

# Kernel selbst kompilieren

---

Tobias Groschup  
Proseminar Linux für Umsteiger  
Betreuer: Julian M. Kunkel, Olga Mordvinova  
Heidelberg, den 13.5.2008

# Inhalt

---

- Vorbereitung
- Konfiguration
- Kompilervorgang und Installation
- Anmerkungen
- Quellen

Vorbereitung

# Vorbemerkungen

---

- Der Kernel IST Linux
- inzwischen kaum noch erforderlich
- Anwendungsfälle:
  - ➔ Exotische Hardware
  - ➔ Spezielle Software, zB Lustre
  - ➔ „mehr“ Performance erwünscht
  - ➔ Neugier

Man kann nicht oft genug erwähnen, dass Linux NUR den Kernel bezeichnet, und nicht alles andere außen rum (die Typischen GNU tools mit Linuxkern heißt GNU/Linux)

Heutzutage liefern die Distributoren für ihre Distribution und für praktisch alle Anwendungsfälle passende Kernel mit. Genauso werden Updates meistens schnell weitergegeben.

Früher war dies seltener der Fall, bzw in die Kernel waren wesentlich weniger Treiber integriert. Daher waren Anweder zum selbstkompilieren gezwungen, sobald sie an die Grenzen ihres vorkompilierten Kernels gestoßen sind.

Mehr Performance wird bei den üblichen (im Vergleich zu früher: Un-)Mengen an Speicher und Rechenleistung nicht spürbar erzeugt. Daher ist dass keine keine wichtige Optimierung.

# Woher?

---

- zum Selbstopilieren benötigt man den Quellcode
- gute Quellen:
  - ➔ <http://www.kernel.org>
  - ➔ Paketsystem der Distribution
    - zB Paketname unter Ubuntu: linux-source (Metapaket mit aktuellem Quellcode)

Kernel.org ist die Hauptquelle für den Kernel Sourcecode. Hier beziehen die Distributoren ihre Quellen her. Hier stellt auch Linus Torvald die „offizielle“ Version online.

Ubuntu stellt auch alte Versionen unter linux-<version> bereit

# Kernel.org: Seite & Ordnerstruktur

## Index of /pub/linux/kernel/v2.6

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>
<a href="#">Parent Directory</a>		-
<a href="#">incr/</a>	07-May-2008 01:46	-
<a href="#">pre-releases/</a>	18-Dec-2003 15:50	-
<a href="#">snapshots/</a>	07-May-2008 07:05	-
<a href="#">stable-review/</a>	29-Apr-2008 19:01	-
<a href="#">testing/</a>	03-May-2008 19:42	-
<a href="#">ChangeLog-2.6.0</a>	18-Dec-2003 03:04	12K
<a href="#">ChangeLog-2.6.1</a>	09-Jan-2004 07:08	189K
<a href="#">ChangeLog-2.6.2</a>	04-Feb-2004 04:06	286K
<a href="#">ChangeLog-2.6.3</a>	18-Feb-2004 04:11	300K
<a href="#">ChangeLog-2.6.4</a>	11-Mar-2004 03:17	321K
<a href="#">ChangeLog-2.6.5</a>	04-Apr-2004 03:52	358K
<a href="#">ChangeLog-2.6.6</a>	10-May-2004 02:52	487K
<a href="#">ChangeLog-2.6.7</a>	16-Jun-2004 05:50	761K
<a href="#">ChangeLog-2.6.8</a>	14-Aug-2004 06:02	883K
<a href="#">ChangeLog-2.6.8.1</a>	14-Aug-2004 11:12	263
<a href="#">ChangeLog-2.6.9</a>	19-Oct-2004 17:44	1.2M
<a href="#">ChangeLog-2.6.10</a>	24-Dec-2004 22:30	1.5M
<a href="#">ChangeLog-2.6.11</a>	02-Mar-2005 15:36	1.4M
<a href="#">ChangeLog-2.6.11.1</a>	09-Mar-2005 00:41	1.2K
<a href="#">ChangeLog-2.6.11.2</a>	09-Mar-2005 08:17	235
<a href="#">ChangeLog-2.6.11.3</a>	13-Mar-2005 06:55	8.0K
<a href="#">ChangeLog-2.6.11.4</a>	16-Mar-2005 00:11	1.0K
<a href="#">ChangeLog-2.6.11.5</a>	19-Mar-2005 06:40	5.0K
<a href="#">ChangeLog-2.6.11.6</a>	26-Mar-2005 03:31	2.8K
<a href="#">ChangeLog-2.6.11.7</a>	07-Apr-2005 19:01	5.6K
<a href="#">ChangeLog-2.6.11.8</a>	30-Apr-2005 01:43	5.9K
<a href="#">ChangeLog-2.6.11.9</a>	11-May-2005 22:47	2.1K
<a href="#">ChangeLog-2.6.11.10</a>	16-May-2005 18:17	3.0K
<a href="#">ChangeLog-2.6.11.11</a>	27-May-2005 05:57	14K
<a href="#">ChangeLog-2.6.11.12</a>	12-Jun-2005 02:58	563

