

Der Linux-Bootprozess und die Init-Scripte

Linux-Proseminar Sommersemester 2002
Universität Ulm

Tobias Nick Fenster

24. April 2002

Inhalt:

1 Zusammenfassung	03
2 Booten mit LILO von Festplatte	04
2.1 BIOS, CMOS und Bootstrap-Loader	04
2.2 Bootvorgang	04
2.2.1 Bootmanager	04
2.2.2 Kernelauswahl	05
2.2.3 Kernel	05
2.3 Init-Scripte	06
2.3.1 Init	06
2.3.2 Runlevels	06
2.3.3 Init-Scripte	07
3 Alternativen	08
3.1 Bootmanager	08
3.1.1 ChOS und GRUB	08
3.1.2 LoadLin	08
3.1.3 NT-Loader	08
3.2 Booten über Netzwerk	09
3.2.1 Initialisieren	09
3.2.2 DHCP	09
3.2.3 Kernel	09
3.2.4 NFS oder Ramdisk	09
3.2.4.1 Init? (Ramdisk)	09
3.2.4.2 Dateisystem (Ramdisk)	10
3.2.5 Init!	10
3.2.6 Runlevel	10
4 Quellen	11

1 Zusammenfassung

Das Betriebssystem erweckt einen Computer erst zum Leben und ermöglicht es, mit ihm zu arbeiten. Doch ein Betriebssystem muss zunächst einmal gestartet oder "gebootet" werden. Dazu wird unter Linux ein Bootprozess ausgeführt und anschliessend das System initialisiert, also konfiguriert und grundlegende Funktionen gestartet.

Dabei ist es möglich, ein Linux-System von einer Festplatte zu Booten, wobei ein Bootloader zum Einsatz kommt. Dabei gibt es diverse Varianten, welcher Bootloader verwendet werden soll. Eine andere Möglichkeit unter Linux besteht darin, das System über ein Netzwerk zu starten. Dabei ist nicht einmal eine Festplatte nötig, da alle benötigten Informationen von einem Server zum Zeitpunkt des Systemsstarts geholt werden können.

2 Booten mit LILO von Festplatte

Um mit einem Computer arbeiten zu können, wird ein Betriebssystem benötigt. Ein solches Betriebssystem muss allerdings zuerst einmal geladen werden. Wie das Betriebssystem Linux geladen wird, soll im Folgenden ausgeführt werden.

2.1 BIOS, CMOS und Bootstrap-Loader

Beim Starten eines Computers wird das BIOS (Basic Input/Output System) gestartet. Dieses BIOS befindet sich in einem ROM (Read-only Memory), also einem Speicher, aus dem gelesen, aber nicht flüchtig geschrieben werden kann. Dieses führt zunächst einen sogenannten "Power-On-Self-Test" durch. Dabei wird unter anderem ermittelt, wie viel Systemspeicher sich in dem Computer befindet, ob eine Grafikkarte zur Ausgabe und eine Tastatur zur Eingabe vorhanden sind und ähnliches mehr. Ausserdem kennt der Computer ein CMOS (Complementary Metal Oxid Semiconductor), ein RAM-Speicher, in den auch flüchtig geschrieben werden kann (RAM = Random Access Memory). In diesem wird beispielsweise die Uhrzeit aber auch Einstellungen zu vorhandenen Festplatten oder anderen Laufwerken festgehalten. Das CMOS wird meist über ein Setup gesteuert, das gestartet werden kann, direkt nachdem der Computer in Betrieb genommen wurde.

Am Ende des BIOS befindet sich der Bootstrap-Loader, der mit Hilfe der Informationen aus dem CMOS einen Bootsektor sucht. Ein Bootsektor ist ein spezieller Sektor, der sich am Anfang eines Speichermediums wie einer Festplatte, einer CD-ROM oder einer Diskette befinden kann. Gekennzeichnet ist ein Bootsektor dadurch, dass sich in seinen letzten beiden Bytes, also an der Stelle $0x1FE = 510$ die Zahl $0xAA55 = 43603$ befindet. Diese Zahl wird auch als magische Zahl bezeichnet. Der Bootstrap-Loader geht also gemäß der Informationen aus dem CMOS durch die verfügbaren Speichermedien und sucht nach einem Bootsektor. In einem Bootsektor befindet sich in den ersten 446 Bytes ein Programm, das den weiteren Bootverlauf anstößt, sowie in den folgenden 64 Bytes weitere Verwaltungsinformationen (meist eine Partitionstabelle). Der Bootstrap-Loader lädt also das im Bootsektor befindliche Programm und übergibt die Kontrolle an dieses. Alles bis hierhin ist noch völlig unanhängig vom verwendeten Betriebssystem.

