Praktikum angewandte Systemsoftwaretechnik

Aufgabe 5

Alexander Würstlein

Lehrstuhl Informatik 4

2016-12-08

Code im Linux-Kern

Der 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 geladen

Code im Linux-Kern

Der 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 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 dazu kann man für große Teile des Linux-Kerns eine Beschreibung der Interfaces ähnlich Doxygen/Javadoc generieren

```
> cd <KERNEL_SOURCES>
> make htmldocs
```

Die gebaute Dokumentation landet in `Documentation/DocBook/`

Entwickeln im Linux-Kern

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

```
> cd <KERNEL_SOURCES>
> make htmldocs
```

  Die gebaute Dokumentation landet in `Documentation/DocBook/`
- 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

- Die meisten Geräte können mehrfach vorhanden sein
  - Daten für die Instanzen müssen dynamisch allokiert werden
  - Beim Entfernen des Gerätes muss man sie dynamisch wieder freigeben
- Dynamische Speicherverwaltung - wie macht man 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

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

`Documentation/DocBook/kernel-hacking/`

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

`Documentation/DocBook/kernel-api/`

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

Gerätetreiber in Linux – Module

Ein einfaches Kernelmodul

```
#include <linux/module.h>
#include <linux/kernel.h> /*printk*/

int __init simple_module_init(void)
{
        printk("module␣loaded\n");
}
void __exit simple_module_exit(void)
{
        printk("module␣unloaded\n");
}
module_init(simple_module_init);
module_exit(simple_module_exit);

MODULE_LICENSE("GPL");
```

simple_module.c

```
obj-m += simple_module.o

all:
    make -C <KERNEL_SOURCE> \
        M=$(PWD)
clean:
    make -C <KERNEL_SOURCE> \
        M=$(PWD) clean
```

Makefile

Gerätetreiber in Linux – Module

Ein einfaches Kernelmodul

```
#include <linux/module.h>
#include <linux/kernel.h> /*printk*/

int __init simple_module_init(void)
{
        printk("module␣loaded\n");
}
void __exit simple_module_exit(void)
{
        printk("module␣unloaded\n");
}
module_init(simple_module_init);
module_exit(simple_module_exit);

MODULE_LICENSE("GPL");
```

simple_module.c

```
obj-m += simple_module.o

all:
    make -C <KERNEL_SOURCE> \
        M=$(PWD)
clean:
    make -C <KERNEL_SOURCE> \
        M=$(PWD) clean
```

Makefile

Kann man einfach laden

```
> insmod simple_module.ko
```

Gerätetreiber in Linux – Module

Ein einfaches Kernelmodul

```
#include <linux/module.h>
#include <linux/kernel.h> /*printk*/

int __init simple_module_init(void)
{
        printk("module␣loaded\n");
}
void __exit simple_module_exit(void)
{
        printk("module␣unloaded\n");
}
module_init(simple_module_init);
module_exit(simple_module_exit);

MODULE_LICENSE("GPL");
```

simple_module.c

```
obj-m += simple_module.o

all:
    make -C <KERNEL_SOURCE> \
        M=$(PWD)
clean:
    make -C <KERNEL_SOURCE> \
        M=$(PWD) clean
```

Makefile

Kann man einfach laden

```
> insmod simple_module.ko
```

... und entladen

```
> rmmod simple_module
```

Hardware – Universal Serial Bus (USB)

- 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
- Vier unterschiedliche Kommunikationsmechanismen:
  - *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
  - *Control Transfers*: Unregelmäßige Anfragen vom PC an das Gerät
    z.B. Enumeration Sequence
- Geschwindigkeitsstufen
  - Low-Speed bis Super-Speed+
  - 1,5 Mbit/s bis 10 Gbit/s

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: `Documentation/DocBook/writing_usb_device_driver`
  - Registrieren eines USB-Gerätetreibers im System
  - Anschließen und Entfernen von USB-Geräten
  - Kommunikation mit dem Gerät
  - Asynchrone USB-Transfers mittels **U**SB **R**equest **B**locks (`URB`)
  - Für die Aufgabe sind synchrone USB-Transfers ausreichend

