LPI 102: Kernel und Bootvorgang

Johannes Kloos

Di, 25.4.2004

Kernelversion

- Frage: Welcher Kernel läuft auf meinem System?
- uname -a gibt alle Informationen aus

```
Linux sushi 2.4.25 #1 SMP Wed Feb 18
  18:43:27 CET 2004 i686 GNU/Linux
```

- Interessant: Kernel-Version, Hardware-Plattform

  Befehle: `uname -r`, `uname -m`

Kernel und Module

- Kernel = zentrales Steuerprogramm des Rechners, enthält Speicherverwaltung, Prozessverwaltung, Dateisystem, Netzwerkschicht, Gerätetreiber, ...
- Nicht immer werden alle Komponenten benötigt
- Deswegen: Nachladbare Module mit Gerätetreibern, Dateisystemen, Netzwerkprotokollen...

Welche Module sind geladen?

- Normale Kernel sind modular, also werden auf jedem System Module nachgeladen
- Liste der geladenen Module anzeigen: lsmod
- Alternativ: Inhalt von /proc/modules betrachten
- Modulabhängigkeiten:

```
ext3                    58532   6  (autoclean)
jbd                     38632   6  (autoclean) [ext3]
```

Module laden

- Wie lädt man Module?

  möglicher Befehl: `insmod modulname` bzw. `insmod pfad`

- Wie entfernt man Module?

  möglicher Befehl: `rmmod modulname`

- Modulabhängigkeiten beachten!

Module laden ohne Abhängigkeitsprobleme

- Befehl: `modprobe modulname` bzw. `modprobe dateiname`
- Lädt das Modul und ggf. vorher alle Module, von denen es abhängig ist.
- `modprobe -r modulname` entfernt Module.
- Wie werden Abhängigkeiten ermittelt?

  Datei: `/lib/modules/<kernel-version>/modules.dep`
- Wie erzeugt man diese Datei?

  Befehl: `depmod`

- Wird normalerweise beim Systemstart oder bei der Installation des Kernels gemacht.

Moduloptionen

- Viele Module haben Optionen, z.B. zu verwendende I/O-Ports oder IRQs.
- Übergabe der Parameter beim Laden: `insmod 3c509 irq=5` bzw. `modprobe 3c509 irq=5`
- Wie kann man ermitteln, welche Parameter ein Modul hat?

  Befehl: `modinfo modulname`

modules.conf

- Man will beim Laden eines Modules nicht immer alle Parameter mitangeben müssen
- Es wäre schön, einfach `modprobe eth0` sagen zu können
- Datei: `/etc/modules.conf`. Enthält u.a. Zeilen folgender Art:
- Aliase: `alias eth0 3c509`
- Optionen: `options apm realmode_power_off=1`
- Kann auch noch wesentlich tiefgreifendere Dinge

Automatisches Laden von Modulen

- Problem:

```
# mount -t isofs /dev/cdrom /mnt
Error: fs type isofs not supported by kernel
# modprobe isofs
# mount -t isofs /dev/cdrom /mnt
```

- Der Kernel wird normalerweise so konfiguriert, dass Module automatisch geladen werden.

- Hierzu wird `/sbin/modprobe` (oder ein anderes Programm, was mit Hilfe von sysctl angegeben wurde) aufgerufen und versucht, das fehlende Modul zu laden.

Kernel bauen

- Distributionskernel sind nicht immer für die Anwendung des Systems geeignet
- Es gibt distributionsspezifische Hilfsmittel zum Kernel-Compilieren (z.B. `make-kpkg` bei Debian)
- Kernel-Sourcecode entweder aus Quellpaket des Distributors oder von `http://www.kernel.org` herunterladen
- Vielfältige Kernel-Patches verfügbar

Kernel bauen (2)

- Zum Bauen werden diverse Pakete benötigt, u.a. make, gcc, binutils
- Ausgepackte Kernel-Sources normalerweise unter `/usr/src/linux`
- Interessantes für Anwender: `Documentation` (insbes. `Documentation/Changes`) und `README`
- Vorgehen: Kernel konfigurieren, `make`, `make modules`, installieren

Kernel konfigurieren

- Kernel-Konfiguration steht in `.config`
- Erzeugen mittels `make config`, `make menuconfig`, `make xconfig`
- `make oldconfig`: übernimmt eine alte `.config`-Datei und fragt bei unbeantworteten Fragen noch

Kernel kompilieren

- `make` baut das Kernel-Image (das später vom Bootloader geladen wird)
- `make modules` baut die Module

Installation

- Kernel-Images stehen in `/boot`
- Aktuelles Image üblicherweise `/boot/vmlinuz`, vorheriges Image unter `/boot/vmlinuz.old`
- `make install` benennt `/boot/vmlinuz` in `/boot/vmlinuz.old` und installiert neues Image als `/boot/vmlinuz`
- `make modules_install` installiert Module in `/lib/modules/version` und führt `depmod` aus

