Linux Tutorium
Systemkonfiguration

Der Bootvorgang

Beim starten des Linux-Systems muss vor allem eins getan werden: laden und starten des Kernels.

Der Linux-Kernel zum starten befindet sich normalerweise im Filesystem an diesem Ort:

`/boot/`**`vmlinuz`**

Bei manchen Konfigurationen ist das Verzeichnis `/boot` eine eigens dafür eingerichtete Partition auf der Festplatte.

Im Verzeichnis `/boot` befindet sich neben der komprimierten Kernel-Datei `vmlinuz` noch weitere Dateien, die für den Bootvorgang von Bedeutung sind. Dies ist insbesondere das für das starten des Systems verantwortliche Programm `grub` (**GR**and **U**nified **B**ootloader).

Der Bootvorgang - grub

`grub` besteht im wesentlichen aus folgenden Dateien:

`stage1`	für den Bootsektor
`stage2`	der eigentliche Programmcode des Bootloaders
`menu.lst`	Konfiguration des Bootmenüs

Hinzu kommen noch ein paar Filesystem Treiber.

Beim starten des Computers wird der erste Sektor der Festplatte geladen und gestartet (der sog. Bootsektor). Dieser Bootsektor hat eine Größe von nur 512Bytes.

Das File `stage1` enthält ein Mini-Programm welches nur in der Lage ist `stage2` zu laden und zu starten. Es muss sehr klein sein, damit es in den Bootsektor passt.

Der Bootvorgang - grub

Das Problem hierbei ist, das der Computer beim Starten nichts von der Struktur des Filesystems auf der Festplatte weiß.

Die Programme `stage1` und `stage2` sind daher sehr spezielle Programme.

Das Programm `stage2` kann lesend auf das Filesystem zugreifen, in dem sich der zu startende Linux-Kernel befindet. Es lädt die Datei `menu.lst` und zeigt das Bootmenü an.

Mit folgenden Befehl installiert man grub in den Bootsektor der Festplatte:

```
grub-install '(hd0)'
```

Die Ausführung dieses Befehls ist jedoch bei einem Korrekt eingerichteten System nicht notwendig und kann bei falscher Anwendung zu einem nicht mehr Bootfähigen System führen!

Der Bootvorgang - grub

In der Datei `menu.lst` wird festgelegt wie und mit welchen Parametern der Kernel gestartet werden soll.

Beispiel:

```
title    Debian GNU/Linux
root     (hd0,0)
kernel   /boot/vmlinux-2.6.18 root=/dev/hda1 ro
initrd   /boot/initrd.img-2.6.18
```

`title`	gibt einen Menütitel der Bootkonfiguration an
`root`	gibt die Root-Partition des Systems an
`kernel`	ist der zu startende Kernel mit Parametern
`initrd`	gibt eine Start RAM-Disk für den Kernel an (später dazu mehr)

Der Bootvorgang - initrd

Nachdem eine Konfiguration gestartete wurde lädt `grub` den Kernel in den Speicher und startet diesen.

Ab hier übernimmt zum ersten mal der Kernel das System.

Der Kernel hat nun jedoch ähnliche Probleme, wie zuvor der `grub` Bootloader: keine Kenntnisse über das Filesystem der System-Partition und keine Kenntnisse über die Hardware.

Aus diesem Grunde bekommt der Kernel vom Bootloader die Init-RAM-Disk (intird) übergeben. Diese ist ähnlich einer kleinen „Festplatte“ und enthält alle Treiber, die zum starten des Systems von der System-Partition notwendig sind.

Dies betrifft insbesondere Chipsatz-Treiber und Filesystem-Treiber.

Der Bootvorgang - initrd

Der genaue Aufbau der `initrd` ist von Distribution zu Distribution unterschiedlich. Grundsätzlich enthalten ist jedoch eine Shell (sash oder sh), ein paar wichtige Shell-Befehle, Kernel-Module (Treiber) und ein Shell-Skript dem Namen “`init`“.

