Patricia Jung

Linux anhand des Bootprozesses verstehen lernen

Modul IF LIN 06 der Linux-Akademie im Rahmen der Informatica Feminale 2008 in Bremen

1 Vom Einschalten zum Login-Prompt: Ein Überblick

1. Wenn das BIOS seine Überprüfungen beendet hat, lädt es den Startcode des Betriebssystems aus dem *Master-Boot-Record* (MBR) der Festplatte[1].

2. Auf Linuxrechnern – egal, ob nur Linux oder auch andere Betriebssysteme installiert sind – liegt an dieser Stelle in der Regel ein *Bootmanager* oder *Bootloader*, der es erlaubt, zwischen mehreren Operativsystemen zu wählen (das können auch verschiedene Linuxinstallationen sein). Er startet das passende OS, d. h., er lädt die Kerneldatei in den Speicher, erkenntlich an der Ausgabe `Loading` *`kernel-name`*.

3. Hat es der Bootloader geschafft, den Kernel zu finden und damit begonnen, ihn in den Hauptspeicher zu laden, ist seine Aufgabe erledigt. Die Kerneldatei enthält den Kernel meist in komprimierter Form und beginnt mit einem kleinen Programm, mit dem sich Linux anschließend gewissermaßen selbst „an den Haaren herbeizieht“, sich also entpackt. Diesen Schritt erkennt frau an der Ausgabe `decompressing kernel`.

4. Das so geladene Betriebssystem prüft Speicher und Hardware und lädt entsprechende Treiber. Dabei kommen zunächst diejenigen zum Zuge, die fest in den Kernel einkompiliert sind.

5. Nun bindet der Kernel die *Root-Partition* (in der Regel zunächst nur lesbar – *read only*, damit eine Überprüfung des Dateisystems möglich bleibt) ein. Benötigt er zu diesem Zweck Treiber, die als Module ausgelagert sind, (etwa für das verwendete Dateisystem), muss er zunächst eine *Initial RAM-Disk* (`initrd`) laden.[2]

6. Es folgt der Aufbau der Prozesstabelle: Als Prozess Nummer 1 startet der Kernel `init`. Ehe frau mit dem Rechner etwas anfangen kann, muss dieser eine Menge Arbeit verrichten, die traditionell in seiner Konfigurationsdatei `/etc/inittab` beschrieben ist.[3]

   Er hat momentan nur die Dateien und Programme zur Verfügung, die auf der Root-Partition zu finden sind. Das sollten mindestens die Verzeichnisse `/sbin` (Programme, die i. A. nur von der Superuserin benötigt werden), `/lib` (Bibliotheken, die von den Programmen in `/sbin` und `/bin` geladen werden), `/etc` (Konfigurationsdateien) und `/bin` (Programme, die auch für normale User von Interesse sein können) sein.

7. `init` überprüft das Root-Dateisystem und remountet es – sofern es keine ernsthaften Fehler gab – *read-write*.

---

[1] . . . oder dem Bootsektor des im Setup eingestellten Bootlaufwerks.

[2] Diese erzeugt man in der Regel mit einem `mkinitrd`-Skript, das sich jedoch von Distribution zu Distribution unterscheidet.

[3] Ubuntu arbeitet seit Version 6.10 (Edgy Eft) an einem Event-basierten alternativen Bootkonzept namens Upstart,[URL 1] das einen alternativen Init-Daemon benutzt, der keine `inittab` seriell abarbeitet, sondern separat definierte *Jobs* ausführt. Dieses zukunftsträchtige neue Init-Konzept hat das Potential, den Bootvorgang zu beschleunigen; es ist abwärtskompatibel zum Sys-V-Init angelegt.

8. `init` liest die Datei `/etc/fstab` (*File System Table*) aus und mountet die dort mit `auto` eingetragenen weiteren Partitionen.

9. `init` lädt Kernelmodule, u. a. weitere Hardwaretreiber, initialisiert die Netzwerkunterstützung und startet eine Reihe Dienste, darunter ggf. auch den *Display-Manager*, der ein grafisches Login erlaubt. Dazu ruft der Prozess sogenannte *Initskripte* auf.

10. `init` startet die *virtuellen Konsolen* oder TTYs (*Terminal Types*). Der entsprechende *Getty*[4]-Prozess sorgt dafür, dass auf der Textkonsole der Loginprompt erscheint.

2 Der Bootloader

Fast alle Linuxinstallationen booten unter Nutzung eines der folgenden drei Bootloader:

`Syslinux` Die Bootloader des Syslinux-Projekts[URL 2] sind darauf spezialisiert, Linux von Wechselmedien (Disketten, CDs) oder vom Netzwerk zu booten.

LILO Der *Linux Loader* ist der klassische Linux-Bootmanager für Festplatteninstallationen.

GRUB Der *Grand Unified Bootloader* als zunehmend verbreitete Alternative ist mittlerweile ein GNU-Projekt.[URL 3]

2.1 LILO

Wenn der Bootmanager den Kernel laden will, befindet sich noch kein Betriebssystem und damit auch keine Unterstützung für Dateisysteme im Hauptspeicher. Der Linuxkernel ist zwar nichts weiter als eine Binärdatei, aber das Wissen um `/boot/vmlinuz` o. ä. nutzt uns mangels Dateisystem zu diesem Zeitpunkt herzlich wenig. Der Bootloader muss also wissen, an welcher physikalischen Stelle auf der Festplatte sich der Kernel befindet.

