

Praktikum angewandte Systemsoftwaretechnik (PASST)

Kernelmodule / Aufgabe 5

27. Juni 2019

Tobias Langer, Stefan Reif, Michael Eischer
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

Rückblick

- Fehlersuche mit `git bisect`
 - Bisektionsalgorithmus auf Commit Historie
 - Erfordert übersetzbare & testbare Historie
- Abgrenzung Open Source und Free Software
 - Mehr oder weniger restriktive Lizenzmodelle
- Linux als Open Source Projekt
 - Transparente Kommunikation über E-Mails & Mailinglisten
 - Bewertung von Beiträgen im Kreuzgutachten
 - Konventionen im Linuxkern

Motivation

“Every good work of software starts by scratching a developer’s personal itch.”

The Cathedral and the Bazaar – Eric S. Raymond

“Every good work of software starts by scratching a developer’s personal itch.”

The Cathedral and the Bazaar – Eric S. Raymond

Linux ist Open Source:

- Erlaubt gemeinsame Überarbeitung und Pflege von Code
- Erlaubt Entwicklung eigener Erweiterungen
 - Systemschnittstellen
 - Gerätetreiber für neue Geräte
 - ...

Kernkomponenten

- Verwaltung von Systemkomponenten
- Unmittelbarer Teil des Kerns
- Notwendig für ein funktionales System
 - Prozessorzuteilung, Speicherverwaltung, ...

Gerätetreiber

- Schnittstelle zwischen Kern und Geräten
- Implementiert gerätespezifische Aufgaben
 - Initialisierung & Ansteuerung von Geräten
- Bei Bedarf dynamisch ladbar als *Module*

Entwickler können auf Infrastruktur zurückgreifen:

- Wiederverwendbare Komponenten & Werkzeuge
 - Helfermacros
 - Datenstrukturen
 - ...
- Designkonzepte
 - Schnittstellen in den Userspace
- ...und jede Menge Dokumentation

Lernziele

Im Anschluss an diese Aufgabe solltet Ihr...

- die Funktionsweise von USB-Geräten erklären
- die Funktionsweise des `sysfs`-Dateisystems beschreiben
- mithilfe der Dokumentation die Funktionsweise von Linuxsubsysteme selbst recherchieren
- eigenständig ladbare Module für Linux entwickeln

...können.

Agenda

Gerätetreiber unter Linux

Universal Serial Bus

USB-Temperatursensor

Hinweise zu QEMU und USB-Geräten

Zusammenfassung

Aufgabe 5

Gerätetreiber unter Linux

Gerätetreiber

- Machen Geräte in System nutzbar
- Kommunikation zwischen Kernel und (virtuellen) Geräten
 - Gerätespezifische Kommunikation zum Gerät hin
 - Einheitliche Schnittstelle zum Kernel

Gerätetreiber unter Linux

- Gerätetreiber sind idR. Module
- Module sind dynamisch ladbar

Aufgaben des Treibermoduls

1. (De-)Registrierung des Treibermoduls im System
2. Verwaltung von Geräteinstanzen
3. Kommunikation mit dem Gerät
4. Optional: Bereitstellen von Userspaceschnittstellen
 - `sysfs`: Kommunikation via Strings
 - `ioctl`: Kommunikation via Binärdaten

(De-)Registrierung von Modulen (1/2)

Ein einfaches Kernelmodul

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

int __init simple_module_init(void)
{
    printk(KERN_INFO "module loaded\n");
    return 0;
}

void __exit simple_module_exit(void)
{
    printk(KERN_INFO "module unloaded\n");
}

module_init(simple_module_init);
module_exit(simple_module_exit);

MODULE_LICENSE("GPL");
```


(De-)Registrierung von Modulen (2/2)

- Makefile

```
obj-m += simple_module.o
```

```
all:
```

```
    make -C <KERNEL_SOURCE> M=$(PWD)
```

```
clean:
```

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

- Kann man einfach laden

