BOOTSTRAPPING

Konzepte von Betriebssystem-Komponenten

Friedrich-Alexander-Universität
Erlangen-Nürnberg
Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Sommersemester 2005

Krasovytskyy Oleg
ok82@gmx.net

Gliederung.

1. Einleitung.

Wie funktioniert es nun, einen Rechner, der ja beim Einschalten erst mal kein Betriebssystem hat, zu starten? Es gibt ja nichts, was der Computer erst mal selbst weiß. Dieses generelle Problem, ein System zu starten, wird "Bootstrapping" genannt (Problem deswegen, weil der Computer zum Zeitpunkt des Einschaltens nur sehr wenige Ressourcen benutzen kann, da ja alle Treiber und Erweiterung erst durch das Betriebssystem geladen werden, dies aber noch nicht aktiv ist).

Der Begriff "Booting" oder "Bootstrapping" geht übrigens auf eine Geschichte des Barons von Münchhausen zurück, der sich an seinen eigenen Schnürsenkeln (engl. bootstraps) aus einem Sumpf zog, und beschreibt gut das Problem, das ein Rechner nach dem Einschalten zu bewältigen hat.

2. Initialisierung der Hardware

Als erstes muss untersucht werden, ob Hardware richtig angeschlossen ist und funktionsfähig ist. Es wird zum Beispiel nach dem Hauptspeicher geschaut, die Grafikkarte oder der Grafikchip untersucht.

2.1 Bootstrap Programm.

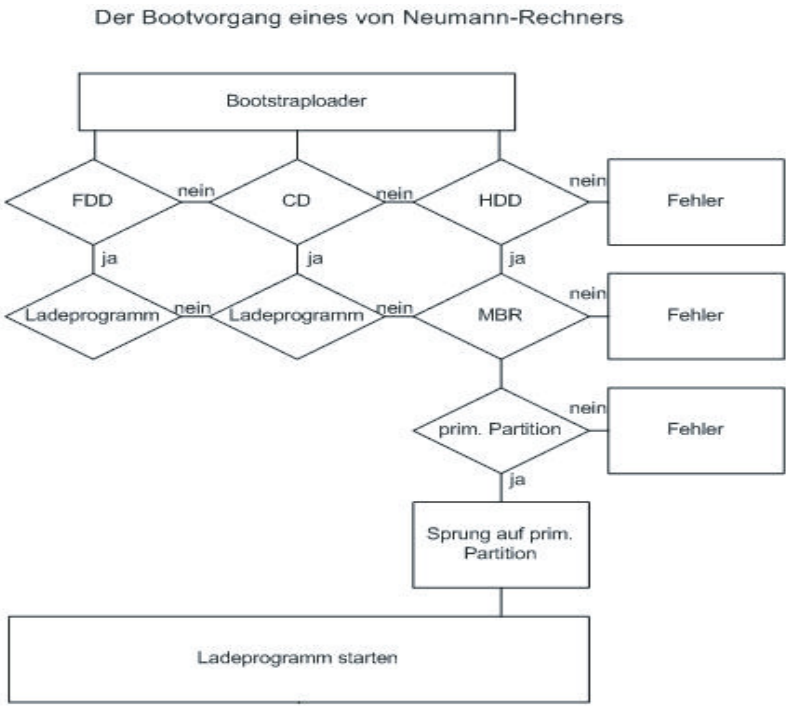
Um den Rechner hochzufahren benötigt man ein initiales Programm – Bootstrap-Programm. Das Programm ist relativ simpel. Es ist im nichtflüchtigen ROM-Speicher (EEPROM) in der Hardware gespeichert. Dessen Aufgabe ist es die CPU-Register, Geräte-Kontroller und weitere Hardware zu initialisieren. Dazu soll es noch fähig sein den Kernel des Betriebsystems zu lokalisieren, ihn in Hauptspeicher zu laden und anschliessend auszuführen. Da es im ROM gespeichert ist, bereitet es ein Problem den Code des Programms auszuwechseln. Und das ist ein grosser Nachteil, da die Betriebsysteme sich
ständig weiterentwickeln. Deshalb entwickelte man bei Computern eine zweistufige Technik:

1. Es wird nur ein **winziges** Bootstrap-Programm (**BIOS-Basic Input Output System**) im ROM gespeichert, dessen Aufgabe ist es
2. das **ganze** Bootstrap-Programm von der **Platte** zu laden. Und dieses kann leicht geändert werden.

Dieses volle bootstrap Programm ist an einer festen Position auf der Festplatte gespeichert, die **boot block/sector** genannt wird. Die Platte, die den boot sector enthält wird **boot disk (system disk)** genannt.