2.2 Bootvorgang

2.2.1 Bootmanager

Nun beginnt der eigentliche Bootvorgang: auf Linux-Systemen wird zum Grossteil der Linux-Loader, kurz LILO, verwendet. In diesem Fall ist im Bootsektor ein Teil von LILO, der First-Stage-Boot-Loader enthalten. Der First-Stage-Boot-Loader hat lediglich die Aufgabe, den nächsten Teil von LILO, den Second-Stage-Boot-Loader zu laden. Die zweigeteilte Architektur des LILO liegt darin begründet, dass im Bootsektor ja nur 446 Bytes zu Verfügung stehen. In diese 446 Bytes wird also nur ein Programm gelegt, das dann den eventuell wesentlich größeren, eigentlichen Bootloader startet. Wenn dieser zweite Teil des LILO, der Second-Stage-Boot-Loader erfolgreich geladen wurde, erscheint eine Eingabe zur Auswahl des gewünschten Betriebssystems. Vor dieser Eingabe, die auch Boot-Prompt genannt wird, steht im Falle eines erfolgreichen Ladens beider Boot-Loader LILO:.
Wenn es ein Problem gab, erscheint nur ein Teil und einer der folgenden Fehler ist aufgetreten:
L: Die erste Stufe des LILO kann die zweite Stufe nicht finden
LI: Die zweite Stufe wurde geladen, kann aber nicht ausgeführt werden.
LIL: Die zweite Stufe kann die Datei mit Informationen über die Lage der Betriebssysteme nicht finden.
LIL?: Die Daten des LILO wurden an eine falsche Speicheradresse geladen.
LIL-: Die Datei mit Informationen über die Lage der Betriebssysteme ist fehlerhaft.

2.2.2 Kernelauswahl

Der Kern eines Linuxsystems ist der sogenannte Kernel. Dieser steuert vereinfacht gesagt die Nutzung der Hardware durch die Software. So entscheidet der Kernel durch das "Scheduling", welcher Prozess wann den Prozessor "benutzen" darf. Mit dem "Virtual Memory Management", das ebenfalls der Kernel übernimmt ist gemeint, dass der Kernel entscheidet, welche Teile welcher Programme sich im Arbeitsspeicher befinden und welche Teile in den virtuellen Speicher, den "Swap" ausgelagert werden. Ausserdem steuert der Kernel das Dateisystem, die Interprozess-Kommunikation und einige netzwerk-bezogene Dinge.

An der Eingabe des LILO (dem Bootprompt) kann nun entschieden werden, welcher Kernel geladen werden soll, da es sehr viele Möglichkeiten gibt, einen Kernel zu konfigurieren. Somit können mehrere unterschiedlich konfigurierte Kernel erstellt werden und beim Booten kann gewählt werden, welche Kernelkonfiguration verwendet werden soll. Welche verschiedenen Kernel zum booten zur Verfügung stehen, wird in der Datei lilo.conf eingestellt, die sich im Verzeichnis /etc befindet. Darin wird für jeden vorhandenen Kernel eine Bezeichnung definiert (zum Beispiel SuSE für den von SuSE standardmäßig konfigurierten), das dann am Bootprompt gewählt werden kann. Wenn eine gültige Bezeichnung eingegeben wurde, wird der zugehörige Kernel geladen, also in den Systemspeicher verschoben.

