timeout=` ist unendlich.

`verbose=level`

Hiermit können Fehler- und Diagnosemeldungen für `lilo` eingeschaltet werden. Je größer der Wert ist, desto mehr Meldungen werden ausgegeben. Dieser Eintrag entspricht der Anzahl der `-v`-Optionen beim `lilo`-Aufruf.

Image-bezogene Optionen

Die global wirksamen Optionen können auch innerhalb der image-bezogenen Konfigurationsblöcke benutzt werden. `lilo` bietet über die global wirksamen Optionen hinaus noch folgende weitere Optionen, die nur innerhalb eines image-bezogenen Blocks benutzt werden können:

`append=string`

Fügt die angegebene Zeichenkette an die Kernel-Kommandozeile an.

`literal=string`

Verwendet die angegebene Zeichenkette als Kernel-Kommandozeile.

`ramdisk=size`

Sorgt für die Erstellung einer RAM-Disk in der entsprechenden Größe (in Kbyte).

`read-only`

Veranlasst den Kernel, das `root`-Dateisystem nur im Lesezugriff anzuhängen. Damit kann auch für dieses Dateisystem ein `fsck` durchgeführt werden. Dies ist Standard bei allen Distributionen. Nach dem Prüfen des Dateisystems wird es erneut, diesmal im Schreibzugriff, gemountet. Die Parameter für die Anzahl und den Abstand der Prüfungen können mit dem Programm `tune2fs` konfiguriert werden.

`read-write`

Das `root`-Dateisystem wird in Lese- und Schreibmodus angehängt.

`root=root_device`

Das angegebene Device wird als `root`-Dateisystem verwendet.

`vga=mode`

Ermöglicht die Veränderung der Bildschirmauflösung an der Konsole. Mit dem Parameter `vga=ask` hält der Kernel beim Booten an und fragt nach der gewünschten Auflösung. Der dort eingegebene Wert kann dann für den nächsten Systemstart hier angegeben werden. Danach muss noch `/sbin/lilo` ausgeführt werden. Alternativ steht eine Framebuffer-Konsole zur Verfügung, hier wird die Grafikkarte im Grafikmodus betrieben, die Auflösung können Sie ebenfalls per Kernel-Parameter (z. B. `vesafb=` für den VESA-Treiber) angeben.

`initrd=Image`

Die Datei `Image` wird als RAM-Disk geladen. Ist dies nicht gewünscht, kann am `lilo`-Prompt `noinitrd` angegeben werden.

Es ist möglich, für eine Image-Datei mehrere Konfigurationen vorzunehmen, da `lilo` diese nicht im Image, sondern in so genannten Image-Beschreibungen speichert. Für einen Eintrag kann ein weiterer Name mit `alias=Aliasname` vergeben werden.

lilo und der Windows-Boot-Manager

Die Konfiguration des Windows-Boot-Managers erfolgt in der Datei `BOOT.INI`. Diese Datei ist möglicherweise mit den Attributen »System«, »Read-Only« und »Versteckt« gekennzeichnet. Bevor Sie diese Datei unter Windows bearbeiten können, müssen Sie diese Attribute möglicherweise mit dem Befehl `ATTRIB` entfernen. Die Datei `BOOT.INI` ist eine Textdatei, die den weiteren Verlauf des Systemstarts beschreibt.

Das Listing 3.2 zeigt eine einfache Boot-Konfiguration für Windows NT, Linux und MS-DOS. Dabei ist auf der `C:`-Platte MS-DOS (hier liegt auch die Datei `BOOT.INI`) und auf der `D:`-Partition Windows NT in einem NTFS-Dateisystem installiert. Der Standard-MBR bootet hierbei zunächst die aktive Partition, das ist die `C:`-Platte. Von dort aus wird der weitere Boot-Vorgang gesteuert.

```
timeout=30
default=scsi(0)disk(1)rdisk(0)partition(3)\WINNT
[operating systems]
scsi(0)disk(1)rdisk(0)partition(3)\WINNT="Windows NT 4.0"
scsi(0)disk(1)rdisk(0)partition(3)\WINNT="NT [VGA-Modus]"
    /basevideo /sos
C:\bootsect.lnx="Linux"
C:\ = "MS-DOS"
```

Listing 3.2 Eine einfache Boot-Konfiguration für Windows

Der Boot-Menü-Eintrag für Linux in der vorletzten Zeile verweist auf die Datei `c:\bootsect.lnx`. Diese enthält den von `lilo` erzeugten Boot-Sektor, der dann den Linux-Kernel lädt. Dafür habe ich in der Datei `/etc/lilo.conf` den Eintrag `boot=/mount/dos/bootsect.lnx` vorgenommen, damit der von `lilo` erzeugte Boot-Sektor in diese Datei geschrieben wird (meine DOS-Partition ist unter `/mount/dos` eingehängt). Vor dem ersten Start von `lilo` muss diese Datei existieren und (mindestens) 512 Byte groß sein, da `lilo` eine Sicherheitskopie des alten Boot-Sektors erstellen will. Man kann die Datei mit `dd if=/dev/zero of=/mount/dos/bootsect.lnx bs=512 count=1` erzeugen.

Vorteilhaft ist hier, dass man die Windows-Konfiguration praktisch unverändert lässt und damit viele Probleme umgeht. Der einzige mir bekannte Nachteil ist, dass man mindestens eine Partition haben muss, auf die beide Systeme zugreifen können. `lilo` muss dort den Boot-Sektor ablegen, der Windows-Boot-Manager muss ihn von dort lesen können.

Resümee

Der Boot-Lader `lilo` ist derzeit weit verbreitet, er hat aber einige gewichtige Nachteile. Der wichtigste ist, dass die Konfiguration sehr statisch ist. Nach jeder Änderung an der Konfigurationsdatei `/etc/lilo.conf`, der Message-Datei oder einem der Kernel-Images muss in jedem Fall das Programm `lilo` aufgerufen werden, das dann die entsprechende Map-Datei neu erstellt.

Vorteilhaft ist, dass `lilo` relativ einfach ist, da der Boot-Sektor keinerlei Informationen über den Aufbau des Dateisystems hat. Nachteilig ist, dass man nicht schnell mal einen beliebigen Kernel booten kann. Wenn man im Falle eines Falles alle in `lilo` eingerichteten Images verloren hat, dann wünscht man es sich, einen beliebigen Kernel aus einem beliebigen Verzeichnis starten zu können.

3.4.2 Der Boot-Lader GRUB

Wenn man `GRUB` (<http://www.gnu.org/software/grub/grub.en.html>) als Boot-Lader verwendet, dann ist es wie bei Workstations möglich, die verschiedensten Kernel zu booten. Ein weiterer Vorteil ist, dass `GRUB` nicht nur Linux-Kernel booten kann, sondern auch für *BSD-Kernel verwendet werden kann. Sollte man also beide Systeme auf seinem System haben, dann benötigt man nur einen Boot-Lader. Weitere unterstützte Systeme sind Mach und The Hurd, dabei hält sich `GRUB` an den Multiboot-Standard, so dass für andere Betriebssysteme diese Infrastruktur bereits bereitsteht.

`GRUB` besteht im Wesentlichen aus einem Boot-Sektor, der den weiteren Lader startet (Stage1), und dem eigentlichen Lader, der in der Lage ist, verschiedene Dateisysteme (wie Extended-2, BSD FFS oder DOS FAT; das Dateisystem wird automatisch erkannt) zu lesen. Neben dem Booten von Unix-Kerneln können auch beliebige andere Partitionen angesprochen werden, so dass man `GRUB` auch als Boot-Menü und damit zum Booten von Windows oder OS/2 verwenden kann.

Die Konfiguration von GRUB

Beim Booten versucht `GRUB` zunächst, seine Konfigurationsdatei zu lesen. Wenn eine solche existiert, dann kann der Anwender das zu startende System aus einem Menü auswählen. Existiert keine Konfigurationsdatei, dann wechselt `GRUB` in die Kommandozeile (diese kann auch aus dem Menü heraus angesprungen werden).