2.2 Bootprozess.

Wenn man den Rechner einschaltet bzw. ein Neustart gemacht wird, beginnt der Prozessor an einer festgelegten Adresse mit der Abarbeitung des im ROM abgelegten BIOS. Dieses führt einen Test der angeschlossenen Geräte durch **(POST-Power On Self Test)** und durchsucht folgende Geräte in einer (heutzutage konfigurierbaren) Reihenfolge nach boot sectoren.



Standardmäßig ist z.B. die Boot-Reihenfolge (default boot sequence):
Diskettenlaufwerk (FDD), CD-Laufwerk, Festplatte (HDD),
d.h. das BIOS versucht zuerst von dem Diskettenlaufwerk (Laufwerk A:) zu booten (den boot sector dort zu finden). Befindet sich eine Diskette im Laufwerk, erkennt aber das BIOS, dass sich kein Betriebssystem auf der Diskette befindet (no system disk = keine bootfähige Diskette) so beschwert sich das BIOS und fordert den Benutzer auf, eine andere Diskette einzulegen. Befindet sich gar keine Diskette im Laufwerk, so versucht das BIOS als nächstes das **CD-Laufwerk.** Falls es hier auch keine boot-fähige CD im Laufwerk gab, wird anschließend von der **Festplatte (C:)** gebootet.

Hier liest das BIOS von der ersten angeschlossenen Platte den ersten Sektor **(Master Boot Record)** in den Hauptspeicher. Dieser beinhaltet ein kleines Programm (**Boot-Loader**) und die **Partitionstabelle**. In dieser Tabelle sind alle Partitionen, in die die Festplatte aufgeteilt ist, vermerkt. Von diesen wird eine, die bootfähig (enthält den boot sector) und aktiv ist, gewählt.

Als nächstes wird von dieser Partition wiederum der erste Sektor **(Boot-Record)** gelesen und die Kontrolle an ihn übergeben. Das Programm im boot sector liest das Wurzelverzeichniss der Partition auf der Suche nach dem **Kernel** des Betriebsystems, lädt ihn in den Speicher und führt ihn aus.

Ein Beispiel für Boot-Loader ist **GRUB** (**GR**and **U**nified **B**oot-loader). Er ist in zwei Stufen aufgeteilt:

1. Die erste Stufe (**stage1**) kann in den MBR (zusammen mit der Partitionstabelle), in boot sector einer Partition oder einer Diskette geschrieben werden. Aufgabe von Programmcode dieser Stufe, die sehr klein ist (512 Byte), die zweite Stufe zu laden. Hier wird vermerkt an welcher physikalischen Stelle der Festplatte die "stage2" zu finden ist.

2. Die zweite Stufe (**stage2**) stellt die eigentlichen Funktionen des Boot-Loaders bereit: u. a. das Auswahlmenü, die GRUB-Shell und den Programmcode, mit dem die Steuerung des Rechners an das Betriebssystem übergeben wird.

Der größte Unterschied zu anderen Boot-Loadern (z.B. LILO) besteht darin, dass GRUB mehrere Dateisysteme direkt unterstützt. Dadurch, dass GRUB noch vor dem Booten auf das Dateisystem zugreifen kann, ist es nicht notwendig die physikalische Position der GRUB-Konfigurations-Datei und auch vom Kern des Betriebsystems zu kennen, solange nur die Angabe der Partition und der Pfad des zu bootenden Kernels in der Menüdatei richtig eingetragen ist.

3. Initialisierung der Software.

Nachdem Boot-Record in den Hauptspeicher geladen wird, durchsucht er dasWurzelverzeichniss der Partition nach der Datei, die dann das Betriebsystem lädt. Diese Datei unterscheidet sich von System zu System.

3.1 Starten von Windows 2000/NT.

ntldr.exe: Unter Windows 2000/NT heißt die Datei ntldr.exe. Das Programm führt weitere Initialisierungen durch, z.B. das Mini-Datei-System, dessen Aufgabe ist es, andere Systemdateien über die unterschiedlichen Pfade zu finden. Jetzt wird die Datei **boot.ini** eingelesen, auf dem Bildschirm wird dabei ein Menü angezeigt, das auflistet welche Betriebssysteme noch geladen werden können.

ntdetect.com: Falls Windows ausgewählt wird, wird das Programm ntdetect geladen und ausgeführt. Dieses untersucht den Typ der Hardware-Komponenten und gibt diese Information an **ntoskrnl.exe** weiter. Es lädt noch alle restlichen Treiber in den Speicher, die in der Registrierung aufgelistet sind. Jetzt wird die Kontrolle an das Programm ntoskrnl.exe übergeben.

ntoskrl.exe: Führt noch einige Initialisierungen durch und startet den ersten Prozess **smss.exe** (Sessions-Manager). Dieser startet einen weiteren crss.exe (Win32-Umgebung). ntoskrnl macht noch weitere Aufgaben, und startet am Ende den Anmeldedienst - **winlogon.exe.** An dieser Stelle ist das System hochgefahren und läuft. Winlogon erzeugt den Authentifizierungs-Manager (**lsas.exe**) und den Vater aller Dienste (**services.exe**).