2.2.2 Kernel

Der Kernel befindet sich irgendwo auf einem Speichermedium, wo ihn LILO mittels der Informationen aus seiner Konfigurationsdatei findet. LILO lädt dann den Kernel in den Systemspeicher, dekomprimiert und startet ihn. Dabei liest LILO den Kernel Sektor für Sektor, da LILO noch keinen Dateisystem-Zugriff auf das Speichermedium hat, sondern den Zugriff über das BIOS. Daher wird in den meisten Linux-Distributionen eine /boot-Partition angelegt, die sich komplett in den ersten 1024 Zylindern eines Speichermediums befindet und in diese werden alle möglichen Kernel geschrieben, da alte BIOS-Versionen nur auf die ersten 1024 Zylinder einer Festplatte zugreifen konnten. Je nachdem wie gross der Kernel ist, wird er in einen bestimmten Speicherbereich geladen, der 512 kByte gross ist oder in einen größeren Speicherbereich, der ein Mbyte fasst. Der Kernel liest die BIOS-Angaben über Grafikkarte und ihr Ausgabeformat sowie Informationen über alle elementaren Schnittstellen des Mainboards. Ausserdem prüft der Kernel den vorhandenen Prozessor auf bekannte Fehler und umgeht diese. Des weiteren werden grundlegende Netwerkdienste initialisiert wie zum Beispiel die unterstützten Protokolle. Danach wird ein Kernel-Thread für das Programm init gestartet, das wiederum selbst zwei weitere Kernel-Threads für kswapd und bdfush startet, die den Kernel bei der Verwaltung des virtuellen Speichers und des Buffer-Caches unterstützen. Der Kernel startet dann setup, ein Systemcall, der weitere Funktionalitäten des Kernels anstößt. Die im Kernel enthaltenen Treiber versuchen hierbei, die vorhandene Hardware zu initialisieren und für die Benutzung vom Kernels aus bereit zu machen. Anschliessend werden Partitionen auf vorhandenen Festplatten geprüft und das Root-Filesystem, also die Wurzel aller Dateisysteme gemountet. Mounten bedeutet, dass eine entsprechende Partition für das Linuxsystem verfügbar gemacht wird.

Ein Dateisystem bringt so etwas wie eine hierarchische Struktur auf eine Festplatte. Es sorgt dafür, dass man über Verzeichnisse, Unterverzeichnisse und Dateien auf den Inhalt einer Partition zugreifen kann. Bei jedem Booten wird das Dateisystem jeder gemounteten Partition auf seine Konsistenz überprüft.

Der Kernel mountet also das Wurzelverzeichnis, um Zugriff beispielsweise auf init zu haben. Der Kernel mountet allerdings nur Read-Only, also nur zum Lesen, weil init in einem späteren Schritt die Dateisysteme überprüfen wird. Init prüft dann das Dateisystem mit fsck und mountet es read-write, also mit Lese- und Schreibzugriff erneut, wenn alles ok war bzw. fsck die Fehler beheben kann. Wenn fsck die Fehler nicht automatisch beheben kann, bleibt das betreffende Filesystem read-only gemountet und Linux bootet in den Einzel-Benutzermodus, in dem der Systemadministrator dann das Dateisystem reparieren kann. Das meistverbreitete Dateisystem unter Linux ist ext2, aber es gibt Alternativen wie ReiserFS, ext3 und einige mehr.

Damit ist die Verbindung vom Kernel zum Dateisystem gegeben und das Programm init kann nun gestartet werden. Der bereits erstellte Kernel-Thread wird zum normalen Prozess und startet.

2.3 Init-Scripte

2.3.1 Init

Es gibt verschiedene Varianten von init, da sich aber unter Linux größtenteils das SystemV init durchgesetzt hat, sei hier nur dieses erwähnt. Ausserdem ist die Konfiguration von init sehr stark distributionsabhängig, weshalb hier nur das Bootkonzept von SuSE Linux dargelegt werden soll. Init ist der erste Prozess, der nach dem Kernel gestartet wird und erhält deshalb die Prozessnummer 1. Init kann auch nicht beendet werden, selbst wenn es das Signal SIGKILL erhält, das jeden anderen Prozess sofort stoppt. Der Kernel sucht init an einigen verschiedenen Stellen, aber die korrekte Stelle wäre /sbin/init. Das komplette weitere Hochfahren des Systems wird gemäß seiner Konfiguration in /etc/inittab von init gesteuert.