Deshalb reicht es beim LILO nicht, die Konfigurationsdatei `/etc/lilo.conf` zu editieren (per Hand oder per Installationstool), denn er kann beim Booten nicht in diese Datei schauen. Daher notiert frau mit dem Kommando `/sbin/lilo`, wo auf der Festplatte ein (hoffentlich startbarer) Kernel liegt, und schreibt einen passenden Startcode in den Bootsektor. Damit wird der Bootmanager überhaupt erst installiert.

Es gilt also, zum einen zwischen dem Bootloader LILO und dem Kommando `lilo` zu unterscheiden, das den Bootmanager erstellt und schreibt. Zum anderen reicht es nicht, einen Kernel zu kompilieren. Damit er gestartet werden kann, muss ihn die Administratorin in die Lilo-Konfigurationsdatei `/etc/lilo.conf` als neue Bootmöglichkeit eintragen und anschließend `/sbin/lilo` aufrufen. Solange frau noch nicht sicher ist, dass der neue Kernel auch ordentlich bootet und funktioniert, sollte sie den Eintrag für den bislang geladenen Kernel noch nicht entfernen, sondern ihn als alternative Bootmöglichkeit anbieten.

[4] Mnemonisch: *Get TTY*

✎ Welche Bootmöglichkeiten bietet die folgende `/etc/lilo.conf`?

```
root=/dev/hda2
map=/boot/map
delay=20
[...]

default=Linux2.4

image=/boot/vmlinuz-2.4.17
    label=Linux2.4
    read-only
    optional

image=/boot/vmlinuz-2.4.17-686
    label=Linux2.4deb
    read-only
    initrd=/initrd.img

other=/dev/hda1
    label=win
```

(Musterlösung im Anhang)

LILO selbst ist allerdings zu groß für den 512 Byte großen Bootsektor. Dort befindet sich nur der sogenannte *First-Stage-Loader*. Konnte er geladen werden, erkennt frau das daran, dass ein `L` auf dem Bildschirm erscheint. Der First-Stage-Loader lädt – wenig überraschend – einen *Second-Stage-Loader*, was ein `I` auf dem Bildschirm bestätigt. Dieser zweite Teil sorgt für ein zweites `L` und versucht, die `map`-Datei (meist `/boot/map`) von der Festplatte zu lesen[5], in der das `lilo`-Kommando die Speicherorte der aus `lilo.conf` bekannten, bootbaren Betriebssysteme vermerkt hat. War er dabei erfolgreich, erscheint ein `O` auf dem Bildschirm. Jetzt kann es nur noch sein, dass der zu bootende Kernel kaputt ist.

2.2 GRUB

Mittlerweile benutzen die meisten Distributionen GRUB als Standard-Bootloader. Sein großer Vorteil ist es, dass frau ihn dank einer eingebauten Shell auch benutzen kann, wenn seine Konfiguration in `/boot/grub/menu.1st` oder `grub.conf` kaputt ist. Damit lassen sich auch Betriebssysteme laden, die diese Konfigurationsdatei noch gar nicht spezifiziert.

```
$ grub

    GNU GRUB  version 0.95  (640K lower / 3072K upper memory)
```

[5] Achtung, auch hier ist noch kein Dateisystem im Spiel, LILO weiß nur, an welcher physikalischen Stelle auf der Festplatte er suchen muss!

```
 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename. ]

grub> help
blocklist FILE                          boot
cat FILE                                chainloader [--force] FILE
clear                                   color NORMAL [HIGHLIGHT]
configfile FILE                         device DRIVE DEVICE
displayapm                              displaymem
find FILENAME                           geometry DRIVE [CYLINDER HEAD SECTOR [
halt [--no-apm]                         help [--all] [PATTERN ...]
hide PARTITION                          initrd FILE [ARG ...]
kernel [--no-mem-option] [--type=TYPE] makeactive
map TO_DRIVE FROM_DRIVE                 md5crypt
module FILE [ARG ...]                   modulenounzip FILE [ARG ...]
pager [FLAG]                            partnew PART TYPE START LEN
parttype PART TYPE                      quit
reboot                                  root [DEVICE [HDBIAS]]
rootnoverify [DEVICE [HDBIAS]]          serial [--unit=UNIT] [--port=PORT] [--
setkey [TO_KEY FROM_KEY]                setup [--prefix=DIR] [--stage2=STAGE2_
terminal [--dumb] [--no-echo] [--no-ed terminfo [--name=NAME --cursor-address
testvbe MODE                            unhide PARTITION
uppermem KBYTES                         vbeprobe [MODE]

grub> quit
```

Auch GRUB besteht aus mehreren Teilen: Den 512 Byte großen *First-Stage-Loader* installiert der Befehl `grub-install /dev/`*`hda`* in den Bootsektor des angegebenen Festspeichers.[6] Dieser lädt in der Regel und abhängig davon, auf welchem Filesystem der *Second-Stage-Loader* sich befindet, einen von mehreren zur Verfügung stehenden, immer noch kleinen *Stage-1.5-Loadern*. Der Second Stage Loader braucht keine Map-Datei, sondern findet die Betriebssysteme anhand der in der Konfigurationsdatei oder am Prompt angegebenen Pfade. Das hat den angenehmen Nebeneffekt, dass frau GRUB nach Kernelupdates etc. nicht neu installieren muss. Alle *GRUB-Stages* findet frau in `/boot/grub/`.