In der Regel wird man die wichtigsten Systeme und Optionen in einer Konfigurationsdatei festlegen. Der Name dieser Datei ist beliebig, er muss nur bei der Installation des Boot-Sektors angegeben werden. Das Listing 3.3 zeigt eine einfache Konfiguration; eine vollständige Dokumentation der Optionen finden Sie in den Quellen zu `GRUB`.

```
# Wartezeit bis zum Start des Default-Image
timeout 30

# Booten des Linux-Kernels von /dev/hda2
title Linux Boot
root      (hd0,1)
kernel    (hd0,1)/boot/vmlinuz-2.0.34

# Booten von Windows NT
title Windows NT boot menu
root      (hd0,1)
makeactive
chainloader +1
```

Listing 3.3 Eine einfache GRUB-Konfiguration

Die Konfigurationsdatei für GRUB kann unter Unix genauso bearbeitet werden wie unter DOS oder Windows. Wichtig ist nur, dass es eine Textdatei ist. Die Kodierung des Zeilenendes ist mit CR+LF oder mit LF erlaubt. Kommentare beginnen mit einer Raute (#) in Spalte eins.

GRUB hat, genau wie `lilo`, globale und lokale Optionen. Globale gelten für GRUB selbst bzw. für alle Images. Die globalen Optionen sind:

`timeout` *Sekunden*

Dieser Eintrag setzt den Timeout, die Zeit in Sekunden, die GRUB wartet, bevor das Default-Image gestartet wird.

`default` *Nummer*

Bestimmt das Image, das gestartet wird, wenn der Benutzer keine Eingabe vornimmt. Die Zählung der Images beginnt bei Null, so dass der Default von Null das erste Image bootet.

`fallback` *Nummer*

Wenn beim Booten des ausgewählten Image ein Fehler aufgetreten ist, dann versucht GRUB, das Fallback-Image zu starten. Das funktioniert natürlich nur, wenn der Fehler noch innerhalb von GRUB auftritt.

`password` *Passwort neue_Konfiguration*

Ohne diesen Eintrag kann jeder die einzelnen Images verändern oder neue Einträge generieren. Außerdem ist es möglich, im Kommandomodus einen beliebigen Kernel zu wählen und mit beliebigen Parametern zu booten. Dieser Eintrag macht diese Funktionen unzugänglich, solange nicht das korrekte Passwort eingegeben wird. Danach wird eine neue Konfigurationsdatei verwendet, in der man beispielsweise bereits Images für den Single-User-Modus definieren kann. Mit der Option `--md5` wird das Passwort mittels MD5 verschlüsselt gespeichert. Ein neues Passwort können Sie mit dem GRUB-Kommando `md5crypt` verschlüsseln und dann hier eintragen.

Jedes Image beginnt mit dem Eintrag `title` und wird implizit durch den Befehl `boot` beendet. Die Image-bezogenen Optionen sind:

`root Partition [hdbias]`

Verwendet die Partition `Partition` als root-Partition. Diese Partition wird von GRUB gelesen, um weitere Daten an den nächsten Lader weiterzugeben. Wenn die root-Partition nicht mit BIOS-Mitteln gelesen werden kann, weil sie z. B. hinter der 1-Gbyte-Grenze liegt, dann können Sie den Eintrag `rootnoverify` verwenden, der auf das Lesen der Partition verzichtet. Der optionale Parameter `hdbias` ist nur für BSD-Systeme interessant.

`kernel Kernel-Image ...`

GRUB lädt das angegebene Kernel-Image, das im Multiboot a.out-, ELF-, Linux- oder *BSD-Format vorliegen kann. Das Kommando selbst ignoriert den Rest der Zeile, dieser wird an den Kernel als Kommandozeile übergeben.

`initrd Datei ...`

Lädt die initiale RAM-Disk für einen Linux-Kernel und übergibt die weiteren Parameter an den Kernel.

`makeactive`

Markiert die als `root` angegebene Partition als »aktiv«. Das ist Voraussetzung für einige Betriebssysteme. Es können nur primäre Partitionen als aktiv markiert werden.

`chainloader`

Lädt die Datei oder den angegebenen Sektor als Chain-Loader. GRUB übergibt hierbei die Kontrolle z. B. an einen anderen Boot-Lader wie den von OS/2 oder Windows.

`pause Text`

Der angegebene Text wird auf den Bildschirm ausgegeben und GRUB hält danach an. Ein `^G` im Text sorgt dafür, dass ein Beep ausgegeben wird.

`uppermem Kilobytes`

Normalerweise ist GRUB in der Lage, auch mehr als 64 Mbyte Speicher zu erkennen und dem geladenen System mitzuteilen. Nur bei älteren Rechnern sollte es notwendig sein, mit Hilfe dieser Option die korrekte Speichergröße einzustellen.

`boot`

Startet das angegebene Image.

`module Datei ...`

Lädt ein Multiboot-Modul. Module können komprimiert gespeichert werden und werden automatisch dekomprimiert.

`modulenounzip Datei ...`

Wie die Option `module`, es wird aber nie die automatische Dekompression versucht.

GRUB hat noch keine Dateisysteme gemountet, kann aber Zugriffe auf Dateien über die Strukturen der verschiedenen Dateisysteme durchführen. Daher muss zu jedem Dateinamen zusätzlich der Name der Partition, auf der diese Datei gespeichert ist, angegeben werden. GRUB unterstützt alle unter Linux verbreiteten Dateisysteme. Die Syntax für die Angabe einer Partition ist:

`(Disk[,Partition[,BSD-Slice]])`

Als Disk kann `fd0` oder `fd1` für das erste bzw. zweite Diskettenlaufwerk verwendet werden; dann darf keine Partition angegeben werden. Wenn man `hd0/hd1` verwendet, kann der MBR angesprochen werden, indem keine Partition angegeben wird. Bei den Partitionen beginnt die Zählung wieder mit Null, so dass die erste Partition der ersten Festplatte als `(hd0, 0)` bezeichnet wird. Diese Bezeichnungen sind durch die Device-Namen in den freien BSD-Varianten beeinflusst und daher nicht die unter Linux gewohnten.

Zusätzlich zu den normalen Partitionen, wie sie von DOS, Windows, OS/2 und Linux verwendet werden, kann GRUB auch mit den Slices der *BSD-Systeme umgehen. Dabei wird an die Partitionsnummer der Buchstabe angehängt, der die entsprechende Slice identifiziert.

Wenn GRUB einen Dateinamen erwartet, dann können Sie diesen direkt an die Partition als absoluten Pfad anfügen. Bei einigen Funktionen ist es außerdem möglich, einen Offset oder eine Sektornummer anzugeben (mit `+Sektornummer`).

Wenn Sie die Kommandozeile von GRUB benutzen, dann werden Sie die Vervollständigung zu schätzen lernen. Der Editor ist an die Funktionen und die Tastenbelegung der `bash` angelehnt, aufgrund der geringen Größe aber nicht ganz so leistungsfähig. Die `[Tab]`-Taste zeigt entweder eine Liste der möglichen Befehle an oder mögliche Eingaben.

Die wichtigsten Kommandos für die interaktive Nutzung sind `kernel` zur Auswahl des zu startenden Kernels und `boot` zum Starten des ausgewählten Kernels. Beachten Sie, dass im Gegensatz zu einigen Boot-PROMs an Workstations die Kernel-Parameter nach der `kernel`-Option anzugeben sind. Das Kommando `boot` kennt keine weiteren Optionen.