Systemstart

- Ablauf: BIOS oder andere Firmware -> Bootloader -> Kernel -> `init`
- Gängige Bootloader auf Intel-x86-Plattformen: LILO und GRUB (es gibt noch mehr, z.B. syslinux). LILO und GRUB können noch andere Systeme ausser Linux booten.
- Bootloader können dem Kernel Boot-Optionen mitgeben
- `init` startet die diversen Prozesse für System und Benutzer

LILO

- Der ältere der beiden angesprochenen Bootloader
- Kleiner und kompakter, aber mit weniger Funktionalität
- Speichert die Plattenposition des Kernelimages, muss also nach Installation eines neuen Kernels die Positionen neu ermitteln.
- Konfiguration in `/etc/lilo.conf`
- Installation eines neuen Kernels: ggf. `lilo.conf` anpassen, dann `lilo` aufrufen

LILO (2)

- Befehlszeile: image optionen, wobei image der Name eines Kernel-Images ist.

  Beispiel: `linux root=/dev/hda3`

- `lilo.conf` - ein Beispiel:

```
boot=/dev/hda
root=/dev/hda2
lba32
image=/vmlinuz
    label=Linux
    read-only
```

- Optionen an Kernel: In image-Block `"append=optionen"`

GRUB

- Der neuere Bootloader. Viel Funktionalität - kann z.B. auch über Netz booten.
- GRUB kann Dateisysteme lesen, muss also nicht updatet werden, wenn ein neuer Kernel installiert wird.
- Befehlszeile:

```
kernel (gerät)/pfad optionen
boot
```

Beispiel:

```
kernel (hd0,0)/boot/vmlinuz
boot
```

GRUB (2)

- Konfigurationsdatei: `/etc/grub.conf`
- Ein Beispiel:

```
default=0
timeout=10
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
title Red Hat Linux (2.4.20-8)
        root (hd0,0)
        kernel /boot/vmlinuz-2.4.20-8 ro root=LABEL=/
        initrd /boot/initrd-2.4.20-8.img
```

Systemstart

- Bootloader lädt kernel
- Kernel initialisiert Hardware
- Kernel startet `init`
- was dann?

init

- Konfigurationsdatei von init: `/etc/inittab`

- Optionen auf Kernel-Befehlszeile, die der Kernel nicht versteht, werden an init weitergereicht

- wichtige Optionen: single (starte das System im Single-User-Modus), emergency (starte sofort eine Shell), 1 ... 5 (starte das System in Runlevel 1 bis 5)

- init startet zunächst alles, was als `"sysinit"` in `inittab` markiert ist

- danach geht init in ein bestimmtes runlevel (entweder das auf der Befehlszeile angegebene oder das, was per initdefault in `inittab` angegeben ist).

Runlevel

- init kann für bis zu neun verschiedene runlevel konfiguriert werden. Ein runlevel ist als Menge von laufenden Prozessen zu verstehen.

- Normale Aufteilung:

0	Systemhalt
1	Single user
2,3,4	Multi user ohne X
5	Multi user mit X
6	reboot

- Runlevel im Betrieb wechseln: `telinit runlevel` oder `init runlevel`

inittab

- Beispiel für einen `inittab`-Eintrag:

  `3:23:respawn:/sbin/getty 38400 tty3`

- Erstes Feld: Name für diesen Eintrag (ein oder zwei Zeichen)
- Zweites Feld: In welchen Runlevels starten?
- Drittes Feld: Wann und wie starten?
- Viertes Feld: Befehlszeile

System herunterfahren und rebooten

- System in Single User-Modus bringen: `telinit 1` oder `shutdown now Nachricht`
- System in fünf Minuten anhalten: `shutdown -h +5 Nachricht`
- System um Mitternacht rebooten: `shutdown -r 24:00 Nachricht`
- Einfach nur Nachricht an alle Benutzer: `wall Nachricht`

Bootmeldungen

- Nach dem Start von syslog: Meldungen im Systemlog, also normalerweise in `/var/log/messages`
- Vorher?
- Meldungen des Kernels: `dmesg`
- Andere Meldungen kann mit mit `bootlogd` aufzeichnen

Welche Prozesse werden von init gestartet?

- Start-/Stop-Skripte in `/etc/init.d` – (fast) jeder Daemon sollte so eins haben
- Symlinks in `/etc/rc0.d` bis `/etc/rc6.d` geben an, was in welchem Runlevel wann gestartet oder beendet werden soll
- Alternativ: `/etc/runlevel.conf` bei Debian mit `file-rc`