`menu.1st` muss frau nicht von Hand schreiben und pflegen, sondern kann dazu das Programm `update-grub` verwenden, das u. a. alle `vmlinuz-*`-Dateien in `/boot` automatisch aufnimmt. Die Datei besteht – wie `lilo.conf` – aus einem Abschnitt mit generellen Optionen und den Booteinträgen selbst:

```
[...]
default         0
timeout         3
hiddenmenu
[...]
title           Ubuntu, kernel 2.6.12-10-686
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.12-10-686 root=/dev/hda1 ro quiet splash
```

[6] `/dev/hda` wäre also der erste IDE-Master, meist die erste IDE-Festplatte.

```
initrd          /boot/initrd.img-2.6.12-10-686
savedefault
boot

title           Ubuntu, kernel 2.6.12-10-686 (recovery mode)
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.12-10-686 root=/dev/hda1 ro single
initrd          /boot/initrd.img-2.6.12-10-686
boot
```

Hier startet per Default nach einer Wartezeit von `3` Sekunden der oberste in der Datei angegebene Kernel. Ein Druck auf (Esc) während dieser Zeit öffnet ein Auswahlmenü, das normalerweise verborgen bleibt.

Der Parameter `root` hat nichts mit dem Root-Filesystem des Rechners zu tun – das gibt frau in der `kernel`-Zeile als Kernel-Parameter an –, sondern meint die Wurzelpartition für GRUB, also die, auf der sich das Verzeichnis `/boot` befindet.

Leider führt GRUB eine eigene Schreibweise für die Ansprache von Devices ein: `hd0,0` entspricht der Partition `hda1` oder `sda1`; `hd0` spricht den Bootsektor (MBR) des ersten IDE- bzw. SCSI-Laufwerks an. Da GRUB nicht zwischen IDE- und SCSI-Geräten unterscheidet, kann die Zuordnung auf Rechnern mit „gemischten“ Festplatten spannend werden; frau entnimmt sie der Datei `/boot/grub/device.map`.

In der Grub-Shell ließe sich der oben angegebene Kernel wie folgt laden:

```
grub> kernel (hd0,0)/boot/vmlinuz-2.6.12-10-686 root=/dev/hda1
   [Linux-bzImage, setup=0x1400, size=0x1644e8]
grub> initrd (hd0,0)/boot/initrd.img-2.6.12-10-686
   [Linux-initrd @ 0xfdd0000, 0x10f9ed bytes]
grub> boot
```

3 Der Kernel

Der Kernel protokolliert alles, was er tut, darunter auch den Bootvorgang mit der Initialisierung der Hardware, nicht nur auf dem Bildschirm,[7] sondern in einem sogenannten *Ringpuffer*. Das ist ein Speicherbereich fester Größe, in dem jeweils die neuesten Kernelmeldungen stehen – sobald er voll ist, überschreibt der Kernel ältere Einträge.

Damit frau auch später noch auf die Bootmeldungen zugreifen kann, konservieren viele Distributionen sie im Verzeichnis `/var/log`. Bei Ubuntu und Red Hat heißt die entsprechende Datei `dmesg`, bei Suse `boot.msg`.

✎ Sieh Dir das Boot-Protokoll des Kernels nachträglich an! Was ist dabei zum Beispiel passiert?

[7] . . . , was viele Distributionen durch nettere Grafiken ersetzen, um Nicht-Techies nicht zu verschrecken, und nur auf Anforderung ausgeben.

✎ Überprüfe, ob `dmesg` noch die Bootmeldungen ausgibt!

4 Die Mutter aller Prozesse: `init`

Als Prozess mit der *Prozessidentifikationsnummer* (PID) 1, also als allererster Prozess, vom Kernel gestartet, hat `init` zunächst nur Programme und Dateien zur Verfügung, die auf der – zunächst nicht beschreibbaren[8] – Root-Partition liegen. Darunter fallen z. B. der Befehl `/sbin/fsck` und die Dateisystem-spezifischen Filesystem-Überprüfungs- und Reparaturprogramme wie `/sbin/e2fsck` für Ext2/Ext3, der Befehl `/bin/mount` zum Einbinden von Dateisystemen in den Dateibaum und die Datei `/etc/fstab`, die die *Filesystem Table* enthält.

4.1 Die `/etc/fstab`

`init` sieht in der Datei `/etc/fstab` nach, welche Partitionen er außer der Root-Partition noch einbinden, sprich: *mounten*, muss. Bevor der Prozess das tut, überprüft er das darauf enthaltene Dateisystem (so möglich) auf Konsistenz und repariert es ggf.

