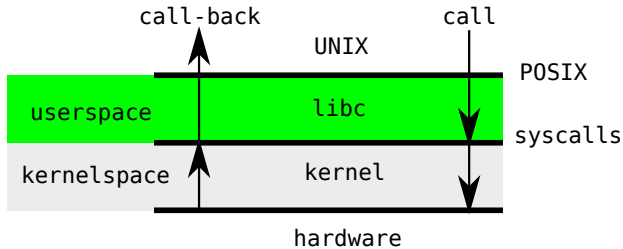


Um was geht es ?

- ▶ 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

	<i>Host</i>	RaspberryPi
A	Editiere <i>a-module.c</i>	
B	sh scripts/module.sh <i>a-module.ko</i>	
C	scp a-module.ko roo@ip:/work	
D		insmod <i>a-module.ko</i>
E		teste
F		rmmod <i>a-module.ko</i>
	→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-iocctl.c—h
 - ▶ Einstellungen mit ioctl
- simple-hw.c
 - ▶ Zugriff auf die Hardware
- simple-irq.c
 - ▶ interrupts
 - ▶ *foreground* der *handler*
 - ▶ *background*
 - ▶ *tasklet* die Verbindung *foreground*→*background*

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 Schleufe

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

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

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`
 - ▶ \rightarrow *major*
- ▶ `mknod devi c major i`
 - ▶ Devicefile beliebige minor
- ▶ `cat devi`
 - ▶ lese von `devi`
- ▶ `echo "hello" > devi`
 - ▶ schreibe auf `devi`

Remark: `devi` 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 FPGA

- ▶ iomem

```
fpga=(FPGA* __iomem) ioremap(FPGA_BASE, FPGA_SIZE);  
                        /* reserve memory */  
iounmap(fpga);
```


Test

siehe simple-device.c Test

- ▶ insmod
- ▶ mknod
- ▶ read: `cat device`
- ▶ write: `cat file > device`