```
$ insmod simple_module.ko
```

- ... und entladen

```
$ rmmod simple_module
```

- Geräte können mehrfach vorhanden sein
 - Dynamische Verwaltung von Geräteinstanzen notwendig
 - Allokation und Freigabe von Daten im Kern
- Dynamische Speicherverwaltung - wie geht das im Kern?
 - `malloc()`, `free()` funktionieren im Linuxkern nicht einfach so
 - Eigene API: `kmalloc()`, `kzalloc()`, `kfree()`

Userspaceschnittstelle mit sysfs

- Kernelobjekte (kobjects)
 - Große Teile des Kerns sind aus kobjects aufgebaut
- Virtuelles Dateisystem unter /sys/
 - Unix Philosophie: „*Everything is a file*“
 - Zugriff auf Kernelobjekte über FileIO
 - sysfs-Struktur spiegelt die Objektstruktur im Kern wieder

sysfs-Treiberschnittstelle

- Repräsentation von Geräten als Verzeichnis unter sys
- Erzeugen von Dateien: `device_create_file(&dev, attr)`
- Löschen von Dateien: `device_remove_file(&dev, attr)`

- `Documentation/`
 - Anleitungen, Erklärungen und Beschreibung
 - ...insbesondere für die Konzepte des Linuxkerns
- `Documentation/output/`
 - Generierte Kernel-API Beschreibung (Sphinx)

```
$ cd <KERNEL_SOURCES>
```

```
$ make htmldocs
```

- Documentation/
 - Anleitungen, Erklärungen und Beschreibung
 - ...insbesondere für die Konzepte des Linuxkerns

- Documentation/output/
 - Generierte Kernel-API Beschreibung (Sphinx)

```
$ cd <KERNEL_SOURCES>
```

```
$ make htmldocs
```

- Allerdings: 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
- Im Zweifelsfall: Code von anderen als Referenz

[Documentation/output/kernel-hacking/index.html](#)

- Guter Einstieg in die Kernel-Entwicklung
- Übersicht über Besonderheiten des Kernelcodes

[Documentation/output/core-api/kernel-api.html](#)

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

`Documentation/filesystems/sysfs.txt`

- Interfacebeschreibung für `sysfs`
 - Erzeugung von Verzeichnissen im Dateisystem
 - Erzeugung von Attributen im Dateisystem

`Documentation/kobject.txt`

- Abstrakte Objektorientierung im Linuxkernel
- Konzept- und Interfacebeschreibung zu `kobjects`

Universal Serial Bus

Asymmetrischer Bus (Baum)

- Ein Host (PC) (Wurzelknoten) und viele Functions (angeschlossene Geräte, Blätter)
- Kommunikation wird ausschließlich vom Host initiiert
- Keine autonome Kommunikation von Geräte

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
- SuperSpeed+ (10 Gbit/s), USB 3.1 Gen 2
- SuperSpeed USB 20Gbps (20 Gbit/s), USB 3.2 Gen 2x2

Vier unterschiedliche Kommunikationsmechanismen:

- **Control Transfers:** Unregelmäßige Anfragen vom PC an das Gerät
z.B. Enumeration Sequence
- **Interrupt Transfers:** Periodische Kommunikation; begrenzte Antwortzeit
z.B. Maus, Tastatur
- **Bulk Transfers:** Aperiodisch; für große Pakete ohne zeitliche Garantien
z.B. USB-Storage-Device
- **Isochronous Transfers:** Periodische, kontinuierliche Datenströme
z.B. Webcam

Weiterführende Informationen

- USB-Spezifikation
 - Spezifikation für USB 2.0 unter `/proj/i4passt/doc`
- Beyond Logic
 - www.beyondlogic.org/usbnutshell/usb1.shtml

Linuxkern implementiert Host-Controller-Treiber (HCD)

- Grundlagen des USB-Protokolls
- Direkte Verwendung der USB-Transferarten möglich
- „Herstellerspezifische“ Treiber bauen darauf auf
- `<linux/usb.h>`

Linux Kommunikationsprimitiven für USB

- **USB Request Blocks (URB)**
- Endpoints
- Pipes
- Control Transfers

- USB-Geräte bieten Endpunkte (Endpoints) an
 - Art und Anzahl sind gerätespezifisch
 - ...mindestens jedoch Endpunkt 0
- Host spricht mit Gerät über Kanäle (Pipe)
 - Kanäle sind mit Endpunkten verbunden
- Control Transfers finden über Endpunkt 0 statt

Entwicklung eines USB-Gerätetreibers

1. (De-)Registrierung des Treibermoduls im System
2. Verwaltung von USB-Geräteinstanzen
3. Kommunikation mit dem USB-Gerät
 - Asynchrone USB-Transfers per URB
 - **Synchrone USB-Transfers für die Aufgabe ausreichend**
4. Bereitstellen von Userspaceschnittstellen

Doku zur Entwicklung von USB-Treibern

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

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
 - Auslesen der Temperatur- und Statusinformationen der Sensoren
 - Rescan der angeschlossenen Temperatursensoren
 - Reset des kompletten Gerätes

Control Transfers

- **Festverdrahtete** (Konfiguration etc.) und **gerätespezifische** Befehle
- Abwicklung über Endpunkt 0 (muss vorhanden sein)
- Bidirektional: Ein Transfer erlaubt Senden und Empfangen

Parameter für Control Transfers¹

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

¹vgl. USB 2.0 Spec. Abschnitt 9.3

Der **Request-Type** für die Befehle ist immer gleich²

- Datentransferrichtung ist vom Gerät zum PC
- Anfragen sind vendor-spezifisch
- Ziel der Anfrage ist das Gerät
request type 0xc0

²vgl. USB Spec. 248

Abfrage nach Status & Zahl der unterstützten Sensoren

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;
```

