Devices revisited

Hans Buchmann FHNW/IME

20. März 2018

Um was geht es ?

C und Scripts

- Verbindung
  - *kernel-space* $\leftrightarrow$ *user-space*
- call-backs
  - verschiedene Formen
- Info `lxr.free-electrons.com`
- Schritt für Schritt mit `git`
  - `git log`
  - `git show`

Um was geht es ?

- Kernel-Modules: **L**oadable **K**ernel **M**odule
  - `insmod`
  - `rmmodule`
- device
  - *major minor*
  - `devicefile` = *major minor*
- *kernel-space* $\leftrightarrow$ **devicefile** $\leftrightarrow$ *user-space*

Um was geht es ?

kernel-space ↔ user-space

Wie merkt der *kernel*-space

- ob ein **LKM**
  - eingefügt
  - entfernt

  wird

Wie merkt der *user*-space

- ob sich in einem **LKM**
  - etwas tut

  wird

Setup
Programmentwicklung

Host	**BBB**
start	
`minicom -D/dev/ttyUSB0`	
	start
	`ifconfig usb0 192.168.7.7`
	`/sbin/sshd`
`ssh root@192.168.7.7`	
`sshfs root@192.168.7.7: mount`	

ready to develop

simple-device-*.c
Schritt für Schritt

- simple-device-1.c:
  - debug mit printk
  - call-back: init/exit
  - struct file_operations
- simple-device-2.c:
  - call-back: read/write fast ohne code
  - device File: Verbindung
- simple-device-3.c:
  - das Zusammenspiel der Parameter: len und ofs
- simple-device-4.c
  - Verbindung mit *user-space*, das Verzeichnis /sys/class
- simple-device-5.c
  - Verbindung mit *user-space*, das Verzeichnis /sys/class/simple-device/*

simple-device-1.c

- init/exit
- `struct file_operations fops;`
- include file: `linux/fs.h` im kernel code

Code

linux/fs.h

```
struct file_operations {
/* ... */
 ssize_t (*read) (struct file *,
                  char __user *,
                  size_t ,
                  loff_t *);
 ssize_t (*write) (struct file *,
                   const char __user *,
                   size_t ,
                   loff_t *);
 /*... */
} __randomize_layout;
```

Remark: Was ist `__randomize_layout` ?

Code

```
/* the call backs defined and **initialized** */
static struct file_operations fops =
{
 /* do nothing for the moment*/
};

/* init */
 Major = register_chrdev(0, DEVICE, &fops);
/* exit */
 unregister_chrdev(Major,DEVICE);
```

simple-device-2.c
empty read/write

- *kernel-space*
  - `file_operations`: read/write
    - mit `printk` call-back anzeigen
  - verschiedene `return` Werte für read/write
- *user-space*
  - `mknod device c` *`Major`* `0`
  - read: `cat device`
  - write wenig Bytes `echo abcd > device`
  - viel Bytes `cat file > device`

Code

```
printk("simple_read len=%d= *ofs= %lld buffer*=0x%p\n",
                                len,      *ofs,          buffer);
```

