

Linux - der Bootvorgang

Moritz Kirchner

22.04.2008

Inhaltsverzeichnis

1	Einleitung	3
2	BIOS	3
3	Bootloader oder Bootmanager	4
4	Kernel	4
5	init-deamon	5
6	Alternativen	6
6.1	Alternativen zum BIOS	6
6.1.1	coreboot	6
6.1.2	OpenFirmware / EFI	7
6.2	Datenträger	8
6.3	Alternative Bootloader / -manager	9
6.3.1	LILO	9
6.3.2	Loadlin	9
6.4	Kernel	10
6.5	Alternativen zum init-deamon	10
6.5.1	Upstart / initNG	10
7	Quellen	11

1 Einleitung

Der Bootvorgang unter Linux ist eine Geschichte voller Irrtümer und Missverständnisse. Um sich klar zu machen, wie der Bootvorgang im klassischen Sinne abläuft, betrachten wir zuerst die wichtigsten Komponenten in der Reihenfolge, in der sie zum Booten notwendig sind, und werfen danach einen Blick über den klassischen Tellerrand hinaus auf die Alternativen zu den einzelnen Komponenten. Die wichtigsten Komponenten entsprechen den ersten fünf Punkten im Inhaltsverzeichnis.

Auf Strom wird in dieser Zusatzinformation zum Vortrag nicht weiter eingegangen, dennoch ist er essentiell wichtig zum Booten denn ohne Strom läuft ein PC nicht. Der nächste Punkt in der Liste ist das BIOS.

2 BIOS

BIOS heißt „Basic Input Output System“. Funny fact am Rande: bios im griechischen bedeutet „Leben“, was vermutlich ein beabsichtigter Zufall ist. In erster Linie ist das BIOS jedoch ein Baustein auf jedem Mainboard der „älteren“ Generation. Das BIOS steht vor dem sog. „Bootstrap“-Problem. Es soll Software laden, wozu es aber Software benötigt. Der Name kommt von der englischen Version des Baron Münchhausen, der sich selbst an seinen Stiefelriemen (bootstraps) aus dem Schlamm zog. Das BIOS führt nach dem Systemstart einen Power-On-Self-Test durch. Das erkennt man daran, dass es einmal kurz piept, wenn alles in Ordnung ist, und andere Piep-Codes ausgibt, wenn etwas fehlschlägt. Danach erkennt es im besten Falle die Hardware, initialisiert sie und früher wurden die Schnittstellen direkt einem DOS oder Windows-System zur Verfügung gestellt, z.B. der Tastaturtreiber. Heute läuft das ein bisschen anders, aber dazu später mehr. Nachdem das BIOS die Hardware fertig initialisiert hat, hat es die Aufgabe, den Bootdatenträger zu finden. Ist dies eine Diskette ist das nicht besonders schwierig, weil eine Diskette nur einen einzigen Bootsektor hat. Der Punkt in der Gliederung heißt „Datenträger“ weil man mit dem Datenträger von dem gebootet wird sehr gut herumspielen kann. Später werden wir noch sehen, dass es zum booten nicht einmal zwingend einen lokalen Datenträger braucht. Ist der Bootdatenträger nun allerdings die Festplatte und existieren mehrere Partitionen ist das erkennen nicht mehr ganz so einfach. Deswegen hat man sich auf einige Konventionen geeinigt. Der erste Bootsektor wird Master Boot Record (MBR) genannt. Jeder Bootsektor ist 512 Byte groß und steht von Sektor 0 bis 0x1FE. Er endet mit der „magic Number“ 0xAA55. Wenn man mehrere Partitionen hat, kann man eine oder mehrere davon mit dem bootable Flag

markieren. Dies geschieht in der Partitionstabelle, in der auch festgehalten wird, wo die Partition anfängt, sodass das BIOS bzw. der Bootloader zum Sektor 0 springen und den Bootloader laden kann.

