Kernel Module

Hans Buchmann FHNW/ISE

2. Juni 2020

Um was geht es ?

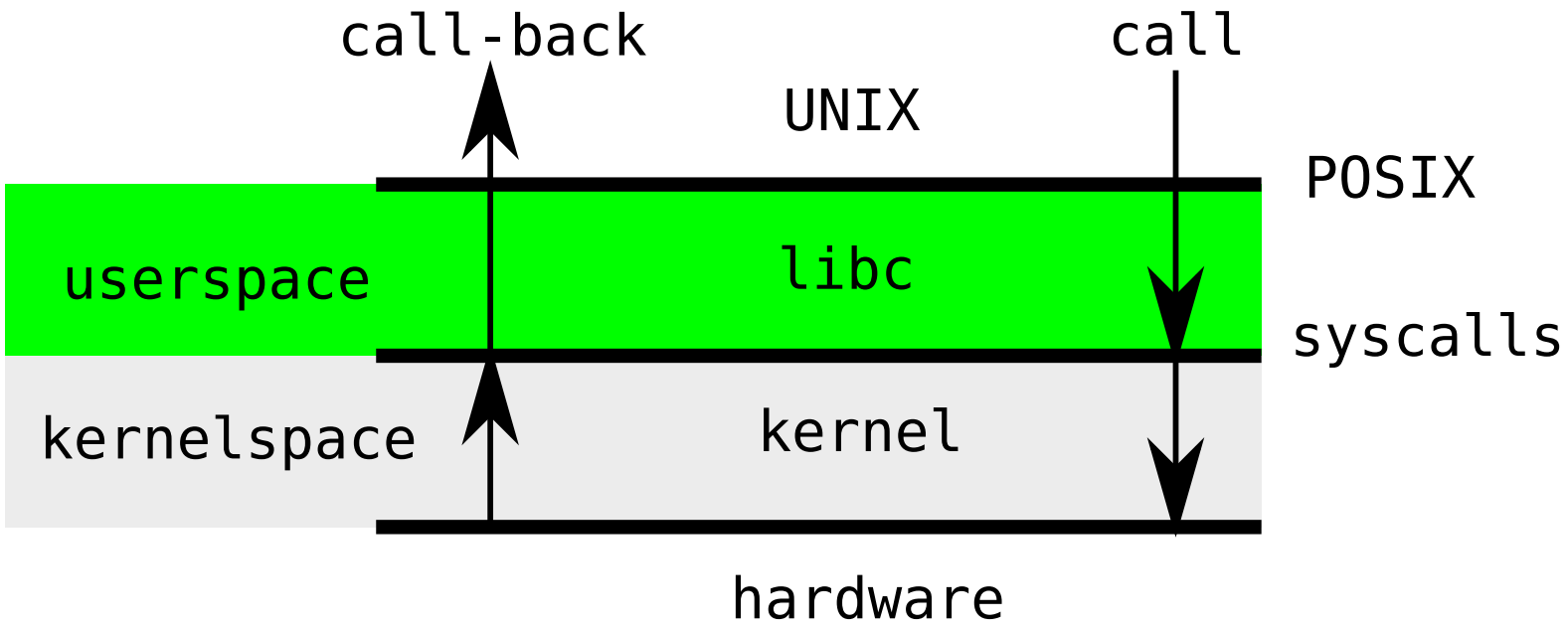
- Code für dem *kernel*: Drivers
- Den *kernel* nicht immer neu kompilieren
- Module laden/löschen

Informationen

- `tldp.org/LDP/lkmpg/2.6/html/`
- `Module`
- `www.kernel.org/doc/`
- `lxr.free-electrons.com`

urserspace vs. *kernelspace*

Systemcalls



userspace geschützt, limitierte Zugriffsmöglichkeiten

kernelspace ungeschützt, unlimitierte Zugriffsmöglichkeiten

Syscall aus *user* Sicht

`syscall-c.c`↑

```
/* write(f,void* buffer,unsigned len) */
char s[]="Hello World\n";
        /*01234567890 */
syscall(4,0,s,12);   /* we are in userspace */
    /*  |-------------------- code for write */
```

