Praktikum angewandte Systemsoftwaretechnik (PASST)

Kernel-Module / Aufgabe 5

13. Dezember 2018

Tobias Langer, Stefan Reif, Michael Eischer,
Bernhard Heinloth und Florian Schmaus

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Kernel-Module

Code im Linux-Kern

Großteil des Codes im Linux-Kern besteht aus Gerätetreibern:

- Kernkomponenten (Prozessor, IRQ-Controller, Timerbausteine, ...)
  - Werden **immer** für ein funktionales System benötigt
  - Unmittelbarer Teil des Kerns
- Peripheriegeräte
  - Bussysteme: PCI(e), SATA, USB, ...
  - Treiber für einzelne Geräte und Geräteklassen:
    Tastatur, Maus, Grafikkarten, Festplatten, Soundkarten, ...
- Abhängig von der Hardware werden Module für die Geräte bei Bedarf geladen

Code im Linux-Kern

Großteil des Codes im Linux-Kern besteht aus Gerätetreibern:

- Kernkomponenten (Prozessor, IRQ-Controller, Timerbausteine, ...)
  - Werden **immer** für ein funktionales System benötigt
  - Unmittelbarer Teil des Kerns
- Peripheriegeräte
  - Bussysteme: PCI(e), SATA, USB, ...
  - Treiber für einzelne Geräte und Geräteklassen: Tastatur, Maus, Grafikkarten, Festplatten, Soundkarten, ...
- Abhängig von der Hardware werden Module für die Geräte bei Bedarf geladen

Aufgabe 5

Entwicklung eines Kerneltreibers für ein USB-Gerät

Entwickeln im Linux-Kern

- Dokumentation (ja, es gibt sie!)
  - `Documentation/` enthält Anleitungen, Erklärungen, Beschreibung von Konzepten für die verschiedensten Teile des Linux-Kerns

Entwickeln im Linux-Kern

- Dokumentation (ja, es gibt sie!)
  - `Documentation/` enthält Anleitungen, Erklärungen, Beschreibung von Konzepten für die verschiedensten Teile des Linux-Kerns
- Zusätzlich kann man für große Teile des Linux-Kerns eine Beschreibung der Interfaces ähnlich Doxygen generieren

```
01 $ cd <KERNEL_SOURCES>
02 $ make htmldocs
```

... landet in `Documentation/output/`

Entwickeln im Linux-Kern

- Dokumentation (ja, es gibt sie!)
  - `Documentation/` enthält Anleitungen, Erklärungen, Beschreibung von Konzepten für die verschiedensten Teile des Linux-Kerns
- Zusätzlich kann man für große Teile des Linux-Kerns eine Beschreibung der Interfaces ähnlich Doxygen generieren

```
01 $ cd <KERNEL_SOURCES>
02 $ make htmldocs
```

  … landet in `Documentation/output/`
- Für beides gilt: Always take with a grain of salt
  Linux hat keine stabile API innerhalb des Kerns
  - Dokumentation kann veralten
  - sich auf eine alte Version des Interfaces beziehen
  - oder schlichtweg falsch sein
- Die beste Dokumentation ist oft der Code von anderen

Entwickeln im Linux-Kern (1/2)

- Die meisten Geräte können mehrfach vorhanden sein
  - Daten für die Instanzen eines Gerätes müssen dynamisch allokiert werden
  - Beim Entfernen des Gerätes muss man sie dynamisch wieder freigeben
- Dynamische Speicherverwaltung - wie geht das im Kern?
  - `malloc()` und `free()` funktionieren im Linux-Kern nicht einfach so
  - Dafür gibt es eine eigene API: `kmalloc()`, `kzalloc()`, `kfree()`

Entwickeln im Linux-Kern (2/2)

- Wie unterscheidet sich Kernel-Code sonst noch von Userlevel-Code?