3 Bootloader oder Bootmanager

Nachdem jeder Bootsektor auf Festplatten allerdings nur 512 Byte groß ist, einige Bootloader aber weitaus mehr Platz brauchen, werden die Bootloader in verschiedene Stages aufgeteilt. Der momentan gängigste Bootloader ist vermutlich GRUB, der „GRand Unified Bootloader“. Er braucht etwas mehr Platz, da er viele verschiedene Filesysteme liest. Unter anderem ext2, ext3, fat, ntfs usw. In der derzeit aktuellen Version 0.97 ist GRUB in drei Stages, die Stage1, Stage 1.5 und Stage 2 unterteilt. Die Stage1 liegt im MBR und hat im Prinzip nur die Aufgabe, die Stage 2 zu laden. Diese befindet sich „normalerweise“ unter `/boot/grub/stage2`, wenn man bei Linux von „normal“ sprechen kann. Die Stage 1.5 wird dann wichtig, wenn etwas an der Stage2 verändert oder diese ausversehen verschoben wird. Die Stage 1.5 liest genau ein Filesystem und ist normalerweise für das Filesystem installiert, auf welchem sich die Stage2 befindet. Daher sollte man vielleicht darauf achten, die Stage 1.5 anzugleichen, wenn man sich ein neues Filesystem für seine `/boot/-`Partition zulegt. In späteren Versionen, also ab Grub 2.0, bekommt Grub seinen eigenen Kernel und wird modularisiert aufgebaut. Das macht den Bootmanager sehr viel flexibler und kitzelt vermutlich auch noch etwas Performance heraus. Der Bootmanager hat jedenfalls die Hauptaufgabe, den Kernel zu starten.

4 Kernel

Kernel heißt übersetzt soviel wie Samen oder Samenkorn. Man merkt also schon deutlich, dass es etwas Grundlegendes sein muss. Der Kernel liegt vorerst komprimiert auf der Festplatte (oder dem entsprechenden Datenträger) und wird komprimiert in den Speicher geladen. Außerdem werden die Dekompressionsroutinen in den Speicher geladen, da man sonst mit dem Kernel nicht besonders viel anfangen könnte. Wird der Kernel einmal geladen, initialisiert dieser die Hardware, schaltet den Prozessor in den Protected Mode (was genau das ist, lernt man in Info 3, Betriebssystem und Netzwerke) und macht allerhand Magie, wie z.B. den PCI Bus aktivieren und den RAM initialisieren, das interessiert für das Verständnis des Bootvorgangs aber nicht weiter. Details findet man im Netz. Wichtig ist, zu wissen, dass der Ker-

nel verschiedene Module hat und verschiedene Module braucht. Der Kernel ist das Grundlegende des Betriebssystems, unter anderem enthält er die meisten Treiber. Um nun aber das System starten zu können, könnte man zu dem Punkt kommen, an dem man `/usr/sbin/init` starten möchte, aber nicht auf die Datei zugreifen kann, da sie auf einer SCSI Festplatte liegt. Das Kernelmodul für SCSI Festplatten scheint aber relativ groß und instabil zu sein, denn ansonsten würde es wohl kaum als Beispiel für ein Modul herangezogen werden, welches man nicht unbedingt in den Kernel kompilieren möchte. Vielleicht ist das auch schlicht nicht möglich, genaueres ist aber auch unwichtig. Um das Modul dennoch beim Bootvorgang benutzen zu können, hat man sich einen schlaunen Workaround ausgedacht. Man benutzt ein temporäres Filesystem im Arbeitsspeicher, die sog. `initrd`, „init Ram Disc“. Dort liegt das Modul und kann vom Kernel benutzt, d.h. geladen, werden, bleibt dort aber nicht dauerhaft liegen. Nachdem man nun auf die SCSI Festplatte zugreifen kann, kann der Kernel `/usr/sbin/init` suchen und starten.

