4.7 Datenträger/Bootvorgang

4.7.1 Firmware

Nach dem Start oder dem Reset eines Computers befindet sich der Programmzähler der CPU auf einem definierten Startwert, auf einer bestimmten Programmadresse. Von dieser Adresse holt die CPU ihren ersten Befehl.

Bei vielen CPU-Architekturen ist der Startwert 0. Bei den heutigen PCs, die alle vom IBM-PC des Jahres 1981 abstammen, ist der Startwert dagegen 0xFFFFF ($= 2^{20} - 16$)[1].

An dieser Programmadresse *muss* bei jedem Computer ein Speicherbereich liegen, der schon beim Start definierte Inhalte aufweist, also ein ROM[2]. In diesem ROM ist ein Programm enthalten, das dazu dient, das System zu starten (beim PC) und/oder zu betreiben (bei *embedded Systems*). Eine solches Programm heißt *Firmware*.

Beim PC nennt man es je nach Version BIOS (*Basic Input/Output System*) oder UEFI (*Unified Extensible Firmware Interface*), wobei UEFI die neuere Version ist.

Nun könnte so eine Firmware durchaus das ganze Betriebsystem enthalten. Dann wären allerdings jede Änderung und jedes Update des Betriebsystems sehr aufwändig. Deshalb hat man es sich beim PC so ausgedacht, dass die Firmware nur einen kleinen Teil der Betriebssystem-Funktionialität bekommt. Sie darf nur die Hardware testen, den sogenannten POST ausführen (*Power On Self Test*). Danach sucht sie nach verfügbaren Massenspeichern (HDD, SSD, DVD, USB-Stick, ...), lädt davon ein Programm und startet es.

Der Vorgang besteht aus zwei Schritten (Abbildung 1):

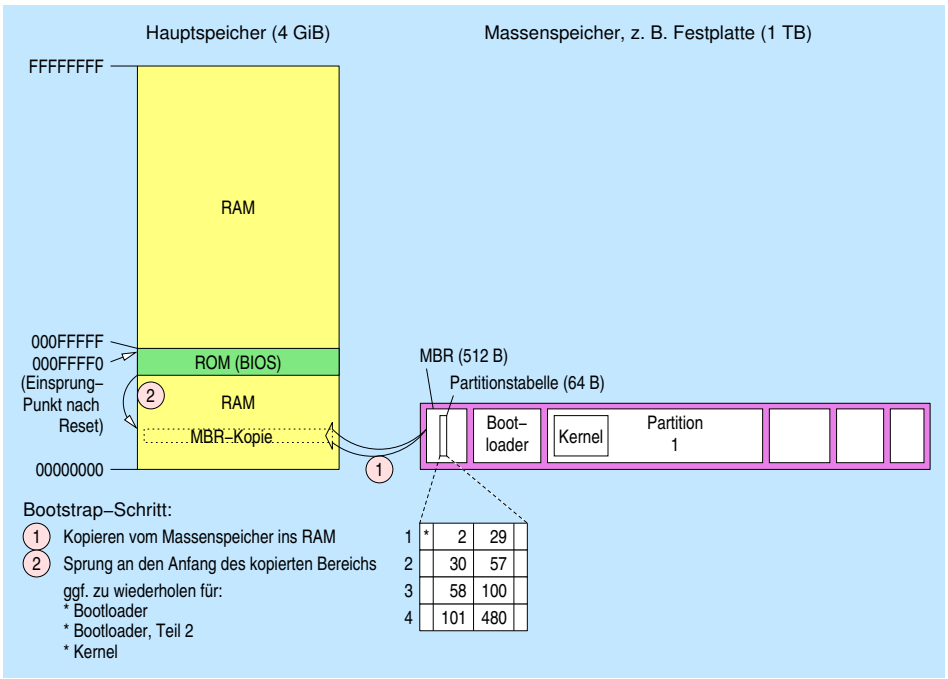

Abbildung 1: Bootstrap-Schritt

Schritt 1 Der Bootcode wird in einen Bereich am Anfang des RAMs geladen.

Schritt 2 Die CPU springt an den Beginn dieses Bereichs. Wenn dort ein funktionierendes Programm steht, kann es weitergehen. Wenn nicht, bleibt das System stehen.

