# Der Linux – Bootvorgang

Lars Oergel

18. Januar 1999

## Inhaltsverzeichnis


**1 Das BIOS („Basic Input-/Output–System“)** 2

**2 Laden des Master–Boot–Records (MBR)** 2

2.1 Der Bootloader LILO („LInux LOader“) . . . . . . . . . . . . . . . . 2

**3 Starten des Kernels** 3

3.1 Auspacken des Images . . . . . . . . . . . . . . . . . . . . . . . . . . 3

3.2 Initialisieren der Geräte . . . . . . . . . . . . . . . . . . . . . . . . . 3

3.3 Mounten des Root–Filesystems . . . . . . . . . . . . . . . . . . . . . 3

3.4 Init laden und starten . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**4 Init — /etc/inittab — Konzept der Runlevels** 4

**5 Start-/Stoppskripte zur Steuerung der Systemdienste** 4

5.1 Ablauf eines Runlevel–Wechsels . . . . . . . . . . . . . . . . . . . . . 5

**6 Fertig!** 5

**A Beispiel für /etc/inittab** 6

**B Beispiel für ein Start-/Stoppskript (lpd)** 7

**C Beispiel für das rc3.d – Verzeichnis** 8






# Der Linux[1] – Bootvorgang

Bis ein Linux–System nach dem Einschalten des Rechners so läuft, daß sich „normale“ Benutzer einloggen und mit dem System arbeiten können, müssen verschiedene Schritte durchlaufen werden. Diese werden hier kurz erläutert. Weiterführende Informationen und Verweise finden sich in den Fußnoten, deren Beachtung zum grundsätzlichen Verständnis der Zusammenhänge aber nicht erforderlich ist.

## 1 Das BIOS („Basic Input-/Output–System“)

Unmittelbar nach dem Einschalten wird das BIOS aktiviert. Das ist ein fest in den Rechner „eingebautes“ Programm zur Ansteuerung und Konfiguration der einzelnen Hardwarebestandteile. Es testet rudimentär die Funktionsfähigkeit von z.B. dem Floppylaufwerk, dem Speicher oder auch der Grafikkarte. Außerdem wird das System initialisiert, d.h. in einen definierten Ausgangszustand gebracht. I.d.R. kann man durch Drücken der Delete–Taste[2] in ein spezielles Konfigurationstool des BIOS' gelangen, in dem verschiedene Parameter gepflegt werden können. Im Anschluß sucht das BIOS den Datenträger, von dem gebootet werden soll (also z.B. Floppy, CD–Rom oder Festplatte – je nach Konfiguration[3]).

## 2 Laden des Master–Boot–Records (MBR)

Der allererste Datensektor auf einer Festplatte ist der sog. „Master–Boot–Record“. Er beinhaltet zum einen den sog. „Urlader“ und die „Partition–Table“. Das BIOS lädt nun den MBR und startet den Urlader. Dieses winzig kleine Programm (max. 446 Bytes groß[4]) interpretiert die Partition–Table und lädt und startet den Bootsektor der Partition, die in der Partition–Table als Bootpartition gekennzeichnet ist.

### 2.1 Der Bootloader LILO („LInux LOader“)

Bei Linux–Systemen wird i.d.R. der LILO[5] als Urlader benutzt, da er weit mehr kann, als nur ein einzelnes Betriebssystem zu booten. Mit seiner Hilfe können einem zu startenden Linux–Kernel verschiedene Start–Parameter mit auf den Weg gegeben werden. Außerdem bietet er ein Boot–Menü an, das es erlaubt, verschiedene Betriebssysteme (ja, auch DOS/Windows und Linux) auf einer Festplatte zu halten und jeweils beim Start des Rechners auszuwählen, mit welchem Betriebssytem man gerade arbeiten möchte.

Soll der LILO ein Linux–System, d.h. einen Linux–Kernel starten, so wird das sog. „Kernel–Image“ in den Speicher geladen und anschließend gestartet.