Aufgaben

- Systemcall für *Host*/**BBG** mit **C** und **C++**

Teil I

Input/Output

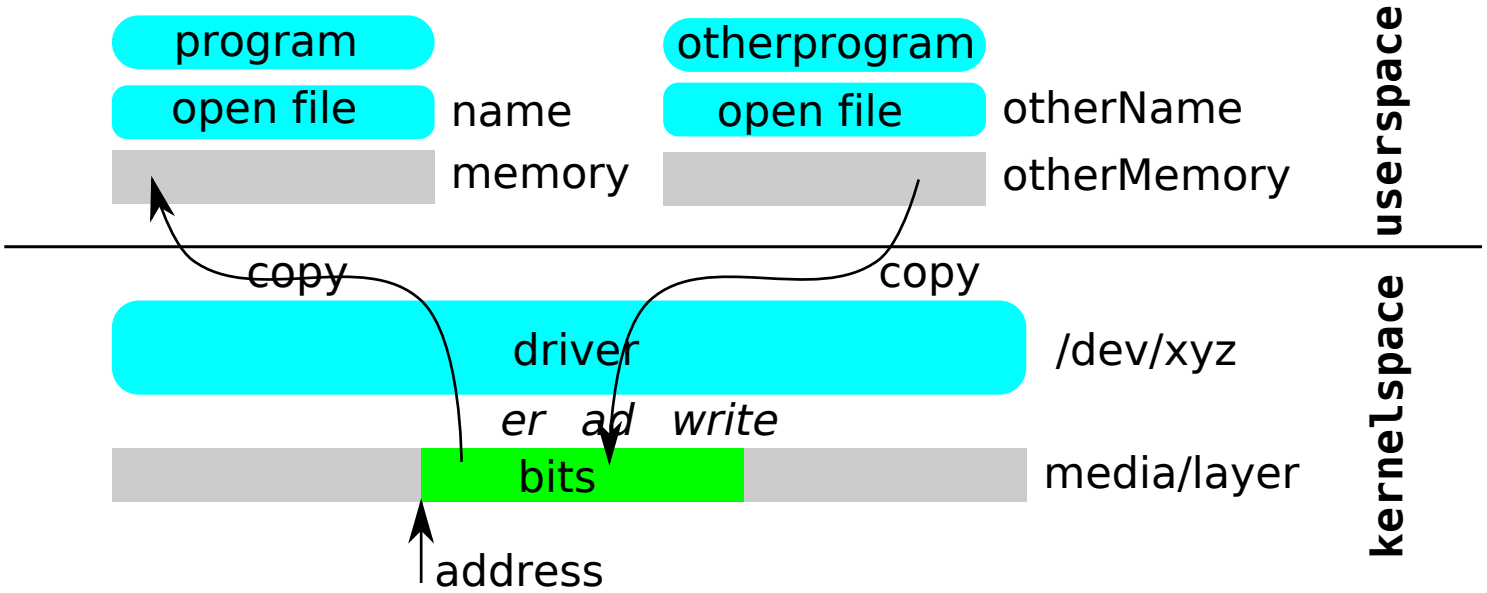
Ziele

im userspace

- `open`, `read`/`write`, `close`
- zwei *flavours*
  - low-level I/O
  - I/O on streams

Alles ist ein File

etwas genauer



Beispiele

- `low-level-io.c`
- `io-level-on-streams.c`

Aufgaben

- `low-level-io.c`/`io-level-on-streams.c` für *Host* und **BBG**

Teil II

Module

Modul für *Host*

Code `scr/*`

Script `tools/module.sh` für einfachen Aufruf

Test

- `dmesg -w`
- `sudo insmod simple-module.ko` wir sind in `src`
- `lsmod | grep simple` ist installiert
- `sudo rmmod simple-module` deinstalliert
- Der File `proc/modules`

Modul für **BBG**

plus Modul für *Host*

Host

```
.
├── doc
├── src
├── target
├── tc -> ../17-build/tc
└── tools
```

Target

```
.
├── bin
├── dev
├── etc
├── home
├── lib
├── libexec
├── linuxrc -> bin/busybox
├── made-2017.12.22
├── proc
├── sbin
├── share
├── sys
├── usr
└── var
```