Die Installation von GRUB

Die Installation von GRUB ist etwas schwieriger als diejenige von `lilo`. Bei `lilo` wird zunächst eine Konfigurationsdatei (`/etc/lilo.conf`) erstellt und diese dann von einem normalen Linux-Programm ausgewertet. Viele Distributionen enthalten ein bei der Installation automatisch startendes Konfigurationstool für `lilo`, so dass der Anwender zunächst wenig mit der Technik zu tun hat.

Das ist bei GRUB anders. Bisher enthält praktisch keine Distribution diesen Boot-Lader, außerdem fehlen Tools zur einfachen Konfiguration. Zudem besteht GRUB

nur aus den entsprechenden Boot-Sektoren, so dass jede Konfiguration ausschließlich beim Booten vorgenommen werden kann. Mit einigen Tricks kann man sich das Leben allerdings wieder etwas vereinfachen.

Am besten beginnen Sie damit, eine Boot-Diskette mit GRUB zu erstellen. Das kann unter Linux mit den in Listing 3.4 dargestellten Befehlen geschehen. Anschließend können Sie von dieser Diskette booten. Sie landen zunächst in der Kommandozeile. Hier können Sie ein Kernel-Image auswählen und booten oder GRUB installieren.

```
(linux):# dd if=bin/stage1 of=/dev/fd0 bs=512 count=1
(linux):# dd if=bin/stage2 of=/dev/fd0 bs=512 seek=1
```

Listing 3.4 Erstellen einer generischen GRUB-Diskette

Die weitere Installation erfolgt mit dem Befehl `install`. Wenn Sie den Rechner mit der generischen Boot-Diskette gestartet haben, dann müssen Sie den vollständigen Befehl eingeben. Wenn Sie bereits eine vorkonfigurierte GRUB-Installation haben, dann können Sie diesen Befehl auch in die Konfigurationsdatei aufnehmen. Die Syntax des `install`-Befehls lautet:

```
install Stage1 [d] Ziel Datei Adresse [p] [Konfiguration]
```

GRUB wird die Datei `Stage1` laden und prüfen, ob diese eine gültige GRUB-Datei ist. In diese Datei wird die Block-Liste der Stage2-Datei eingefügt und das Ergebnis als `Ziel` installiert. Die Adresse gibt an, in welchen Hauptspeicherbereich der Stage2 geladen werden soll. Unter Linux wird man `0x8000` verwenden, bei FreeBSD `0x2000`, wenn man den `Stage1_5`-Lader verwendet. Wenn die Option `p` angegeben ist oder eine Konfigurationsdatei angegeben wurde, dann wird in den ersten Sektor der Stage2-Datei der Name dieser Datei aufgenommen.

Das klingt alles sehr kompliziert, ist aber nur halb so wild. Nehmen Sie einen Stift und einen Zettel und schreiben Sie sich den Befehl auf. Die Befehle in Listing 3.5 habe ich verwendet, um GRUB zunächst nur auf einer Diskette zu installieren.

```
install (fd0)+1 (fd0) (hd0,2)/boot/grub/stage2 0x8000 p \
(hd0,2)/boot/grub/menu.lst
```

Listing 3.5 Installation von GRUB auf Diskette

Die Parameter im Einzelnen bedeuten:

```
(fd0)+1
```

Lese den ersten Sektor vom Diskettenlaufwerk und verwende diesen als Boot-Sektor.

(fd0)

Schreibe den Boot-Sektor wieder auf das Diskettenlaufwerk zurück, nachdem dort die Sektornummern der Stage2-Datei gespeichert wurden. Bei der Installation auf Festplatte geben Sie hier das Ziel an; entweder (hd0) für den MBR der ersten Platte oder (hd0, Partition) für die entsprechende Partition.

(hd0,2)/boot/grub/stage2

Verwende diese Datei aus der angegebenen Partition als Stage2-Lader. Die Sektornummern dieser Datei werden im Boot-Sektor der ersten Stage vermerkt. Der Programmcode hier liest dann eventuell die Konfigurationsdatei, lädt den gewünschten Kernel usw.

0x8000

Lade die Stage2 an die Adresse 0x8000. Unter Linux wird man keinen anderen Wert verwenden, unter FreeBSD kann es sinnvoll sein, Stage1_5 einzusetzen und die Adresse 0x2000 zu verwenden.

P

Vermerke den Namen der Konfigurationsdatei im Stage2-Lader.

(hd0,2)/boot/grub/menu.lst

Die Datei /boot/grub/menu.lst auf der Partition /dev/hda3 wird als Konfigurationsdatei verwendet.

Das große Problem bei der Installation von GRUB ist, dass kein Linux-Programm die Konfiguration vorbereitet, so dass man zumindest einmal den komplizierten `install=-`Befehl eingeben muss. Später kann man diesen Befehl allerdings in die Konfigurationsdatei aufnehmen, so dass er vor dem eigentlichen Booten ausgeführt wird.

Ein weiteres Feature von GRUB ist, dass zum Booten des Kernels auch BOOTP oder DHCP verwendet werden kann. Genaueres dazu finden Sie in der Info-Dokumentation zu GRUB.

Lohnt sich GRUB wirklich?

GRUB hat einige sehr nützliche Vorteile, vor allem die Verwendbarkeit für Linux und *BSD-Systeme und die sehr flexible Konfiguration zur Laufzeit. Dafür muss man aber auch einige Nachteile in Kauf nehmen, insbesondere die komplexe Konfiguration und die geringe Unterstützung durch Tools und Distributionen. Dennoch kann sich der Einsatz von GRUB lohnen. Im Zweifelsfall installieren Sie GRUB auf einer Diskette und probieren es aus.

3.4.3 Der Boot-Lader LOADLIN

LOADLIN und `loadlinX` von Hans Lermen sind Programme, die unter MS-DOS laufen und es erlauben, Linux von MS-DOS aus zu starten. `loadlinX`² ist ein Präprozessor für LOADLIN, der eine Umsetzung der Kommandozeilenparameter vornimmt. Zusammen mit dem UMSDOS-Dateisystem ermöglicht LOADLIN ein »Reinschnuppern« in Linux, ohne dass eine Neupartitionierung der Festplatte notwendig wird. Diese Möglichkeit stellt insbesondere für Linux-Neulinge eine einfachere und sicherere Alternative zu `lilo` dar. LOADLIN ist auch dann nützlich, wenn etwa eine Soundkarte zunächst unter DOS mit einem speziellen Programm initialisiert werden muss, bevor sie unter Linux benutzt werden kann.

Um LOADLIN zu benutzen, muss zunächst Linux installiert werden (manche Distributionen verwenden LOADLIN auch auf den Bootmedien). Der bei den meisten Distributionen folgende Schritt, `lilo` in den Boot-Sektor zu installieren, wird jedoch übersprungen. Das Erstellen einer Boot-Diskette hingegen muss erfolgen, damit Linux zunächst von der Diskette gebootet werden kann. Der Kernel (`/vmlinuz`) wird dann auf die DOS-Partition kopiert. Sofern die DOS-Partition noch nicht durch die Installationsroutine gemounted wurde, kann dies z. B. via `mount -t msdos /dev/hda1 /mnt per` Hand nachgeholt werden.

Das Programm LOADLIN bekommt als Parameter den DOS-Dateinamen des zu startenden Kernel-Image übergeben, z. B. `c:\linux\vmlinuz`. Das zweite Argument, z. B. `root=/dev/hda2`, enthält die als `root`-Dateisystem zu benutzende Partition. Nun wird nur noch die Angabe benötigt, ob die `root`-Partition `read-only` (`ro`) oder `read-write` (`rw`) eingehängt wird. Eine typische Kommandozeile sehen Sie in Listing 3.6.

```
C:\> c:\linux\loadlin c:\linux\vmlinuz root=/dev/hdb2 rw
```

Listing 3.6 Ein typischer Aufruf von loadlin

Wenn UMSDOS als Dateisystem verwendet wird, ist zu beachten, dass anstelle von LOADLIN der Präprozessor `loadlinX` benutzt werden sollte. Bei der Verwendung von `loadlinX` wird im Parameter `root=` als Argument nur noch das zu benutzende Laufwerk, z. B. die primäre DOS-Partition `C:` oder auch eines der Diskettenlaufwerke `A:` oder `B:`, übergeben. Eine Kommandozeile unter Verwendung von `loadlinX` sehen Sie in Listing 3.7.

```
C:\> c:\linux\loadlinX c:\linux\vmlinuz root=c: rw
```

Listing 3.7 Ein typischer Aufruf von loadlinX

2. Auch DOS 6.2 unterscheidet nicht zwischen Groß- und Kleinschreibung bei Befehlen; das große X in `loadlinX` dient nur zur Hervorhebung der unterschiedlichen Programmnamen.

