Kompilieren und Installieren eines neuen Kernels

- Kurzakte –

Installation

als tar.gz-Archiv: Download: www.kernel.org

- das entpackte Verzeichnis gehört nach /usr/src/*Kernelversion*
- Erstellen eines Symlinks auf /usr/src/linux: ln –s /usr/src/*Kernelversion* /usr/src/linux

aus den Distributions-CD´s:

- Paket „kernelsources“ installieren (SUSE)

Konfiguration des Kernels

cd /usr/src/linux	• in das neue Verzeichnis wechseln (Symlink auf /usr/src/Kernelversion)
Makefile	• der Linux-Quellbaum enthält in jedem Verzeichnis ein Makefile • Anpassen der Zeile EXTRAVERSION im Toplevel-Makefile bei Kerneln der gleichen Version, aber verschiedenen Konfigurationen
make config **make menuconfig** make xconfig make gconfig	• ein sehr einfaches textorientiertes Programm zur Kernelkonfiguration (umständlich, da immer ***alle*** Fragen beantwortet werden müssen) • menügeführte Konfiguration, erlaubt Einzeländerungen **(Nutzung empfohlen!)** benötigt Paket ncurses-devel, um auf der Konsole Fenster, Menüs darzustellen • grafisch unterstützte Konfiguration des Kernels, nutzt Tcl/Tk (QT bei K. 2.6) • seit Kernel 2.6 zusätzliche grafische Konfiguration mit Zugriff auf Gtk+ statt Qt
/usr/src/linux/.config	• speichert oben erstellte Konfigurationsanweisungen • Speichern verschiedener Kernelkonfigurationen durch einfaches Umbenennen
make oldconfig	• beim Wechsel auf eine neue Kernelversion werden nur Fragen zu neu hinzugekommenen Features gestellt • dazu muss die alte .config-Datei in das neue Kernelverzeichnis kopiert werden
/proc/config.gz	• hier stehen alle gesetzten Optionen des aktuellen Kernels (lesen mit zcat)

Übersetzung des Kernels und der Module

make dep nicht bei Kernel 2.6!	• Erstellen von Abhänigkeitsinfos zwischen Quell- und Include-Dateien • nur bei erster Kompilierung nötig (erzeugt .depend in den Verzeichnissen)
make clean	• Löschen der alten Objekt-Dateien (*.o), damit diese neu übersetzt werden • sonst werden nur geänderte Quellcode-Dateien neu kompiliert
make mrproper	• wie make clean und löscht zusätzlich die .config (neue Konfiguration nötig)
make distclean	• entfernt Editor-Backup-Dateien und Patchrückstände
make bzImage	• Kompilieren, Komprimieren und Kopieren des neuen Kernel nach /usr/src/linux/arch/i386/boot/bzImage bei Intel-Systemen • kompiliert alle ausgewählten .c-Dateien in Objektdateien (.o), die dann später zum Kernel zusammengefasst (linked) werden
make bzdisk	• wie bzImage, wobei die Dateien auf eine eingelegte Diskette kopiert werden (dieser Schritt kann übergangen werden)
make modules	• Module übersetzen (nicht fest in den Kernel eingebunden) • die übersetzten Module (*.o) bleiben in den Quellcode-Verzeichnissen

Installation des neuen Kernels

make modules_install	• Kopieren der Module nach /lib/modules/*Kernelversion* • automatischer Start von "depmod" zur Analyse der Modulabhängigkeiten • Ablage der Informationen in /lib/modules/Kernelversion/modules.dep

cp /usr/src/linux/arch/i386/boot/bzImage /boot/*neuerKernelName*

- Kopieren der neu erstellten Kerneldatei in das Zielverzeichnis (alten Kernel nicht überschreiben!)

cp /usr/src/linux/System.map /boot/System.map.*Kernelversion*

- Kopieren der neuen System-Map
- enthält die Kernelsymbole, damit die Kernelmodule die Kernelfunktionen korrekt aufrufen können

