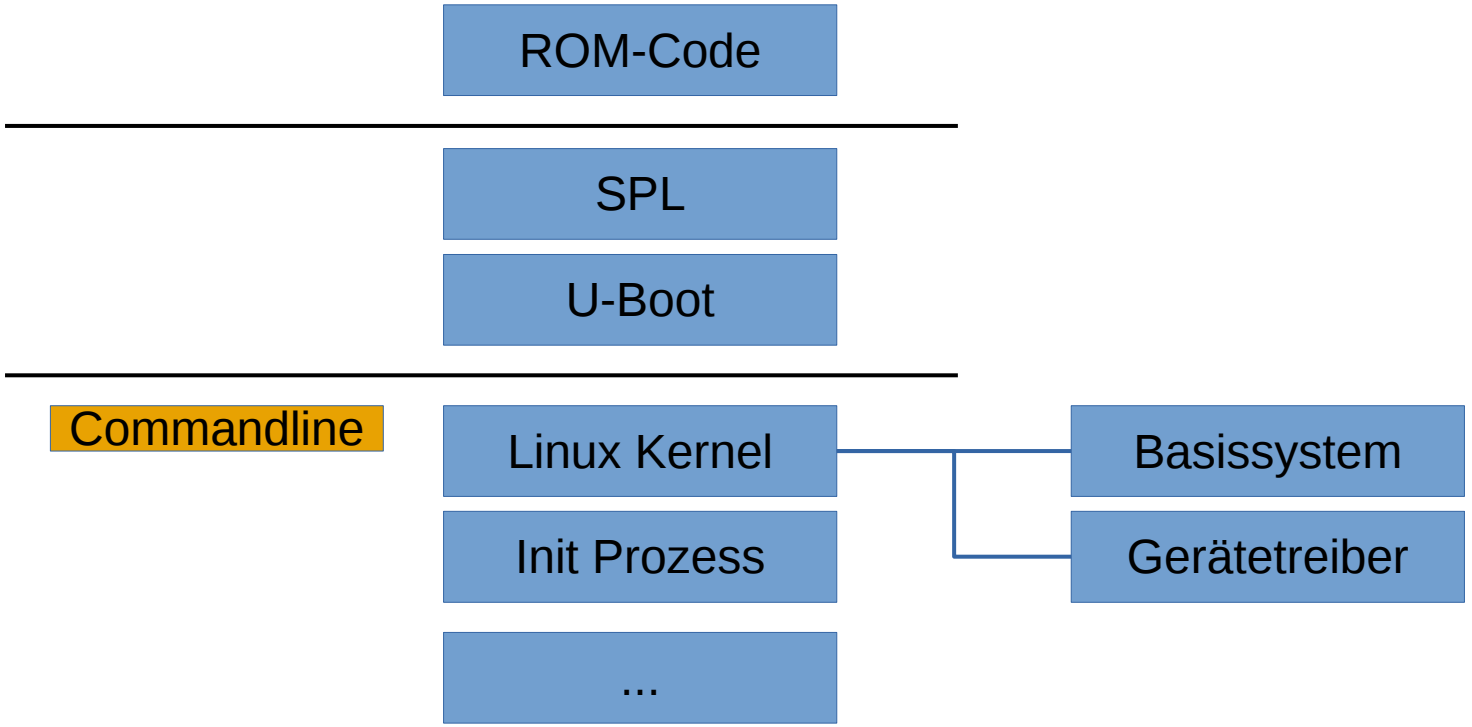
Linux Treiber 2

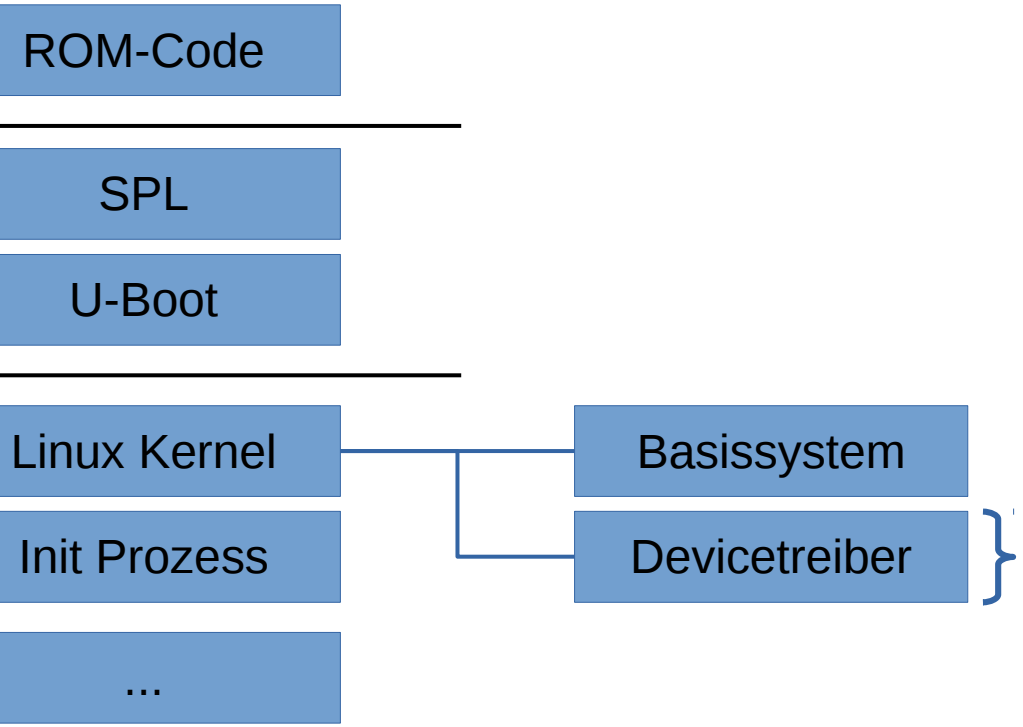
EIM/INFM

Frank Erdrich
frank.erdrich@emtrion.de

Aktueller Stand



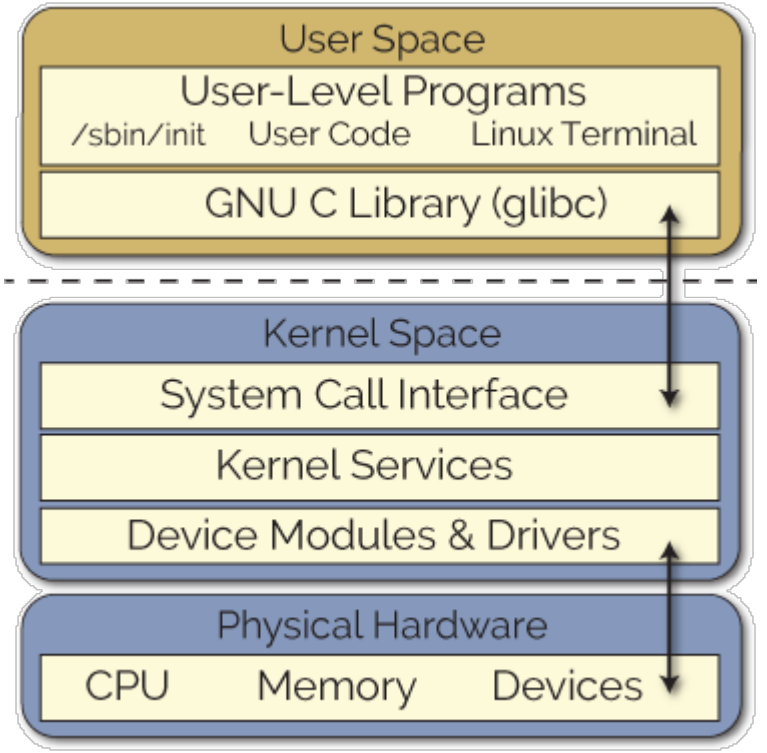
Aktueller Stand



- Schnittstelle zur Hardware
- laufen im Kernelspace
- Char- und Blocktreiber (/dev/mydriver)
- Information im SysFS (Steuerung/Daten)
- Erweiterte Steuerung über IOCTL
- kthreads, kmalloc

Gerätetreiber

- Interface zur Hardware
- Treibertypen
  - Character
  - Block
  - Other (bspw. Netzwerk und USB)



Quelle: http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction

System Calls

- Was sind System Calls?

