Linux Driver Workshop

An introduction to Linux Driver Development

Johannes Roith

05.10.2025

About me

- Embedded Software Developer
- Embedded Linux YouTube Channel
- My website with links to my GitHub, Mastadon, LinkedIn, ...
- One driver of mine made it into the Linux Kernel

Agenda

1. The Linux Kernel
2. The I2C bus
3. A Linux I2C driver
4. Makefile for compiling the I2C driver
5. Module verwalten in einer Shell
6. Adding I2C devices over sysfs
7. PCF8574 IO Expander
8. Accessing the I2C bus
9. Creation of sysfs entries

Backup

10 Linux Kernel programming on a Raspberry Pi

11 A true Hello World Kernel Module

12 The macro module_i2c_driver

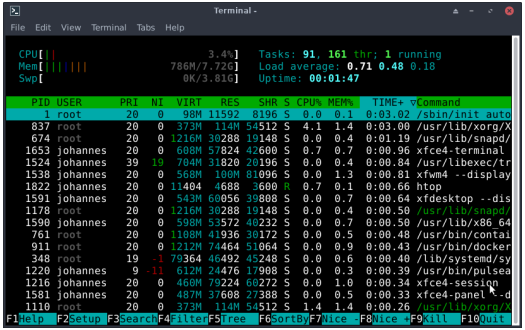
Material for the workshop

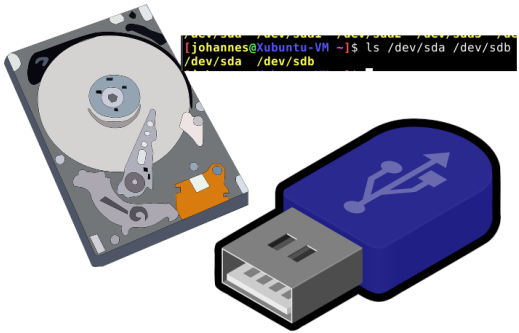
https://github.com/Johannes4Linux/linuxdays.cz25

The Linux Kernel

- Kernel of an operating system: hardware abstraction layer
- Uniform interface (API Systemcalls) independent from PC architecture
- Tasks of the Linux-Kernels:
  - Memory management
  - Process management
  - Multitasking
  - Load balancing
  - Access to hardware over drivers
- Applications are using systemcalls (open, close, read, write, ioctl, ...): they don't need knowledge about the underlying hardware
- Linux: modular monolithic Kernel with loadable modules

The Linux Kernel

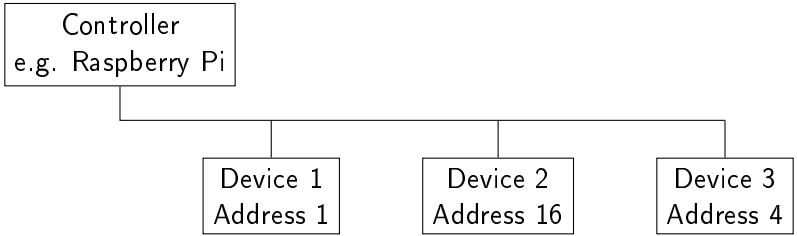




The I2C bus

- Simple two wire bus
- Data line: *SDA*
- Clock line: *SCK*
- Supported frequencies: 100kbit/s, 400kbit/s, 1Mbit/s
- Pull-Up resistor on both signals necessary

The I2C bus


Controller
e.g. Raspberry Pi
Device 1
Address 1
Device 2
Address 16
Device 3
Address 4

A Linux I2C driver

Header and compatible devices

```
/* Required Header */
#include <linux/i2c.h>

/* Name all compatible devices*/
static struct i2c_device_id my_ids[] = {
        {"my_dev"},
        {} /* Empty element signalizes end of list */
};
MODULE_DEVICE_TABLE(i2c, my_ids);
```

A Linux I2C driver

Probe- and Remove functions

```
/* function is called, when a compatible I2C device is added to the system
    */
static int my_probe(struct i2c_client *client)
{
        printk("Hello says I2C client with address: 0x%x\n", client->addr);
        return 0;
}

/* function is called, when a compatible I2C device is removed from the
    system */
static void my_remove(struct i2c_client *client)
{
        printk("Bye, bye, I2C\n");
}
```

A Linux I2C driver

Bundle driver struct

```
/* Bundle compatbile devices, probe and remove functions and driver info
    into driver struct */
static struct i2c_driver my_driver = {
        .probe = my_probe,
        .remove = my_remove,
        .id_table = my_ids,
        .driver = {
                .name = "my-i2c-driver",
        }
};
/* Register driver at the OS */
module_i2c_driver(my_driver);
/* Information about the driver*/
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Johannes Roith");
MODULE_DESCRIPTION("A Hello World I2C driver");
```

Makefile for compiling the I2C driver

```
# Kernel Header Makefile compiles rgb_brd.c to rgb_brd.o file automatically
obj-m += rgb_brd.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Module verwalten in einer Shell

- `lsmod` lists all loaded modules
- `dmesg` shows the kernel's log
- `insmod rgb_brd.ko` load the module `rgb_brd.ko` into the kernel
- `rmmod rgb_brd` removes the module <`rgb_brd` from the kernel
- `modinfo ./rgb_brd.ko` shows the meta-data (author, licence, description, ...) of the module `rgb_brd.ko`
- `modprobe uio` loads the module `uio` together with all its dependencies

Adding I2C devices over sysfs