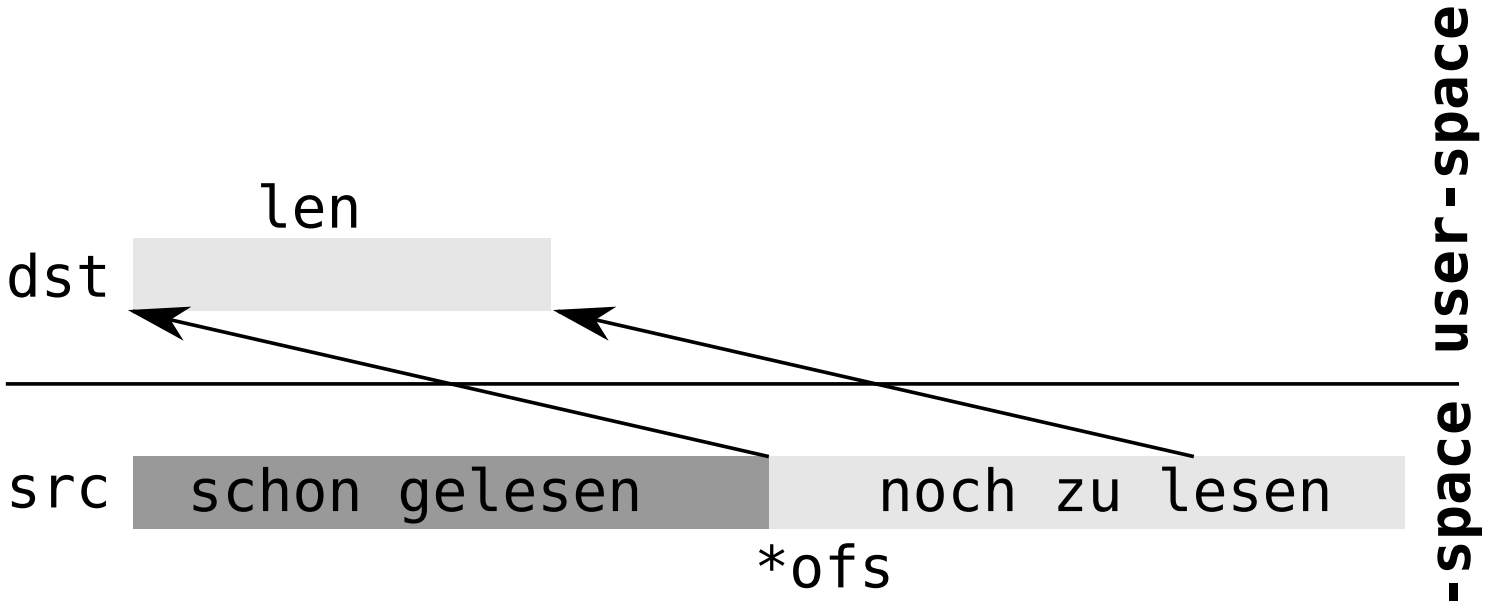
Remark: wie `printf` im *user-space*

```
static struct file_operations fops =  /* the call backs */
{
 read :simple_read, /* register call-backs */
 write:simple_write,
};
```

simple-device-3.c
read/write

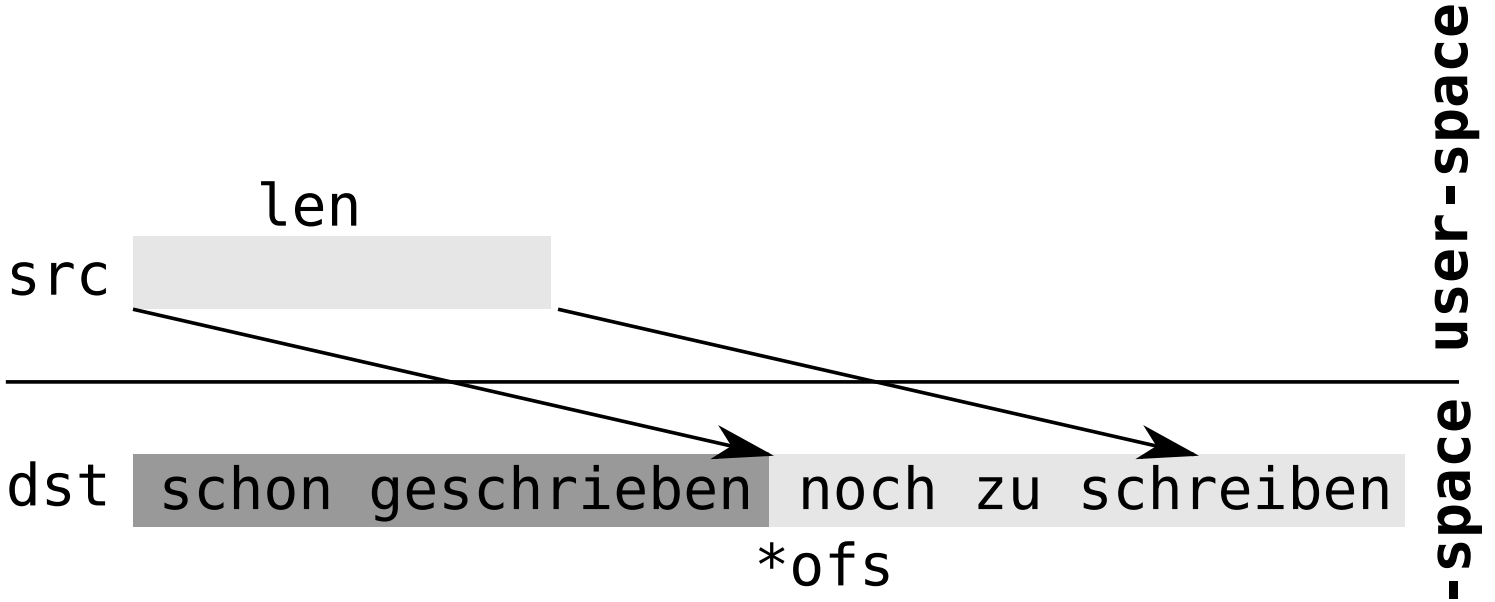
- read/write:
  - `copy_to_user`/`copy_from_user`
  - das Zusammenspiel:
    - `len` und `*ofs`