5 init-deamon

Das kann mit verschiedenen Spielarten passieren. Früher wurde das im sog. „BSD-Style“ getan, was bedeutet, dass der `init` Prozess die `/etc/rc` auswertet und entsprechend der vorliegenden Skripte munter Prozesse startet. Im weiteren Verlauf wird der im Prinzip noch aktuelle, aber dennoch schon überholte „systemV-Style“ beschrieben. Beim SystemV-Style bzw. `sysvinit` (die Bezeichnungen sind da nicht eindeutig) wertet der `init` Prozess die `/etc/inittab` aus und kennt verschiedene Runlevels:

RL 0	halt
RL 1	Single User (ohne Netzwerk)
RL 2	Multiuser (ohne Netzwerk)
RL 3	Multiuser und Netzwerk
RL 4	Unused
RL 5	Multiuser, Netzwerk und X
RL 6	restart

Jedes Runlevel hat seine eigene Bedeutung und beinhaltet verschiedenste Funktionalität. In der `/etc/inittab` wird nach dem gesetzten Standardrunlevel gesucht und dieses wird initialisiert. Verschiedene Distributionen können verschiedene Standardrunlevels benutzen, selbst wenn sie aufeinander basieren. So kennt Debian als Standardrunlevel das Runlevel 2, Ubuntu benutzt Runlevel 5. Wenn ihr eure Kollegen ärgern wollt und zufällig eine unbeobachtete `root-shell` findet, könnt ihr ja mal ausprobieren, was passiert, wenn man

das Standardrunlevel auf 6 oder 0, setzt und wie lange euer Kollege braucht, um herauszufinden, was genau mit seinem PC los ist. Um aber wirklich die Funktionalität der einzelnen Runlevels auszunutzen, legt man die Scripte im entsprechenden Ordner an. Die Ordner heißen `/etc/rcX.d`, wobei das X für die Nummer des entsprechenden Runlevels steht. Genauer gesagt legt man symlinks auf die Scripte in `/etc/init.d` an, und zwar in folgendem Format:

```
S<Nummer><Scriptname> bedeutet /etc/init.d/<Scriptname> start  
K<Nummer><Scriptname> bedeutet /etc/init.d/<Scriptname> stop
```

Die Logik, die dahinter steckt, ist nicht besonders schwierig: S steht für „start“, K für „kill“. Beim Eingang in ein Runlevel (z.B. init 2) werden die Scripte in aufsteigender Reihenfolge der Nummern ausgeführt, verlässt man ein Runlevel werden die Scripte in ansteigender Reihenfolge zur 100 hin gestoppt. Nochdazu sollte man einige Konventionen beachten, wenn man die Runlevels bei einem Rechner bearbeitet, der von mehreren Menschen genutzt bzw. administriert wird, aber dazu findet man mit Hilfe von Google auch Genaueres.

Ist der Rechner also im entsprechenden Runlevel, ist der Bootvorgang soweit abgeschlossen und man könnte meinen, dieser Text würde hier enden. Aber Linux wäre nicht Linux, wenn man nicht für jede Komponente nicht mindestens eine Alternative in petto hätte.

6 Alternativen

6.1 Alternativen zum BIOS

6.1.1 coreboot

Fangen wir noch einmal von vorne an. Das BIOS ist proprietäre Software und man kann es dekompileieren, was aber ziemlich illegal ist, weil Intel nicht gerne öffentlich bewiesen sieht, dass seit Jahren am BIOS herum geflickschustert wird. Da aber kein echter Linux-Benutzer auch nur irgend einen Fetzen proprietärer Software auf seinem Rechner haben möchte, gibt es mehrere Möglichkeiten: Eine sehr interessante davon ist das Projekt „coreboot“. Das aus dem LinuxBIOS hervorgegangene Projekt hat sich zur Aufgabe gemacht, ein freies BIOS zu entwickeln und wird mittlerweile sogar ein bisschen von der Industrie unterstützt. coreboot soll den Bootvorgang beschleunigen und hat nette Spielereien wie die Möglichkeit des Fernzugriffs übers Netzwerk, was es besonders für Cluster-Systeme interessant macht. Leider werden mo-

mentan nur sehr wenige Mainboards unterstützt und es braucht noch eine Payload, also etwas wie die OpenFirmware, um wirklich booten zu können.

