

Wir bauen einen Kernel

Der Linux Kernel

Die Quelldateien des Linux Kernels auf kernel.org enthalten

- ▶ weit mehr als 10 Millionen Code-Zeilen und
- ▶ über 1.000 Makefiles

Die Quellen des Kernels

- ▶ Die Eigenschaften des Linux-Kernels sind konfigurierbar.
- ▶ Es werden zahlreiche Prozessorarchitekturen unterstützt.

Der Linux-Kernel wird mit Hilfe von `make` gebaut.

Das Werkzeug `autoconfig` kommt **nicht** zum Einsatz

Eine erste Orientierung bietet

`make help`

Die Konfiguration des Kernels

Die *Standardkonfiguration* findet man im Hauptverzeichnis der Kernel-Quellen in der (umfangreichen) Datei **.config**:

```
CONFIG_ARM=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_GENERIC_GPIO=y
CONFIG_HAVE_PROC_CPU=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_TRACE_IRQFLAGS_SUPPORT=y
...
```

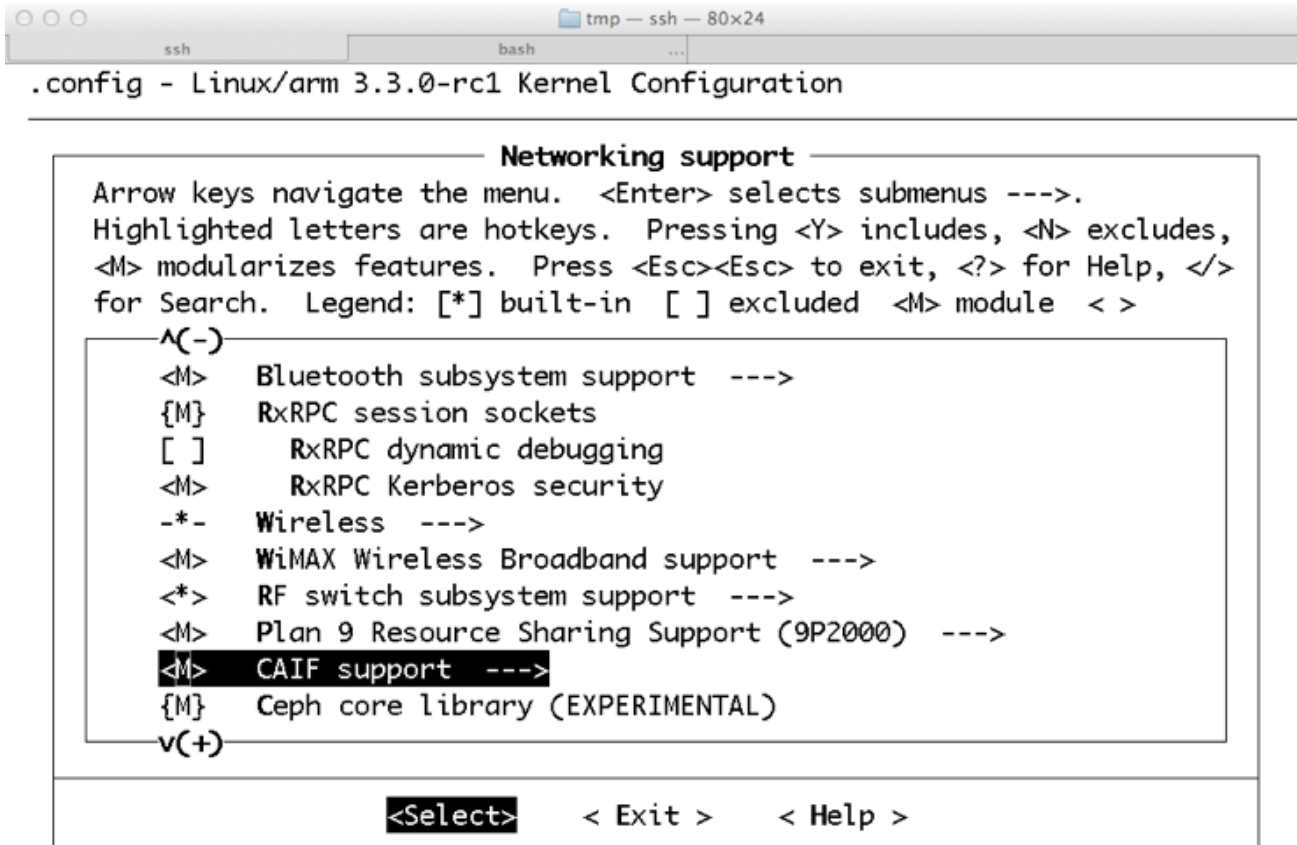
Anpassen der Standardkonfiguration

Wegen der sehr hohen Komplexität der zahlreichen Konfigurationsmöglichkeiten wird die Konfigurationsdatei meistens nicht mit einem Texteditor bearbeitet.