Zunächst werden allerdings noch die System-Logger von init gestartet und dienen dazu, Meldungen aller Prozesse zu protokollieren. Dabei wird unterschieden in Kernmeldungen und Meldungen anderer Prozesse: Kernmeldungen werden vom Dämon klogd behandelt, die anderen von syslogd. Hauptsächlich wird in die Datei /var/log/messages protokolliert, aber die System-Logger sind in der Datei /etc/syslog.conf konfigurierbar. Dabei werden die Meldungen nach dem Programm, von dem sie kommen und deren Prioritäten kategorisiert.

2.3.2 Runlevels

Runlevel	Funktion
0	System halt
1/S	Einzelbenutzer-Modus
2	Mehrbenutzer-Modus ohne Netz
3	Mehrbenutzer-Modus mit Netz
5	grafischer Mehrbenutzer-Modus mit Netz
6	Reboot

In der etc/inittab werden sogenannte Runlevel definiert sowie die Programme und Dämonen, die in jedem dieser Runlevel gestartet werden. SuSE Linux kennt sieben verschiedene Runlevel: 0 bis 3, 5, 6 und S. Dabei steht Runlevel 0 für das Anhalten des Systems, Die Runlevel 1 und S starten das System in einem Einzelbenutzer-Modus, in dem sich nur der Systemadministrator anmelden kann und diesen zur Konfiguration des Systems nutzen. Runlevel 2 ist ein Mehrbenutzer-Modus, in dem allerdings keine Netzwerkunterstützung zur Verfügung steht. Diese steht dann im Runlevel 3 bereit. Ein grafischer Login wird zusätzlich im Runlevel 5 gestartet. Das Runlevel 6 schliesslich steht für einen Reboot des Systems. In der /etc/inittab wird ein Default-Runlevel festgelegt, der gestartet wird, wenn am LILO-Prompt nichts anderes angegeben wurde.

Da sich alle im weiteren verwendeten Dateien in /etc befinden, werde ich diesen Pfad voraussetzen und nur die ab hier relativen Pfade nennen. Wenn nun also entweder das Standard-Runlevel oder ein anderes gestartet wird, werden alle dafür festgelegten Programme und Dämonen gestartet. Beim Wechsel von einem Runlevel in ein anderes werden alle Programme oder Dämonen des alten Runlevels gestoppt und alle des neuen gestartet. Ein Beispiel: der Systemadministrator – denn nur er darf die Runlevel wechseln – befindet sich in Runlevel 3 und führt init 5 aus. Um nun unnötige Starts und Stopps zu vermeiden, werden nur diejenigen Stopps in Runlevel 3 ausgeführt, die keine Start in Runlevel 5 haben, da ein solcher Prozess ja unnötigerweise zunächst gestoppt und dann wieder gestartet würde. Woher weiss init, welche Prozesse in einem Runlevel zu starten oder zu stoppen sind? Die inittab sagt ihm, dass er init.d/rc mit dem neuen Runlevel als Parameter ausführen muss, in diesem Fall also init.d/rc 5. Dieses Script wiederum vergleicht die Verzeichnisse init.d/rc3.d und init.d/rc5.d, denn in diesen befinden sich die zu startenden und zu stoppenden Scripte für Runlevel 3 bzw. 5.

Scripte, die hier mit einem S beginnen, werden gestartet, diejenigen, die mit K beginnen werden gestoppt. Um eine Reihenfolge festlegen zu können, haben diese Scripte auch nach dem ersten Buchstaben (K oder S) eine Zahl, die der Reihe nach abgearbeitet werden. Ein Script S10Etwas wird

demnach vor einem anderen Script S20EtwasAnderes gestartet. Ebenso ist es mit den Kill-Scripten, K10Etwas wird also vor K20EtwasAnderes gestoppt. Dabei gilt es allerdings zu beachten, dass die Reihenfolge der Kill-Scripte umgekehrt zu der der Start-Scripte sein sollte. So sollte beispielsweise ein Script, das eine Firewall aufbaut geladen werden, bevor das Netzwerk gestartet wird, etwa in der Reihenfolge S01Firewall S05network. Beim stoppen sollte nun aber zuerst das Netzwerk gestoppt werden, bevor die Firewall endet, also muss die Reihenfolge bei den Kill-Scripten genau umgekehrt zu den Start-Scripten sein, etwa K18network K22Firewall.