```
# Change to I2C device folder
cd /sys/bus/i2c/devices/i2c-1

# Add I2C device mydev with address 0x12
echo "my_dev 0x12" | sudo tee new_device

# Removes I2C device with address 0x12
echo "0x12" | sudo tee delete_device
```

Exercise

- Implement the kernel module `rgb_brd` on the Raspberry Pi in the folder `ex_1` as follows:
  - The name of the compatible device should be `rgb_brd`
  - The probe function should print out the I2C address of the device into the kernel's log
  - Another kernel's log message should be written to the log when removing the device
- Compile the kernel with the Makefile
- Load the module
- Check that the module is loaded
- Add a compatible I2C device
- Check the Kernel's log
- Remove the module

PCF8574 IO Expander

- Write access writes output values P0 - P7
- Read access reads current values P0 - P7
- Button connected to P0
- For input operation: Set Output to 1, Button pulls pin to GND. When reading a 1, the button is not pushed, when reading a 0 it is pushed
- Red LED connected to P1, green to P2, blue to P3
- Set output to 0: LED is On
- Set output to 1: LED is Off

Bit:	0	1	2	3	4	5	6	7
Value for:	P0	P1	P2	P3	P4	P5	P6	P7

Accessing the I2C bus

```
struct i2c_client *my_client;
```

The `struct i2c_client` is used, to manage an I2C device in the kernel. With the pointer `my_client` we can access the device, e.g. for reading or writing data.

```
s32 i2c_smbus_read_byte(struct i2c_client *my_client);
```

Reads a byte from the I2C device `my_client`. If an error occurs, the function returns a negative error code, else the read byte.

```
s32 i2c_smbus_write_byte(struct i2c_client *my_client, u8 value);
```

Writes the byte `value` to the I2C device `my_client`. If an error occurs, the function returns a negative error code, else 0.

Exercise

- Copy the file `ex_1/rgb_brd.c` to `ex_2` and edit the copy in this folder.
- Light up the RGB LED in a color of your choice. You can do so, by writing to P1-P3 of the PCF8574 in the probe function of the driver.
- Turn off the RGB LED in the remove function
- Compile and test the module
- Additional task: Read the state of the button on P0 in the probe function and write it to the Kernel's log.

Creation of sysfs entries

- *sysfs*: Virtual filesystem
- Display and management of *Kernel Objects* (`kobject`)
- Allows interaction with the driver
- *Kernel Object*: Folder in *sysfs*
- *Kernel Object* can have attributes (represented as files) over which the driver can exchange data with user space.
- Procedure: Implement show and store functions, create attribute, create Kernel Object, link sysfs files with Kernel Object

Show and store functions and attribute

```c
/* Required Header */
#include <linux/kobject.h>

static ssize_t mydev_show(struct kobject *kobj, struct kobj_attribute *attr,
     char *buffer)
{
        return sprintf(buffer, "Hello world!\n");
}

static ssize_t mydev_store(struct kobject *kobj, struct kobj_attribute *attr
    , const char *buffer, size_t count)
{
        printk("I got %s\n", buffer);
        return count;
}

static struct kobj_attribute mydev_attr = __ATTR(my_attr, 0660, mydev_show,
    mydev_store);
```

Create kobject and link it with the attribute

```
struct kobject * my_kobj;
/* in init or probe function */
int status;

my_kobj = kobject_create_and_add("my_kobj", my_kobj);
if (!my_kobj) {
        printk("Error creating kernel object\n");
        return -ENOMEM;
}

status = sysfs_create_file(my_kobj, &mydev_attr.attr);
if (status) {
        printk("Error creating /sys/my_kobj/my_attr\n");
        return status;
}
```

Delete kobject and attribute

```
/* in exit or remove function */
sysfs_remove_file(my_kobj, &mydev_attr.attr);
kobject_put(my_kobj);
```

Exercise

In `ex_3` you can find a demo driver for creating a Kernel Object and an Attribute. Modify the driver as follows:

- Rename the Kernel object *rgb_led*
- Create the attribute *led* of the Kernel object *rgb_led*
- Implement the store function for this attribute, so you can control the RGB LED over it. By writing the string `011` to the attribute, the red LED should be set t0 0, the green LED to 1 and the blue LED to 1.
- Additional Task: Create a second attribute *button* of the Kernel Object *rgb_led*. Implement a show function to read out the current state of the button.

Linux Kernel programming on a Raspberry Pi

- Update packages with: `sudo apt update && sudo apt upgrade -y`
- Install Kernel Headers: `sudo apt install -y raspberrypi-kernel-headers`
- Install build tools like gcc, make, ...: `sudo apt install -y build-essential`
- Reboot, to start updated kernel: `sudo reboot`

A true Hello World Kernel Module

```c
#include <linux/module.h>
#include <linux/init.h>
int __init my_init(void)
{
        printk("hello_kernel - Module was loaded.\n");
        return 0;
}
void __exit my_exit(void)
{
        printk("hello_kernel - Module was removed\n");
}
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Johannes Roith");
MODULE_DESCRIPTION("A simple hello world LKM");
module_init(my_init);
module_exit(my_exit);
```

The macro module_i2c_driver

The macro `module_i2c_driver(my_driver)` creates the following code:

```
static int i2c_driver_init(void)
{
        return i2c_add_driver(&my_driver);
}
module_init(i2c_driver_init);

static void i2c_driver_exit(void)
{
        i2c_del_driver(&my_driver);
}
module_exit(i2c_driver_exit);
```