Das Skript `init` lädt alle benötigten Kernel-Module (Treiber), initialisiert ein paar wichtige Dinge (`/proc`, `/sys` usw.) und mountet die System-Partition als root-Filesystem.

Wenn das Root-Filesystem gemountet ist, wird auf die System-Partition gewechselt, dort `init` (/sbin/init) gestartet und danach die `initrd` verworfen.

Der Bootvorgang - init

Beim Starten von `init` auf der Root-Partition (System) öffnet `init` die Datei: `/etc/inittab`.

Mit `/etc/inittab` wird gesteuert, wie das System gebootet werden soll, hierzu gibt es die sog. „Runlevels“:

0	System runterfahren
1	Sigle-User-Mode, nur ein Benutzer, meist root
2	Multiuser ohne Netzwerk
3	normaler Multiuser, ohne X
4	reserviert, falls nicht definiert wie 3
5	Multiuser mit X
6	Reboot, Neustart des Systems

Debian benutzt eine etwas andere Runlevel Aufteilung, hier ist nur Runlevel 0, 1, 2 und 6 definiert, wobei 2 den o.g. Runlevel 5 entspricht.

Der Bootvorgang - init

Mit dem Befehl `init` lässt sich ein Runlevel aktivieren. Beim Ausführen von:

```
init 6
```

wird z. B. das System neu gebootet.

Beim Starten des Systems selbst wird jedoch der default Runlevel gestartet, welcher in /etc/inittab angegeben ist:

```
# The default runlevel

id:2:initdefault:
```

Beim konfigurieren des Systems empfiehlt es sich, den default Runlevel auf einen anderen wert einzustellen, z.B. auf 3 (oder 1 bei Debian).

Der Grund hierfür ist, das bei der Änderung der Konfiguration, insbesondere X, das System in einen nicht benutzbaren zustand geraten kann.

Der Bootvorgang - init

In `/etc/inittab` wird weiterhin angegeben wie jeder einzelne Runlevel aktiviert wird:

```
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
   :
```

Hier wird bei jedem Runlevel die Datei `/etc/init.d/rc` gestartet und dabei die Nummer des Runlevels angegeben.

Die Datei `/etc/init.d/rc` ist ein Shell-Skript, welches dafür sorgt, das alle Dateien aus den Verzeichnissen `/etc/rc0.d/` bis `/etc/rc6.d/` je nach Runlevel gestartet werden.

Der Bootvorgang - init

Ein solches Runlevel Verzeichnis hat in etwa folgenden Aufbau:

```
K01gdm
K11anacron
   :
S40umountfs
S60umountroot
S90halt
```

Hierbei steht ein „S“ am Anfang für Start und ein „K“ für Kill (Stop), gibt also an, ob ein Skript beim ausführen des Runlevels gestartet oder gestoppt wertden soll.

Die Nummer gibt an, in welcher Reihenfolge die die Skripte ausgeführt werden sollen.

Der Bootvorgang - init

Die Runlevel-Verzeichnisse geben an, wie das System konfiguriert und welche Dienste gestartet werden sollen.

Die Dateien in den Runlevel-Verzeichnissen sind jedoch nur Symbolische Links auf die eigentlichen Start-Skripte, die sich in `/etc/init.d/` befinden.

Diese Skripte akzeptieren beim starten in der Regel die Parameter `start` und `stop` und oftmals noch `restart`.

Ist ein Dienst z.B. abgestürzt oder wurde neu Konfiguriert, so ist ein Neustart erforderlich. Wurde z.B. der Apache neu Konfiguriert, so kann man mit:

```
/etc/init.d/httpd restart
```

den Dienst neu starten, ein Neustart des Systems ist nicht nötig.