Zahl der Probes bleibt über Lebenszeit konstant!

Abfrage von Status/Temperatur aller unterstützter Sensoren

Aufrufparameter		Antwort
request	3	<pre>struct probe_status {</pre>
value	0	<pre> uint8_t serial[6];</pre>
index	0	<pre> uint8_t type;</pre>
		<pre> uint8_t flags;</pre>
		<pre> uint8_t temperature[2];</pre>
		<pre> uint32_t timestamp;</pre>
		<pre> uint8_t padding[2];</pre>
		<pre>}__packed;</pre>
		<pre>struct probe_status answer[supported_probes];</pre>

- Flags: 0x01: Sensor vorhanden. Sonst: Slot unbenutzt
- Mehrere Bytes umfassende Werte sind little-endian
- Temperatur: 12-bit 2er-Komplement, in sechzehntel Grad³

³Für ds18b20 mit type == 0x28

Fragt die angeschlossenen Sensoren erneut ab

Aufrufparameter		Antwort
request	2	<code>struct rescan_reply {</code>
value	0	<code>uint8_t answer;</code>
index	0	<code>};</code>

- Im Erfolgsfall zwei Antworten möglich
 - 23 Neuerkennung wird gestartet
 - 42 Neuerkennung wird schon durchgeführt

Fragt die angeschlossenen Sensoren erneut ab

Aufrufparameter		Antwort
request	2	<code>struct rescan_reply {</code>
value	0	<code>uint8_t answer;</code>
index	0	<code>};</code>

- Im Erfolgsfall zwei Antworten möglich
 - 23 Neuerkennung wird gestartet
 - 42 Neuerkennung wird schon durchgeführt
- Mit `value = 0x01` kann Fortschritt abgefragt werden
 - 23 Neuerkennung abgeschlossen
 - 42 Neuerkennung wird noch durchgeführt

Setzt den Temperatursensor zurück

Aufrufparameter

request	4
value	0
index	0

- Das Geräte sollte keine Antwort schicken

Setzt den Temperatursensor zurück

Aufrufparameter

request	4
value	0
index	0

- Das Geräte sollte keine Antwort schicken
- Bereitstellen eines Puffers schadet trotzdem nicht

Hinweise zu QEMU und USB-Geräten

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

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

Lösung: **udev**

```
ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="05dc", MODE="666"  
/etc/udev/rules.d/99-usbtemp.rules
```

Zusammenfassung

Gerätetreiber unter Linux

- Entwicklung als ladbare Module
- Zugriff auf Kernelobjekte per `sysfs`
- Verwendung von Kernel-API

Universal Serial Bus

- Eigentlich kein richtiger Bus
- Verschiedene Kommunikationsmöglichkeiten
- Abstraktion unter Linux als Endpoints & Pipes

Aufgabe 5

- Einarbeiten in die benötigten APIs im Linuxkern
 - Dokumentation
 - Codebeispiele
- Programmieren des Gerätetreibers
 - Ordentliche Speicherverwaltung
 - Linux Coding Style beachten
- Programmieren einer Userspace-Anwendung

Abgabe: Bis 11. Juli 2019 durch Vorführung in einer Rechnerübung

- sysfs-Einträge für USB-Temperatursensor
- Dateieintrag zur Temperaturabfrage für jeden Sensor

```
$ 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 USB-Temperatursensors

```
$ echo 1 > reset
```


- Temperaturen periodisch aus `/sys` auslesen
- 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`)

Fragen?