2.3.3 Init-Scripte

Die Initscripte, die sich alle unter init.d befinden, werden entweder direkt von init aufgerufen oder in einem der Runlevel. Da sich aber alle Scripte in init.d befinden, sind die Aufrufe in init.d/rcX.d nur Links auf diese Scripte. Dies verhindert, dass ein Script, das in mehreren Runlevels verwendet wird, versehentlich nur in einem geändert wird, weil ein Link lediglich ein Verweis auf eine Datei ist. Wenn also in den verschiedenen rcX.d-Verzeichnissen auf das gleiche Script unter init.d verwiesen wird, kann man einfach dieses ändern und alle Scripte unter rcX.d verweisen auf das geänderte Script. Da hier also nur Links verwendet werden, müssen die Scripte alle bestimmte Parameter verstehen. Wird ein Script mit Anfangsbuchstabe S aufgerufen, führt dies dazu, dass diesem der Parameter start übergeben wird, beginnt es mit K, wird ihm stop übergeben. Ein Eintrag S10Etwas, der auf das Script etwas linkt, würde also zum Aufruf "etwas start" führen. Ausserdem verstehen die Scripte die Parameter status für eine Statusmeldung, restart für das Stoppen und anschließende erneute Starten, reload für ein Neuladen der Konfiguration eines Scripts sowie force-reload für ein Neuladen der Konfiguration oder ein restart, falls das Neuladen nicht unterstützt wird.

Einige Scripte in init.d werden wie bereits erwähnt nicht Runlevel-spezifisch ausgeführt sondern direkt von init. Dies sind;

- ⑩ boot: Dieses Script wird nur genau einmal ausgeführt und startet den Kernel-Dämon zum automatischen Laden von Kernel-Modulen, es prüft die vorhandenen Dateisysteme, Netzwerkkonfigurationen werden vorgenommen sowie das Ansprechen von Plug-and-Play-Geräten vorbereitet. Ausserdem sind diesem Script alle Scripte unter init.d/boot.d zugeordnet. Als letztes wird init.d/boot.local aufgerufen.
- ⑩ boot.local: Hier werden weitere Programme angegeben, die gestartet werden sollen. Dies ist die Stelle, an der ein Benutzer von ihm erwünschte Programme eintragen sollte.
- ⑩ boot.setup: Beim Übergang vom Einzelbenutzer-Modus in irgendeinen anderen Modus wird dieses Script ausgeführt. In ihm werden zum Beispiel Tastaturbelegung und Konsolenkonfiguration festgelegt.

Damit ist der Bootvorgang beendet und ein oder mehrere Benutzer können sich an einem Terminal oder an einem graphischen Login anmelden, je nach dem in welches Runlevel gebootet wurden

3 Alternativen

3.1 Bootmanager

3.1.1 Chos und GRUB

Die beiden Bootmanager Chos und GRUB folgen demselben Prinzip wie der LinuxLoader, sie schreiben einen ersten Teil in den Master Boot Record des zu bootenden Datenträgers und ermöglichen dem User dann, das gewünschte Betriebssystem zu laden.

Der wohl größte Unterschied und auch größte Vorteil von GRUB gegenüber den anderen beiden ist, dass GRUB nicht auf die BIOS-Möglichkeiten zum Laden eines Kernels angewiesen ist. Stattdessen unterstützt er schon den Zugriff auf eine Reihe von verschiedenen Dateisystemen, so dass er nach einem bestimmten Dateinamen suchen kann. Wurde also ein Kernel neu kompiliert oder verschoben und damit seine Sektoren geändert, müssen LILO und Chos davon informiert werden. GRUB dagegen kann nach dem Dateinamen suchen. Das heißt, wenn ein Kernel neu kompiliert wurde und LILO oder Chos wurden nicht davon informiert, sind diese beiden Bootmanager nicht mehr in der Lage, korrekt zu starten. GRUB dagegen wird den Kernel finden, so lange er den gleichen Dateinamen hat wie zuvor.