- COPYING
- CREDITS
- Documentation
- Kbuild
- MAINTAINERS
- Makefile
- README
- REPORTING-BUGS
- arch
- block
- crypto
- drivers
- fs
- include
- init
- ipc
- kernel
- lib
- mm
- net
- samples
- scripts
- security
- sound
- usr
- virt

6

Hier sind alle Versionen des aktuellen Major-Releases (2.x) hinterlegt. Da sehr unübersichtlich, ist im Verzeichnisbaum von Kernel.org hinterlegt, was die aktuelle Version ist: LATEST-IS-<version>

Changelog beinhaltet die Veränderungen zum letzten Kernel

Weiter unten stehen die Patch-Dateien, deren Funktion weiter hinten erklärt wird.

Rechts steht der Verzeichnisbaum, der in einem Archiv mit den einer aktuellen Kernelquelle verpackt ist. Dateien fangen mit einem Großbuchstaben an, Ordner mit einem kleinen (ausser Documentation)

Unter arch stehen zB Quellcodedateien für die unterstützten Architekturen (wie x86 oder PowerPC).

Unter drivers die Treiber, unter fs die Dateisysteme. In Documentation stehen Hilfedateien, die jedoch nur interessant sind, wenn man selbst am Quellcode mitschreiben möchte.

# Voraussetzungen

---

- root-Zugang
- aktuelle Source (haben wir schon)
- Kompiler, zB gcc
- Buildsystem, hier make
  - ➔ Ubuntu: build-essential
- eventuell Möglichkeit zur graphischen Konfiguration:
  - ➔ ncurses, gtk, qt

Den root-Zugang benötigen wir, da wir in Systemverzeichnisse schreiben müssen, in die normale Benutzer keinen Zugriff haben.

make nimmt uns einen Großteil der Arbeit ab - es weiß wie vorzugehen ist, und fragt uns, an den Stellen an denen wir Entscheidungen treffen sollen (zB die Konfiguration).

Graphische Konfiguration entweder per Xserver (wie im Vortrag, xconfig) oder ncurse auf der Kommandozeile. Geht auch menügeführt. Bei der aktuellen Technik ist die graphische Konfiguration die Methode der Wahl, da sie sehr übersichtlich ist. Andere Arten sind in speziellen Anwendungsfällen schneller. Alle Konfigurationstools erlauben die selben Einstellungen, nur die Darstellung unterscheidet sich.

Rest der benötigten Ubuntu-Pakete: <http://wiki.ubuntuusers.de/Kernel?highlight=%28kernel%29>

# Konfiguration

# Was konfiguriere ich?

---

- fast alles im Kernel lässt sich einstellen, zB
  - ➔ Unterstütze Hardware
  - ➔ Unterstütze Dateisysteme
  - ➔ Methode des Bootens
  - ➔ Benutzung von Modulen

Hardwareunterstützung per Treiber, direkt im Kern oder als Modul ausgelagert.