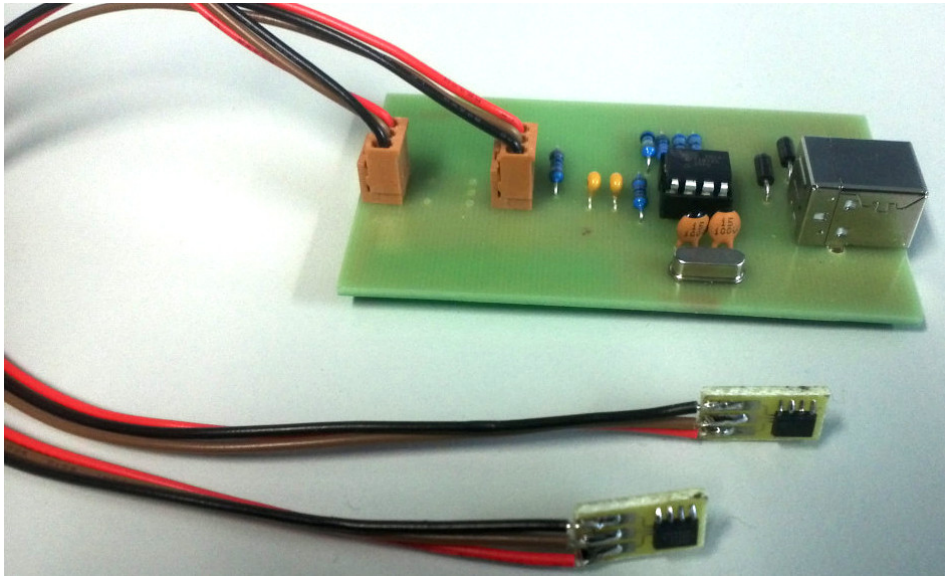
USB: *Endpoints* und *Pipes*

- 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
- Allerdings müssen alle Geräte 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



- Bauanleitung und Quellen zur Firmware und Userspacetreiber
  - `http://www.poempelfox.de/ds1820tousb/`
  - `git clone https://git.informatik.uni-erlangen.de/ds1820tousb`
- Steuert mehrere Temperatursensoren über 1-Wire-Bus an
- Steuerung vom PC aus mittels USB Control Transfers möglich
  - Rescan der angeschlossenen Temperatursensoren
  - Temperatur- und Statusinformationen der einzelnen Sensoren
  - Reset des kompletten Gerätes

USB: Control Transfers

- Abwicklung über den immer vorhandenen Endpunkt 0
- *Festverdrahtete* (Konfiguration etc.) und *gerätespezifische* Befehle
- Parameter für Control Transfers (vgl. USB Spezifikation 9.3)

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	1 Byte	Länge des Datenpaketes

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

```
request         3
value           0
index           0
```

Antwort

```
struct probe_status {
    uint8_t  serial[6];
    uint8_t  type;
    uint8_t  flags;
    uint8_t  temperature[2];
    uint32_t timestamp;
    uint8_t  padding[2];
}__packed;
struct probe_status
    answer[supported_probes];
```

- Liefert immer Status für alle unterstützten Sensoren
- Flags
  - 0x01: Sensor ist vorhanden, ansonsten ist der Slot unbenutzt
  - 0x02: Sensor wird parasitär mit Spannung versorgt
- Mehrere Bytes umfassende Werte sind little-endian
- Temperatur ist ein 9-bit Zweierkomplement-Wert

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

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
- 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
  - Struktur von sysfs spiegelt die Objektstruktur im Kern wieder
    - kobjects erscheinen im sysfs als Verzeichnis
    - Erzeugen von Dateien durch sysfs_create_file(&kobject,attr)
    - Löschen von Dateien mit sysfs_remove_file(&kobject,attr)

sysfs - Temperatursensoren

sysfs-Einträge sollen folgende Funktionen bereitstellen

- Temperatur jedes Sensors durch Lesen einer eigenen Datei

```
> ls /sys/bus/usb/devices/4-1.5:1.0/
bInterfaceNumber bNumEndpoints modalias power ...
temp0 temp1 temp2 rescan reset
> cat /sys/.../temp1
23.4
```

- 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

Reale USB-Geräte an eine KVM weiterleiten:

- Ein bestimmtes Gerät

```
-usbdevice host:bus.addr
```

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

- Ganze Geräteklassen

```
-usbdevice host:vendor_id:product_id
```

  Für unsere Temperatursensoren ist das 16c0:05dc:

```
-usbdevice host:16c0:05dc
```

- Problem: KVM benötigt Lese- und Schreibrechte auf das Gerät
  Lösung: udev

```
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 einem Graphen ausgeben
- mehr Punkte wenn der Graph „live“ aktualisiert wird
- mögliche Werkzeuge u.A.:
  - gnuplot
  - rrdtool
  - R
  - root (`root.cern.ch`)

Aufgabe 5

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

Abgabe: bis 2017-01-27 durch Vorführung in einer Rechnerübung