6.1.2 OpenFirmware / EFI

Statt eines BIOS haben nun aber z.B. die neuen Intel Macs ein sogenanntes EFI, ein „Extensible Firmware Interface“. Das kommt daher, dass Macs, damals noch mit PowerPC Architektur, schon immer eine etwas eigene Rolle spielen und bisher mit der Open Firmware booteten. Das BIOS ist von Haus aus nicht 64bit fähig. Die Unterschiede im Groben zwischen der OpenFirmware und dem EFI sind, dass EFI von Intel gepusht wird und die Open Firmware ein offener Standard ist, den sich einige kluge Leute einmal ausgedacht haben. Die grundlegende Funktionsweise der beiden Einrichtungen ist relativ faszinierend, da sie alle Geräte über vereinheitlichte Schnittstellen dem Betriebssystem zur Verfügung stellen. Das ist daher praktisch, da sowohl die OpenFirmware als auch das EFI System bzw. CPU-Architektur unabhängig läuft und z.B. Hersteller von Grafikkarten nur noch einen Treiber basteln müssen, der mit der OpenFirmware funktioniert, und diese die entsprechenden Funktionen dann weiterreicht. Allerdings ist die OpenFirmware je nach Standpunkt, sicherlich zumindest für den Laien, nicht besonders komfortabel, da sie den sogenannten F-Code versteht, eine hardwarenahe, assemblerähnliche Programmiersprache. Wenn man nun statt über das Netzwerk oder die Festplatte von einer USB-CD-Rom booten möchte, sollte man etwas in derart schreiben:

Mit

```
dev / ls
```

bekommt man den Baum der eingebundenen Geräte angezeigt

```
Open Firmware, 1.0.5
To continue booting the MacOS type:
BYE
To continue booting from the default boot device type:
BOOT
ok
0 > dev / ls
FF828F80: /PowerPC,604@0
FF829230: /l2-cache@0,0
FF8299F0: /chosen@0
FF829B20: /memory@0
FF829C68: /openprom@0
FF829D28: /AAPL,RDM@FFC00000
FF829F40: /options@0
FF82A618: /aliases@0
FF82A858: /packages@0
FF82A8E0: /debloader@0,0
FF82B0E0: /disk-label@0,0
FF82B620: /obp-tftp@0,0
FF82DA60: /mac-files@0,0
FF82E258: /mac-parts@0,0
FF82E9B8: /aix-boot@0,0
FF82EE30: /fat-files@0,0
FF830400: /iso-9660-files@0,0
```

```

FF830D48: /xcoff-loader@0,0
FF831708: /terminal-emulator@0,0
FF8317A0: /bandit@F2000000
FF832990: /gc@10
FF832DC8: /53c94@10000
FF834650: /sd@0,0
FF835280: /st@0,0
FF835EF8: /mace@11000
FF836D70: /escc@13000
FF836EC8: /ch-a@13020
FF837578: /ch-b@13000
FF837C28: /awacs@14000
FF837D10: /swin3@15000
FF838E18: /via-cuda@16000
FF8399A8: /adb@0,0
FF839A98: /keyboard@0,0
FF83A1E8: /mouse@1,0
FF83A298: /pram@0,0
FF83A348: /rtc@0,0
FF83A810: /power-mgt@0,0
FF83A930: /mesh@18000
FF83C498: /sd@0,0
FF83D0C8: /st@0,0
FF83DD0: /nvram@1D000
FF83FB70: /pci106b,1@B
FF83FD48: /ATY,XCLAIM@D
FF848968: /wayne.device@E
FF8494D8: /wayne.device@F
FF83DF68: /bandit@F4000000
FF84A190: /pci106b,1@B
FF84A368: /pci1234,5678@D
FF84A670: /pci1000,3@E
FF84A908: /TRUV,TARGA2000@PCI@F
FF83F1C0: /hammerhead@F8000000
ok
0 >