Dateisysteme genauso

Bootmethoden: mit/ohne initrd (oder initramfs)

Modulbenutzung: sinnvoll, hält Kernel klein, kleiner Kernel ist schnell geladen  
(lässt sich in manchen Fällen sogar im BIOS-Rom speichern: <http://www.coreboot.org>)

# Wie konfiguriere ich? #1

---

- Zentrales Werkzeug: das `make` File
- Verschiedene Arten:
  - ➔ `make config`
  - ➔ `make menuconfig`
  - ➔ `make xconfig` bzw `make gconfig`

`make config`: Urahn der Konfigurationswerkzeuge, Terminal-basiert, Frage/  
Antwort

`menuconfig`: n-curse Oberfläche, Zeichenbasiert, „halb“graphisch

`xconfig`: ganz graphisch (benötigt qt), genauso `gconfig` (mit gtk Frontend)

qt ist das Framework, auf dem die KDE Desktopumgebung basiert. gtk ist die Grundlage für Gnome. Die Wahl zwischen den beiden ist dem Benutzer überlassen, wirklich groß sind die Unterschiede nicht.

## Wie konfiguriere ich? #2

---

- `make defconfig`
  - ➔ ergibt eine (hoffentlich) lauffähige Minimalkonfiguration
- `make oldconfig`
  - ➔ benutzt vorhandene `.config`
  - ➔ erhält man aus `/proc/config.gz` (oder `/boot/config-<aktuelle Version>` unter Ubuntu)
  - ➔ Achtung! gibt meistens Probleme
- `make randomconfig`

`.config` ist die Datei, in der die Konfiguration gespeichert wird, und aus dem später beim Kompilieren die nötigen Einstellungen gelesen wird. Man kann sich so mit verschiedenen `.config` Dateien verschiedene Konfigurationen des Kernels speichern.

`/proc/config.gz` ist manchmal nicht aktiviert (zB in Ubuntu standardmäßig ausgeschaltet)

`oldconfig` fragt nach neuen Konfigurationen im neuen Kernel.

Hat jedoch Probleme beim einem Versionswechsel, zB von 2.4 auf 2.6, und ergibt nicht immer einen lauffähigen Kernel.

`Make randomconfig` erstellt eine zufällige Konfiguration für den Kernel, das muss nicht immer laufen. Es gibt noch weitere Optionen wie `make noconfig`, die einfach nichts auswählt, und `make allconfig`, die alles auswählt. Diese sind aber alle nur für Testzwecke interessant.

# Konfiguration

---

- empfehlenswert: Prozessortyp einstellen
- Module/Initrd nutzen
- benutze Dateisysteme wenigstens als Module einbinden
- Sicher nicht benötigte (zB ISA-Netzwerkkarten) Treiber entfernen
- Andere Treiber als Modul auslagern
- Kernelhacking/Magic\_SysReq key hilft in Notsituationen
- gute Übersicht unter:
  - ➔ <http://www.kubieziel.de/computer/kernel-gensetup.html>

Richtiger Prozessortyp gibt mehr Performance und mehr Features. (zB No Execution Bits oder Hardware-Virtualisierung)

Option für Namen des Kernels: append to kernel release

Methode, um durch die Konfiguration zu kommen: Es müssen nicht alle möglichen Optionen bekannt sein. Es reicht, nur das zu ändern, was man ändern muss. Das Konfigurationstool informiert über Abhängigkeiten.

Die angezeigte Hilfe gibt in den meisten Fällen genug Information, dass man weiß, wonach man suchen muss, wenn etwas unklar ist. Angegebene Default-Optionen sind auch hilfreich.

Bei unbekanntenen Optionen kein Risiko eingehen, und Distributions-Voreinstellung nutzen.

Treiber deaktivieren hilft, dank der Verwendung von Modulen, auch nur noch, die Kompilierzeit klein zu halten.

Denn, wenn irgendwelche Komponenten als Module ausgelagert sind, dann benötigen sie keine Platz im RAM und werden dynamisch nachgeladen.