Es gibt mehrere **menügesteuerte Editoren** für die Konfiguration. Die Übersicht liefert wieder **make help**

Wir arbeiten hier mit einem Werkzeug:
make menuconfig

Menüsteuerung



```
tmp - ssh - 80x24
.config - Linux/arm 3.3.0-rc1 Kernel Configuration

Networking support
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

^(-)
<M> Bluetooth subsystem support --->
{M} RxRPC session sockets
[ ] RxRPC dynamic debugging
<M> RxRPC Kerberos security
*- Wireless --->
<M> WiMAX Wireless Broadband support --->
<*> RF switch subsystem support --->
<M> Plan 9 Resource Sharing Support (9P2000) --->
<M> CAIF support --->
{M} Ceph core library (EXPERIMENTAL)
v(+)
```

<Select> < Exit > < Help >

Mit Cursor- und anderen Tasten wird die Konfiguration eingestellt

Abspeichern

- ▶ Der Kernel wird menügesteuert konfiguriert.
- ▶ Bevor man den Konfigurator verlässt, wird man gefragt, ob die Änderungen gespeichert werden sollen.
- ▶ Wird die Frage bejaht, wird die zugehörige Konfiguration in **.config** abgelegt.

Standardkonfigurationen

- ▶ Es ist oft schwierig zu entscheiden, wie der Kernel für eine bestimmte Hardware konfiguriert werden muss.
- ▶ Wenn die Konfiguration nicht zur Hardware passt, kann der Kernel nicht gestartet werden.
- ▶ Für gängige Plattformen gibt es aber bereits Standardkonfigurationen.

Wir prüfen das für unseren Fall:

```
lothar@ubuntu:~/kernel/linux$ make help | grep bcm
bcm2709_defconfig      - Build for bcm2709
bcm2835_defconfig      - Build for bcm2835
bcm_defconfig          - Build for bcm
bcmrpi_defconfig       - Build for bcmrpi
```


Spezielle Konfigurationen

- ▶ Die abgebildete Ausgabe von `make help` wurde mit Hilfe eines für Raspberry Pi *spezialisierten* Kernel-Systems durchgeführt. Wir erhalten die passende Konfiguration über `bcm2709_defconfig`.
- ▶ Der Kernel für den Raspberry Pi 1 ist anders als der für die Version 2. Um einen Kernel für diese Version zu bauen, muss eine weitere Umgebungsvariable gesetzt werden:

`KERNEL=kernel7`

Ein Kernel für den Raspberry Pi

Wir generieren eine Konfigurationsdatei für den Raspberry Pi:

```
make bcm2709_defconfig
```

- ▶ Die `.config`-Datei enthält jetzt die zugehörige Konfiguration.
- ▶ Über den Konfigurator kann sie noch weiter angepasst werden.

Das Dateiformat

Das Ergebnis der Kernelkonstruktion ist eine **komprimierte** Datei, die den ausführbaren Kernel enthält:

arch/arm/boot/zImage

Die Datei kann auf die SD-Karte des Targets kopiert werden und passend in **kernel17.img** umbenannt werden.

Das kann im laufenden Betrieb des Pis gemacht werden, da die Boot-Partition unter **/boot** eingehängt ist.

Los geht's

Wir starten die Konstruktion des Kernels und aktivieren dazu 4 CPUs damit es etwas schneller geht:

```
make -j4
```

`make` kommentiert sein Vorgehen. Wir wählen hier Standard (silent). Mit geeigneten Parametern fällt die Kommentierung umfangreicher aus.

Wir analysieren im folgenden die Schlussphase des Konstruktionsprozesses.

Das Ergebnis

Die ELF-Datei (Executable and Linking Format) **vmlinux** enthält den eigentlichen Linux-Kernel (*kernel proper*).

Grundsätzlich ist die Konstruktion des Kernels damit abgeschlossen.

Noch nicht passend

Aus Platzgründen sollte unser Kernel aber komprimiert werden.

...und dann brauchen wir wieder Software, die den Kernel auspackt.