- `src/Makefile` anpassen
- `cp src/`*`module`*` target/home/root`

Ziel

`simple-module.c`

- Herstellung
- install/deinstall
- elementare call-backs

simple-module.c↑

init/exit

```
module_init(simple_init); /* register   :called by kernel */
module_exit(simple_exit); /* deregister:called by kernel */
```

- call-back
- register/deregister
- `printk` wie `printf`

  ```
  printk(KERN_INFO "%d %x",val1,val2);
  ```

  für debug

Aufgaben

- `src/simple-module.c` für *Host*/**BBG**
- Machen Sie eine 'ewige Schlaufe'

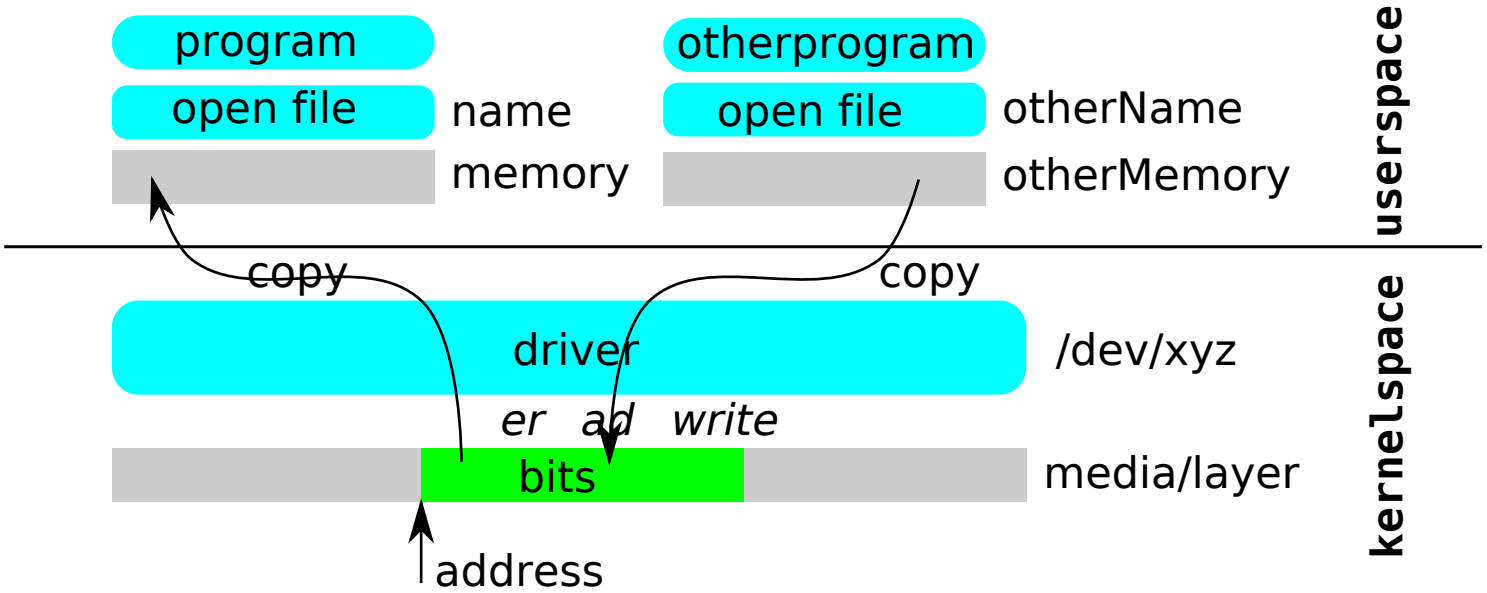
Ziel

`simple-device.c`↑

- Verbindung *userspace-userspace*
  - alles ist ein File
- *devicefile*
  - `mknod` *`device-file type`* `major minor`
- die elementaren Operationen

Alles ist ein File

etwas genauer



simple-device.c↑:im alles ist ein File

Die elementaren Operationen im *userspace*

- `open`
- `read`
- `write`
- `close`

Die elementaren Operationen im *userspace*

der Befehl `cat`

- `cat` **device**
  - `open,read,close`
- `cat file >` **device**
  - `open,write,close`

Der Devicefile `device`

- ist ein File
- bezeichnet ein *device*
- ist normalerweise im Verzeichnis `dev`
  - muss aber nicht

Beispiele

- `/dev/ttyUSB0` die serielle Schnittstelle
- `/dev/mmcblk0` die SD-Karte auf **BeagleBoneGreen**
- `/dev/random`, `/dev/urandom`
- ...