Da das beim traditionellen *ext2fs*-Dateisystem sehr lange dauern kann, wird ein solcher *Filesystemcheck* mit dem Programm `fsck` nur dann durchgeführt, wenn eine bestimmte (mit dem Befehl `tune2fs` einstellbare) Anzahl Bootvorgänge überschritten oder der Rechner nicht ordnungsgemäß heruntergefahren wurde. Deshalb setzen die meisten Distributionen mittlerweile von sich aus Journaling-Dateisysteme wie den Ext2-Nachfolger Ext3 oder ReiserFS ein.

Welches Dateisystem auf einer Partition aufgebracht ist, verrät die jeweils dritte Spalte in der `fstab`:[9]

```
proc             /proc           proc    defaults                      0  0
/dev/hda1        /               ext3    defaults,errors=remount-ro    0  1
UUID=96d47f59-2435-4280-8ac2-0588d4f85cab /usr   ext3 defaults         0  2
LABEL=var        /var            ext3    defaults                      0  2
/dev/hda5        none            swap    sw                            0  0
/dev/hdc         /media/cdrom0   udf,iso9660 user,noauto               0  0
nfsserver:/homes /home     nfs rsize=8192,wsize=8192,timeo=14,intr,bg 0  0
```

[8] Der Kernel bindet die Root-Partition zunächst nur *read only* ein, damit `init` zunächst die Möglichkeit hat, das darauf liegende Dateisystem auf Konsistenz zu prüfen. Erst wenn das erledigt ist, *remountet* `init` das Wurzeldateisystem *read-write*. So lässt sich verhindern, dass etwas auf ein kaputtes, inkonsistentes Dateisystem geschrieben wird.

[9] Der Eintrag `auto` besagt, dass `mount` das selbst rausfinden soll. Allerdings dauert das Mounten mit `mount -t auto` etwas länger, als wenn an dieser Stelle bereits der Dateisystemtyp steht.

In der ersten steht, welches Gerät eingehängt werden soll, meist in Form der jeweiligen Gerätedatei, bei lokalen Datenträgern aber immer öfter auch in Form eines Labels oder eines 16 Byte großen *Universally Unique Identifier* (UUID).

Letzteren erhält das Dateisystem beim Anlegen automatisch; frau kann ihn aus dem Superblock der Partition auslesen:

```
# dumpe2fs -h /dev/hda1
dumpe2fs 1.38 (30-Jun-2005)
Filesystem volume name:   /
Last mounted on:          <not available>
Filesystem UUID:          e50c0be3-fa5a-4082-8e30-6720dac7eb48
[...]
```

Das maximal 16 Zeichen lange Label lässt sich mit dateisystemspezifischen Tools wie `tune2fs` (Option `-L`) oder `reiserfstune` (Option `-l`) auch nachträglich setzen.

Die zweite Spalte in der `fstab` gibt an, unterhalb welchen Verzeichnisses der Inhalt dieses Dateisystems zugänglich gemacht werden soll, während die vierte Spalte Optionen für das `mount`-Kommando auflistet.[10] `init` mountet nur die Geräte, die hier die Option `auto` stehen haben. Sie ist Bestandteil von `defaults`.

✎ Welche Optionen umfasst `defaults`?

(Musterlösung im Anhang)

Das vorletzte Feld in der `fstab` hat eher historischen Wert: Ihm entnimmt der – heute eher selten benutzte – Backupbefehl `dump` (so er denn die `fstab` zu Rate zieht), aller wieviel Tage er die entsprechende Partition sichern soll.

Die Zahl in der letzten Spalte legt fest, in welcher Reihenfolge die entsprechenden Dateisysteme beim Booten (oder bei einem `mount -a`) geprüft (und damit auch gemountet) werden: Eine 0 bedeutet: kein Filesystemcheck, alle mit 1 versehenen Dateisysteme kommen zuerst dran, dann folgen die mit der 2 usw. (bis 9).

4.2 Runlevel

Nun, da `init` das komplette Linux-Dateisystem zur Verfügung hat, findet es in der Datei `/etc/inittab` beschrieben, welche Aufgaben es weiterhin erfüllen soll. Zunächst muss die „Mutter aller Prozesse“ wissen, in welchen Betriebszustand, in welchen *Runlevel*, sie schalten soll.

Ein Unixrechner kann ohne Weiteres zur Laufzeit aus einer unvernetzten Workstation zu einem Internetserver (und umgekehrt) werden. Einzige Voraussetzung ist, dass entsprechende

[10] Diese Optionen sind dieselben, die frau `mount` mit dem Parameter `-o` übergeben kann.

Betriebszustände definiert sind. Wie diese Runlevel aussehen und welche Nummern sie tragen, unterscheidet sich von Distribution zu Distribution. Immerhin dokumentieren die meisten Distributoren in der `/etc/inittab`, welche groben Eigenschaften die verschiedenen Betriebszustände haben:

```
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.
```

Identisch definiert sind lediglich die Runlevel 0 (*halt*) und 6 (*reboot*), fast immer auch ein *Single-User-Modus* 1. Bei letzterem handelt es sich um einen Wartungsmodus, in den die Superuserin zum Reparieren gehen kann. Hier gibt es oft noch kein Netzwerk, keine Dienste und nur das, was auf dem Root-Filesystem zu finden ist.