[1] Diese Adresse lag 16 Bytes vor dem Ende des im *real mode* erlaubten Adressraums von 1 MiB.

[2] Wie dieses ROM nun technisch ausgeführt ist, kann unterschiedlich sein. Bei *embedded Systems* ist es oft ein MROM, beim PC ist es heutzutage ein Flash-EPROM.

Man nennt diesen Vorgang *Bootstrap*, und zwar nach der Redensart „sich selbst an den Schuhen (Boots) aus dem Sumpf ziehen“, vergleichbar mit der Geschichte von Baron von Münchhausen, der „sich selbst an den Haaren aus dem Sumpf zieht“. Tatsächlich hat das Rechnersystem am Anfang nur wenig Möglichkeiten, sich selbst aus dem Sumpf zu ziehen: Ein maximal 512 Bytes großes Programm, einen alten CPU-Modus, nur wenig RAM und keine Kenntnisse über Dateisysteme (nicht einmal über FAT).

Deshalb folgt auf den Bootcode fast immer ein zweite Bootstrap-Stufe, in der der richtige Bootloader geladen wird. Der kann z. B. ein Menü enthalten für die Auswahl zwischen verschiedenen Betriebssystemen. Und in der dritten Stufe könnte dann der Betriebssystem-Kern geladen werden.

4.7.1.1 Fall 1: BIOS-Firmware

Der Sektor Nr. 0 einer Festplatte heißt MBR (*Master Boot Record*). Er hat eine Größe von 512 Bytes und enthält darin

a) die Partitionstabelle (64 Bytes, **immer vorhanden**)

b) den Bootcode (440 Bytes, **optional**)

Falls das MBR keinen Bootcode enthält (Anfangsbyte ist 0), dann wird der Bootcode aus dem Sektor Nr. 0 (*boot record*) der Bootpartition benutzt. Zur Erinnerung: Die Bootpartition ist die Partition, bei der in der Partitionstabelle im MBR das Boot-Flag gesetzt ist[3].

Anders ausgedrückt: Man kann das Programmstück für den ersten Bootstrap-Schritt anstatt aus dem MBR auch aus dem Bootrecord der Bootpartition holen. In diesem Fall wird die Partitionstabelle des MBR nur dazu benutzt, die Bootpartition zu ermitteln (welche es ist) und zu finden (wo sie ist).

4.7.1.2 Fall 2: UEFI-Firmware

Auch bei UEFI wird nach der Initialisierung des Systems versucht, einen Bootloader zu lesen, allerdings keinen einzelnen Sektor, sondern den ganzen Bootloader auf einmal. Er liegt in einer gesonderten Partition mit dem Namen ESP (*efi system partition*).

Allerdings gibt es hier die Möglichkeit, in einer *secure boot* genannten Betriebsart zu diesem Bootloader einen Hash (eine Art Prüfsumme) zu berechnen. Diese Prüfsumme wird zunächst verglichen mit einer Liste (blacklist) von Prüfsummen in einer internen Datenbank DBX (*forbidden signatures database*) . Diese liegt im NVRAM, einem winzigen RAM, auf den die Firmware Zugriff hat. In einem weiteren Schritt wird nachgesehen, ob der Bootloader eine bekannte Signatur hat. Falls nein, wird sein Hash mit einer anderen Liste (whitelist) der Datenbank DB (*signatures database*) verglichen. Falls ja, wird die Signatur zuerst mit der einen (DBX) und dann mit der anderen (DB) Datenbank verglichen. Erst nach Bestehen dieser Tests kann es mit dem Start weitergehen. Ohne *secure boot* entfällt dieser Teil. Darüberhinaus gibt es bei den meisten Mainboards einen Kompatibilitätsmodus CSM (*compatibility support mode*), bei dem sich die UEFI-Firmware wie ein BIOS verhält. Tabelle 1 listet die zu UEFI gehörenden Mini-Datenbanken im NVRAM aus.

