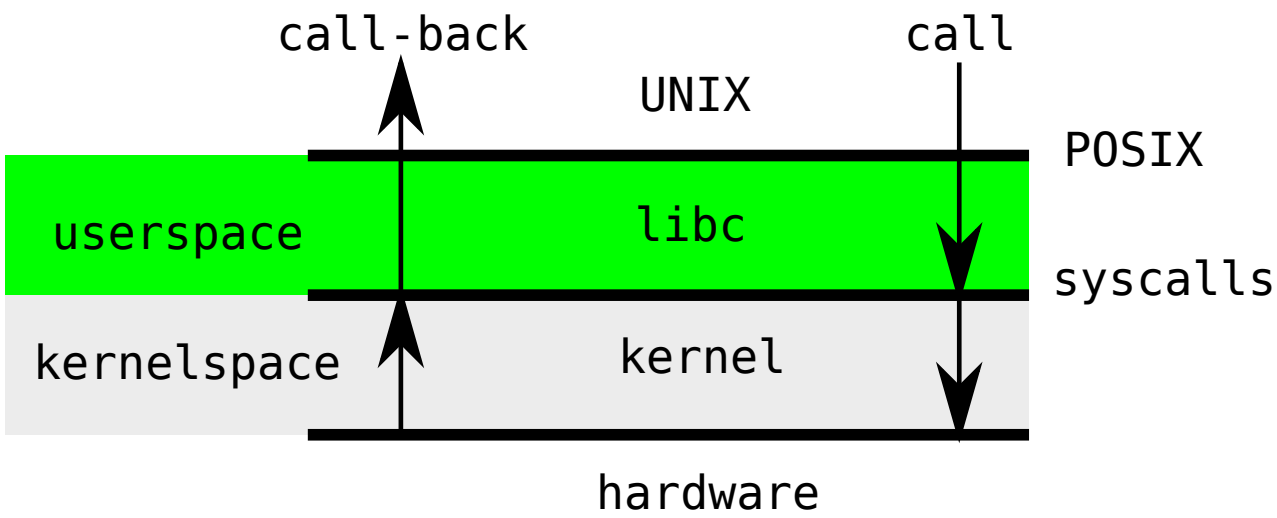
Um was geht es ?

- GNU/Linux *kernelmodule*'s
  - userspace
  - kernelspace: wo die *kernelmodule* sind
- Herstellung
- Interrupts

Userspace-Kernelspace

zwei verschiedene Welten



`syscall.c`

Beispiel userspace

- `syscall`
- der File `sys/syscall.h`

Verzeichnisstruktur

auf dem *Host*

```
17-build
├── modules ..................... source and generated files
│   └── Makefile ................ for the modules
└── scripts
    └── modules.sh .............. wrapper to make
```

Der Workflow

die Orte

Host	`17-build/modules`
RaspberryPi	`/work`
Verbindung	mit `scp` (secure copy)
Server	`sshd` **RaspberryPi**
Client	*Host*

Der Workflow

	Host	**RaspberryPi**
A	Edititiere *a-module.c*	
B	`sh scripts/module.sh` *a-module.ko*	
C	`scp a-module.ko roo@ip:/work`	
D		`insmod` *a-module.ko*
E		teste
F		`rmmod` *a-module.ko*
	$\rightarrow$A	

Die Schritte

simple-module.c	▶ `insmod`/`rmmod` ▶ debug mit `printk`
simple-device.c	▶ das Konzept *File*: alles ist ein File ▶ `read`, `write` & Co.
simple-ioctl.c|h	▶ Einstellungen mit `ioctl`
simple-hw.c	▶ Zugriff auf die Hardware

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
- `dmesg` für `printk`
- Probiere ewige Schlaufe

Ziel

simple-device.c

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

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 **RaspberryPi**
- `/dev/random`, `/dev/urandom`
- ...

Beispiele
/dev/random, /dev/urandom, /dev/sda

- `cat /dev/random | hexdump -C`
  - sammelt das Rauschen: langsam
- `dd if=/dev/sda count=1 | hexdump -C`
  - der MBR

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

minor 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

à la C++

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

Test

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

Remark: *device* in `/work`

Ziel

simple-ioctl.c|h

- 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 call-ioctl.c Test

- `insmod`
- `mknod`
- `./call-ioctl` *`device`*

Ziel

simple-hw.c

- iomem
  - mapping

ioremap/iounmap

- der `struct` GPIO
- iomem

```
gpio=(GPIO* __iomem)ioremap(GPIO_BASE,GPIO_SIZE);
                                /* reserve memory */
iounmap(gpio);
```

Test

siehe simple-device.c Test

- `insmod`
- `mknod`
- userspace

```
userspace-led theDevice on|off
```