Zu beachten ist, dass bei der Benutzung von UMSDOS das schreibgeschützte Einfügen des Dateisystems (`ro/read-only`) nicht möglich ist. Spezielle Boot-Parameter nur für `LOADLIN` sind nicht vorhanden.

Ein wesentlicher Vorteil von `LOADLIN` ist, dass weder der MBR noch der Startsektor der Linux-Partition verändert werden muss. Damit kann man problemlos zwischen den verschiedenen Systemen wechseln und diese auch aktualisieren. Hat man `lilo` in den MBR installiert, so wird er dort z. B. von der NT- oder Windows 95-Installation überschrieben. Außerdem stört er dort, wenn man Linux wieder entfernt.

Um das Laden mit `LOADLIN` zu automatisieren, können Sie unter DOS einen Boot-Selektor wie `BOOT.SYS` verwenden. Unter Linux sollten Sie ein Skript `/sbin/installkernel` erstellen, das den Kernel beim `make install` auf die DOS-Partition kopiert.

3.4.4 Der Boot-Lader SYSLINUX

`SYSLINUX` von H. Peter Anvin ist ein Boot-Lader, der den MS-DOS-Boot-Sektor bzw. die Dateien `IO.SYS` und `MSDOS.SYS` auf Disketten ersetzt. `SYSLINUX` selbst ist ein MS-DOS-Programm, mit dem aus MS-DOS-formatierten Disketten Linux-Boot-Disketten erstellt werden können. Unter DOS (oder mittels `mcopy`) werden eine oder mehrere Kernel-Dateien auf die MS-DOS-Diskette kopiert. Anschließend werden mit dem DOS-Befehl aus Listing 3.8 das Image in der Datei `a:vmlinux` als zu ladendes Kernelimage und die Boot-Parameter `root=/dev/hda1` als zu übergebendes Argument eingetragen. `SYSLINUX` ändert den Boot-Sektor der Diskette und überträgt die Datei `LDLINUX.SYS` darauf.

```
c:\> syslinux a: vmlinux root=/dev/hda1
```

Listing 3.8 Ein typischer Aufruf von `syslinux`

Beim Starten von einer mittels `syslinux` erzeugten Boot-Diskette wird, ähnlich wie bei `lilo`, nach dem Drücken einer der Tasten `[Umschalt]`, `[Alt]`, `[Umschalt-Fest]` oder `[Rollen]` ein Boot-Prompt angezeigt. An dieser Stelle ist es möglich, andere Parameter als den Default-Boot-Parameter und einen anderen als den fest eingestellten Kernel-Dateinamen anzugeben, sofern er auf der Diskette vorhanden ist.

Falls auf der Diskette eine Datei namens `LINUXMSG.TXT` existiert, wird deren Inhalt beim Starten angezeigt. In diesem Fall wird auch der Boot-Prompt ausgegeben, auch wenn keine der oben genannten Tasten gedrückt wurde.

Mit `syslinux` erstellte Boot-Disketten können ohne Probleme mit dem DOS-Programm `DISKCOPY` kopiert werden. Da jedoch immer mehr Distributionen mit boot-fähigen CDs ausgestattet sind, wird dieses Verfahren nur noch relativ selten verwendet.

3.5 Start des Kernels

Nachdem einer der Boot-Lader das Kernel-Image, z. B. `/vmlinuz`, von der aktiven Partition in den Hauptspeicher geladen hat, muss der Kernel zunächst dekomprimiert werden. Der zusätzliche Aufwand, den die CPU hierbei leisten muss, fällt nicht ins Gewicht, da sie zu dieser Zeit überwiegend »Däumchen dreht« und die meiste Zeit auf das Eintreffen von Daten aus dem »Flaschenhals« DiskIO wartet. Insgesamt ist das Laden komprimierter Daten sogar schneller. Würde demgegenüber in einem System die CPU den Flaschenhals darstellen, so wäre das Benutzen unkomprimierter Daten günstiger; beim gegenwärtigen Stand der Bus- und Festplattentechnik ist dies aber in absehbarer Zeit nicht zu erwarten. Ein weiterer, in diesem Fall aber nicht so gewichtiger Vorteil der Kompression ist die Ersparnis von Plattenplatz.

Die Verwendung komprimierter Kernel wurde notwendig, da sie unkomprimiert größer als 704 Kbyte wurden (640 Kbyte »konventioneller DOS-Speicher« zuzüglich 64 Kbyte ungenutzter Datenbereich der Grafikkarten unter DOS) und der Prozessor sich zu diesem Zeitpunkt noch im Real Mode befindet, in dem nur Speicheradressen unterhalb von 1 Mbyte zur Verfügung stehen.

Die Einführung der komprimierten Kernel hat das eigentliche Problem nur verdeckt. Durch die vielen Treiber, die in den Kernel eingebunden werden können, stößt man jetzt wieder auf die Beschränkungen des Real-Modes, da der Kernel zunächst in den Speicher unterhalb 1 Mbyte geladen werden muss. Früher oder später wird der Boot-Lader im Protected-Mode laufen, so dass die Größe des Kernels nur noch durch den realen Hauptspeicher begrenzt ist.

Daher hat man mit dem `bzImage` und der initialen RAM-Disk `initrd` Möglichkeiten geschaffen, um diese Grenzen zu umgehen. Wenn Ihr Kernel wirklich so groß geworden ist und Sie keine Module verwenden können, so finden Sie die notwendigen Informationen in der Datei `./Documentation/initrd.txt` in den Kernel-Quellen bzw. in der `lilo`-Dokumentation.

Die letzte Amtshandlung des Boot-Laders ist die Übergabe der Programmkontrolle an das entpackende (und logischerweise selbst nicht komprimierte) »Präfix«, das beim Kompilieren des Systems mittels der in dem Verzeichnis `/usr/src/linux/arch/i386/boot/compressed` liegenden Programme `xtract` und `piggyback` angefügt wurde. Der Prozessor wird jetzt in den Protected-Mode umgeschaltet, damit die Dekompressionsroutine das Image auf (virtuelle) Adressen ab drei Gigabyte (`0xc0000000`) entpacken kann. Der Programmzähler wird auf diese Adresse gesetzt. Somit wird die Programmkontrolle ein weiteres Mal, und diesmal an den eigentlichen Kernel, übergeben.

Unter dem Aspekt der Konfigurationsmöglichkeiten betrachtet, können beim Systemstart nach der Auswahl des Betriebssystems zwei Abschnitte unterschieden werden:

- Ermittlung und Überprüfung der Hardware-Ausstattung und
- Abarbeiten der Konfigurationsdateien und automatisches Starten von Programmen.

Im ersten Abschnitt (siehe: `arch/i386/boot/setup.S`) werden bereits vor dem Entpacken des Kernels Teile der Hardware-Ausstattung ermittelt. Dazu werden die entsprechenden BIOS-Variablen (für Disketten und Festplattenparameter sowie die angeschlossene Grafikkarte) ausgelesen und an einem sicheren Platz im Hauptspeicher vermerkt. Weitere Hardware-Tests erfolgen nach der Umschaltung in den 32-Bit-Modus, dazu jedoch später mehr (zur exakten Reihenfolge vgl.: `./linux/init/main.c`). Zunächst werden jedoch interne Funktionen und Tabellen des Systems initialisiert:

- Die Tabelle für die Speicherseitenverwaltung,
- die Belegung der Fehlerinterrupts und der IRQs,
- die Prozesstabelle.