Abk.	Name	Bemerkungen
DB	signature DB	durch Betriebssystem aktualisierbar
DBX	forbidden signature DB	durch Betriebssystem aktualisierbar
KEK	key exchange keys	Aktualisierung nur mit Schlüssel aus KEK möglich
PK	platform key	Nur ein Schlüssel des Mainboard-Herstellers

Tabelle 1: UEFI-Datenbanken im NVRAM

Für solche Linux-Systeme, die im *secure boot*-Modus laufen sollen, gibt es einen Bootloader mit dem Namen Shim. Seine Signatur ist bei allen UEFI-Datenbanken enthalten.

[3] Das heißt, im MBR muss mindestens die Partitionstabelle vorhanden sein.

4.7.2 Bootloader

Der Bootloader dient dazu, den Kernel und die *initial ramdisk* ins RAM zu laden und zu starten. Komfortabel ist es, wenn der Bootloader ein Auswahlmenü für mehrere Betriebssysteme oder für mehrere Betriebsarten eines Betriebssystems anbietet. Bekannte Bootloader für BIOS sind:

a) LILO

b) GRUB 1 (genannt Legacy)

c) GRUB 2

Für UEFI gibt es:

a) ELILO

b) GRUB 2

c) Shim und PreLoader als Vorstufe im *secure-boot*-Modus

4.7.3 Kernel und *initial ramdisk*

Der Kernel ist im Grunde ein ganz normales Programm (bei Debian: `/boot/vmlinuz*`). Er muss aber für sich alleine lauffähig sein. Oft liegt er im Verzeichnis `/boot`. Das kann innerhalb des Wurzeldateisystems liegen. Für ältere BIOS- oder Bootloaderversionen ist es nötig, dass dieses Verzeichnis auf einer eigenen Partition innerhalb der ersten 1024 Zylinder einer Festplatte liegt.

Zunächst initialisiert der Kernel seine (eingebauten) Treiber. Auf Dateien und Verzeichnisse hat er zunächst keinen Zugriff. Warnungen und Fehlermeldungen speichert er deshalb zunächst im RAM, im sogenannten Ringpuffer des Kernels. Erst am Ende des Bootvorgangs wird er diese Meldungen in die Datei `/var/log/dmesg` speichern[4].

Im Kernel sind in der Regel schon sehr viele Treiber einkompiliert, aber nicht alle, die es jemals geben könnte. Nun kann der Fall eintreten, dass das Wurzeldateisystem, welches der Kernel einbinden soll, auf einer exotischen Hardware liegt, für die es im Kernel keinen Treiber gibt.

Zu diesem Zweck gibt es außer dem Kernel im Verzeichnis `/boot` noch eine zweite Datei, die *initial ramdisk* (bei Debian: `/boot/initrd.img*`). Sie enthält in sich das Abbild eines kleinen Dateisystems; in den abgebildeten Dateien liegen weitere Treiber für die genannte exotische Hardware.

Der Kernel bindet also zuerst die *initial ramdisk* ein. Erst dann wird (eventuell mit Hilfe eines Treibers aus der *initial ramdisk*) das Wurzeldateisystem eingebunden. Zum Schluss wird ein Programm mit dem Namen `init` gestartet.

4.7.4 Init-Prozess

Der Init-Prozess ist der einzige Prozess, der vom Kernel quasi von Hand gestartet wird. Er hat die Prozess-ID PID=1. Alle weiteren Prozesse werden durch den Init-Prozess gestartet und sind damit seine Kind-Prozesse. Der Init-Prozess hat noch eine zweite Aufgabe: Alle Prozesse auf dem System, die von ihren Eltern-Prozessen nicht wie erwartet behandelt werden, werden Kind-Prozesse von Init.

Es gibt bei Linux drei gängige Versionen von Init:

a) Sys-V-init: Die traditionelle UNIX-Variante, die ihren Namen von AT&T UNIX Version 5 ableitet – erkennbar an der Datei `/etc/inittab` und Skripten im Verzeichnis `/etc/init.d`

b) Upstart: Bei Ubuntu verwendete Variante, – erkennbar am Verzeichnis `/etc/init`

c) Systemd: Bei Fedora und Debian benutzte Variante, – erkennbar am Verzeichnis `/etc/systemd`

[4] Mit dem Befehl `dmesg` kann man sie dann als Benutzer nachlesen.