System Calls

- „Services“ des Kernel
  - File-IO, Fork, ioctl, exit, ...
- Switch vom Userspace in den Kernelspace
- Definierte Menge an Calls
  - Liste aller aktuellen Syscalls unter
    http://man7.org/linux/man-pages/man2/syscalls.2.html
- Werden selten erweitert

System Calls

- Architekturspezifisch in Assembler
- Trap mit Syscall-Nummer wird generiert
  - Auf ARM wird ein SWI generiert
- Definierte Register halten Parameter
- Siehe https://w3challs.com/syscalls/?arch=arm_strong

System Calls

- Sichert diverse Register
- Switch von Userspace Stack auf Kernel Stack
- Dispatcher führt gewünschte Funktion aus

- Kernel muss Parameter prüfen (Security)
  - Speziell bei Pointern

System Calls

- copy_to_user()
- copy_from_user()
- Kopieren Speicher von Userspace in Kernelspace und vice versa
- Prüfen dabei, ob die Adressen im richtigen Bereich liegen
- Vermeiden Kernelcrashes durch falsche Speicherzugriffe

System Calls

- Beispiel open() -> Syscall #5
  (siehe Linux-Quellen
  `arch/arm/include/uapi/asm/unistd.h`)

```
open(const char *filename, int flags, umode_t mode)
```

R0	R1	R2	...	R7
c. char *filename	int flags	umode_t mode	...	0x900005

System Calls

- Was, wenn ein Treiber einen neuen/eigenen System Call benötigt?
- Beispiele:
  - I²C → setzen der Slave Adresse
  - Filesystem → Informationen zur Partition
  - Serielle Schnittstelle → termios
  - ...

IO Controls

- Erweitern die System Calls
- Implementiert in Treibern
- Sollten nicht bestehende Syscalls nachimplementieren (etwa File-IO)
- Beispiel termios:
  - IOCTL: setzen der Baudrate, Start-/Stopbits
  - Syscall: read/write über /dev/tty*

IO Controls

- Erweitern die System Calls
- Implementiert in Treibern
- Sollten nicht bestehende Syscalls nachimplementieren (etwa File-IO)
- Beispiel termios:
  - IOCTL: setzen der Baudrate, Start-/Stopbits
  - Syscall: read/write über /dev/tty*

IO Controls

- ioctl selbst ist ein System Call
  - ARM: Nummer 0x900036
- Jede Treiberklasse hat ein ioctl-Feld
  - Char- und Blockdevicetreiber:
    `struct file_operations → unlocked_ioctl`
  - Netdevices:
    `struct net_device_ops → ndo_do_ioctl`

IO Controls

- Definieren eines IOCTL → Macros
  - `_IO(type, nr)` → Parameterlos
  - `_IOR(type, nr, datatype)` → copy_from_user
  - `_IOW(type, nr, datatype)` → copy_to_user
  - `_IORW(type, nr, datatype)` → Read/Write

IO Controls

- „type“ der IOCTLs muss unique im gesamten Kernel sein
- Siehe Documentation/ioctl/ioctl-number.txt
- nr nummeriert alle ioctls im Treiber durch

```
#include <linux/ioctl.h>

#define MYDRIVER_MAGIC   ‚k‘

#define MYDRIVER_IOCWRITEPROT _IO(MYDRIVER_MAGIC, 0)
#define MYDRIVER_IOCWRITEUNPROT _IO(MYDRIVER_MAGIC, 1)
#define MYDRIVER_IOCGETLENGTH _IOR(MYDRIVER_MAGIC, 2, int)
#define MYDRIVER_IOCWRITETEST _IOW(MYDRIVER_MAGIC, 3, int)
```