Anschließend wird der Scheduler gestartet und der Bildschirm initialisiert. Ab hier werden Meldungen nicht nur auf den Bildschirm ausgegeben, sondern auch mittels des `syslogd` verwaltet und möglicherweise in einer Datei gespeichert. Mehr zur Verwendung von `syslogd` finden Sie im Abschnitt 4.4.4, »System-Logs«. Diese Meldungen können später mit dem Programm `dmesg` erneut angezeigt werden.

An dieser Stelle erfolgt, im Zusammenhang mit der Initialisierung der entsprechenden Gerätetreiber, die bereits erwähnte zweite Phase des Hardware-Tests. Die Hardware wird mittels verschiedener Tests untersucht (so genanntes Auto-Probing oder Auto-Detect). Anhand der Bildschirmmeldungen können Sie verfolgen, welche Geräte gefunden wurden. Wird hier für eine Controller-Karte keine oder eine falsche Meldung (z. B. andere IO-Ports oder Interrupts) ausgegeben, so werden Sie dieses Gerät später nicht ansprechen können. Eine Ursache für solche Probleme kann darin bestehen, dass mehrere Karten auf einen Interrupt oder dieselben IO-Ports eingestellt sind (entweder mittels Jumper oder Software-Konfiguration). Die automatische Hardware-Erkennung kann mit Hilfe von Kommandozeilenparametern für den Kernel beeinflusst werden. Lesen Sie dazu auch die aktuellen `README`-Dateien in den Kernel-Quellen.

Jetzt wird der Prozessor vom privilegierten (auch Kernel-Mode) in den nicht privilegierten Modus³ (User-Mode) geschaltet; ab jetzt läuft der erste Linux-Prozess (mit der »Prozessnummer« 0). Dieser Prozess unterscheidet sich nicht grundsätzlich von allen anderen Linux-Prozessen, läuft aber nur dann, wenn kein anderer Prozess aktiv ist. Es handelt sich beim Prozess 0 um den so genannten `idle`-Prozess, der im Fall der Prozessor-Nichtnutzung 100% der Systemressourcen belegt.

3. Der privilegierte Modus ist nicht zu verwechseln mit dem Protected-Mode.

Vom Prozess 0 aus wird mittels des Systemaufrufs `fork()`⁴ der Prozess 1 gestartet, der wiederum die weitere Initialisierung des Systems in Gang setzt. Hier ist die Hardware im Prinzip initialisiert, die `root`-Partition angehängt und das Programm `init` wird ausgeführt. `init` wird bei Unix-Systemen häufig als »Vater aller Prozesse« bezeichnet, was für Linux nicht ganz korrekt ist; eigentlich ist es hier der `idle`-Prozess mit der Nummer 0. Es wird noch ein weiterer Prozess gestartet, der für das regelmäßige Update der internen Informationen des Dateisystems auf der Festplatte sorgt. Das Verhalten dieses Prozesses kann mit dem Programm `bdflush` verändert werden.

Gemäß des Filesystem-Hierarchie-Standards (FHS-Version-2.0) sollte sich das `init`-Programm in `/sbin/init` befinden. Die tatsächliche Suchreihenfolge, die in `linux/init/main.c` festgelegt ist, lautet jedoch `/sbin/init`, `/etc/init` und `/bin/init`. Wenn `init` dort nicht gefunden wird, wird versucht, `/bin/sh` zu starten, um die Instandsetzung eines derart »unordentlichen« Systems zu ermöglichen. Mit aktuellen Kernel-Versionen ist es möglich, mit dem Kommandozeilen-Parameter `init=/bin/sh` eine Shell anstelle von `init` zu starten. Achten Sie hier darauf, dass dieser Befehl mit `root`-Privilegien gestartet wird, und verwenden Sie, falls erforderlich, die Option `password=` in der `/etc/lilo.conf`.

Zum Abschluss der Initialisierung meldet sich der Kernel mit seiner Versionsnummer sowie seinem Erstellungsdatum. Zuvor wird jedoch das `init`-Programm gestartet. Damit ist der Systemstart aus der Sicht des Kernels abgeschlossen. Das System ist aber zu diesem Zeitpunkt noch nicht benutzbar, da keine Anmeldung erfolgen kann und keine Hintergrundprozesse (Dämonen) gestartet sind.

Das Verfahren wird etwas komplizierter, wenn ein Kernel mit der `initrd`-Option gestartet wird. In diesem Fall wird vor dem Start von `init` die initiale RAM-Disk gemountet und das dort enthaltene Programm `linuxrc` gestartet. Damit ist es möglich, einen vollständig modularisierten Kernel zu verwenden, der auch die Festplattentreiber dynamisch lädt. In Zukunft wird man versuchen, viele Funktionen des Auto-Probing hier unterzubringen.

3.5.1 Parameter für den Kernel

Viele Initialisierungen (z. B. IO-Adressen für Erweiterungskarten) des Kernels können durch Kommandozeilenparameter beeinflusst werden. Diese Parameter werden dem Kernel vom Boot-Lader übergeben.

4. Dieses `fork()` arbeitet insofern anders, als der Prozess 0 eine Sonderrolle einnimmt: Er arbeitet mit demselben Speicherbereich wie Prozess 1, insbesondere mit demselben Stack. Deshalb führt Prozess 0 auch nur Operationen aus, die den Stack nicht benutzen. Zitiert nach [Beck2001].

`debug`

Setzt `console_loglevel` auf 10 und schaltet somit die ausführlichen Konsolenmeldungen ein.

`mem=size`

Setzt das Ende des physikalischen Speichers. Dies kann notwendig sein, wenn die RAM-Größe nicht automatisch ermittelt werden kann, z. B. bei Rechnern mit mehr als 64 MB RAM. Es kann auch nützlich sein, um defekten Speicher nicht zu verwenden, ohne diesen auszubauen oder die Performance eines Systems mit weniger Speicher zu prüfen.

`no387`

Ein eventuell vorhandener mathematischer Coprozessor wird nicht benutzt.

`no-hlt`

Verhindert das Benutzen bzw. Ausführen der HLT-Instruktion neuerer Intel-(kompatibler-)Prozessoren, wenn das System idle ist. Durch die Verwendung dieses Befehls wird die CPU weniger heiß und Laptops verbrauchen deutlich weniger Strom. Allerdings kann diese Einstellung bei manchen Rechnern dazu führen, dass diese beim Booten einfach hängen bleiben.

`reserve=port1,num1[,portn,numn]`

Nimmt IO-Ports und -Bereiche vom Auto-Probing aus. Dies ist zum Beispiel dann notwendig, wenn das Auto-Probing zum »Hängen« des Systems führt. `port1` stellt die erste Adresse dar, `num1` gibt die Anzahl der ab `port1` auszublenkenden Adressen an.

`root=device`

ändert das Gerät, das als `root`-Dateisystem verwendet wird. Hiermit werden die Angaben im Kernel-Image überschrieben. `device` ist entweder ein Geräte-name wie etwa `/dev/hda3` oder eine hexadezimale Gerätenummer. Das Root-Device kann auch mit Hilfe des Programms `rdev` in das Kernel-Image geschrieben werden.

`ro` bzw. `rw`

Die root-Partition wird read-only oder read-write in den Verzeichnisbaum eingefügt.

3.6 Tipps und Tricks zur Boot-Konfiguration

Es ist leicht, das System unbenutzbar zu machen, wenn man an der Boot-Konfiguration herumbastelt. Leider sind derartige Fehler nicht einfach zu beheben, so dass man hierbei wirklich vorsichtig sein sollte. Auf den folgenden Seiten möchte ich einige Hinweise geben, wie man derartige Fehler verhindern bzw. wie man sein System retten kann.