4.7.5 Beispiel für einen Bootloader: GRUB2

Ein häufig benutzer Bootloader unter Linux ist GRUB2, die zweite Version des *grand unified bootloader*-Programms.

GRUB benutzt im Vergleich zu Linux eine andere Bezeichnung der Massenspeicher:

a) `(hd0)` ist die erste Platte (wie `/dev/sda`)

b) `(hd0,3)` ist die dritte Partition der ersten Platte (wie `/dev/sda3`)

Die Erstinstallation von GRUB erfolgt mit dem Befehl `grub-install`, gefolgt von der Bezeichnung der Platte oder der Partition. Hier wird GRUB im MBR der ersten Platte installiert:

Terminal
```
root@debian964:~# grub-install "(hd0)"
```

Und hier im Bootsektor der dritten Partition der ersten Platte:

Terminal
```
root@debian964:~# grub-install "(hd0,3)"
```

Die eigentlichen Binärdateien, die GRUB ausmachen, sind

- `/boot/grub/i386-pc/boot.img`
- `/boot/grub/i386-pc/boot.img`
- weitere Module in `/boot/grub/i386-pc/`

Sie werden beim Installieren in die Boot-Region kopiert[5].

Die Konfiguration liegt in der Datei `/boot/grub/grub.cfg`. Diese Datei sollte allerdings nicht von Hand geschrieben werden (sie wird bei jedem Kernel-Update überschrieben), sondern sie wird von folgendem Befehl erzeugt (untere Zeile: Alternative):

Terminal
```
root@debian964:~# update-grub
root@debian964:~# # /usr/sbin/grub-mkconfig -o /boot/grub/grub.cfg
```

Das Programm `update-grub` holt sich die Einzelheiten aus den eigentlichen Konfigurationsdateien im Teilebaum `/etc`:

- `/etc/default/grub`
- `/etc/grub.d/[0-9][0-9]*`

Die Datei `/etc/default/grub` liefert die allgemeinen Einstellungen für das Menü. Mit der folgenden Zeile kann man dafür sorgen, dass ohne Eingreifen des Nutzers immer der oberste Eintrag ausgewählt wird (GRUB fängt offenbar immer bei 0 an zu zählen):

```
1 GRUB_DEFAULT=0
```

Mit der Zeile `update-grub` wird die Änderung in die Installation eingebaut – fertig.

Die Dateien `/etc/grub.d/*` enthalten Skripte, die in sortierter Reihenfolge das Bootmenü aufbauen. Dabei liegt `10_linux` weit vorne, andere Betriebssysteme mit `30_os-prober` und `40_custom` weiter hinten. Wer ein anderes Betriebssystem im Menu vorne haben will, ändert einfach die Nummern und ruft wieder `update-grub` auf.

Die Befehle in den Skripten in `/etc/grub.d/*` werden *nicht* direkt nach `/boot/grub/grub.cfg` kopiert. Stattdessen werden sie (von `update-grub`) *ausgeführt*, und ihre Ausgaben landen dann in `/boot/grub/grub.cfg`.

So könnte man ein Passwort setzen, indem man in die Datei `/boot/grub/grub.cfg` die Zeilen schreibt:

[5] Außerdem gibt es noch Bibl.-Dateien in `/usr/lib/grub`.

```
set superusers="schueler"
password schueler schule
```

Damit das passieren kann, schreibt man in das Ende der Datei `/etc/grub.d/00_header` die Zeilen:

```
echo set superusers="schueler"
echo password schueler "schule"
```

Falls aber das Passwort von `schueler` Steuerzeichen enthält, ist es besser, mit einem Here-Document zu arbeiten und dies hier an das Ende von `/etc/grub.d/00_header` zu schreiben:

```
cat << HEINZ
set superusers=schueler
password schueler schule
HEINZ
```

Das bewirkt das Gleiche, ist aber sauberer, weil man den `echo`-Befehl umgeht. Nach dem Aufruf von `update-grub` ist die Neuerung in der Datei `/boot/grub/grub.cfg` eingebaut (kann man mit einem Editor ansehen) und steht beim nächsten Neustart zur Verfügung.