✎ Finde anhand der Grub-Konfigurationsdatei auf Seite 5 heraus, mit welchem Bootparameter der Kernel von sich aus in den Single-User-Modus bootet!

(Musterlösung im Anhang)

Der Rechner lässt sich also zum Rebooten bringen, indem `root` einfach den Runlevel wechselt: `telinit 6` oder `init 6`.[11]

Unter den Runleveln 2–5 gibt es meist einen, der den X-Server startet und ein grafisches Login erlaubt; ansonsten ist ihnen gemeinsam, dass sie Multiuser-Betrieb ermöglichen. Die Runleveldefinitionen kann frau aber ändern, auch die Runlevel 7–9 sind gültig, werden aber in der Praxis nie benutzt.

Wenn der Kernel beim Laden keine andere Option per „Kommandozeilenparameter“ mitgeteilt bekam, baut `init` den *Default-Runlevel* auf, der in der `inittab` niedergelegt ist.

✎ Finde in der `/etc/inittab` heraus, in welchen Runlevel Dein Rechner von sich aus startet. Welche Zeile könnte das sein?

(Musterlösung im Anhang)

Den aktuellen Runlevel gibt das Kommando

```
$ runlevel
N 2
```

aus: Dabei steht in der zweiten Spalte der aktuelle Runlevel und in der ersten, aus welchem Runlevel der Rechner dahin gelangte: Das `N` (wie *none*) bedeutet, dass er in den aktuellen Runlevel gebootet hat.

[11] Auf Linux-Systemen ist `telinit` (zum Merken: *Tell init*) lediglich ein Link auf `init`. Der Befehl sendet Signale an den laufenden Init-Prozess und ist aus Sicht der guten Praxis die saubere Lösung. `init` selbst aufzurufen, funktioniert aber auch.

4.3 System-V-Init

Traditionell gibt es unter Unix zwei *Init-Systeme*: *Simple Init* und *System-V-Init*. Da außer Exoten wie Slackware kaum eine Distribution auf ersteres setzt, beschäftigen wir uns hier nur mit letzterem.

Hierbei liegen in einem Verzeichnis namens `init.d` verschiedene *Start-Stopp-Skripte*, die den Rechner softwareseitig netzwerkfähig machen und verschiedene Dienste konfigurieren und starten: pro Dienst ein Skript. Ruft frau sie mit dem Parameter `start` auf, startet der entsprechende Service, mit dem Parameter `stop` wird er beendet. Auch andere Parameter wie `restart` sind möglich, spielen beim Booten und Herunterfahren des Rechners allerdings keine Rolle, sondern erlauben es der Systemadministratorin, Dienste bei Konfigurationsänderungen schnell zu aktualisieren etc.

Beim Sys-V-Init gibt es zudem Verzeichnisse namens `rc`*`Runlevel`*`.d`. Dort liegen Verweise (*Links*) auf besagte Init-Skripte, die frau manuell oder mit Hilfe distributionsspezifischer Tools wie `update-rc.d` (Debian/Ubuntu) oder `chkconfig` (SuSE, Red Hat) setzen kann:

```
$ ls -l /etc/rc2.d/
insgesamt 0
[...]
lrwxrwxrwx  1 root root 18 2006-01-05 16:18 S10sysklogd -> ../init.d/sysklogd
lrwxrwxrwx  1 root root 15 2006-01-05 16:18 S11klogd -> ../init.d/klogd
lrwxrwxrwx  1 root root 14 2006-01-05 16:42 S12dbus -> ../init.d/dbus
lrwxrwxrwx  1 root root 13 2006-01-05 16:44 S13gdm -> ../init.d/gdm
lrwxrwxrwx  1 root root 13 2006-01-05 16:41 S14ppp -> ../init.d/ppp
lrwxrwxrwx  1 root root 17 2006-01-21 14:52 S16openvpn -> ../init.d/openvpn
lrwxrwxrwx  1 root root 18 2006-01-05 22:57 S18firewall -> ../init.d/firewall
lrwxrwxrwx  1 root root 16 2006-01-05 16:42 S19cupsys -> ../init.d/cupsys
[...]
lrwxrwxrwx  1 root root 16 2006-01-05 16:19 S20pcmcia -> ../init.d/pcmcia
lrwxrwxrwx  1 root root 17 2006-01-05 17:01 S20postfix -> ../init.d/postfix
[...]
lrwxrwxrwx  1 root root 13 2006-01-05 17:20 S20ssh -> ../init.d/ssh
[...]
lrwxrwxrwx  1 root root 14 2006-01-05 16:41 S89cron -> ../init.d/cron
[...]
```

Sie definieren, welche Skripte tatsächlich ausgeführt werden: Beginnt der Name des Links mit einem S, wird das verlinkte Skript beim Wechsel in dieses Runlevel (z. B. beim Hochfahren des Rechners) mit dem Parameter `start` aufgerufen. Ist der erste Buchstabe ein K (wie *kill*), ruft `init` es beim Wechsel in einen anderen Runlevel (beispielsweise 1 oder 6 zum Herunterfahren) mit dem Parameter `stop` auf.