```

und mit:

```
boot /ht@0,f2000000/pci@2/usb@b/disk@1:2,\install\yaboot
```

startet man den Bootloader yaboot von der ersten Partition der CD-Rom, die am USB hängt. Die OpenFirmware sowie das EFI sind beide programmierbar. Allen denjenigen, die das ausprobieren möchten, wünsche ich an dieser Stelle viel Spaß. Von der etwas seriöseren, wirtschaftlicheren Seite betrachtet, bedeutet das, dass man in so ein EFI z.B. Digital Right Management einbauen kann oder sonstige Funktionen, die momentan aber wohl noch auf Entwicklungsniveau und nicht Standard sind. Die OpenFirmware kann übrigens auch als Bootmanager benutzt werden. Möchte man z.B. übers Netzwerk booten, so funktioniert das mit dem Kommando:

```
boot enet:0
```

6.2 Datenträger

Ein guter Grund, keine Festplatte in seinem Rechner verbaut zu haben, ist, keine bewegten Teile in seinem System haben zu wollen. Keine Bewegung heißt kein Geräusch. So kann man z.B. die Standardmethode wählen und eine CF-Karte mit einem entsprechenden Adapter zu einem IDE-Device aufbohren. Da das aber noch nicht interessant oder geekig genug ist, kann man

das BIOS oder die vorgestellten Alternativen auch dazu bringen, über das Netzwerk zu booten. Die entsprechende Vereinbarung zwischen BIOS und Netzwerkkarte nennt sich „PXE“, „Preeboot eXecution Environment“. Das ermöglicht, unabhängig von einem lokalen Datenträger oder einem Betriebssystem zu booten, und benutzt dabei allerhand an Protokollen. Im ungebooteten Zustand hat der Rechner natürlich noch keine IP-Adresse. Deswegen muss er diese mit einem DHCP-Server aushandeln, der ihm auch gleich mitteilt, wo er die Dateien zum Booten findet. Dann muss der Rechner über das TFTP (Trivial File Transfer Protocol) die Dateien in den Hauptspeicher laden und kann weitermachen. Möchte man komplett auf einen lokalen Datenträger verzichten, bietet es sich an, das /-Filesystem im Hauptspeicher anzulegen und den Inhalt über NFS (Network FileSystem) zu beziehen. Der Vorteil dieses Verfahrens ist die Betreuung mehrerer Rechner. Man muss nicht von einem Rechner zum anderen laufen, um Änderungen oder Aktualisierungen vorzunehmen, sondern man nimmt diese Änderungen am zentralen Image vor, das geladen wird, und schon sind ab dem nächsten Neustart alle Rechner, die das entsprechende Image beziehen, aktualisiert.

6.3 Alternative Bootloader / -manager

6.3.1 LILO

Der mehr oder minder Vorgänger von GRUB ist LILO, der „Linux Loader“. Mehr oder minder deshalb, da GRUB nicht wirklich eine neuere Version von LILO ist, sondern LILO vielmehr immer weiter verdrängt. LILO ist etwas schlanker, da dieser Bootmanager nicht vom Filesystem abhängig ist, sondern blockorientiert. Dies kann sowohl als Vor- als auch als Nachteil betrachtet werden. Einerseits ist es damit möglich, von exotischeren Filesystemen zu booten, andererseits muss man stets darauf achten, wenn man sich z.B. seinen Kernel neu kompiliert, ein „lilo“ auszuführen, da ansonsten natürlich die Adresse des Kernels nicht mehr stimmt und sie nicht neu an LILO weitergegeben wird. Und den Kernel, ohne auf das Filesystem zuzugreifen zu finden, wird vermutlich relativ schwierig.

6.3.2 Loadlin

Benutzt man lieber MS-Dos bzw. Windows-Systeme, könnte man sich auch mit Loadlin beschäftigen, ein Bootloader, der Linux aus einem laufenden System heraus bootet. Mein erster Gedanke war, das Vista-Dual-Boot-Problem so möglicherweise lösen zu können, hatte aber nicht mehr genug Zeit, um genauer recherchieren zu können. Zumindest kann man wohl per

Mausklick Windows beenden und Linux booten, was mitunter recht praktisch ist. Allerdings ist der Bootloader nicht mehr allzu verbreitet, sondern war früher wichtig, als der Kernel nicht hinter dem Sektor 1024 liegen durfte, auch ein Relikt aus BIOS-Zeiten, und man so dennoch den Weg zum Kernel und damit auch zum init-daemon finden konnte.