`Documentation/output/kernel-hacking/index.html`

Guter Einstieg in die Kernel-Entwicklung
Liefert eine Übersicht über die Besonderheiten der Entwicklung von Kernel-Code

`Documentation/output/core-api/kernel-api.html`

Enthält eine Interfacebeschreibung für viele Kernkomponenten und Bibliotheken (u.a. ein Subset der C-Bibliothek)

Gerätetreiber in Linux – Module (1/2)

Ein einfaches Kernelmodul

```
01 #include <linux/module.h>
02 #include <linux/kernel.h> /*printk*/
03 int __init simple_module_init(void)
04 {
05     printk(KERN_INFO "module loaded\n");
06     return 0;
07 }
08 void __exit simple_module_exit(void)
09 {
10     printk(KERN_INFO "module unloaded\n");
11 }
12 module_init(simple_module_init);
13 module_exit(simple_module_exit);
14
15 MODULE_LICENSE("GPL");
```

Gerätetreiber in Linux – Module (2/2)

- Makefile

```
01 obj-m += simple_module.o
02
03 all:
04     make -C <KERNEL_SOURCE> M=$(PWD)
05
06 clean:
07     make -C <KERNEL_SOURCE> M=$(PWD) clean
```

- Kann man einfach laden

```
01 $ insmod simple_module.ko
```

- … und entladen

```
01 $ rmmod simple_module
```

Hardware – Universal Serial Bus (USB) (1/2)

- Asymmetrischer Bus (Baum)
  - Ein Host (PC) (Wurzelknoten) und viele Functions (angeschlossene Geräte, Blätter)
  - Kommunikation wird ausschließlich vom Host initiiert
  - Geräte können nicht autonom miteinander kommunizieren
- Geschwindigkeitsstufen
  - Low-Speed (1,5 Mbit/s), USB 1.0
  - Full-Speed (12 Mbit/s), USB 1.0
  - High-Speed (480 Mbit/s), USB 2.0
  - SuperSpeed (5 Gbit/s), USB 3.0
  - SuperSpeedPlus (10 Gbit/s), USB 3.1 Gen 2

Hardware – Universal Serial Bus (USB) (2/2)

- Vier unterschiedliche Kommunikationsmechanismen:

  Control Transfers Unregelmäßige Anfragen vom PC an das Gerät

  z.B. Enumeration Sequence

  Bulk Transfers Aperiodisch; für große Pakete ohne zeitliche Garantien

  z.B. USB-Storage-Device

  Interrupt Transfers Periodische Kommunikation; begrenzte Antwortzeit

  z.B. Maus, Tastatur

  Isochronous Transfers Periodische, kontinuierliche Datenströme

  z.B. Webcam

Gerätetreiber in Linux - USB-Geräte

- Tiefere Ebenen des USB-Protokolls sind in Form eines Host-Controller-Treibers (HCD) schon implementiert
- Benutzung der unterschiedlichen USB-Transferarten direkt möglich
- Diese Funktionalität kann über `<linux/usb.h>` eingebunden werden
- Writing USB Device Drivers
  - Registrieren eines USB-Gerätetreibers im System
  - Anschließen und Entfernen von USB-Geräten
  - Kommunikation mit dem Gerät
  - Asynchrone USB-Transfers per **U**SB **R**equest **B**locks (URB)
  - Synchrone USB-Transfers für die Aufgabe ausreichend

  `Documentation/output/driver-api/usb/writing_usb_driver.html`

USB: Endpoints und Pipes

- USB-Geräte bieten Kommunikationsendpunkte (Endpoints) an
- Auf Hostseite spricht man mit einem Gerät über einen Kanal (Pipe), der mit einem bestimmten Endpunkt (Endpoint) verbunden ist
- Art und Anzahl der Endpunkte sind gerätespezifisch
- Alle Geräte müssen mindestens den Endpunkt 0 bereitstellen, der für Control Transfers benötigt wird (u.a. für die Konfiguration)