Die verschiedenen Skripte werden jeweils in der Reihenfolge der auf den ersten Buchstaben folgenden Zahlen abgearbeitet.

Zum Glück halten sich im Zuge der Umsetzung der *Linux Standard Base* (LSB)[URL 4] nach Jahren der Uneinigkeit immer mehr Distributoren (wieder) an dieses einfache Konzept. Bei

nicht mehr ganz taufrischen Distributionen gilt es jedoch aufzupassen: So reicht es bei alten SuSE-Ausgaben nicht, dass Startlinks vorhanden sind – um einen Dienst zu starten, muss zusätzlich noch eine Variable in der Datei `/etc/rc.config` gesetzt werden.[12] Auch wenn sich die LSB-Vorgabe `/etc/init.d` als Container-Verzeichnis für die Init-Skripte inzwischen weitgehend durchgesetzt hat, sollte frau im Zweifelsfall lieber auch `/sbin` (alte SuSE-Distributionen), `/etc` oder `/etc/rc.d` durchsuchen. Auf älteren Distributionen findet frau zudem oft eine Datei namens `rc.local`, die ohne irgendwelche Verlinkerei ausgeführt wird, wenn das Runlevel konfiguriert wurde.

✎ Wo liegen die Initskripte und die jeweiligen Links auf Deinem Rechner?

(Musterlösung im Anhang)

✎ Welches Initskript ist dafür verantwortlich, dass der Cron-Daemon startet?

(Musterlösung im Anhang)

✎ Finde heraus, welche Init-Skripte im Default-Runlevel Deines Rechners beim Booten *nicht* gestartet werden.

(Musterlösung im Anhang)

Auch wenn viele Distributionen die Start-Stopp-Skripte möglichst kompliziert implementieren und mit zusätzlichen Informationen für die Runlevel-Linkerstellungstools anreichern, reicht es, wenn darin eine einfache `case`-Anweisung festlegt, was in dem Fall zu tun ist, wenn das Skript mit dem Parameter `start` bzw. `stop` aufgerufen wird:

```
#! /bin/sh

N=/etc/init.d/firewall

set -e

case "$1" in
  start)
if [ -x /etc/firewall ]
then
               /etc/firewall
else
echo "/etc/firewall does not exist." >&2
exit 1
fi
;;
  stop)
iptables -F
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
```

[12] Dieses Konzept war übrigens FreeBSD, einem anderen freien Unix, abgeschaut.

```
;;
  *)
echo "Usage: $N {start|stop}" >&2
exit 1
;;
esac

exit 0
```

4.4 /etc/inittab

Das alles ist schön und gut, doch woher weiß `init`, welche Skripte es starten soll? Tatsächlich tut es das gar nicht selbst, sondern überlässt diese Aufgabe einem anderen Programm oder Skript. Auch hier ist keine Magie dabei: Alles steht fein säuberlich in der Konfigurationsdatei `/etc/inittab`.

Wie wir schon beim Herausfinden des Default-Runlevels gesehen haben, besteht diese aus vierspaltigen Zeilen, wobei der Doppelpunkt als Spaltentrenner gilt.

- Die erste Spalte nimmt einen eindeutigen Bezeichner für den Eintrag auf.
- Die zweite gibt (ohne Trennzeichen) die Runlevel an, denen die Zeile gilt.
- Die dritte enthält ein vordefiniertes Stichwort, das `init` sagt, worum es in dieser Zeile geht.
- Die vierte Spalte (die im Falle des Stichworts `initdefault` leer bleibt), gibt das auszuführende Kommando an.

Enthält die `inittab` keine `initdefault`-Zeile, fragt `init` beim Booten auf der Konsole nach, welchen Runlevel es wählen soll.

Noch bevor `init` den Default-Runlevel initialisiert, führt das Programm beim Booten die Befehle aus den Zeilen aus, die das Stichwort `sysinit` enthalten:

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

Bei manchen Linux-Distributionen sorgt das (distributionsabhängige) Boot-Skript `rc.boot` dafür, dass ein benutzbares System hochfährt, auf dem allerdings noch kaum ein Dienst läuft. In neueren Distributionen sollten die entsprechenden Aufrufe in jeweils eigene Init-Skripte ausgelagert werden, die ins `/etc/rcS.d`-Verzeichnis verlinkt werden.[13] `/etc/init.d/rcS` führt diese dann im Single-User-Modus S aus.

Mit `wait`-Einträgen initialisiert `init` den jeweiligen Runlevel und tut dabei genau das, was `wait` bedeutet: Auf das Ende des angegebenen Kommandos warten:

[13] Wobei sich frau z. B. bei Ubuntu doch fragt, ob Netzwerk- und DNS-Konfiguration hier wirklich hingehören…

```
# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
[...]
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

Im hier abgedruckten Beispiel (Debian/Ubuntu) ruft `init` also das Programm `/etc/init.d/rc` mit der jeweiligen Runlevel-Nummer als Argument auf. Da es sich dabei um ein Shellskript handelt, lässt sich gut nachvollziehen, was dieses Skript tut:

```
#! /bin/sh
[...]
# Optimization feature:
# A startup script is _not_ run when the service was
# running in the previous runlevel and it wasn't stopped
# in the runlevel transition (most Debian services don't
# have K?? links in rc{1,2,3,4,5} )
[...]
# Start script or program.
#
startup() {
  case "$1" in
        *.sh)
                $debug sh "$@"
                ;;
        *)
                $debug "$@"
                ;;
  esac
}
[...]
  runlevel=$RUNLEVEL
[...]
  previous=$PREVLEVEL
[...]
  export runlevel previous

  # Is there an rc directory for this new runlevel?
  if [ -d /etc/rc$runlevel.d ]
  then
# First, run the KILL scripts.
if [ $previous != N ]
then
for i in /etc/rc$runlevel.d/K[0-9][0-9]*
```

```
do
# Check if the script is there.
[ ! -f $i ] && continue

# Stop the service.
startup $i stop
done
fi
[...]
  fi
# eof /etc/init.d/rc
```

Die `inittab` definiert daneben auch, was geschieht, wenn frau die Tastenkombination (Strg)+(Alt)+(Entf) drückt:

```
# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

✎ Speziell auf Rechnern wie im hiesigen Pool, auf denen Leute auch per `ssh` eingeloggt sein dürfen/sollen, möchte frau es u. U. verbieten, dass Nutzerinnen mit Konsolenzugang den Rechner einfach herunterfahren. Wie implementierst Du dieses Verbot?

(Musterlösung im Anhang)

Sobald das Netzwerk eingerichtet ist und auch sonst alle gewünschten Dienste laufen, wird es Zeit, dass sich auch Nutzerinnen am System anmelden können. Während der grafische Login-Manager, in der Regel einer der *Display-Manager* `xdm`, `kdm` oder `gdm`, per Init-Skript startet, ist für die Konsolenlogins auf den virtuellen Terminals (TTYs) wieder die `inittab` zuständig:

```
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
[...]
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
[...]
```

Im Beispiel wird dazu in den Runleveln 2–5 der Befehl `/sbin/getty 38400 tty1` ausgeführt, in den Runleveln 2 und 3 zusätzlich der Befehl `/sbin/getty 38400 tty2`.[14]

[14] Nur der Vollständigkeit halber: 38400 gibt die Baud-Rate, also die Anzahl der Symbole des übertragenen Signals pro Sekunde an, `tty1` und tty2 das zu benutzende Device-File unterhalb des Verzeichnisses `dev`.

Wichtig hierbei: Die ID in der ersten Spalte muss der Nummer des `/dev/tty?`-Geräts entsprechen.

`/dev/tty7` ist meistens dem Display-Manager vorbehalten. Zwischen den virtuellen Terminals schaltet frau mit der Tastenkombination (Strg)+(Fx) bzw. von der grafischen Oberfläche mit (Alt)+(Strg)+(Fx) um.

Das Stichwort `respawn` in der `inittab` sorgt übrigens dafür, dass der angegebene Befehl wiederholt wird, sobald das entsprechende Terminal (durch Ausloggen) geschlossen wird.

Online-Ressourcen

URL 1 Upstart: http://upstart.ubuntu.com/

URL 2 Syslinux: http://syslinux.zytor.com/

URL 3 GRUB: http://www.gnu.org/software/grub/

URL 4 Linux Standard Base: http://www.freestandards.org/en/LSB

5 Lösungen

✎ Welche Bootmöglichkeiten bietet die folgende `/etc/lilo.conf`?

```
root=/dev/hda2
map=/boot/map
delay=20
[...]

default=Linux2.4

image=/boot/vmlinuz-2.4.17
    label=Linux2.4
    read-only
    optional

image=/boot/vmlinuz-2.4.17-686
    label=Linux2.4deb
    read-only
    initrd=/initrd.img

other=/dev/hda1
    label=win
```

Auf diesem Rechner hat frau die Möglichkeit zum Booten zweier verschiedener Linuxkernel (`/boot/vmlinuz-2.4.17` und `/boot/vmlinuz-2.4.17-686`, der die Hilfe einer Initial RAM-Disk benötigt) sowie einer Windows-Installation. Anhand der Labels kann die Benutzerin im Bootmenü aussuchen, was gebootet wird. Trifft sie keine Entscheidung, kommt nach 20 Zehntelsekunden Wartezeit der Linux-Kernel mit dem Label `Linux2.4` zum Zuge (`default`), als Root-Partition dient `/dev/hda2`. Ein # leitet eine Kommentarzeile ein. Alle übrigen Parameter sind zum Beispiel in der Manpage zu `lilo.conf` nachzulesen. So bedeutet die `optional`-Option, dass `/sbin/lilo` den entsprechenden Kernel nur dann in die Map-Datei aufnehmen soll, wenn er zum entsprechenden Zeitpunkt existiert.

✎ Finde anhand der Grub-Konfigurationsdatei auf Seite 5 her aus, mit welchem Bootparameter der Kernel von sich aus in den Single-User-Modus bootet!

Mit dem Parameter `single`, alternativ `S` für das Runlevel S, in dem nur die Skripte aus `rcS.d` gestartet werden. Gibt man stattdessen die Runlevelnummer `1` als Kernelparameter ein, führt `init` auch die Skripte aus `rc1.d` aus.