Chos bietet als Zusatzfunktion standardmäßig ein farbiges und frei konfigurierbares Menu zum booten und kann darüber hinaus auch festlegen, welches Betriebssystem beim nächsten Start des Computers geladen werden soll. Man legt als im laufenden Zustand fest, welches Betriebssystem oder welcher Kernel beim nächsten Start erwünscht ist und dieser wird dann gebootet.

3.1.2 Loadlin

Loadlin verfolgt einen komplett anderen Ansatz als die anderen bisher erwähnten Bootmanager: loadlin startet Linux, nachdem ein DOS oder Windows gestartet wurde. Dabei wird der gewünschte Linux-Kernel auf die Windows-Partition oder -Festplatte kopiert und hier loadlin.exe mit dem Pfad des Kernels als Parameter ausgeführt. Wurde der Kernel als zum Beispiel nach c:\kernel.lin kopiert, wird loadlin c:\kernel.lin eingegeben. Daraufhin startet der Linux-Kernel wie bereits beschrieben. Ein Vorteil bei diesem Verfahren ist beispielsweise, dass auch auf alten BIOS-Versionen das Booten eines Kernels hinter dem 1024. Zylinder möglich, da Windows bereits den Zugriff auf die volle Festplatte ermöglicht. Wurde ein Linux also erst nach Windows installiert und befindet sich eventuell ganz ausserhalb der ersten 1024 Zylinder, kann man zunächst Windows booten und von dort aus dann Linux, wobei der Kernel selbst auch in der Windows-Partition hinter dem 1024. Zylinder liegen darf.

3.1.3 NT-Loader

Es ist auch möglich, den NT-Bootloader zu verwenden um Linux zu booten, wenn neben Linux ein WindowsNT auf dem Rechner installiert ist und es möglich sein soll, beide Betriebssysteme zu booten. Dabei ist es allerdings wohl nicht möglich, Linux ohne einen Linux-Bootloader zu starten, wie dies umgekehrt möglich wäre. Statt dessen wird der Linux-Bootloader der Wahl (Chos, LILO, GRUB) in den Bootsektor der Linux-Wurzelpartition installiert. Der Linux-Bootloader darf nicht in den Master Boot Record installiert werden, denn hier sitzt ja der NT-Loader! Nun wird der Bootsektor auf die WinNT-Festplatte oder -Partition kopiert, beispielsweise direkt nach c:\bootsec.lin. Nun muss noch dem NT-Loader mitgeteilt werden, dass er auch Linux booten kann, so dass Linux als Auswahl im Bootmenu erscheint. Das geschieht, indem die Datei boot.ini um den Eintrag
c:\bootsec.lin="Linux"
erweitert wird.

3.2 Booten über Netzwerk

3.2.1 Initialisieren

Ein Booten über Netzwerk ist in vielen verschiedenen Konfigurationen denkbar. An dieser Stelle sei die Vorgehensweise des LTSP – Linux Terminal Server Project - verwendet. Dazu ist ein DHCP-Server nötig. Auf der Workstation, also der Maschine, die über das Netzwerk gebootet werden soll, wird eine Netzwerkkarte mit Ethernet-BootROM verwendet, das heisst, dass die Netzwerkkarte einen ROM hat, in dem weiterer Bootcode enthalten ist. Diesen lädt das BIOS, wenn es in einem "normalen" System den Bootstrap-Loader starten würde. Dieser Ethernet-Bootcode sucht nach einer Netzwerkkarte, die er zum booten verwenden kann. Wenn diese gefunden wurde, erfolgt ihre Initialisierung. Ausserdem ist in der Workstation keine Festplatte notwendig, sie kann allerdings für verschiedene Zwecke wie Swapping eingesetzt werden.