# Wichtige Optionen - initrd

---

- ermöglicht es, den Kernel klein zu halten
- enthält ein temporäres Root-Dateisystem
- wird vom Bootmanager in den Arbeitsspeicher geladen
- Inhalt wird über `/etc/mkinitrd` festgelegt
- Kernel muss nur ein Dateisystem enthalten

Der Bootmanager greift i.A. auf die Bios-Routinen zurück. Durch die Bios-Routinen kann der Bootloader den Kernel und die `initrd` in den Arbeitsspeicher laden. Moderne Kernel umgehen das Bios durch Treiber, und sind dadurch schneller.

`initramfs: /etc/initramfs-tools/modules`

Vorteile von `initramfs`: 1. Kernel benötigt kein Dateisystem mehr. 2. keine maximale Größe der Ramdisk. 3. Zugriffe auf Ramdisk werden nicht nochmal im Arbeitsspeicher gepuffert

# Wichtige Optionen - Module

---

- Module aktivieren
- Vorteile von Modulen:
  - ➔ halten den Kernel klein und flexibel
- Nachteile:
  - ➔ können ein Sicherheitsrisiko darstellen

Module sind in den seltensten Fällen ein Risiko: es lohnt sich, praktisch alles an Treibern und Dateisystemunterstützung auszulagern. Wird dann nur geladen, wenn es gebraucht wird. Und verbraucht keine Zeit beim Boot. Für mehr: siehe Vortrag über Module.

# Abschluss der Vorbereitung

---

- `make deb`
  - ➔ löst Abhängigkeiten der Sourcefiles auf
  - ➔ prüft auf Vollständigkeit
- `make clean`
  - ➔ löscht Überreste alter `make` Vorgänge
- **alternativ:** `make distclean`
  - ➔ löscht zusätzlich noch alte Konfigurationsdateien, aber auch eben erstellte

`make deb` & `make clean` sind vor dem Starten des Kompiliervorgangs wichtig, da sie viele Fehler, die sich erst später bemerkbar machen können, verhindern.

`make distclean` ist keine Option, die hier angewandt werden soll: Sie würde unsere ganze Konfigurationsarbeit zunichte machen.

# Kompiliervorgang und Installation

# Kompilervorgang

---

- endlich „finales“ `make` :
  - ➔ erstellt Kernelimage (`vmlinux`)
    - Steht in `./arch/i386/boot/compressed`
  - ➔ kompiliert Module, schreibt sie nach `/lib/modules/<kernelname>`
  - ➔ zusätzlich eine `system.map`
    - hilft beim Debuggen
    - nicht notwendig

`make vmlinux` erstellt nur den Kernel, `make modules` kompiliert die Module dazu. Dann müssen die Module noch installiert werden: `make modules_install`  
Option `-j` führt mehrere Threads aus (zB auf Zweikernmaschinen bietet sich `-j2` an)  
`system.map` enthält Speicheradressen von Namen und Symbole des Linuxkernels. Ist für uns unwichtig, aber später vielleicht für Debug-Aufgaben zu gebrauchen.  
für `initramfs` reicht `update-initramfs`

Geschätzte Dauer: ca 1 Stunde für den Übersetzungsvorgang. Gemessen auf einem 2 GHz Prozessor in einer Virtuellen Maschine. Die Geschwindigkeit ist natürlich vom benutzen System und dem zu kompilierenden Kernel abhängig: weniger Module und Treiber im Kernel führen zu kürzeren Kompilierzeiten.

# Installation: Ort & initrd

---

- kopiere neuen Kernel und system.map nach `/boot/`
- Kernel sollte `vmlinuz-<version>` heißen
- initrd wird in `/boot` erstellt:
  - ➔ `mkinitrd -k vmlinuz-<version> -i initrd-<version>`
- system.map, initrd und Kernelimage müssen die gleiche Endung haben und liegen am praktischsten im selben Verzeichnis

In der initrd stehen Module, die der Kernel braucht, aber die aus Platzgründen nicht in ihm selbst stehen, zB das root-Dateisystem, oder Treiber für diverse Geräte.