Der erste und wichtigste Tipp ist, dass man immer ein zweites Boot-Medium greifbar haben sollte. Das kann entweder die Boot-Diskette (oder CD-ROM) der Distribution sein oder eine selbsterstellte Boot-Diskette. Wenn Sie einen eigenen Kernel erstellen (siehe auch Abschnitt 4.2.1, »Konfigurieren des Kernels«), dann können Sie den neuen Kernel zunächst mittels `make bzdisk` auf eine formatierte Diskette schreiben und von dieser booten.

Wenn Sie `lilo` verwenden, dann sollten Sie zusätzlich zu den beiden Kerneln `/boot/vmlinuz` und `/boot/vmlinuz.old`, die bei der Kernel-Installation mit `make install` bzw. `make bzlilo` automatisch erzeugt werden, eine zusätzliche Version aufheben. Wichtig ist, dass nach jeder Änderung an der Datei `/etc/lilo.conf` oder einem Kernel-Image unbedingt neu aufgerufen werden muss. Warum? `lilo` merkt sich beim Aufruf die Sektoren, die das Kernel-Image enthalten, in der Map-Datei und diese Sektoren ändern sich.

Bevor Sie `lilo` tatsächlich einsetzen, können Sie mit der Option `boot=/dev/fd0` in der Datei `/etc/lilo.conf` dafür sorgen, dass `lilo` nur den Boot-Sektor auf der Diskette verändert. Wenn Sie damit zufrieden sind, dann können Sie später den Boot-Sektor an eine geeignete Stelle auf der Festplatte kopieren (bzw. die `lilo`-Konfiguration ändern und `lilo` erneut aufrufen). Ein wichtiger Vorteil ist hier, dass Sie nicht in die Konfiguration Ihres laufenden Systems eingreifen, von Nachteil ist allerdings, dass Sie von Diskette booten müssen (mit der üblichen Gefahr von Boot-Sektor-Viren). Ich persönlich setze diese Funktion gerne zum Test einer neuen Konfiguration ein.

Wenn Sie die Wahl haben, wo Sie den `lilo`-Boot-Sektor installieren möchten, dann verwenden Sie möglichst den Boot-Sektor einer Linux-Partition und markieren Sie diese als »aktiv« (boot-fähig). Damit lassen Sie den Master-Boot-Record unverändert und können mittels `fdisk` schnell ein anderes System aktivieren. In der `lilo`-Konfiguration können dann andere Systeme zur Auswahl angeboten werden.

Wenn Sie bereits Windows auf Ihrem System haben und den Boot-Manager dieser Systeme verwenden wollen, dann geht das natürlich auch. Das hat den Vorteil, dass man das bisherige Boot-Verfahren praktisch unverändert lassen kann. Mehr zu diesem Thema finden Sie im nächsten Abschnitt.

Wenn Sie Boot-Probleme haben, dann ist das wichtigste Ziel, keine Daten zu verlieren und möglichst schnell wieder ein lauffähiges System zu erhalten. Je nach Fehlerquelle und Situation sind verschiedene Strategien erforderlich. Die folgende Liste versucht, einen Überblick über die notwendigen Schritte zu geben – ein Kochrezept für jede Situation kann es aber nicht sein. Seien Sie vorsichtig!

- Bei Fehlern in den Boot-Skripten, die von `init` aufgerufen werden, kann es ausreichen, im Single-User-Mode zu booten und dann das Skript zu korrigieren oder zu deaktivieren.

- Wenn das Programm `init` nicht mehr startet, dann kann man möglicherweise mit der Kernel-Option `INIT=/bin/sh` eine Shell starten und mit Reparaturmaßnahmen beginnen. Vielleicht haben Sie die Stand-Alone-Shell `sash` installiert – diese enthält viele Befehle als Builtins, die man zur Rettung gebrauchen kann.
- Falls der aktive Kernel nicht bootet, so kann man möglicherweise eine ältere Kernel-Version (`/boot/vmlinuz.old`) verwenden. Achtung: Wenn Sie den neuen Kernel einfach mittels `make bzlilo` neu übersetzen, überschreiben Sie auch die vorherige Sicherungskopie.
- Das System bootet gar nicht: Verwenden Sie eine Boot-Diskette oder die CD-ROM der Distribution.

In vielen Fällen werden nicht alle Partitionen eingehängt sein, dann können Sie diese manuell mit einhängen. Wenn die Root-Partition nur im Lesezugriff eingehängt ist, so müssen Sie diese erneut mit den richtigen Optionen einhängen. Auf meinem System geht das mit `mount -n -o remount,rw /dev/root`, wobei ich als Root-Device `sda2` verwende.

3.7 Der `init`-Prozess

Das gestartete `init`-Programm arbeitet seine Konfigurationsdatei `/etc/inittab` ab und startet die darin eingetragenen Programme und Prozesse. Wenn kein `init`-Programm gefunden wird, versucht der Kernel, eine Shell zu starten und `/etc/rc` darin auszuführen. Das Format der `/etc/inittab` ist bei `simpleinit` und `System-V-init` unterschiedlich.

Das Mischen verschiedener `inits` und `inittabs` kann das System unbrauchbar machen. Es ist in jedem Fall äußerst wichtig, vor der Umstellung einer `init`-Methode auf eine andere Backups der betroffenen Dateien und Programme zu machen sowie eine Boot- und Root-Diskette einschließlich Editor zu erstellen. Eine Umstellung ist nur dann sinnvoll, wenn Sie sich einen deutlichen Gewinn versprechen. Ein guter Grund ist die Verwendung von `simpleinit` auf einem Rechner mit sehr wenig Speicher, allerdings wird man dieses nur sehr selten wirklich tun. Aus diesem Grund werde ich mich hier auf das `System-V-init` konzentrieren.

3.7.1 Parameter für `init`

Folgende Parameter werden von allen unter Linux verwendeten `init`-Varianten erkannt. Diese Parameter können vom Kernel an `init` weitergereicht werden.

`single`

Startet das System im Single-User-Modus. Alle für Linux verfügbaren `init`-Programme verstehen mindestens den Parameter `single`.

`auto`

Startet das System so, wie es in der Konfigurationsdatei `/etc/inittab` festgelegt wurde.

3.7.2 Die init-Konfiguration in der Datei `/etc/inittab`

Das Programm `init` wird mit Hilfe der Datei `/etc/inittab` konfiguriert. Ein `inittab`-Eintrag sieht sowohl beim echten System-V-`init` als auch beim System-V-ähnlichen `init` schematisch wie folgt aus:

id:runlevel:action:process

`id`

Steht für einen eindeutigen Code zur Identifizierung eines Eintrags. Dieser Code darf bis zu vier Zeichen enthalten.

`runlevel`Runlevel

Kann Werte von 1 bis 6 sowie S und A bis C annehmen (Groß- bzw. Kleinschreibung wird ignoriert). S steht z. B. für den Single-User-Modus, in dem der Systemadministrator üblicherweise Systemarbeiten durchführt. Wenn `init` als `telinit` aufgerufen wird, stehen darüber hinaus auch noch die Runlevel A, B und C zur Verfügung, mit denen spezielle Einstellungen via `inittab` aufgerufen werden können. Dies kann zum Starten bzw. Stoppen von verschiedenen Diensten genutzt werden.

`action`

Gibt an, was in Bezug auf den Prozess bei einem Neustart (des Systems) oder seiner Beendigung geschieht (z. B. Neustart des Prozesses). In Tabelle 3.1 finden Sie eine Übersicht über die möglichen Aktionen.

`process`

Enthält den Namen des zu startenden Programms mit den beim Start zu übergebenden Parametern.