[1] Diese Darstellung beschreibt ein auf PC–Hardware basierendes Linux–System.
[2] Das BIOS verrät in seiner Startmeldung die konkrete Taste.
[3] Ich gehe im folgenden von einer Festplatte aus.
[4] Der MBR ist zwar 512 Bytes groß, enthält aber auch noch die 64 Bytes große Partition–Table und zwei „Magic Bytes“.
[5] Eine hervorragende Dokumentation zu diesem Programm mit einer ausführlichen Einleitung in die Festplattenorganisation und verschiedene Bootkonzepte findet sich im Dokument `/usr/doc/packages/lilo/user.dvi`, zu finden auf jedem „vernünftigen“ Linux–Rechner.

# 3 Starten des Kernels

## 3.1 Auspacken des Images

Da sich der LILO noch der „Speichermanagement"–Funktionen des BIOS' bedienen muß[6], kann er auf maximal 640 KBytes RAM zugreifen[7].

Aktuelle Linux–Kernel–Images sind aber meistens größer als 640 KBytes, so daß ein komprimiertes Image verwendet werden muß, das sich beim Starten zunächst selbst auspackt.

## 3.2 Initialisieren der Geräte

Dann werden die Initialisierungs–Routinen der *fest in den Kernel einkompilierten Gerätetreiber*[8] durchlaufen.

Es werden verschiedene Status- und evtl. Fehlermeldungen angezeigt, die Aufschluß über die durch den Linux–Kernel erkannten Hardwarekomponenten geben können. Da diese bei aktuellen Rechnern aber zu schnell angezeigt werden, als das man sie mitlesen könnte, kann man sie sich bei Bedarf mit Hilfe des `dmesg`[9]–Befehls auch später noch einmal anzeigen lassen.

## 3.3 Mounten des Root–Filesystems

Anschließend wird das Root–Filesystem gemountet.

Auf welcher Festplatte und welcher Partition sich das Root–Filesystem befindet, muß entweder in den Kernel einkompiliert worden sein oder ihm vom LILO durch einen Bootparameter übermittelt worden sein.

Dieses ist eine der kritischsten Operationen beim Bootvorgang. Schlägt sie fehl, ist außer der Meldung „`panic: cannot mount root`" nicht mehr viel los mit dem Rechner.

## 3.4 Init laden und starten

Konnte das Root–Filesystem einwandfrei gemountet werden, wird der `init`–Befehl gesucht[10] und hoffentlich erfolgreich gestartet.

[6] LILO ist ja nur ein ganz kleines Programm. Erst der Linux–Kernel bietet ausgewachsene Speichermanagement–Funktionen, mit denen der gesamte Hauptspeicher aktueller PCs genutzt werden kann.

[7] Diese Speicherausstattung wurde bei Erfindung des PCs als utopisch angesehen. Bis heute hat man diese Beschränkung aus Kompatibilitätsgründen beibehalten.

[8] Im Gegensatz dazu können Treiber für Geräte, die nicht beim Booten benötigt werden, als Module übersetzt werden, die erst bei Bedarf, dynamisch während der Laufzeit des Systems durch den sog. „Kernel–Daemon" geladen werden.

[9] Anzeige des sog. „Kernel–Ring–Buffers"

[10] sehr schön am Ende der Sourcecode–Datei `/usr/src/linux/init/main.c` zu sehen



# 4 Init — /etc/inittab — Konzept der Runlevels

Der init–Prozeß übernimmt nun die Kontrolle über das System. Init „denkt" in verschiedenen, in der Datei `/etc/inittab`[11] konfigurierten Zuständen, „Runlevels" genannt.