Daher sind noch weitere Schritte nötig. Insbesondere wird noch ein so genannter **Bootstrap-Loader** gebaut, der den Kernel auspackt startet.

Boot-Loader und Bootstrap-Loader

Begriffe:

Der Boot-Loader

enthält Anweisungen, die nach dem Einschalten des Targets ausgeführt werden. Er hat nichts mit Linux zu tun. In unserem Fall sind dies Dateien wie **start.elf**, und **bootcode.bin** die die GPU nach dem Einschalten startet.

Der Bootstrap-Loader

wird vom Boot-Loader gestartet und ist für den Start des Kernel verantwortlich.

Der Bootstrap-Loader

```
arch/arm/boot/compressed/head.o
```

`head.o` ist das Objekt, das vom Boot-Loader gestartet wird und das auch für das Auspacken und Starten des Kernels verantwortlich ist.

```
arch/arm/boot/compressed/piggy.gz
```

`piggy.gz` enthält die gepackte Image-Datei des Kernels.

Der Kernel reist Huckepack

```
arch/arm/boot/compressed/piggy.gzip.o
```

Ist erzeugt aus

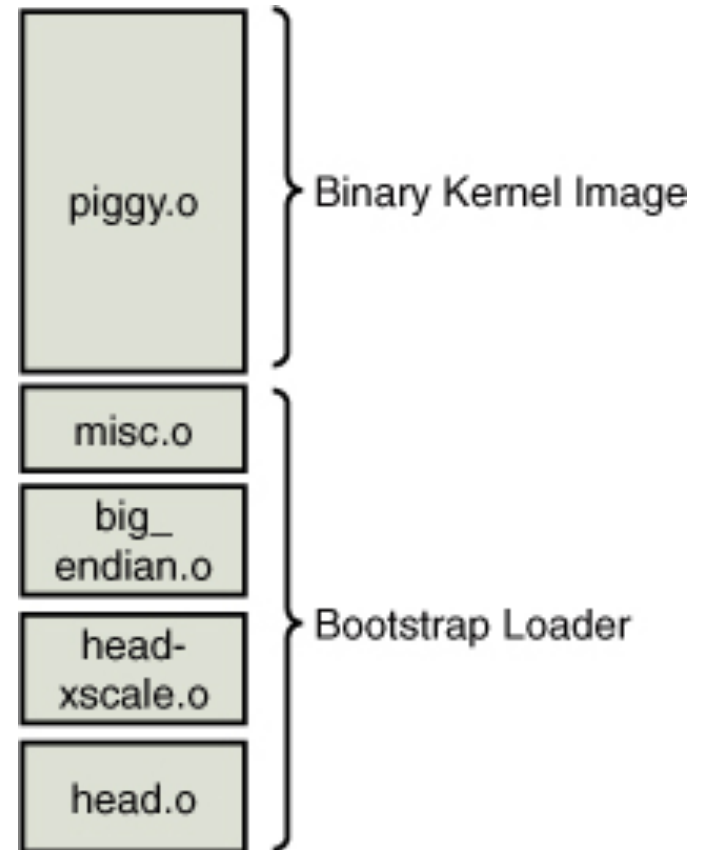
```
arch/arm/boot/compressed/piggy.gzip.S
```

Hier sehen wir den 'Trick', der zum Laden des Kernels verwendet wird:

```
.section .piggydata,#alloc
    .globl    input_data
input_data:
    .incbin   "arch/arm/boot/compressed/piggy.gzip"
    .globl    input_data_end
input_data_end:
```

Der Kernel reist Huckepack

Die Assemblerdatei enthält eine `.incbin`-Anweisung. Diese fügt die Binärdaten aus `piggy.gz` in den Code ein.



Zum Bootstrap-Loader wird also das Kernel-Image hinzugefügt. Daher auch der Name `piggy` (Piggypack=Huckepack) .

Uncompressing Linux...

Weitere Dateien wie

`arch/arm/boot/compressed/misc.c`

werden noch zum Bootstraploader gebunden.

`misc.c` ist übrigens die Quelle der berühmten Ausgabe

```
"Uncompressing Linux..."
```

Der letzte Schliff

Zum Schluss wird der Bootstrap-Loader gebunden.

Der Name der erzeugten Datei ist der gleiche wie der des Kernel Proper.

```
arch/arm/boot/compressed/vmlinux
```

Der letzte Schliff

Diese Datei wird komprimiert:

arch/arm/boot/zImage