Linux Kernel Hacking

Georg Gottleuber

Chaosseminar - CCC Ulm

13. Juli 2009

Betriebssystem - wofür?

Hardware direkt programmieren (Firmware / Bare Metal)

- Fehlertoleranz
- (quasi)parallele “Prozesse”
- Debugging
- Rechte-Management

Betriebssystem

- Kernel als Hardware-Abstraktion, bietet System-API
- Verwaltung der Ressourcen (CPU, Speicher, IO, Zeit)
- Anwendungen als Prozesse

Betriebssystem - wofür?

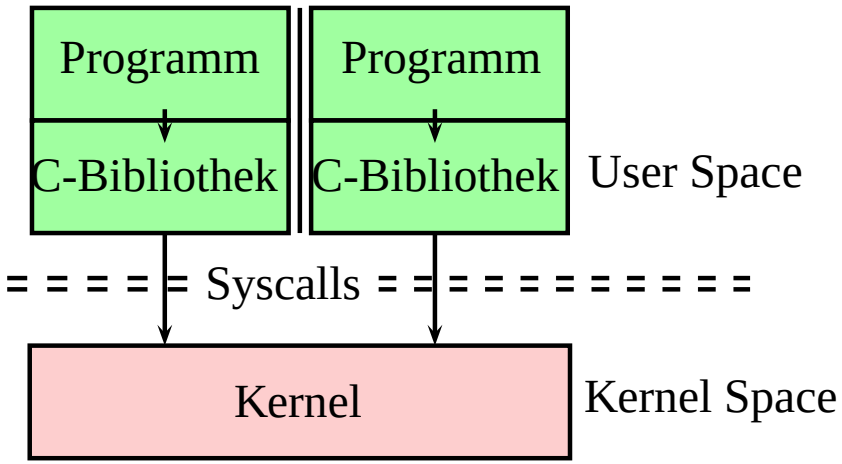
Hardware direkt programmieren (Firmware / Bare Metal)

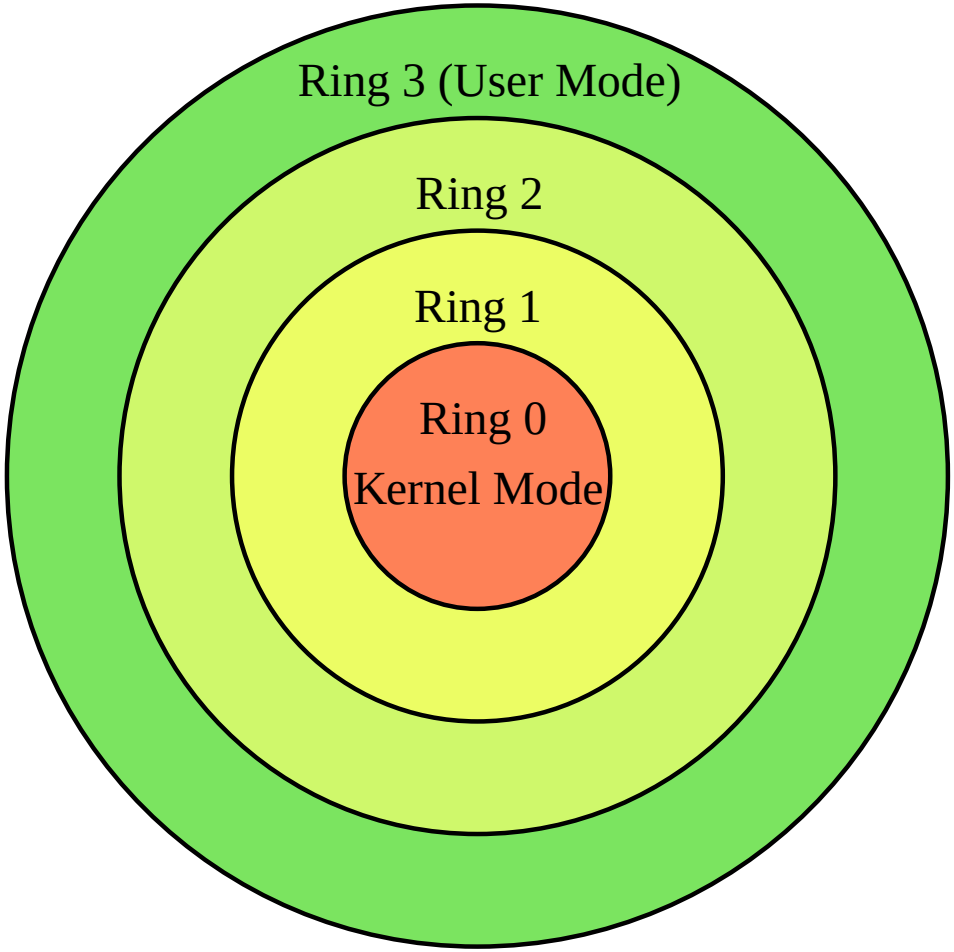
- Fehlertoleranz
- (quasi)parallele “Prozesse”
- Debugging
- Rechte-Management