Die Verbindung *file - device*

Devicefile

Beispiel: `/dev/ttyUSB0`[1]

```
crw-rw---- 1 root uucp   188,  0 11. Nov  20:27 /dev/ttyUSB0
|            |    |      |     |                |
|            |    |      |     |                name
|            |    |      |     minor
|            |    |      major
|            |    group
|            owner
devicetyoe
```

für uns wichtig:

- **major** Code für die *device* Klasse
- **minor** Nummer für ein *device*

[1] gemacht mit `ls -l`

Major:Minor

objektorientierte Interpretation

major Code für die Klasse

major Code für die Instanz

Der Befehl `mknod`

erzeugt einen *Devicefile*

```
Usage: mknod [-m MODE] NAME TYPE MAJOR MINOR

Create a special file (block, character, or pipe)

        -m MODE Creation mode (default a=rw)
TYPE:
        b       Block device
        c or u  Character device
        p       Named pipe (MAJOR and MINOR are ignored)
```

register_chrdev

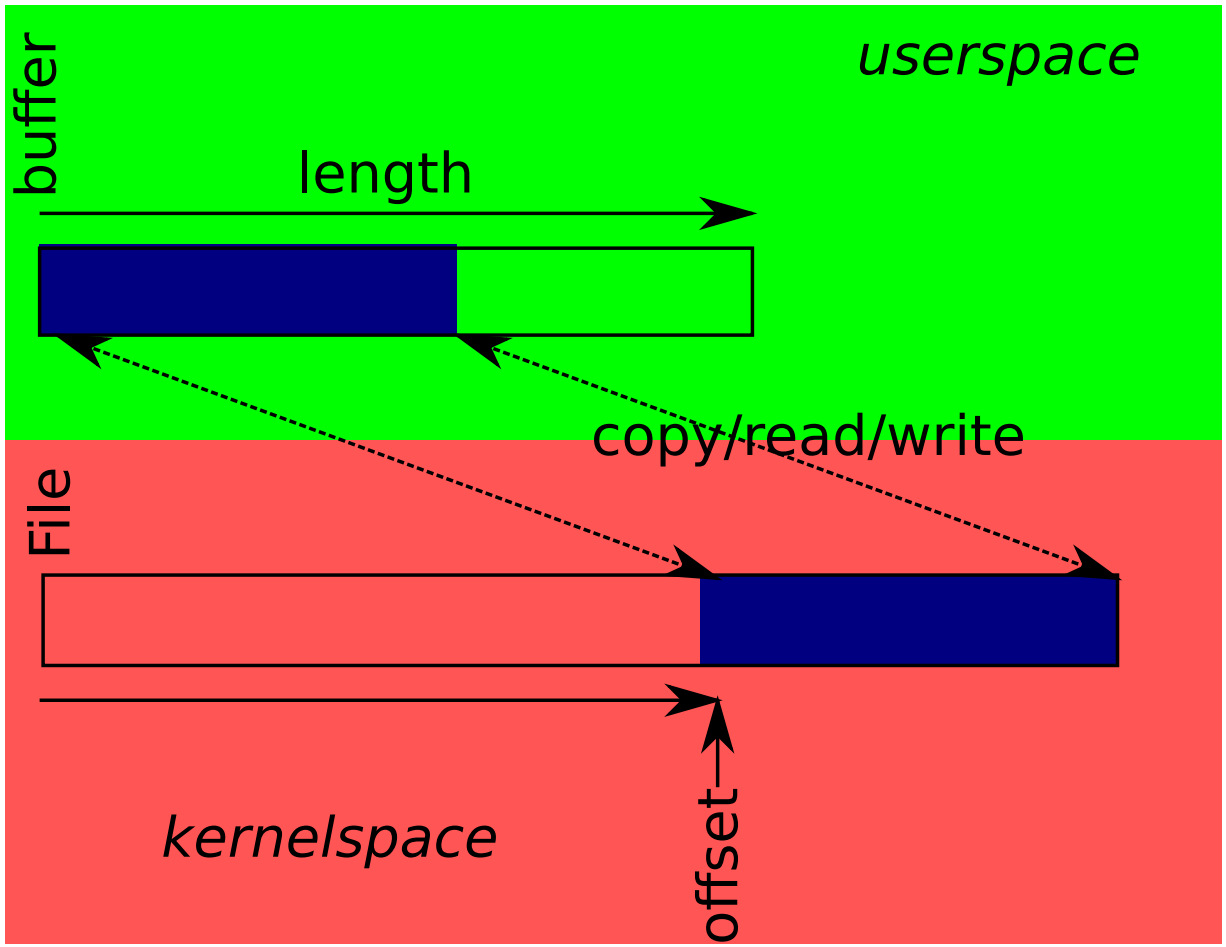
- erzeugt *major*
- `file_operations fops` die Fileoperationen
  - call-backs
  - siehe `linux/fs.h`