So kann man z.B. einen Wartungszustand definieren, in dem sich nur der Systemverwalter einloggen kann. Auch die Frage, ob man sich nur vor einer Textkonsole wiederfindet oder die grafische Oberfläche mit xdm zum Einloggen vor Augen hat, wird mittels der Runlevels beantwortet. In der inittab wird auch der Default–Runlevel festgelegt, d.h. der Runlevel, der nach dem Systemstart eingenommen werden soll.

# 5 Start-/Stoppskripte zur Steuerung der Systemdienste

Um die Handhabung der inittab zu vereinfachen[12], bedient man sich normalerweise[13] einer Reihe von Shell–Skripten[14], für jeden Systemdienst eines. Diese Skripte sind so aufgebaut, daß sie entweder mit dem Parameter `start` oder `stop` aufgerufen werden können, um den jeweiligen Systemdienst zu starten bzw. anzuhalten[15]. Zusätzlich kommt ein spezielles Startup–Skript beim Systemstart zum Aufruf.

Es gibt nun für jeden Runlevel ein Verzeichnis[16], in dem sich symbolische Links auf die Steuerungsskripte befinden. Über diese Softlinks wird gesteuert, in welchem Runlevel welche Systemdienste zur Verfügung stehen sollen. Dazu folgt die Namensgebung der Links einem bestimmten Schema, durch das festgelegt wird, wann welche Skripte mit `start` oder `stop` aufgerufen werden[17].

Dieses Schema sieht folgendermaßen aus:

- Der erste Buchstabe ist entweder ein „`S`" oder ein „`K`". Dabei steht „`S`" für einen Link auf einen Systemdienst, der beim „Betreten" des spezifizierten Runlevels gestartet werden soll und „`K`" für einen Link auf einen Systemdienst, der beim „Verlassen" des spezifizierten Runlevels beendet werden soll.
- Es folgt eine (i.d.R. zweistellige) Zahl, die zur Anordnung der Start- und Stoppvorgänge dient.
- Zur Übersicht folgt noch der Name des eigentlich aufzurufenden Skriptes.

Die *Start*- und *Kill*–Einträge sollten vernünftigerweise einander entsprechen, d.h. zu jedem Start–Link sollte es einen entsprechenden Kill–Link geben. Außerdem sollte darauf geachtet werden, daß aufeinander aufbauende Dienste in der korrekten Reihenfolge gestartet und beendet werden. So wird z.B. der sehr wichtige Kernel–Daemon als allererster Dienst gestartet und als letzter beendet[18].

[11] zum Aufbau der inittab siehe `man inittab`

[12] Man könnte auch alle Systemdienste direkt in der inittab konfigurieren, nur wäre das sehr unübersichtlich, da manche Dienste über sehr aufwendige Start-/Stoppmechanismen verfügen.

[13] Es wird hier das weitverbreitete, sog. „System V" – Bootkonzept beschrieben.

[14] zu finden wahlweise unter `/sbin/init.d` oder `/etc/rc.d/init.d`, je nach Linux–Distribution

[15] siehe dazu das `lpd`–Skript Beispiel im Anhang B

[16] i.d.R. heißt es `rc0.d`, `rc1.d`, `rc2.d` usw.

[17] Der eigentliche Aufrufer ist das `rc`–Skript.

[18] siehe dazu auch das Beispiel rc3.d–Verzeichnis im Anhang C

### 5.1 Ablauf eines Runlevel–Wechsels

Soll nun beim Bootvorgang der Default–Runlevel eingenommen werden oder der Runlevel während des laufenden Betriebes gewechselt werden[19], geschieht folgendes: Der init–Prozeß startet diejenigen Prozesse, die *laut inittab* im neuen Runlevel zusätzlich laufen sollen und stoppt jene, die *laut inittab* im neuen Runlevel nicht mehr laufen sollen. Dieses sind i.d.R. nur die getty–Prozesse zum Erzeugen der Login–Prompts auf den virtuellen Textkonsolen und das sog. `rc`–Skript, das den eben beschriebenen Steuerungsmechanismus mit Hilfe der Shellskripte und der Softlinks in die Tat umsetzt. Dazu wird dieses `rc`–Skript mit dem neuen Runlevel als Parameter aufgerufen[20]. Es geht dann folgendermaßen vor:

1. Zunächst werden alle Dienste des alten Runlevels, die im neuen Runlevel nicht mehr laufen sollen, in der mit den Softlinks abgelegten Reihenfolge gestoppt. Es werden also die entsprechenden „`K`"–Skripte von „`K01*`" bis „`K99*`" mit dem Parameter `stop` aufgerufen, die sich im Verzeichnis für den alten Runlevel befinden. Beim Systemstart wird dieser Schritt natürlich übersprungen.

2. Dann werden alle Dienste des neuen Runlevels, die im alten Runlevel noch nicht liefen, in der mit den Softlinks abgelegten Reihenfolge gestartet. So werden die entsprechenden „`S`"–Skripte von „`S01*`" bis „`S99*`" mit dem Parameter `start` aufgerufen, die sich im Verzeichnis für den neuen Runlevel befinden.

Durch diese Vorgehensweise ist garantiert[21], daß bei jedem Runlevel–Wechsel „ordentlich" alle nicht mehr benötigten Dienste beendet und die neu hinzukommenden Dienste gestartet werden.

Dieser Mechanismus wird übrigens auch zum Herunterfahren des Systems verwendet. Dafür werden i.d.R. die Pseudo–Runlevels 0 (Halt) und 6 (Reboot) verwendet. In den entsprechenden `rc0.d`- und `rc6.d`–Verzeichnissen existiert ein einziger Softlink: Ein Start–Link auf ein Shellskript, das die nötigen Befehle zum Neustart bzw. Anhalten des Systems ausführt.

## 6 Fertig!

Im logischen Ablauf der Startskripte beim Bootvorgang ergibt sich so das Bild, daß ausgehend von den „Low–Level–Diensten" die „High–Level–Dienste" gestartet werden, die es im Endeffekt dem „normalen" Benutzer erlauben, sich entweder über die Textkonsole oder auch mit Hilfe der zum Schluß gestarteten grafischen Oberfläche einzuloggen und regulär mit dem System arbeiten zu können.

[19] Das macht der Systemverwalter mit Hilfe des `telinit`–Befehls.
[20] siehe dazu die `/etc/inittab` im Anhang A
[21] wenn die Softlinks entsprechend eingerichtet worden sind ...



## A Beispiel für /etc/inittab

```
#
# /etc/inittab
#
# This is the main configuration file of /sbin/init, which
# is executed by the kernel on startup. It describes what
# scripts are used for the different run-levels.
#
# All scripts for runlevel changes are in /sbin/init.d.

# default runlevel
id:3:initdefault:

# check system on startup
# first script to be executed if not booting in emergency mode
si:I:wait:/sbin/init.d/boot

# /sbin/init.d/rc takes care of runlevel handling
#
# runlevel 0 is halt
# runlevel S is single-user
# runlevel 1 is multi-user without network
# runlevel 2 is multi-user with network
# runlevel 3 is multi-user with network and xdm
# runlevel 6 is reboot
l0:0:wait:/sbin/init.d/rc 0
l1:1:wait:/sbin/init.d/rc 1
l2:2:wait:/sbin/init.d/rc 2
l3:3:wait:/sbin/init.d/rc 3
l6:6:wait:/sbin/init.d/rc 6

# what to do in single-user mode
ls:S:wait:/sbin/init.d/rc S

# getty-programs for the normal runlevels
# <id>:<runlevels>:<action>:<process>
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
1:123:respawn:/sbin/mingetty --noclear tty1
2:123:respawn:/sbin/mingetty tty2
3:123:respawn:/sbin/mingetty tty3
4:123:respawn:/sbin/mingetty tty4
5:123:respawn:/sbin/mingetty tty5
6:123:respawn:/sbin/mingetty tty6

# end of /etc/inittab
```