read




```
unsigned long copy_to_user(void __user *to,
                           const void *from,
                           unsigned long n)
```

write




```
unsigned long copy_from_user(void *to,
                             const void __user *from,
                             unsigned long n)
```

simple-device-4.c
Verbindung mit *userspace*: /sys

- *kernel-space*
  - MODULE_LICENSE ("GPL")
  - init: simple_module=class_create(THIS_MODULE,"simple_device")
  - exit: class_destroy(simple_class)
- *user-space*
  - ls /sys/class
- noch keine Informationen in /sys/class/simple_device

simple-device-5.c
Verbindung mit *userspace*: /sys

- *kernel-space*
  - init: device_create, MKDEV(Major,0)
  - exit: device_destroy
- *user-space*
  - ls /sys/class/simple_device

Code

```
struct device *device_create(struct class *cls,
                             struct device *parent,
                             dev_t devt,
                             void *drvdata,
                             const char *fmt, ...);

void device_destroy(struct class *cls,
                    dev_t devt);
```

Die Files

`/sys/class/simple_device/simple_device0/`

- der File
  - `uevent`

Die Reihenfolge

Beim `init`

```
simple_class=class_create(...);
Major        = register_chrdev(...);
dev          =device_create(...);
```

Beim `exit`

- Umgekehrt

Notification

call-back: *kernel-space* → *user-space*

- Kernel: *kernel-space* entdeckt ein neues Gerät
- Informiert (notify) User *user-space*: neues Gerät
- User: *user-space* zieht die Konsequenzen

Beispiel

udevadm monitor

- udevadm monitor -k

```
KERNEL[4377.896571] add  /devices/pci0000:00/0000:00:14.0/usb3/3-4 (usb)
KERNEL[4377.896887] add  /devices/pci0000:00/0000:00:14.0/usb3/3-4/3-4:1.0 (usb)
KERNEL[4377.899345] add  /devices/pci0000:00/0000:00:14.0/usb3/3-4/3-4:1.0/ttyUSB0 (usb-serial)
KERNEL[4377.899434] add  /devices/pci0000:00/0000:00:14.0/usb3/3-4/3-4:1.0/ttyUSB0/tty/ttyUSB0 (tty
KERNEL[4377.899470] bind /devices/pci0000:00/0000:00:14.0/usb3/3-4/3-4:1.0/ttyUSB0 (usb-serial)
KERNEL[4377.899513] bind /devices/pci0000:00/0000:00:14.0/usb3/3-4/3-4:1.0 (usb)
KERNEL[4377.899584] bind /devices/pci0000:00/0000:00:14.0/usb3/3-4 (usb)
```

- udevadm monitor -kp
  - mehr Informationen

Unser Ziel
die Basismechanismen

- hotplug
- socket `NETLINK_KOBJECT_UEVENT`

Hotplug
/proc/sys/kernel/hotplug

- Kernel: *kernel-space* entdeckt ein neues Gerät
- Informiert (notify) User *user-space*: neues Gerät
  - call-back: `/proc/sys/kernel/hotplug`
    - enthält *user-space* executable

Remark: Kernel muss für *hotplug* konfiguriert sein

Beispiel

ein Skript im *user-space*

```
#!/bin/ash
#-------------------------
#my-hotplug.sh
#(c) H.Buchmann FHNW 2018
#-------------------------
echo "{------ my-hotplug -----${@}" >> /home/root/my-hotplug.log
env >> /home/root/my-hotplug.log
echo "------ my-hotplug -----}" >> /home/root/my-hotplug.log
```

- `chmod a+x my-hotplug.sh` :executable
- `echo /home/root/my-hotplug.sh > /proc/sys/kernel/hotplug`

Test

- shell1:
  - `tail -f //home/root/my-hotplug.log`
- shell2:
  - `insmod simple-device-5.ko`
  - `rmmod simple-device-5.ko`

Socket

Endpunkt: Buchse einer Verbindung

- empfängt/sendet Bytes
- verschiedene Typen
  - Internet
  - Verbindung vom *kernel*
  - `netstat`
- Alles ist ein File
- Alles ist ein Socket

Socket

- Kernel: *kernel-space* entdeckt ein neues Gerät
- sendet Daten an einen Socket:
- User:*user-space* liest die Daten

Beispiel

`uevent-userspace.c`

- **C**
- funktioniert auch auf dem *Host*
- Schöne Ausgabe
  - `./uevent-userspace | ./split`