3.2.2 DHCP

DHCP steht für Dynamic Host Configuration Protocoll und wird genutzt, um vom Server automatisch eine IP-Adresse, einen Rechnernamen und weitere Parameter zugewiesen zu bekommen. Dies funktioniert so, dass der Etherboot-Code eine Anfrage nach einer Adresse (einen IP-Request) in das lokale Netz schickt, bei der auch die MAC-Adresse der Netzwerkkarte mitgeschickt wird. MAC steht für Media Access Control und die MAC-Adresse ist eine weltweit eindeutige Kennzeichnung jeder Netzwerkkarte. Wenn der DHCP-Prozess auf dem Server die Anfrage von der Workstation erhält, sieht er in einer Liste mit zugelassenen MAC-Adressen nach und falls diese ihm bekannt ist, vergibt er eine IP-Adresse. Des weiteren schickt der DHCP auf dem Server die Netzmaske für das lokale Netz, den Namen des Kernels, der gebootet werden soll, den Pfadnamen des Wurzel-Dateisystems auf dem Server sowie Kommandozeilen-Parameter für den Kernel mit. Die Netzwerkkarte wird nun entsprechend dieser Daten konfiguriert und ist damit voll einsatzbereit.

3.2.3 Kernel

Nun verlangt der Etherboot-Code den Download des Kernels vom Server, der über das TFTP – Trivial File Transfer Protocol – funktioniert. Wenn der Kernel komplett geladen ist, wird dieser in den Systemspeicher verschoben, also das gleiche was ein Second-Stage-Bootlaoder in einem normalen System machen würde. Nun geht die Kontrolle an den Kernel über, der sich zunächst wie ein Kernel auf einem normalen System verhält: er initialisiert das System und alle gefundenen Hardware-Komponenten.

3.2.4 NFS oder Ramdisk

Falls eine Festplatte vorhanden wäre, würde der Kernel sich das Wurzeldateisystem suchen, um weiterarbeiten zu können. Doch es wird ein anderer Weg beschritten werden, der Umweg über eine Ramdisk. Eine Ramdisk ist ein Image eines Dateisystems, das in den Systemspeicher geschrieben wird und sich wie eine Festplatte verhält. Diese Ramdisk wird vorübergehend als Wurzeldateisystem gemountet, was dem Kernel über einen Kommandozeilenparameter der Form `root=/dev/ramdisk` mitgeteilt wird.

3.2.4.1 Init? (Ramdisk)

Sollte der Weg über eine Ramdisk gewählt worden sein passiert folgendes: Der Kernel sucht nun nach einem init, das er ausführen kann, aber über den Kommandozeilenparameter `init=/linuxrc` wird er angewiesen, statt dessen ein Script auf der Ramdisk zu verwenden. Dieses Script sucht zuerst nach (PCI-)Netzwerkkarten und vergleicht gefundene mit einer Liste von bekannten Karten. Trifft es hier eine Übereinstimmung, setzt es die geeigneten Parameter und das passende Modul wird in den Kernel geladen. Nun wird erneut eine Anfrage an den DHCP-Prozess auf dem Server gestellt,

um die Daten auch im User-Space verfügbar zu machen, da der Kernel das Ergebnis der ersten Anfrage "verschluckt". Diese zweite Anfrage stellt nun allerdings der Prozess dhclient, der auch nach einer Antwort vom Server über ein Script das Netzwerkinterface eth0 aktiviert.

3.2.4.2 Dateisystem (Ramdisk)

Bis jetzt war das Wurzel-Dateisystem in der Ramdisk, jetzt wird das eigentlich vorgesehene Dateisystem auf dem Server über das NFS gemountet. Das neue Dateisystem kann allerdings nicht direkt nach / gemountet werden, da hier ja noch die Ramdisk liegt. Deshalb wird das neue Dateisystem zunächst nach /root gemountet, die Ramdisk nach /oldroot verschoben und dann /root nach / verlegt. Nun ist das linuxrc-Script, das als Ersatz für init verwendet wurde, beendet und das eigentliche init kann jetzt gestartet werden.