à la C++

```
class File
{
 protected:
  virtual int open(...)=0;
  virtual int flush(...)=0;
  virtual int read(...)=0;
  virtual int write(...)=0;
  ...
};
```

Transfer userspace ↔ kernelspace

`simple-read` und `simple-write`



Test

- `insmod simple-device.ko`
  - → *major*
- `mknod devi c` *major i*
  - Devicefile beliebige minor
- `cat devi`
  - lese von `devi`
- `echo "hello" > devi`
  - schreibe auf `devi`

Remark: `devi` in `/work`

Aufgaben

- `src/simple-device.c` für *Host*/**BBG**
- Registrierung/Deregistrierung in `sys/class`
  - `class_create`
  - `device_create`

Problem

- Funktioniert für *Host*
- Funktioniert **nicht** für **BBG**

ioctl

Die Kontrolle von Files/devices

- alles ist ein File
  - besser: ein *stream of bits*
- aber:
  - der *stream of bits* muss kontrolliert/konfiguriert werden

Beispiel:Serielle Schnittstelle

Baudrate

- `ioctl-c-example.c` **C** Programm
- `ioctl-cpp-example.cc` **C++** Programm

Typischer Ablauf

1. öffne File: *stream of bits* `fid`
   - `fid=open(name,mode)`
2. konfiguriere *stream of bits*
   - `ioctl(fid,`*`what`*`,`*`parameter`*`)`
3. lese/schreibe
   - `read(fid,data,length)`/`write(fid,data,length)`
4. schliesse:
   - `close(fid)`

Ziel

`simple-ioctl.c|h`↑ und `simple-ioctl-userspace.c`↑

- Einstellungen
- `ioctl(fileId,cmd,data)`

ioctl - control device

userspace

- `man 2 ioctl`

  The **ioctl()** *function manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with* **ioctl()** *requests. The argument d must be an open file descriptor*

- `int ioctl(int d, unsigned long request, ...);`

Im userspace

simple-ioctl-userspace.c↑

- die *requests* simple-ioctl.h↑

```
#define SIMPLE_IOCTL_WRITE _IOW(0x23,5,int)
#define SIMPLE_IOCTL_READ  _IOR(0x23,5,int)
```

- simple-ioctl-userspace.c↑
  - read

```
int val=1;
int res=ioctl(id,SIMPLE_IOCTL_READ,&val);
```

  - write

```
int res=ioctl(id,SIMPLE_IOCTL_WRITE,0x1234);
```

Test

siehe `simple-device.c`↑ Test

- `insmod`
- `mknod`
- `./simple-ioctl-userspace`

simple-hw.c

TODO:

Teil III

Aufgaben

Aufgaben

- `read-device.c scr/simple-device.c` für *Host*/**BBG**
- `call-ioctl.c scr/simple-ioctl.c` für *Host*/**BBG**