Weitere Informationen

- USB Spezifikation (siehe `/proj/i4passt/doc`)
- `http://www.beyondlogic.org/usbnutshell/usb1.shtml`

USB-Temperatursensor

USB-Temperatursensor

- Bauanleitung und Quellen
  - Firmware & Userspacetreiber: `https://gitlab.cs.fau.de/i4/passt/ds1820tousb`
  - Original-Firmware: `http://www.poempelfox.de/ds1820tousb/`
  - neues Layout: `https://gitlab.cs.fau.de/i4/passt/passtboard-v2`
- Steuert mehrere Temperatursensoren über 1-Wire-Bus an
- Steuerung vom PC aus per USB Control Transfers möglich
  - Rescan der angeschlossenen Temperatursensoren
  - Temperatur- und Statusinformationen der einzelnen Sensoren auslesen
  - Reset des kompletten Gerätes
  - Synchrone Transfers für die Aufgabe ausreichend

USB: Control Transfers

- Abwicklung über den immer vorhandenen Endpunkt 0
- **Festverdrahtete** (Konfiguration etc.) und **gerätespezifische** Befehle
- Parameter für Control Transfers

Parameter	Größe	Beschreibung
request type	1 Byte	Charakteristik der Anfrage
request	1 Byte	Nummer der Anfrage
value	2 Byte	1. Parameter für die Anfrage
index	2 Byte	2. Parameter für die Anfrage
length	2 Byte	Länge des Datenpaketes

(vgl. USB 2.0 Spezifikation Abschnitt 9.3)

Befehle für den Temperatursensor

- Der **Request-Type** für die Befehle ist immer gleich (USB Spec S. 248):
  - Datentransferrichtung ist vom Gerät zum PC
  - Anfragen sind vendor-spezifisch
  - Ziel der Anfrage ist das Gerät

    `request type 0xc0`
- Kurze Statusabfrage:

Aufrufparameter

```
request      1
value        0
index        0
```

Antwort

```
struct short_status {
    uint8_t  version_high;
    uint8_t  version_low;
    uint32_t timestamp;
    uint8_t  supported_probes;
    uint8_t  padding;
}__packed;
```

- `supported_probes`: Über die Lebenszeit des Gerätes am Bus konstant

Befehle für den Temperatursensor

- Lange Statusabfrage:

Aufrufparameter

```
01 request      3
02 value        0
03 index        0
```

Antwort

```
01 struct probe_status {
02     uint8_t  serial[6];
03     uint8_t  type;
04     uint8_t  flags;
05     uint8_t  temperature[2];
06     uint32_t timestamp;
07     uint8_t  padding[2];
08 }__packed;
09 struct probe_status
10     answer[supported_probes];
```

- Liefert immer Status für alle unterstützten Sensoren
- Flags: `0x01` Sensor ist vorhanden, sonst ist der Slot unbenutzt
  `0x02` Sensor wird parasitär mit Spannung versorgt
- Mehrere Bytes umfassende Werte sind little-endian
- Temperatur ist ein 12-bit Zweierkomplement-Wert in sechzehntel Grad (für `ds18b20` mit `type == 0x28`)

Befehle für den Temperatursensor

- Neuerkennung aller Sensoren am 1-Wire-Bus:

Aufrufparameter

```
request     2
value       0
index       0
```

Antwort

```
struct rescan_reply {
    uint8_t  answer;
};
```

- Im Erfolgsfall zwei Antworten möglich
  - **23** Neuerkennung wird gestartet
  - **42** Neuerkennung wird schon durchgeführt
- Fortschritt der Neuerkennung per `value = 0x01` abfragbar. Erkennung ist bei Antwort 23 abgeschlossen

Befehle für den Temperatursensor

- Neuerkennung aller Sensoren am 1-Wire-Bus:

Aufrufparameter

```
request     2
value       0
index       0
```

Antwort

```
struct rescan_reply {
    uint8_t  answer;
};
```

- Im Erfolgsfall zwei Antworten möglich
  - **23** Neuerkennung wird gestartet
  - **42** Neuerkennung wird schon durchgeführt
- Fortschritt der Neuerkennung per `value = 0x01` abfragbar. Erkennung ist bei Antwort 23 abgeschlossen
- Reset des kompletten Gerätes:

Aufrufparameter

```
request     4
value       0
index       0
```

- Das Geräte sollte bei diesem Kommando keine Antwort schicken
- Das Bereitstellen eines Empfangspuffers schadet trotzdem nicht

sysfs - Kernelzustand für Benutzer sichtbar machen

- Interaktion mit dem USB-Gerät via sysfs
- Benutzung von sysfs: Documentation/filesystems/sysfs.txt
- In a Nutshell
  - Große Teile des Kerns sind aus kobjects aufgebaut
    - Objektorientierung in C: Documentation/kobject.txt
  - sysfs-Struktur spiegelt die Objektstruktur im Kern wieder
  - Spezielle Schnittstellen für Geräte(-Treiber)
    - Gerät erscheint im sysfs als Verzeichnis
    - Erzeugen von Dateien durch `device_create_file(&dev, attr)`
    - Löschen von Dateien durch `device_remove_file(&dev, attr)`

sysfs - Temperatursensoren

sysfs-Einträge sollen folgende Funktionen bereitstellen:

- Temperatur jedes Sensors durch Lesen einer eigenen Datei abfragbar

```
$ ls /sys/bus/usb/devices/4-1.5:1.0/
bInterfaceNumber bNumEndpoints modalias power ...
temp0 temp1 temp2 rescan reset
$ cat /sys/.../temp1
23.43
```

- Rescan des 1-Wire-Bus

  „Hotplug" von Sensoren auf der Platine

```
$ echo 1 > rescan
```

- Reset des Gerätes

```
$ echo 1 > reset
```

Verbinden von USB-Geräten mit KVM (1/2)

Reale USB-Geräte an eine KVM weiterleiten:

- Aktivieren des USB Treibers

  `-usb`

- Ein bestimmtes Gerät

  `-device usb-host,hostbus=<bus>,hostaddr=<id>`

  Mühsam wegen Hotplug an verschiedenen Ports:
  Bus- und Adress-ID nicht zwingend eindeutig

- Ganze Geräteklassen

  `-device usb-host,vendorid=<vid>,productid=<pid>`

  Für unsere Temperatursensoren ist das `16c0:05dc`

  `-device usb-host,vendorid=0x16c0,productid=0x05dc`

Verbinden von USB-Geräten mit KVM (2/2)

- *Problem:* KVM benötigt Lese- und Schreibrechte fürs Gerät
  Lösung: **udev**

```
01 ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="05dc", MODE="666"
```

/etc/udev/rules.d/99-usbtemp.rules

Userspace-Anwendung zur Darstellung der Messdaten

- Temperaturen periodisch aus `/sys` auslesen
- evtl. geeignet Zwischenspeichern
- zeitlichen Verlauf aller Sensoren in Graphen ausgeben
- Graph soll „live“ aktualisiert werden
- mögliche Werkzeuge u.A.:
  - GNUPLOT
  - MATPLOTLIB (Python)
  - RRDTOOL
  - R
  - ROOT (`root.cern.ch`)

Aufgabe 5

USB-Temperatursensor

- Einarbeiten in die benötigten APIs im Linux-Kern
  - Dokumentation
  - Codebeispiele
- Programmieren des Gerätetreibers für den Temperatursensor
  - Ordentliche Speicherverwaltung
  - Linux Coding Style beachten
- Programmieren der Userspace-Anwendung
- Last, but not least:
  - Die Hardware muss gelötet werden (Schon erledigt)

Abgabe: Bis 10. Januar 2019 durch Vorführung in einer Rechnerübung

Fragen?