Optionen `-k` und `-i` erstellen eine Ramdisk für den neuen Kernel, und nicht für den aktuell Laufenden

# Installation Grub

---

- Grub:
  - ➔ Skript `update-grub` erledigt den Rest
  - ➔ reboot
  - ➔ fertig 😊

Wenn das nicht funktioniert:

(hier wird davon ausgegangen, dass alle wichtigen Daten in /boot stehen)

- in der `menu.lst` von Grub einen neuen Eintrag für den neuen Kernel und die neue Ramdisk anlegen.

- Einträge stehen ganz unten, am besten einen Neuen anlegen.

- bei Kernel da neue Image angeben, genauso wie für die Ramdisk

- boot-Optionen von alten Einträgen übernehmen, ausser man möchte

etwas Spezielles ändern, zB mehr Konsolen Ausgaben beim Boot fürs Debugging

- dann nochmal `update-grub` ausführen

- hier lohnt sich Ordnung: eindeutige Namen für Kernel und Ramdisk mit Versionsnummer helfen, die richtigen Daten einzutragen.

- Details im Vortrag über Bootmanager

# Ubuntu

---

- Ubuntu hat das `.dep` Paketmanagement
- Konfiguration wie schon bekannt
- **anstatt** `make clean, make dep, make`
  - ➔ `sudo make-kpkg clean`
  - ➔ `sudo make-kpkg --initrd --revision <Name> binary`
- Installation per Paketmanager

Option `--initrd` sorgt für Erstellung der Initial Ramdisk  
`--revision` legt den Namen fest, der danach eingegeben wird  
`binary` erstellt ein `.dep` Paket

Die Installation per Paketmanager sorgt dafür, dass der selbstkompilierter Kernel vom Paketmanager korrekt behandelt wird. So behält diese Tool den Überblick über die gesamte Software.

Die Installation per Paketmanager ist denkbar einfach: Doppelklick auf das eben erstellte Paket.

# Anmerkungen

Hier können natürlich nur „einfache“ Fehler mit „einfachen“ Lösungen angesprochen werden.

Für alles, was hier nicht abgedeckt wird (und das ist leider eine Menge. Fehler beim Kompilervorgang und der Behebung könnten wohl ein ganzes Proseminar füllen) gibt es im Internet Hilfe und hilfsbereite User und Entwickler: allgemeine Foren, die Foren der Distribution, wie immer bei Linux gilt: selber fragen macht schlau

# Patches

---

- erlauben ein Update der lokalen Source
- erhältlich zB. auf kernel.org
- klein, ersparen Downloads
- gerade die Differenz zwischen alter und neuer Source
- machen Probleme bei veränderten Source-Daten

Patches spielen gerade die Änderung der neuen Version zur alten ein, nicht mehr. Daher ist der Download des Patches klein.

Falss man selbst die Source verändert hat, geben Patches unter umständen Probleme, falls sie im selben Bereich etwas ändern sollen.

Patches sind fast überflüssig geworden, heutige Bandbreiten ermöglichen es, die ca 50 MB des Kernels schnell zu laden.

# Kompilieren schlägt fehl

---

- Mögliche Ursachen:

- ➔ `make dep/clean` vergessen

- ➔ falscher Compiler installiert

- ➔ Fehler im Verzeichnisbaum

- Lösung:

- ➔ Source neu laden und neu kompilieren

Vergessenes `make clean` oder `dep` sind häufig eine Fehlerursache.

Überbleibsel von alten Kompilervorgängen können zu unerwarteten Ergebnissen führen. Daher `make clean` ausführen.

Fehler im Verzeichnisbaum können durch Patches entstehen. Und genau wie die Reste alter Kompilervorgänge können sie unvorhersehbare Fehler entstehen lassen, meistens scheitert aber schon der Kompilervorgang.

Falls ein vom `make`-File nicht unterstützter Compiler installiert ist, hilft nur einen unterstützten wie `gcc` zu installieren.

