Devices revisited

Hans Buchmann FHNW/IME

19. 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*↔ **devicefile** ↔ *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-3.c
  - Verbindung mit *user-space*, das Verzeichnis /sys

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);
```

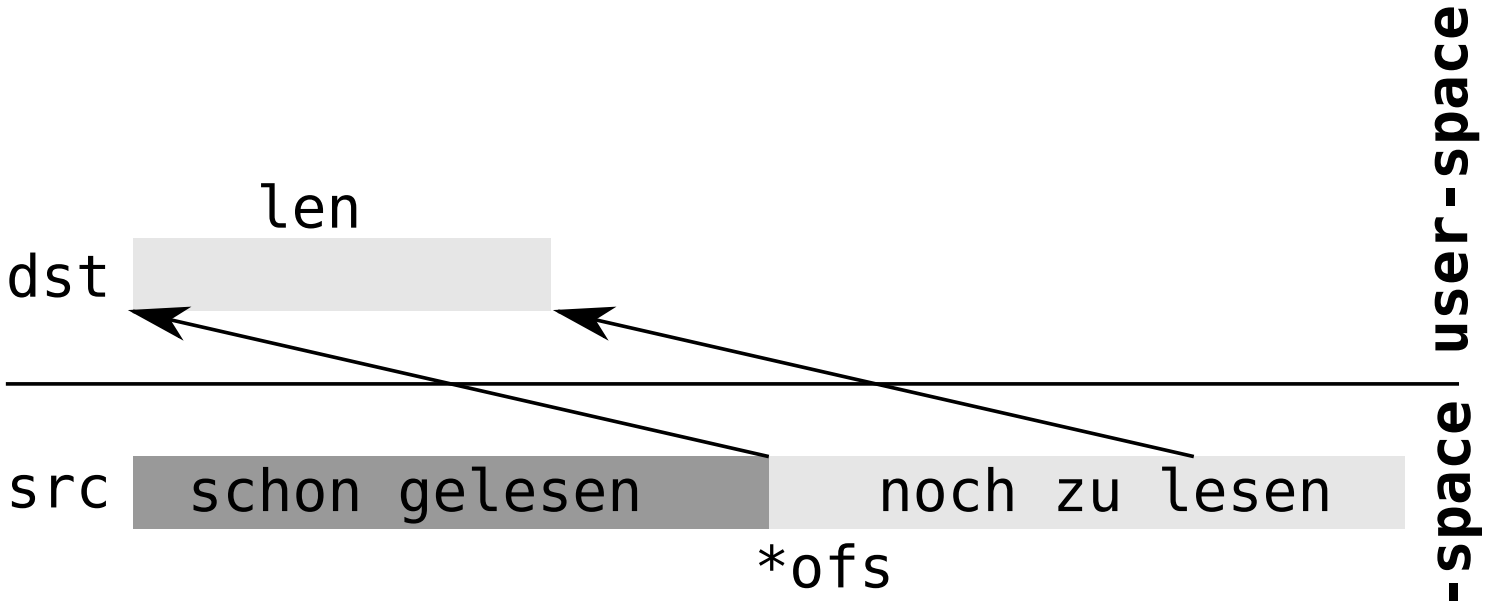
Remark: wie `printf` im *user-space*

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

simple-device-3.c
read/write

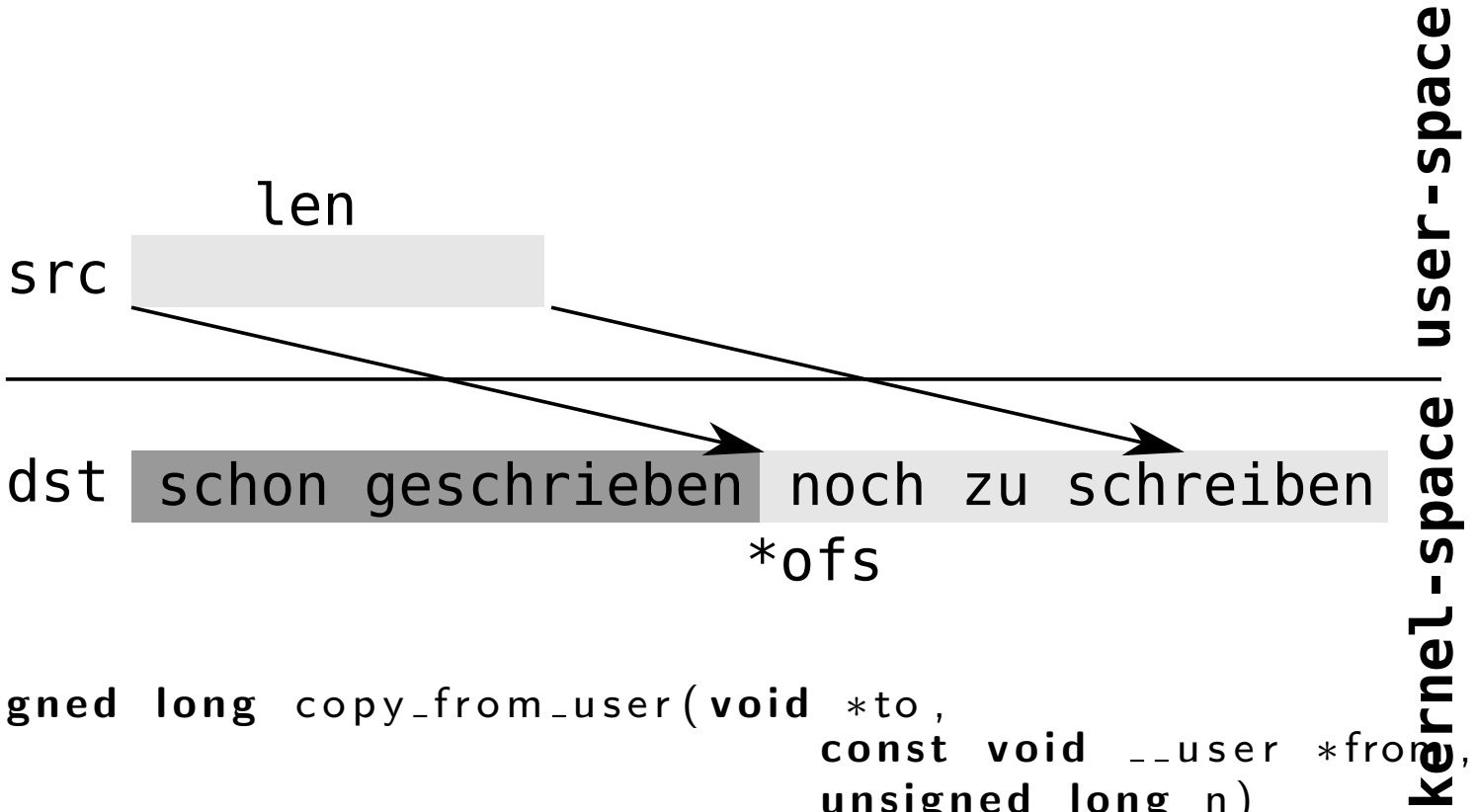
- read/write:
  - `copy_to_user`/`copy_from_user`
  - das Zusammenspiel:
    - `len` und `*ofs`

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,KDEV(Major,0)`
  - exit: `device_destroy`
- *user-space*
  - `ls /sys/class/simple_device`

Die Reihenfolge

Beim `init`

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

Beim `exit`

- Umgekehrt

hotplug