6.4 Kernel

Wer Linux schon etwas intensiver benutzt, der weiß, dass jeder seinen eigenen Kernel kompilieren kann. So gibt es zumindest keine wirkliche Alternative zum Kernel aber vermutlich tausende, alternative Kernels. Das wird in einem der folgenden Vorträge noch behandelt, betrachten wir also direkt die Alternativen des init-daemons.

6.5 Alternativen zum init-daemon

Der init daemon ist etwas veraltet. Er startet Prozesse nicht parallel und ist deshalb etwas langsam. Als Beispiele im Netz wurde der Startvorgang eines Servers beschrieben. Der Daemon init verhält sich wie folgt: Er wartet, bis das Netzwerk initialisiert ist, um dann z.B. Apache zu starten, dann wieder wartet, bis dieser läuft, um dann ein Datenbanksystem zu starten, um dann zu warten, bis dieses geladen ist, um endlich Mailanwendungen starten zu können. Da dieses Vorgehen nicht besonders effizient ist, haben sich einige kluge Köpfe Gedanken gemacht, wie man ihn beschleunigen könnte. Das Ergebnis heißt Upstart bzw. initNG

6.5.1 Upstart / initNG

Nachdem seit Ubuntu 6.10 Upstart verwendet wird, wird sich initNG vermutlich nicht wirklich durchsetzen, wobei diese Prognose von einem Linux-Laien abgegeben wird und möglicherweise völlig wertlos ist. Upstart wie initNG (NG für „next generation“) parallelisieren die Startvorgänge und haben noch einige andere nette Features: Sobald das Netzwerk initialisiert wurde, startet Upstart z.B. gleichzeitig Apache, das Datenbanksystem, und die Mailanwendungen, die dadurch viel früher geladen sind, als das init schaffen würde. Upstart ist außerdem ereignisorientiert, d.h. es wird z.B. nicht von Anfang an ein USB-Treiber geladen und gewartet, bis evtl. ein USB-Flash-Medium angesteckt wird, sondern erst wenn ein Flash-Medium angesteckt wird, werden die entsprechenden Module und Jobs geladen, um die Funktionalität zur Verfügung zu stellen. Dass diese Vorgehensweise vorteilhaft ist, ist vermutlich schnell ersichtlich. In nicht allzuferner Zukunft soll Upstart sogar den

cron-daemon ersetzen. Da er momentan aber sogar noch auf die alten init.d Skripte zugreift und langsam umgearbeitet wird, dauert das vermutlich noch einen Augenblick.

Natürlich ist das alles nur ein kurzer Anriss aller Möglichkeiten, und alle diese Alternativen werden sicherlich unvollständig und möglicherweise vereinfacht und / oder inkorrekt dargestellt. Meine Angaben mache ich ohne Gewähr und ich übernehme keine Verantwortung für Menschen, die sich aufgrund dieses Textes ihr System zerschießen. Think before you type.

Wer sich allerdings wirklich dafür interessiert, findet dazu jede Menge Informationen im Netz. Nach Schlagwörtern googlen bringt hier einiges.

7 Quellen

- http://de.opensuse.org/SDB:Booten_mit_der_initial_ramdisk
- <http://www.thomashertweck.de/kernel26.html>
- <http://de.wikipedia.org/wiki/GRUB>
- <http://de.wikipedia.org/wiki/LILO>
- http://de.wikipedia.org/wiki/Basic_Input_Output_System
- <http://de.wikipedia.org/wiki/Booten>
- <http://de.wikipedia.org/wiki/Upstart>
- <http://de.wikipedia.org/wiki/initNG>
- <http://software.magnus.de/betriebssysteme/artikel/der-bootvorgang-unter-linux.html>
- Dokumentation des PC Pools der Computerlinguisten
- <http://developer.apple.com/technotes/tn/tn1062.html>
- ...