# Neuer Kernel lässt sich nicht booten

---

- Mögliche Ursachen:

- ➔ Bootmanagereintrag vergessen
- ➔ fehlende Treiber für Bootmedium/Dateisystem des Bootmediums
- ➔ Kernel fehlerhaft kompiliert

- Lösungen:

- ➔ Bootmanager reparieren
- ➔ Kernel neu kompilieren

Wenn der Kernel nicht zum booten zu bewegen ist, hilf nichts. Alten Kernel booten, und nochmal den Neuen kompilieren. Die Fehlersuche gestaltet sich leider recht komplex, und ist nicht ohne weiteres durchführbar.

Es lohnt sich also, den alten Kernel aufzuheben, da dieser per Bootmanager geladen werden kann, falls der Neue seine Aufgabe nicht erfüllt.

# Instabiles System

---

- unerklärliche Fehler bei I/O, Systemabstürze etc.
- mögliche Ursachen:
  - ➔ falsch übersetzter Kernel, `make clean/dep` vergessen
  - ➔ Treiber
- Lösung:
  - ➔ Kernel neu übersetzen

Richtige, stabile Treiber (Treiber, die getestet werden sind als solche markiert) sind in den seltensten Fällen die Ursache für Fehler. Ein noch nicht fertiger, oder von den Linuxentwickelern freigegebener Treiber schon häufiger. Treiber, die Probleme verursachen, müssen durch eine Alternative ersetzt werden, oder garnicht verwendet. Beides ist leider nicht immer möglich.

Meistens hat man selbst irgendwelche Einstellungen am Kernel kaputtoptimiert. Diese Option(en) lassen sich meistens finden und zurücknehmen. Dennoch muss dazu der Kernel neu übersetzt werden.

# Monolithischer Kernel?

---

- <=> Mikrokern
- Linux ist monolithisch, vergleiche Wikipedia:
  - ➔ Als **monolithischen Kernel** bezeichnet man einen Betriebssystemkern, in dem nicht nur Funktionen zu Speicher- und Prozessverwaltung und zur Kommunikation zwischen den Prozessen, sondern auch Treiber für die Hardwarekomponenten und möglicherweise weitere Funktionen direkt eingebaut sind.
- trotz Module weiterhin monolithisch
  - ➔ Fehleranfälligkeit durch Module
  - ➔ Kann nicht die Sicherheitsmechanismen moderner Prozessoren nutzen
- früher (1.x) waren sogar Teile von Apache im Kernel kompiliert

Mikrokern: enthält nur grundlegende Funktionen wie Speicher und Prozessverwaltung. Alle anderen werden als eigene Prozesse ausgeführt. Gerätetreiber können im Benutzermodus laufen.

(Interessant ist in dieser Beziehung der MacOS Kernel: entstanden aus einem Mikrokern wurden viele performancekritische Systemteile in den Kernel gezogen, bis dieser von aussen wie ein Monolithischer Kernel aussieht, und auch wieder so performant läuft. Vortrag des CCC: <http://events.ccc.de/congress/2007/Fahrplan/events/2303.en.html>)

Danke für die Aufmerksamkeit

# Quellen

---

- <http://wiki.ubuntuusers.de/Kernel?highlight=%28kernel%29>
- <http://www.linuxhaven.de/dlhp/HOWTO/DE-Kernel-HOWTO.html>
- [http://de.wikipedia.org/wiki/Linux\\_\(Kernel\)](http://de.wikipedia.org/wiki/Linux_(Kernel))
- <http://www.kernel.org/>
- <http://gd.tuwien.ac.at/opsys/linux/tut/LinuxFibel/kconf.htm>
- <http://www.linuxfaqs.de/howto/linuxkernelhowto.php>
- [http://de.gentoo-wiki.com/Kernel\\_manuell\\_kompilieren](http://de.gentoo-wiki.com/Kernel_manuell_kompilieren)
- <http://www.kubieziel.de/computer/kerneloptionen.html>
- <http://www.thomashertweck.de/kernel26.html>
- <http://ubuntuforums.org/showthread.php?t=491387>
- [http://de.opensuse.org/SDB:Booten\\_mit\\_der\\_initial\\_ramdisk#Konzept\\_der\\_initial\\_ramdisk](http://de.opensuse.org/SDB:Booten_mit_der_initial_ramdisk#Konzept_der_initial_ramdisk)