Betriebssystem

- Kernel als Hardware-Abstraktion, bietet System-API
- Verwaltung der Ressourcen (CPU, Speicher, IO, Zeit)
- Anwendungen als Prozesse

Aufbau

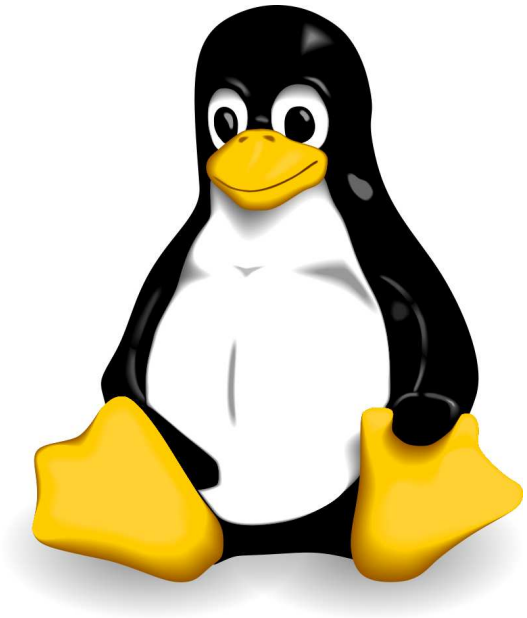




Anzeige der Syscalls mit `strace`

Weshalb mit dem Kernel beschäftigen?

- Kernel selbst entwickeln:
  - Hardware-Unterstützung (eigene Treiber)
  - Neue Funktionalität (Algorithmen, Echtzeit, etc.)
- Security (Kernel-Rootkits)
- (Sehr gründliche) Fehlersuche
- Neugier



Geschichte

- 1983 GNU, Richard Stallman
- 1991 Linux, Linus Torvalds:
  Hello everybody out there using minix -

  I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. [...]

  `http://groups.google.com/group/comp.os.minix/msg/b813d52cbc5a044b`
- $\Rightarrow$ GNU/Linux Distributionen

Metadaten

- GPL (v.2)
- kernel.org
- 350 MB Quellcode, ca. 7.230.573 LoC
- 21 Hardware-Architekturen
- Änderungen durchschnittlich pro Tag:
  - 3.621 Zeilen hinzugefügt
  - 1.550 Zeilen entfernt
  - 1.425 Zeilen verändert
- Über 1000 aktive Entwickler

$\Rightarrow$ Organisatorische Regelungen!

Eigenschaften

- Mehrbenutzer
- Monolithischer Aufbau
- Ladbare Module
- Kernel-Threading
- Multiprozessorfähig (SMP)
- Präemptiv

Was man alles braucht

- Entwicklungstools (Editor, Make, Perl)
- GCC (oder ICC)
- Quellcode:
  - `www.kernel.org`
  - `git clone git://git.kernel.org/pub/scm /linux/kernel/git/torvalds/linux-2.6.git`

Was man tun muss

- `make menuconfig`
- `make && make modules_install`
- Installieren:
  - `make install`
  - `cp arch/x86/boot/bzImage + vim /boot/grub/menu.lst`

Tools

- Git (“Git The Basics Tutorial” excess.org)
- Ctags + Cscope / Global
- LXR “the Linux Cross Referencer”
- KConfig (`Kconfig`-Dateien)
- KBuild (Makefiles)

Kernel Mode != User Mode

- Kein C++
- Keine Programm-Bibliotheken
  (dafür Kernel-API)
- Keine Gleitkommaberechnungen
- Namespace Pollution
  - `EXPORT_SYMBOL(`*myVar*`)`
- Kein Speicherschutz
- Kleiner, statischer Stack (4 oder 8 KiB)

Sightseeing Tour

- Boot
  - `arch/x86/boot/header.S`
  - `arch/x86/boot/main.c`
  - `arch/x86/boot/compressed/head_32.S:startup_32`
  - `arch/x86/kernel/head_32.S:startup_32`
  - `init/main.c:start_kernel()`
    ⇒ `rest_init()`
    ⇒ `kernel_init()`
    (“Kernel Walkthrough” excess.org)
- `/dev/null (drivers/char/mem.c)`

Beispiele

- halloKernel.ko
- minDev.ko
- divDev.ko
- OOM-Killer
- bad.ko

Selber Kernel-Hacken

- Webseiten:
  - `http://lwn.net/Kernel/`
  - `http://kernelnewbies.org/`
  - `https://kerneltrap.org/`
- LKML
- *"Where Linux Kernel Documentation Hides"*, Ottawa Linux Symposium

Submitting code

- *"How to Participate in the Linux Community"*
- `Documentation/SubmitChecklist`
- `Documentation/SubmittingPatches`
- `Documentation/SubmittingDrivers`

Vielen Dank für die Aufmerksamkeit!

Fragen?