Beispielskript

```
#!/bin/bash
case "$1" in      # $1 ist erster parameter

   start)         # wenn paramtere start ist
      echo "MyService wird gestartet!"
      sleep 5     # 5 sec. pause
      ;;

   stop)          # wenn paramtere stop ist
      echo "MyService wird gestoppt!"
      sleep 5
      ;;

   *) # andere oder keine paramter
      echo "Usage: {start|stop}"
      exit 1
      ;;
esac
exit
```

Beispielskript

Diese Beispielskript kann man nun im Verzeichnis /etc/init.d z.B. mit dem Namen `myservice` erzeugen.

Damit das Skript ausgeführt werden kann muss noch das Execute-Bit gesetzt werden:

```
chmod +x myservice
```

Danach in den jeweiligen Runleveln verfügbar machen:

```
cd /etc/rc2.d
ln -s ../init.d/myservice S20mysevice
cd /etc/rc0.d
ln -s ../init.d/myservice K20mysevice
cd /etc/rc6.d
ln -s ../init.d/myservice K20mysevice
```

Kernel-Module

Beim Systemstart müssen auch bestimmte Kernel-Module (Treiber) gestartet werden.

Welche Treiber zur Boot-Zeit geladen werden sollen wird in der Datei `/etc/modules` bzw. `/etc/modules.preload` angegeben. Hier wird einfach pro Zeile der Name eines Kernelmoduls angegeben:

```
hw_random
intel-agp
```

Die Kernel-Module selbst befinden sich normalerweise in einem der Unterverzeichnisse unter `/lib/modules/kernelversion` und haben einen Namen wie z.B. “`nfs.ko`“ bzw. “`nfs.ko.gz`“.

Kernel-Module

Die derzeit im System geladenen Kernel-Module lassen sich mit dem Befehl `lsmod` anzeigen. Das sind normalerweise eine ganz Menge...

Zur Laufzeit des Systems können Module je nach Bedarf geladen (`insmod` oder `modprobe`) oder auch wieder entfernt werden (`rmmod`).

Laden eines Kernel-Moduls (Dateisystem-Treiber FAT):

```
modprobe fat
```

Entfernen eines Moduls (FAT-Treiber):

```
rmmod fat
```

Der Unterschied zwischen `insmod` und `modprobe` besteht darin, das bestimmte Kernel-Module zum funktionieren weitere Kernel-Module benötigen und diese mit `modprobe` automatisch nachgeladen werden.

Kernel-Module

Die Abhängigkeiten der Kernel-Module untereinander werden in der Datei `/lib/modules/kernelversion/modules.dep` verwaltet, welche grob folgenden Aufbau hat:

```
modulname.ko: abhängigkeit1 abhängigkeit2 ...
```

Alle Module, auch wenn sie keine weitere Abhängigkeit besitzen sind hier aufgelistet.

Ohne diese Datei kann `modprobe` die benötigten Module nicht laden. Die `modules.dep` Datei muss immer aktuell sein und muss bei einer Erweiterung des Systems, z.B. Grafikkarten-Treiber, aktualisiert werden. Dies geschieht normalerweise automatisch.

Sollte dennoch das aktualisieren dieser Datei erforderlich sein, so kann man mit dem Befehl: `depmod -a` die Aktualisierung durchführen.

Kernel Kompilieren

Manchmal ist es Notwendig einen neuen Linux Kernel im System zu installieren.

Die Gründe dafür können unterschiedlich sein:

- Systemstabilität
- Sicherheitsprobleme
- Optimierung des Kernels
- neue Treiber oder Funktionalitäten

Um einen Kernel selbst zu Kompilieren benötigt man erstmal den Quellcode. Diesen besorgt man sich entweder von der Distribution die man benutzt (empfohlen) oder einen sog. Vanilla-Kernel von www.kernel.org.

Kernel Kompilieren

Der Kernel-Quellcode befindet sich dann normalerweise in dem Verzeichnis `/usr/src/linux/` bzw. `/usr/src/linux-version/`.

Zuvor muss der Kernel jedoch konfiguriert werden, dazu muss zunächst in das Verzeichnis mit dem Kernel-Quellcode gewechselt werden. Hier gibt es nun mehrere Möglichkeiten, die alle über das `Makefile` gestartet werden, welches auch dafür benutzt wird den Kernel zu kompilieren.