3.2.5 Init!

An dieser Stelle treffen nun die Varianten Ramdisk und direktes NFS-Mounten wieder zusammen, wobei sich natürlich die Konfiguration der Init-Scripte unterscheiden muss: Init startet nun also ganz normal, liest die /etc/inittab und führt die entsprechenden Scripte aus. Dabei bootet es in Runlevel 2, also lokale Multiuser ohne Netzwerk, weil in der inittab dieses Runlevel als default angegeben wurde. Eines der ersten Scripte, die von init ausgeführt werden, ist das Skript rc.local. Dieses legt eine neue Ramdisk der Größe 1 Mbyte an, die alle Dateien enthalten wird, die geschrieben oder irgendwie verändert werden müssen. Die Ramdisk wird nach /tmp gemountet und alle Dateien, die Schreibrechte haben, sind dorthin zu legen. Weil das System sie allerdings eventuell an anderen Stellen sucht, werden von dort Links auf die entsprechenden Dateien unter /tmp angelegt. Nun wird /proc gemountet, dass für die Prozessverwaltung wichtig ist. Die Workstation kann sogar Swapping über NFS betrieben, wofür ein geeignetes Swap-File auf dem NFS-Server nach /tmp/swapfiles gemountet wird. Dabei wird ein neues Swap-File angelegt, falls noch keines für die Workstation existiert. Das Kommando swapon aktiviert das Swap-File. Dann wird /home vom Server auf /home der Workstation gemountet, so dass die Benutzer Zugriff auf ihren eigenen Datenbereich erhalten. Des weiteren werden einige Verzeichnisse im /tmp Verzeichnis angelegt die zur Laufzeit der Workstation notwendig sind. Mit Hilfe eines Scripts wird jetzt versucht, einen passenden X-Server für die vorhandene Grafikkarte zu finden und diesen zu starten.

3.2.6 Die Runlevel

Init führt jetzt das Script set_runlevel aus, das entscheidet in welches Runlevel letzten Endes gebootet wird: runlevel 3 startet lediglich eine bash (eine shell), was vor allem nützlich ist, wenn die Workstation nicht so arbeitet, wie erwartet. Runlevel 4, das ja für eigenen Definitionen offen ist, ist in der Lage eine oder mehrere telnet-Sitzungen zu starten und ermöglicht damit das Ersetzen einfacher serieller Terminals. Runlevel 5, für graphischen Login gedacht, startet schickt einen XDMCP-Request an den Server auf dem ein xdm, kdm, gdm oder ähnliches laufen muss. XDMCP steht für X Display Manager Control Protocol, xdm ist der X Display Manager, kdm der KDE Display Manager und gdm er Gnome Display Manager. Diese werden zur Authentifizierung des Benutzers verwendet.

Somit ist der Bootvorgang über das Netzwerk beendet und man kann sich anmelden.

4 Quellen

- ⑩ From PowerUp To Bash Prompt HowTo von Greg O'Keefe, v0.9, www.linuxdoc.org
- ⑩ Linux System Administrators Guide von Lars Wirzenius, Joanna Oja und Stephen Stafford, Version 0.7, www.linuxdoc.org
- ⑩ Linux Anwenderbuch und Leitfaden für die Systemverwaltung von Sebastian Hetze, Dirk Hohndel, Olaf Kirch und Martin Müller, siebte Auflage, www.lunetix.de
- ⑩ Linux Boot Prompt HowTo von Bart Gortmaker, v1.3, www.linuxdoc.org
- ⑩ SuSE Linux 7.2 System- und Referenzhandbuch von Leah Cunningham, Karl Eichwalder, Stefan Fent, Werner Fink, Peter Findeisen, Dennis Geider, Viviane Glanz, Carsten Groß, Roland Haidl, Björn Jacke, Richard Jelinek, Hubert Mantel, Johannes Meixner, Edith Parzefall, Peter Pöml, Peter Reinhart, Marc Rührschneck, Thomas Schraitle, Klaus G. Wagner und Christian Zoz, 1. Auflage 2001, www.suse.de
- ⑩ "Die Linuxfibel" – Bootmanager von Thomas Ermer und Michael Meyer, www.linuxfibel.de
- ⑩ LILO – Der Linux Loader von Mike Becher, www.linuxdoc.org
- ⑩ Network Boot and Exotic Root HowTo von Brieuc Jeunhomme, Revision 0.2.2, www.linuxdoc.org
- ⑩ Diskless Nodes HowTo von Robert Nemkin, Alavor Vasudevan, Markus Gutschke, Ken Yap und Gero Kuhlmann, v22.6, www.linuxdoc.org
- ⑩ Linux Terminal Server Project von James McQuillan, Martin Herweg und Wolfgang Schweer, v3.0, www.ltsp.org
- ⑩ Manpage für init
- ⑩ Manpage für inittab