Aktion	Beschreibung
<code>respawn</code>	Der Prozess wird nach Beendigung erneut gestartet.
<code>wait</code>	<code>init</code> wartet auf das Ende des Prozesses.
<code>once</code>	Der Prozess wird beim Wechsel in diesen Runlevel gestartet.
<code>boot</code>	Der Prozess wird beim Systemstart erzeugt.
<code>bootwait</code>	Wie <code>boot</code> , aber <code>init</code> wartet auf das Ende des Prozesses.
<code>off</code>	Eintrag ohne Funktion.
<code>ondemand</code>	Start bei der Anforderung des Runlevel, aber kein Wechsel des Runlevel.
<code>initdefault</code>	Default-Runlevel.
<code>sysinit</code>	Start vor <code>boot</code> bzw. <code>bootwait</code> .

Aktion	Beschreibung
powerwait	Signal der UPS über Fehler in der Stromversorgung. Der <code>powerd</code> -Dämon muss laufen.
powerfail	Wie <code>powerwait</code> , aber <code>init</code> wartet nicht auf das Prozessende.
powerokwait	Stromversorgung ist wieder ok.
ctrlaltdel	An der Konsole wurde <code>[Strg]+[Alt]+[Entf]</code> gedrückt.
kbrequest	Bearbeitung einer speziellen Tastenkombination.

Tabelle 3.1 Aktionen in der `inittab`

Das Listing 3.9 zeigt ein Beispiel für die Datei `/etc/inittab`. Eine ausführliche Dokumentation des Formats finden Sie auch in der Manpage zu `inittab(5)`.

Das Programm `telinit` ist ein Link auf `init` und bietet über die zusätzlichen Argumente `a`, `b` und `c` die erweiterte Möglichkeit zur Kontrolle des Systems: Es werden jeweils nur die Einträge mit dem Runlevel `a`, `b` und/oder `c` angesprochen. Es können z. B. nur die Programme des speziellen Runlevel `a` neu gestartet werden. Zwischen den numerischen Runleveln (1 bis 6) kann ebenfalls mit dem Programm `telinit` gewechselt werden. Mit dem Parameter `q` wird das Programm `init` veranlasst, die Datei `/etc/inittab` neu einzulesen.

Dabei sind einige Runlevel reserviert, andere haben eine historisch gewachsene Bedeutung. In Tabelle 3.2, finden Sie eine entsprechende Übersicht, so wie die Runlevel in der »Linux Standard Base« definiert sind.

Durch die differenziertere Verwendung der verschiedenen Runlevel ist die Initialisierung des Systems einfacher und flexibler zu steuern. Außerdem ist es wesentlich übersichtlicher, welche Dämonen in welchem Runlevel gestartet bzw. gestoppt werden sollen. Zum Editieren der Runlevel kann das Programm `tksysv` verwendet werden, das Bestandteil einiger Distributionen ist.

Runlevel	Bedeutung
0	Halt
1	Single-User-Modus (reserviert)
2	Multi-User-Modus ohne Netzwerkdienste
3	Multi-User-Modus als Netz-Server
4	reserviert für den Systemverwalter
5	X-Workstation (<code>xdm</code>)
6	Reboot

Tabelle 3.2 Die verschiedenen Runlevel und deren Bedeutung

Der Ablauf eines Systemstarts ist zunächst ähnlich, wie in den vorigen Kapiteln beschrieben. Zunächst werden, nach dem Laden des Kernels und dem Start von

init, die Dateisysteme geprüft und die stets notwendigen Dämonen gestartet. Anschließend wird das Skript `/etc/rc` gestartet. Dieses Skript erhält als Parameter den gewünschten Runlevel. Den entsprechenden Aufruf in der Datei `inittab` finden Sie in Listing 3.9.

```
# Default runlevel.
id:5:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# /etc/init.d/rc takes care of runlevel handling.
10:0:wait:/etc/init.d/rc 0
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2
13:3:wait:/etc/init.d/rc 3
14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6
```

Listing 3.9 Änderungen in der Datei `/etc/inittab`

Für jeden Runlevel existiert ein Verzeichnis `/etc/rc.d/rcRunlevel.d`. Jedes dieser Verzeichnisse enthält Links auf so genannte Start- und Stop-Skripten. Die Namen der Start- bzw. Stop-Skripten beginnen mit einem `s` bzw. `k`, gefolgt von einer zweistelligen Nummer. Die Skripten werden aufsteigend bezüglich dieser Nummer sortiert gestartet. Damit ist es möglich, zunächst das Netzwerkinterface zu aktivieren und dann die Client- und Server-Prozesse zu starten. Der letzte Teil des Namens beschreibt normalerweise den Typ des Services, der mit diesem Skript gesteuert wird.

Die Skripten selbst sind im Verzeichnis `/etc/init.d` gespeichert. In den Verzeichnissen für die entsprechenden Runlevel werden nur symbolische Links auf die Skripten gelegt, die gestartet bzw. gestoppt werden sollen. Das hat den Nachteil, dass die Konfiguration der gestarteten Dämonen nicht übersichtlich in einer Datei vorgenommen wird, sondern durch Links in einem Dateisystem. Ein Vorteil ist, dass dieses System sehr flexibel ist und auch auf anderen Unix-Systemen eingesetzt wird.

Die Skripten werden mit dem Parameter `start` bzw. `stop` aufgerufen und starten oder stoppen die entsprechenden Subsysteme. Damit ist es sehr einfach möglich, einen einzelnen Dienst manuell anzuhalten und neu zu starten (z. B. weil man die Konfiguration geändert hat oder den Dienst temporär nicht benötigt). In letzter Zeit sind bei einigen Distributionen weitere mögliche Parameter hin-

zugekommen, wie beispielsweise `restart`, `status` oder `list`. Im Zweifelsfall gibt das Skript eine entsprechende Hilfe aus, wenn man es ohne Parameter aufruft.

In Listing 3.10 finden Sie eine Übersicht über die auf meinem System verwendeten Skripten. Eine besondere Bedeutung haben die Stop-Skripten, die beim Wechsel in einen Runlevel bestimmte Services stoppen. Die Namen der Stop-Skripten beginnen mit `K`. Innerhalb eines Runlevels werden die Skripten nach aufsteigender Nummer abgearbeitet.

```
(linux):/etc> ls -F rc*
rc0.d:
S20halt@

rc1.d:
S20single@

rc2.d:
K51syslog@ K60lpd@ S51syslog@ S60lpd@
K55cron@ K95gpm@ S55cron@ S95gpm@

rc5.d:
K51syslog@ K60lpd@ S51syslog@ S60lpd@
K55cron@ K95gpm@ S55cron@ S95gpm@

rc6.d:
S20reboot@
```

Listing 3.10 Skripten für das System-V-init

In der Regel wird sich der Umstieg auf ein anderes `init`-Verfahren nicht lohnen. Andererseits ist die Kompatibilität mit anderen (kommerziellen) Systemen sicherlich sinnvoll, insbesondere wenn man ständig zwischen verschiedenen Systemen wechselt. Dann ist es besonders sinnvoll, die Runlevel so zu belegen, wie man es von anderen Systemen her gewohnt ist.

Bei Rechnern mit wenig Hauptspeicher kann es sinnvoll sein, nur möglichst wenige und kleine Dämonen zu starten. In diesem Fall eignet sich z. B. das `simpleinit` in Kombination mit dem abgespeckten `bdflysh` (ein erweiterter `update`-Dämon) statt eines `System-V-init` mit dem normalen `bdflysh`. Beide Programme sind im Paket `util-linux` enthalten.