Mit dem Aufruf von: `make menuconfig`

bekommt eine recht komfortable Möglichkeit den Kernel zu konfigurieren. Am Ende der Konfiguration entsteht eine Datei mit dem Namen `.config`.

Hat man bereits eine solche Konfigurationsdatei (z.B. befindet sich normalerweise eine in `/boot/config-version`), so kann man diese mit dem Namen `.config` in den Quellcode Kopieren und mit `make oldconfig` den Kernel damit konfigurieren.

Kernel Kompilieren

Nachdem man den Kernel Konfiguriert hat, muss er „nur“ noch kompiliert werden:

```
make bzImage
```

dies nimmt je nach Rechnerleistung einige Zeit in Anspruch (ca. 1h).

Der so Kompilierte Kernel muss dann noch mit:

```
cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz
```

in das `boot` Verzeichnis kopiert werden.

Danach muss noch:

```
make modules_install
```

aufgerufen werden, um die Kernel-Module an den richtigen Platz ins System zu kopieren (`/lib/modules/kernelversion`).

Boot-Konfiguration erstellen

Hat man den Kernel erstellt, muss noch eine entsprechende Boot-Konfiguration für den `grub` Bootloader erstellt werden:

```
title    Mein selbstkompilierter Kernel
root     (hd0.0)
kernel   /boot/vmlinuz root=/dev/hda1 ro
initrd   /boot/initrd.img
```

Ältere Kernel-Versionen, mit denen das System ja bereits läuft, sollte man zunächst in der Boot Konfiguration lassen um bei einem nicht funktionierenden Kernel darauf zurück greifen zu können.

Dank grub ist es ohne weiteres möglich mehrere Kernelversion gleichzeitig zu installieren und nach Bedarf zu booten

Boot-Konfiguration erstellen

Das Erstellen eines neuen Boot-Eintrags ist recht einfach.

Das größere Problem hierbei ist eine passende `initrd` für den neuen Kernel zu erstellen. Leider gibt es hier keinen eindeutigen Weg. Debian basierte Systeme bieten hierfür ein Skript mit dem Namen `mkinitramfs` bzw. `update-initramfs`, RedHat basierte Systeme bieten hierfür `mkinitrd`.

Folgende Befehle erzeugen eine `initrd` mit dem Namen `/boot/initrd.img-2.6.18-6-686` mit der optionalen Kernelversion `2.6.18-6-686`.

Debian basiert:

```
update-initramfs -k 2.6.18-6-686 -c
```

RedHat basiert:

```
mkinitrd /boot/initrd.img-2.6.18-6-686 2.6.18-6-686
```

Boot-Konfiguration erstellen

Eine andere möglichkeit ist eine bereits vorhandene `initrd` zu benutzen um daraus eine für den Kernel passende `initrd` zu erstellen.

Die `initrd` ist komprimiert und muss deshalb erstmal entpackt werden:

```
mv initrd.img initrd.img.gz
gzip -d initrd.img.gz
```

Nun muss der Inhalt der RAM-Disk extrahiert werden:

```
cpio -iv <initrd.img
```

So erhält man im aktuellen Verzeichnis den Inhalt der RAM-Disk, allerdings sollte das RAM-Disk Image selbst noch gelöscht werden:

```
rm initrd.img
```

Die oben genannten Schritte lassen sich vereinfachen durch:

```
zcat /boot/initrd.img | cpio -iv
```

Boot-Konfiguration erstellen

In dieses Verzeichnis kopiert bzw. überschreibt man nun die passenden Kernel-Module.

Hat man die `initrd` vollständig angepasst, so muss wieder ein gültiges, komprimiertes Image erzeugt werden:

```
find . | cpio -H newc -o >/boot/initrd.img
cd /boot
gzip initrd.img
mv initrd.img.gz initrd.img
```

Nun sollte alles soweit eingerichtet sein und mit ein bisschen Glück startet das System mit dem neuen Kernel...