Anpassen des Bootmanagers

GRUB	• in /etc/menue.lst: Kopieren der Linuxsektion • dort Vergabe eines neuen Labels und Anpassen des Kernelpfads/-namens
LILO	• in /etc/lilo.conf: Kopieren der Linuxsektion • dort Vergabe eines neuen Labels und Anpassen des Kernelnamens/-namens • Neustart von LILO mit /sbin/lilo, um Änderungen in MBR zu schreiben

Anpassen der Modulkonfiguration mit eigener modules.conf

/etc/modules.conf	• je nach Kernelversion wird die zugehörige modules.conf geladen if -f /etc/modules.conf-`uname -r`; include /etc/modules.conf-`uname -r` else; include /etc/modules.conf.alt endif

Dokumentation

Dateien unter /usr/src/linux/Documentation bei installierten Kernelquellen
Kernel-HOWTO des Linux Dokumentation Projects (www.tldp.org, www.linuxhaven.de)

Erstellt von Dr. Jörg Meese www.doc-systems.de

1. Kernel-Update als rpm-Version

Check der Kernelversion:
```
cat /proc/kernelversion
```

Check des rpm-Pakets des Kernels:
```
rpm -qf /boot/vmlinuz
```

Kernel und dazugehörige initrd sichern:
```
cp /boot/vmlinuz-$(uname -r) /boot/vmlinuz.old
cp /boot/initrd-$(uname -r) /boot/initrd.old
```

Update des Kernels aus einem rpm heraus:
rpm `-Uvh [--force] <Paketname>` `# bei älterer Version erzwingen`

Schreiben einer neuen RAMDisk (erfolgt eigentlich automatisch):
mk_initrd

2. Kernel-Neubau Version 2.6

Voraussetzungen:
Pakete der Kernelquellen (`kernel-source`) des C-Compilers (`gcc`), der GNU Binutils (`binutils`) und der Include-Dateien für den C-Compiler (`glibc-devel`) müssen installiert sein.
Achtung: möchte man verschiedene Kernelversionen vorhalten, sollte man die Quellen mehrmals in verschiedene Verzeichnisse entpacken und die jeweils aktuellen Quellen mit Symlinks auf `/usr/src/linux` ansprechen.

aktuelle Konfiguration (steht in Datei /proc/config.gz) verändern:
```
zcat /proc/config.gz > .config
make oldconfig
```

Übersetzen des neuen Kernels
```
make clean
make bzImage           oder zusammangefasst: make clean bzImage
make modules
```

Kernel installieren
```
INSTALL_PATH=/boot make install       oder:
       cp  /usr/src/linux/arch/i386/boot/bzImage  /boot/neuerKernelName
make modules_install
```
Achung: Module, deren Funktionalität man jetzt direkt in den Kernel einkompiliert hat, müssen unter /lib/modules/<Version> entfernt werden. Sonst kann es zu unvorhersehbaren Effekten kommen.

Festplatte aufräumen
```
cd /usr/src/linux
make clean
```

Bootmanager anpassen
Damit der alte Kernel (jetzt `/boot/vmlinuz.old`) von GRUB gebootet werden kann, wird in der Datei `/boot/grub/menu.lst` zusätzlich ein Label Linux.old als Boot-Image eingetragen.

Modulmanagement:

`insmod`	lädt Einzelmodule aus /lib/modules/<Version>
`rmmod`	entlädt Einzelmodule, die vom Kernel nicht mehr benötigt werden
`depmod`	erzeugt Datei modules.dep mit den Modulabhängigkeiten
`modprobe`	(ent-)lädt Module unter Berücksichtigung der Abhängigkeiten laut /etc/modprobe.conf, /etc/modprobe.conf.local und Verzeichnis /etc/modprobe.d/ (ehemals modules.conf)
`lsmod`	listet die geladenen Module und von welchen anderen sie verwendet werden (Abhängigkeiten)
`modinfo`	zeigt die einkompilierten Modulinformationen an