Im klassischen Unix-Single-User-Modus startet der Kernel übrigens anstelle von `init` eine Shell (`/bin/sh`). Zu diesem Ergebnis kommt frau, indem sie am Bootprompt den Parameter init=/bin/sh eingibt.

✎ Welche Optionen umfasst `defaults`?

```
$ man mount
[...]
      -o     Options  are  specified with a -o flag followed by a comma sepa¿
             rated string of options.  Some of these options are only  useful
             when  they appear in the /etc/fstab file.  The following options
             apply to any file system that is being mounted  (but  not  every
             file  system  actually honors them - e.g., the sync option today
             has effect only for ext2, ext3 and ufs):
[...]
             defaults
                    Use default options: rw, suid, dev, exec,  auto,  nouser,
                    and async.
[...]
```

✎ Finde in der `/etc/inittab` heraus, in welchen Runlevel Dein Rechner von sich aus startet. Welche Zeile könnte das sein?

Der folgende Beispielrechner bootet ins Multiuser-Runlevel 2:

```
# The default runlevel.
id:2:initdefault:
[...]
# Runlevels 2-5 are multi-user.
```

✎ Wo liegen die Initskripte und die jeweiligen Links auf Deinem Rechner?

Wenn unterhalb des Verzeichnisses `etc` tatsächlich kein Ordner zu finden sein sollte, der sich `init.d` o. ä. nennt, lässt sich der `find`-Befehl zur Hilfe nehmen:

```
find / -name init* -type d
```

Dieser sucht unterhalb des Wurzelverzeichnisses `/` nach einem Verzeichnis (`-type d` – „vom Typ *directory*“), dessen Name mit den Buchstaben `init` beginnt.

Ähnlich lässt sich nach allen symbolischen Links unterhalb des Verzeichnisses `/etc` suchen, die mit dem Buchstaben `S` und einer Ziffer beginnen (und damit prädestiniert dafür sind, auf Initskripte zu zeigen):

```
# find /etc -type l -name S[0-9]*
/etc/rc0.d/S30urandom
/etc/rc0.d/S20sendsigs
[...]
```

Will frau sich gleich mit anzeigen zu lassen, wohin diese Links führen, geht das zum Beispiel so:

```
# find /etc -type l -name S[0-9]* -exec ls -l {} \;
lrwxrwxrwx 1 root root 17 Jan 26 2002 /etc/rc0.d/S30urandom ->
../init.d/urandom
lrwxrwxrwx 1 root root 18 Jan 26 2002 /etc/rc0.d/S20sendsigs ->
../init.d/sendsigs
[...]
```

Die `find`-Option `-exec` führt für jede Fundstelle den dahinterstehenden Befehl aus. Anstelle des geschweiften Klammernpaars setzt `find` den Namen der gefundenen Datei ein. Das Semikolon am Ende wird benötigt, um die damit entstehenden vielen Befehle voneinander zu trennen; und damit die Shell nicht fälschlicherweise meint, dieses Semikolon sei für sie gedacht, muss es mit `\` geschützt werden.

✎ Welches Initskript ist dafür verantwortlich, dass der Cron-Daemon startet?

Gute Kandidaten sind `/etc/init.d/crond` oder `/etc/init.d/cron`. Damit der Daemon beim Booten tatsächlich startet, muss es aber aus dem jeweiligen `rcx.d`-Verzeichnis verlinkt sein, zum Beispiel:

```
$ ls -al /etc/rc2.d/S*cron
lrwxrwxrwx 1 root root 14 Jan 26  2002 /etc/rc2.d/S89cron -> ../init.d/cron
```

✎ Finde heraus, welche Init-Skripte im Default-Runlevel Deines Rechners beim Booten *nicht* gestartet werden.

Ist 2 der Default-Runlevel, gilt es zur Beantwortung dieser Frage, im `init.d`-Verzeichnis ein Skript zu finden, zu dem *kein* mit dem Buchstaben `S` beginnender Link im `rc2.d`-Verzeichnis führt.

Um ganz genau zu sein, darf es zudem auch in `/etc/rcS.d` (so vorhanden) keinen entsprechenden Link geben.

Gute Kandidaten sind hier die Initskripte von Datenbank- oder Internetservern.

✎ Speziell auf Rechnern wie im hiesigen Pool, auf denen Leute auch per `ssh` eingeloggt sein dürfen/sollen, möchte frau es u. U. verbieten, dass Nutzerinnen mit Konsolenzugang den Rechner einfach herunterfahren. Wie implementierst Du dieses Verbot?

```
# What to do when CTRL-ALT-DEL is pressed.
# ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
ca:12345:ctrlaltdel:/bin/echo -e "Sorry, Strg-Alt-Entf ist abgeschaltet."
```

Dieses Skript basiert passagenweise auf der für die Informatica Feminale erarbeiteten Kursunterlage „Linux ist weiblich“ in deren 5. Auflage. Die Autorin bedankt sich bei Sibylle Nägle und Gabriele Pohl für die Mitarbeit an der 1. Auflage (2001) bzw. der 5. Auflage (2005).

Satz: TEX/LATEX unter Linux