Linux Driver Workshop

An introduction to Linux Driver Programming

Johannes Roith

22.03.2024

About me

- Embedded Software Developer
- Embedded Linux YouTube Channel
- My webpage with links to GitHub, Mastadon, LinkedIn, . . .
- On driver from me got accepted in the mainline Linux Kernel

Support my work

- https://www.buymeacoffee.com/johannes4linux
- https://paypal.me/johannes4linux
- Super Thanks for my YouTube videos

Agenda

Backup

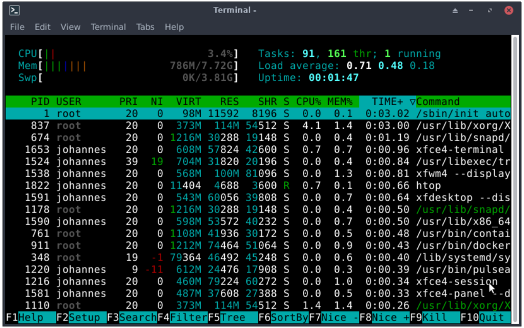
11 A real Hello World Kernel Module

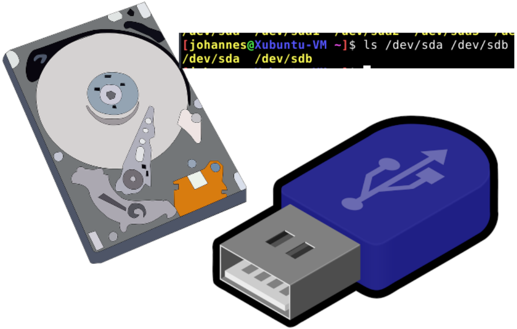
12 The macro module_i2c_driver

The Linux Kernel

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

The Linux Kernel





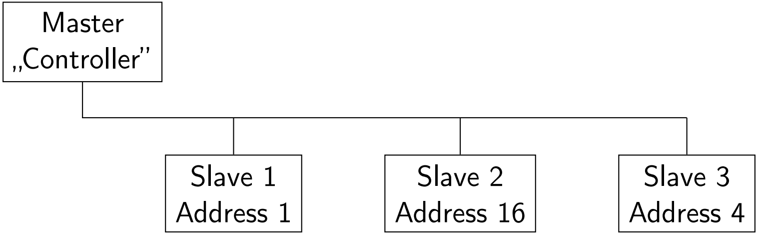
Linux Kernel Programming on a Raspberry Pi

- 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 load new kernel (if installed during update): `sudo reboot`

The I2C Bus

- Simple two-wire bus
- Data signal: *SDA*
- Clock signal: *SCK*
- Frequencies: 100kbit/s, 400kbit/s, 1Mbit/s
- Pull-Up resistor required for both signals

The I2C Bus



Master
„Controller"
Slave 1
Address 1
Slave 2
Address 16
Slave 3
Address 4

A Linux I2C Driver

Header and compatible devices

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

/* List all compatible devices with this driver */
static struct i2c_device_id my_ids[] = {
        {"my_dev"},
        {} /* An empty element signalizes the end of the list */
};
MODULE_DEVICE_TABLE(i2c, my_ids);
```

Probe- and Remove Functions

```c
/* function will be called when a compatbile I2C device is added */
static int my_probe(struct i2c_client *client, const struct i2c_device_id *
    id)
{
        printk("Hello from I2C Slave with addresse: 0x%x\n", client->addr);
        return 0;
}

/* function will be called when a I2C device is deleted */
static void my_remove(struct i2c_client *client)
{
        printk("Bye, bye, I2C\n");
}
```

A Linux I2C Driver

Create driver

```
        /* Combine compatible devices, Probe- & Remove-function to driver */
static struct i2c_driver my_driver = {
        .probe = my_probe,
        .remove = my_remove,
        .id_table = my_ids,
        .driver = {
                .name = "my-i2c-driver",
        }
};
/* Register Driver */
module_i2c_driver(my_driver);
/* Info about Driver*/
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Johannes Roith");
MODULE_DESCRIPTION("A Hello World I2C Driver");
```

Makefile zum Bauen des I2C Treibers

```
# Kernel Header Makefile compiles i2c_hello.c to i2c_hello.o file
    automatically
obj-m += i2c_hello.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
```

Managing modules in a Shell

- `lsmod` lists the loaded modules
- `dmesg` shows the kernel's log
- `insmod <modulname>` loads the module `<modulname>` into the Kernel
- `rmmod <modulname>` removes the module `<modulname>` from the Kernel
- `modprobe <modulname>` loads the module `<modulname>` together with all dependencies
- `modinfo <modulname>` shows the Meta-Daten (Author, License, Description, ...) of the module `<modulname>`

Adding I2C device over sysfs

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

# Add device mydev with I2C address 0x12
echo "mydev 0x12" > new_devices

# Remove I2C device with address 0x12 from system
echo "0x12" > delete_device
```

Exercise

- Implement the Hello World I2C Driver on a Raspberry Pi. The compatible device should be named *rgb_brd*.
- Build the module with a Makefile
- Load the kernel module
- Check if the module is loaded to the kernel
- Add an I2C device over sysfs
- Check the kernel's log
- Delete the device and remove the module from the kernel

PCF8574 IO Expander

- Write access sets the outputs P0 - P7
- Read access reads value of P0 - P7
- Button connected to P0
- For input: Set Port to 1, Button pulls input to GND: If a 1 is read from the port, the button is not pressed, if a 0 is read, the button is pressed.
- Red LED connected to P1, green LED to P2, blue LED to P3
- Output set to 0: LED is on
- Output set to 1: LED is off

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

Access the I2C bus

```
struct i2c_client *my_client;
```

The structure from the type `struct i2c_client` is used, to manage an I2C device in the kernel. Over the pointer `my_client` it is possible to access the device, e.g. to read or write data from/to it.

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

Reads a byte from the I2C device `my_client`. If an error occurs during the operation, a negative error code is returned, else the read value.

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

Writes a byte `value` to the I2C device `my_client`. If an error occurs during the operation, a negative error code is returned, else 0.

Aufgabe

- Let the RGB LED light with a color of your choice, by writing to P1-P3 of the IO Expander in the probe function.
- Turn off the LED in the remove function.
- Compile and test the kernel module.
- Additional Task: Read in the value of the button at P0 in the probe function and print the value to the kernel's log.

Creation of sysfs entries

- *sysfs*: Virtual Filesystem
- Presentation and Management of *Kernel Objects* (`kobject`)
- Enables interaction to drivers from userspace
- *Kernel Objekt*: Folder in *sysfs*
- *Kernel Objekt* offers attributes (present as files) over which we can communicate with the driver from userspace.
- Procedure: Implement Show and Store functions, add attributes, create kernel object, combine sysfs file with attributes

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 kernel object and combine it with 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

- Add the folder *rgb_led* in the sysfs over the kernel module
- Create the file *led* inside the folder *rgb_led*
- Implement the store function to be able to control the three LEDs. By writing the string `011`, the red LED is turned off, the green and the blue on.
- Additional Task: Add the file *button* to the folder *rgb_led*. Implement a show function for this file so that the state of the button can be read out.

A real Hello World Kernel Module

```c
#include <linux/module.h>
#include <linux/init.h>
int __init my_init(void)
{
        printk("hello_kernel - Hello, Kernel\n");
        return 0;
}
void __exit my_exit(void)
{
        printk("hello_kernel - Goodbye, Kernel\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

Das Makro `module_i2c_driver(my_driver)` is replaced with 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);
```