IO Controls

- IOCTL-Funktion

```
static int sample_ioctl(struct inode *i, struct file *f, unsigned int cmd,
unsigned long arg) {
    int data;

    switch (cmd) {
        case MYDRIVER_IOCGETLENGTH:
            copy_to_user((int *) arg, &data, sizeof(int));
        break;
        case MYDRIVER_IOCWRITETEST:
            copy_from_user(&data, (int *)arg, sizeof(int));
            pr_info(„Data written to driver: %d\n“, data);
        default:
        break;
    }
}
```

IO Controls

- Registrieren der IOCTLs

```
static struct file_operations fops = {
    .read = sample_read,
    .write = sample_write,
    .open = sample_open,
    .release = sample_release,
    .unlocked_ioctl = sample_ioctl,
};
```

SYSFS

- Export von Informationen aus Treibern
- Konfiguration von Treibern
- Basis ist das `struct kobject`
  - sowie: `struct device_attribute`
  - und: `struct attribute_group`

SYSFS

- `struct kobject` wird in der Regel vom Kernel angelegt
- Treiberentwickler muss sich nur um die Attribute und die Attributgruppen kümmern
- Attribut beschreibt einen Eintrag (eine Datei) im SysFS
  - Besteht aus Schreib- und Lesefunktion
  - Zugriffsrechte
  - Name

SYSFS

- Anlegen eines Attributs
  - Helper-Macro in linux/device.h
  - Generiert ein struct dev_attr_name

```
#include <linux/sysfs.h>
#include <linux/device.h>

// #define DEVICE_ATTR(_name, _mode, _show, _store)
static DEVICE_ATTR(attr1, 0444, my_show, my_store)   // dev_attr_attr1
static DEVICE_ATTR(attr2, 0444, my_show2, my_store2) // dev_attr_attr2
```

SYSFS

- Show- und Store-Funktionen
  - Werden aufgerufen, wenn auf dem Attribut gelesen oder geschrieben wird (cat/echo)

```
/* Prototypen */
ssize_t (*show)(struct device *dev, struct device_attribute *attr,
                char *buf);

ssize_t (*store)(struct device *dev, struct device_attribute *attr,
                 const char *buf, size_t count);
```

SYSFS

- Attributgruppe fast mehrere Attribute zusammen

```
static DEVICE_ATTR(attr1, 0444, my_show, my_store)   // dev_attr_attr1
static DEVICE_ATTR(attr2, 0444, my_show2, my_store2) // dev_attr_attr2

static struct attribute *my_attributes[] = {
    &attr1.attr,
    &attr2.attr,
    NULL,
};

static struct attribute_group my_group = {
    .name = „mydriver“,
    .attrs = my_attributes,
};
```

SYSFS

- Registrieren der Attribute
  - Entweder über spezielle Funktion
  - Oder: zuweisen der Gruppe an Member eines struct class oder struct device

```
// static struct attribute_group my_group

sysfs_create_group(&sampleDevice->kobj, &my_group);

// Alternativ:
sampleClass->dev_attrs = my_group;

Siehe hierzu
http://kroah.com/log/blog/2013/06/26/how-to-create-a-sysfs-file-correctly/
```

SYSFS

- Aufräumen bei Modul-Exit

```
// static struct attribute_group my_group

sysfs_remove_group(&sampleDevice->kobj, &my_group);
```

Linux Treiber

- Treiber in Linux sind Eventgetrieben
- Wie können zyklische oder dauerhafte Aufgaben abgebildet werden?

KThreads

- Verwendung von Kthreads
- KThreads sind Kernel-Threads
- Ähnlich den pthreads im Userspace

KThreads

- Verwendung:

```
#include <linux/kthread.h>

static struct task_struct *task;

static int task_fn(void *arg) {
    while (!kthread_should_stop()) {
        // do work
        schedule();
    }
}

static int __init sampleInit(void) {
    task = kthread_run(task_fn, NULL, „My Task“);
}

static void __exit sampleExit(void) {
    kthread_stop(task);
}
```