# B Beispiel für ein Start-/Stoppskript (lpd)

```
#!/bin/sh
#
# /sbin/init.d/lpd
#

case "$1" in
  start)
    echo "Starting lpd."
    /usr/sbin/lpd
    sleep 1
    ;;
  stop)
    echo -n "Shutting down lpd:"
    killproc -TERM /usr/sbin/lpd
    echo
    ;;
  *)
    echo "Usage:  $0 {start|stop}"
    exit 1
esac

exit 0

# end of /sbin/init.d/lpd
```



# C Beispiel für das rc3.d – Verzeichnis

```
cip5001:/sbin/init.d/rc3.d # ls -l
total 2
lrwxrwxrwx 1 root root 10 Feb 20 1998  S01kerneld -> ../kerneld
lrwxrwxrwx 1 root root  9 Feb 20 1998  S02serial -> ../serial
lrwxrwxrwx 1 root root 10 Feb 20 1998  S05network -> ../network
lrwxrwxrwx 1 root root  8 Feb 20 1998  S07route -> ../route
lrwxrwxrwx 1 root root  6 Feb 20 1998  S08nfs -> ../nfs
lrwxrwxrwx 1 root root  9 Feb 20 1998  S09syslog -> ../syslog
lrwxrwxrwx 1 root root  6 Feb 20 1998  S12rpc -> ../rpc
lrwxrwxrwx 1 root root 11 Feb 20 1998  S15ypclient -> ../ypclient
lrwxrwxrwx 1 root root  9 Nov 25 21:00 S17autofs -> ../autofs
lrwxrwxrwx 1 root root  9 May 14 1998  S20apache -> ../apache
lrwxrwxrwx 1 root root  7 Feb 20 1998  S20cron -> ../cron
lrwxrwxrwx 1 root root  8 Feb 20 1998  S20inetd -> ../inetd
lrwxrwxrwx 1 root root  6 Feb 20 1998  S20lpd -> ../lpd
lrwxrwxrwx 1 root root  8 Feb 27 1998  S20rwhod -> ../rwhod
lrwxrwxrwx 1 root root 11 Feb 20 1998  S20sendmail -> ../sendmail
lrwxrwxrwx 1 root root  6 Feb 20 1998  S30xdm -> ../xdm

lrwxrwxrwx 1 root root  6 Feb 20 1998  K10xdm -> ../xdm
lrwxrwxrwx 1 root root 11 Feb 20 1998  K20sendmail -> ../sendmail
lrwxrwxrwx 1 root root  8 Feb 27 1998  K20rwhod -> ../rwhod
lrwxrwxrwx 1 root root  6 Feb 20 1998  K20lpd -> ../lpd
lrwxrwxrwx 1 root root  8 Feb 20 1998  K20inetd -> ../inetd
lrwxrwxrwx 1 root root  7 Feb 20 1998  K20cron -> ../cron
lrwxrwxrwx 1 root root  9 May 14 1998  K20apache -> ../apache
lrwxrwxrwx 1 root root  9 Nov 25 21:00 K24autofs -> ../autofs
lrwxrwxrwx 1 root root 11 Feb 20 1998  K25ypclient -> ../ypclient
lrwxrwxrwx 1 root root  6 Feb 20 1998  K28rpc -> ../rpc
lrwxrwxrwx 1 root root  9 Feb 20 1998  K35syslog -> ../syslog
lrwxrwxrwx 1 root root  6 Feb 20 1998  K37nfs -> ../nfs
lrwxrwxrwx 1 root root  8 Feb 20 1998  K38route -> ../route
lrwxrwxrwx 1 root root 10 Feb 20 1998  K40network -> ../network
lrwxrwxrwx 1 root root  9 Feb 20 1998  K50serial -> ../serial
lrwxrwxrwx 1 root root 10 Feb 20 1998  K99kerneld -> ../kerneld
```