3.2 Starten von Unix.

Unter Unix heißt der Kern normalerweise **/unix.**

(1) Startcode von dem Kern ist in Assembler geschrieben:

- Stack für den Kern bereitstellen,
- CPU-Typ erkennen
- Größe der Arbeitsspeicher berechnen und s. w.

(2) die **main-Funktion,** die in C geschrieben ist, wird aufgerufen:

- Ein **Puffer** wird für Nachrichten aufgesetzt, um bei der **Fehlersuche** bei Boot-Problemen zu helfen.
- Die Kern-Datenstrukturen werden alloziert.
- Autokonfiguration des Systems.

Bei der **Autokonfiguration** des Systems werden alle angeschlossene Ein-/Ausgabegeräte erkannt und die nötigen Treiber dazu dynamisch geladen (nicht in allen Systemen wird dynamisches Laden verwendet).

(3) Nun wird der Prozess mit Nummer 0 kreiert (Kernel-Prozess **/unix**). Dieser setzt die Initialisierung weiter fort:

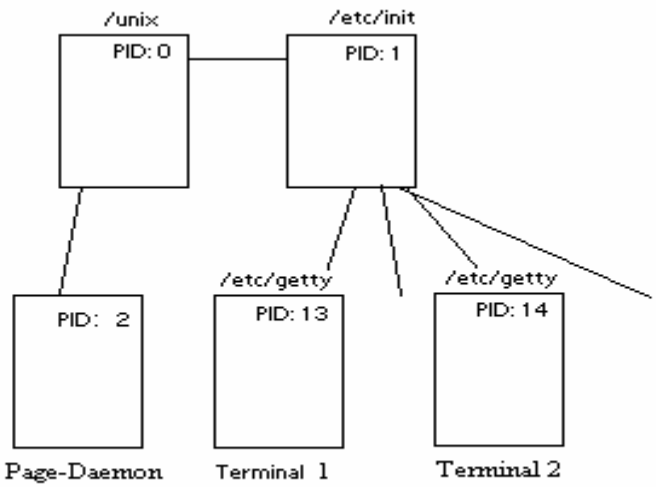
- Einrichten der Echtzeituhr;
- Mounten des Wurzelverzeichnisses;
- Erzeugen von **init (Uhrprozess -** Nummer 1**)** und den **Page-Daemon** (2. Prozess).

(4) Init-Prozess startet abhängig von seinen Parametern entweder Single- oder Multi-User-Modus.

Single-User-Modus: wird eine Shell gestartet und auf deren Beendigung gewartet.

Multi-User-Modus:

startet der init-Prozess einen Sohnprozess für alle zu aktivierenden Anschlüsse (Terminals). Danach wird auf jedem Terminal das Programm **/etc/getty** gestartet und auf dem jeweiligen Bildschirm das **login:** ausgegeben. Dieses Programm ruft **/bin/login** aus, das den Benutzer authentifiziert. Und schließlich wird **/bin/sh** aufgerufen. Hier befindet sich der Benutzer auf der Unix-Kommandoebene und kann Kommandos eingeben.



Beispiel für eine Unix-Systemhierarchie

4. Zusammenfassung.

Moderne Betriebsysteme benutzen ein mehrstufiges Bootstrap-Programm um den Rechner hochzufahren. Ein Teil von diesem Programm befindet sich im nicht flüchtigen Speicher im ROM (BIOS). Das BIOS führt dann Initialisierung von Hardware durch und lädt den ersten Sektor von der Platte (MBR). Das zweite Teil des Programms befindet sich in diesem MBR und kann den ganzen Bootloader von beliebiger Stelle auf der Platte laden. Letzter kann dann das Betriebsystem laden, indem er den Kern des Systems in den Hauptspeicher lädt und die Kontrolle an ihn übergibt.

Quellen:

[1] "Operating System Concepts" Silberschatz, Galvin, Gagne.
Sixth Edition, 2002;
[2] "Moderne Betriebsysteme" Andrew S. Tannenbaum.
2.Auflage, Pearson Studium. 2002;
[3] "Windows NT 4.x Workstation" Jürgen Ortmann.
1997;
[4] "UNIX Grundlagen" Helmut Herold.
4. überarbeitete Auflage, 1999.