Wenn Sie Ihr System von einer Variante des `init`-Programms auf eine andere umstellen wollen, so müssen Sie für alle Fälle eine Boot- und Root-Diskette bereithalten. Nur mit einer Boot-Diskette kommen Sie nicht weiter, da möglicherweise das Programm `init` auf der Festplatte nicht funktionsfähig ist. Im schlimmsten Fall können Sie das System auch nicht im Single-User-Modus starten und müssen von Diskette booten. Glücklicherweise müssen Sie beim Um-

stieg vom System-V-ähnlichen `init` auf das System-V-`init` nur die Datei `/etc/inittab` verändern (und möglicherweise diese Änderungen zurücknehmen). Alle übrigen betroffenen Dateien werden durch gänzlich andere Dateien mit völlig anderen Namen ersetzt.

Das unter Linux verwendete `init` weicht in einer Beziehung von denen anderer Systeme ab: Es wird direkt in den angeforderten Runlevel gewechselt und nicht der Reihe nach alle kleineren Runlevel durchlaufen.

3.7.3 Stoppen des Systems

Unter Linux laufen, wie bei jedem anderen Unix-System, viele Prozesse im Hintergrund. Vor dem Ausschalten des Systems müssen diese Prozesse beendet werden, da sonst möglicherweise Daten verlorengehen. Ein weiterer Grund für das geordnete Stoppen des Systems ist, dass Linux auch Schreibzugriffe im Hauptspeicher puffert. Nach dem Ende eines Programms sind also die Daten noch nicht auf der Festplatte verewigt. Dies erfolgt erst nach einem `sync`, der aber regelmäßig vom `update`-Dämon durchgeführt wird.

Das Standardverfahren zum Stoppen eines Unix-Systems ist der Aufruf des Befehls `shutdown`. Je nach Kommandozeilenparametern kann die Reaktion des Systems unterschiedlich aussehen. Als Parameter muss eine Zeit angegeben werden, nach der das System heruntergefahren wird. Der Zeitpunkt `now` steht für das sofortige Stoppen des Systems, andernfalls wird eine Zeit in Minuten angegeben. Vor dem Shutdown werden alle angemeldeten Benutzer gewarnt, dass eine Systemwartung bevorsteht.

Mit dem zusätzlichen Parameter `-h` (Halt) wird das System angehalten. Der Parameter `-r` (Reboot) sorgt für einen Neustart des Systems. Wird der Parameter `-f` angegeben, so wird ein fast-Reboot ohne Überprüfung der Dateisysteme durchgeführt. Dies ist in den `rc`-Skripten oft nicht mehr implementiert, da das Extended-2-Dateisystem über ein eigenes Flag für ein ordnungsgemäß heruntergefahrenes System verfügt.

Zur Vereinfachung existieren für häufig vorkommende Aufgaben entsprechende Abkürzungen. So steht je nach `init` z. B. `reboot` oder `init 6` für einen Neustart, `halt` oder `init 0` für Halt. Diese Programme sind Bestandteil des jeweiligen `init`-Systems. Weitere Informationen hierzu finden Sie in den entsprechenden Manpages.

Viele DOS/Windows-Benutzer haben sich an die Tastenkombination `[Strg]+[Alt]+[Entf]` zum Neustart ihres Systems gewöhnt. Diese Tastenkombination kann auch unter Linux verwendet werden. Mit dem Befehl `ctrlaltdel` kann das Verhalten des Systems eingestellt werden. Die Option `hard` (Default) beim Drücken der magischen drei Tasten startet das System ohne `sync` neu

(`hard_reset_now()` im Kernel). Bei der Option `soft` wird das Signal `SIGINT` an `init` gesendet, das den weiteren Shutdown steuert.

Mit Windows NT und den Nachfolgesystemen wird diese Tastenkombination als eine Art »Secure Attention Key« verwendet. Ein unbedarfter Anwender wird diese Tastenkombination früher oder später auch auf einem Linux-Rechner ausprobieren und den Rechner versehentlich booten. Wenn Sie damit rechnen müssen, ist diese Einstellung vielleicht nicht allzu praktisch.

Damit nicht jeder Anwender das System stoppen kann, lässt sich die Funktion des Neustarts davon abhängig machen, dass ein autorisierter Benutzer an der Konsole angemeldet ist. Die Namen der berechtigten Benutzer werden dazu einfach in die Datei `/etc/shutdown.allow` aufgenommen. Eine Einschränkung auf einzelne Benutzer ist nur dann sinnvoll, wenn das System in keinem Fall einfach ausgeschaltet werden kann, also der Reset- und der Strom-Schalter nicht erreichbar sind.

3.8 Die init-Skripte im LSB

Aktion	Beschreibung
<code>start</code>	Starten des Dienstes
<code>stop</code>	Stoppen des Dienstes
<code>restart</code>	Stoppen und erneutes Starten des Services. Läuft der Dienst nicht, so wird dieser gestartet.
<code>reload</code>	Konfiguration erneut einlesen (optional)
<code>force-reload</code>	Wie <code>reload</code> , wenn der Dienst das unterstützt. Wenn nicht, dann wird der Dienst gestoppt und neu gestartet.
<code>status</code>	Ausgeben des Status dieses Dienstes

Tabelle 3.3 Standardisierte Aktionen für init-Skripte

Bei der Debian-Distribution wird zur Verwaltung der symbolischen Links in den Runlevel-Verzeichnissen das Programm `update-rc.d` verwendet. Red Hat verwendet das Programm `chkconfig`. Im LSB ist das Programm `/usr/lib/lsb/install_initd` definiert, das spezielle Kommentare in den Skripten auswertet.

```
# Provides: boot_facility_1 [ boot_facility_2 ...]
# Required-Start: boot_facility_1 [ boot_facility_2 ...]
# Required-Stop: boot_facility_1 [ boot_facility_2 ...]
# Default-Start: run_level_1 [ run_level_2 ...]
# Default-Stop: run_level_1 [ run_level_2 ...]
# Short-Description: short_description
# Description: multiline_description
```

Listing 3.11 Kommentare in init-Skripten

Für den Entwickler ist es wichtig, dass die Datei `/lib/lsb/init-functions` in den Skripten eingelesen werden muss und damit verschiedene Funktionen zum Starten, Stoppen und Steuern von Dämonen zur Verfügung stehen. Damit wird es erstmals möglich, auch *init*-Skripte zwischen den Distributionen auszutauschen.

3.9 *init*-Konzepte ohne symbolische Links

So bequem und bekannt wie das Verwalten der symbolischen Links auch ist, es hat auch Nachteile. So kann man die Links nur schwer mittels einer Versionsverwaltung bearbeiten oder eine alte mit einer neuen Version vergleichen. Die Programme `file-rc` (<http://packages.debian.org/file-rc>) und `r2d2` (<http://www.ibiblio.org/pub/Linux/system/daemons/init/>) implementieren die bekannte *init*-Struktur mit Hilfe einer Konfigurationsdatei. Diese enthält Runlevel, die Sortierung darin und den Skriptnamen zum Starten bzw. Stoppen der Dienste (Listing 3.12).

```
# Format:
# <sort> <off-> <on-levels> <command>
01      0,1,6   -           /etc/init.d/xdm
05      -       1           /etc/init.d/single
20      0,1,6   2,3,4,5       /etc/init.d/gpm
...
```

Listing 3.12 Die Konfigurationsdatei `/etc/runlevel.conf`

Für Debian-Benutzer ist die Verwendung von `file-rc` fast vollkommen transparent. Die üblichen symbolischen Links werden von den einzelnen Paketen nicht selbst erstellt, sondern das Programm `update-rc.d` wird aufgerufen. Wenn man dieses Programm austauscht, wie es zum Beispiel das Paket `file-rc` macht, so braucht kein einziges Paket angepasst zu werden. Ein Nachteil ist allerdings, dass die Konfigurationsdatei jeweils neu geschrieben wird, eventuell könnte aber ein Verfahren wie das `/etc/cron.d`-Verzeichnis Abhilfe schaffen. Obwohl die Idee schon einige Jahre alt ist, werden diese Programme nur selten eingesetzt.