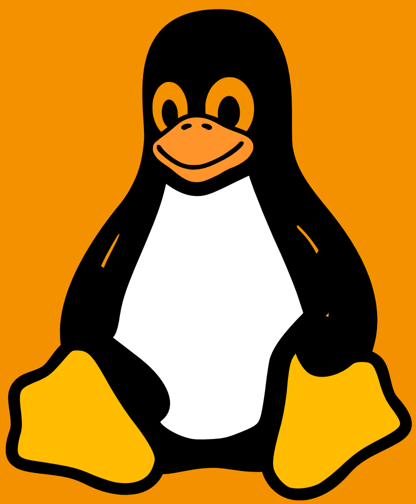
Technische Hochschule Rosenheim
Technical University of Applied Sciences

Prof. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

OS 16 – Drivers



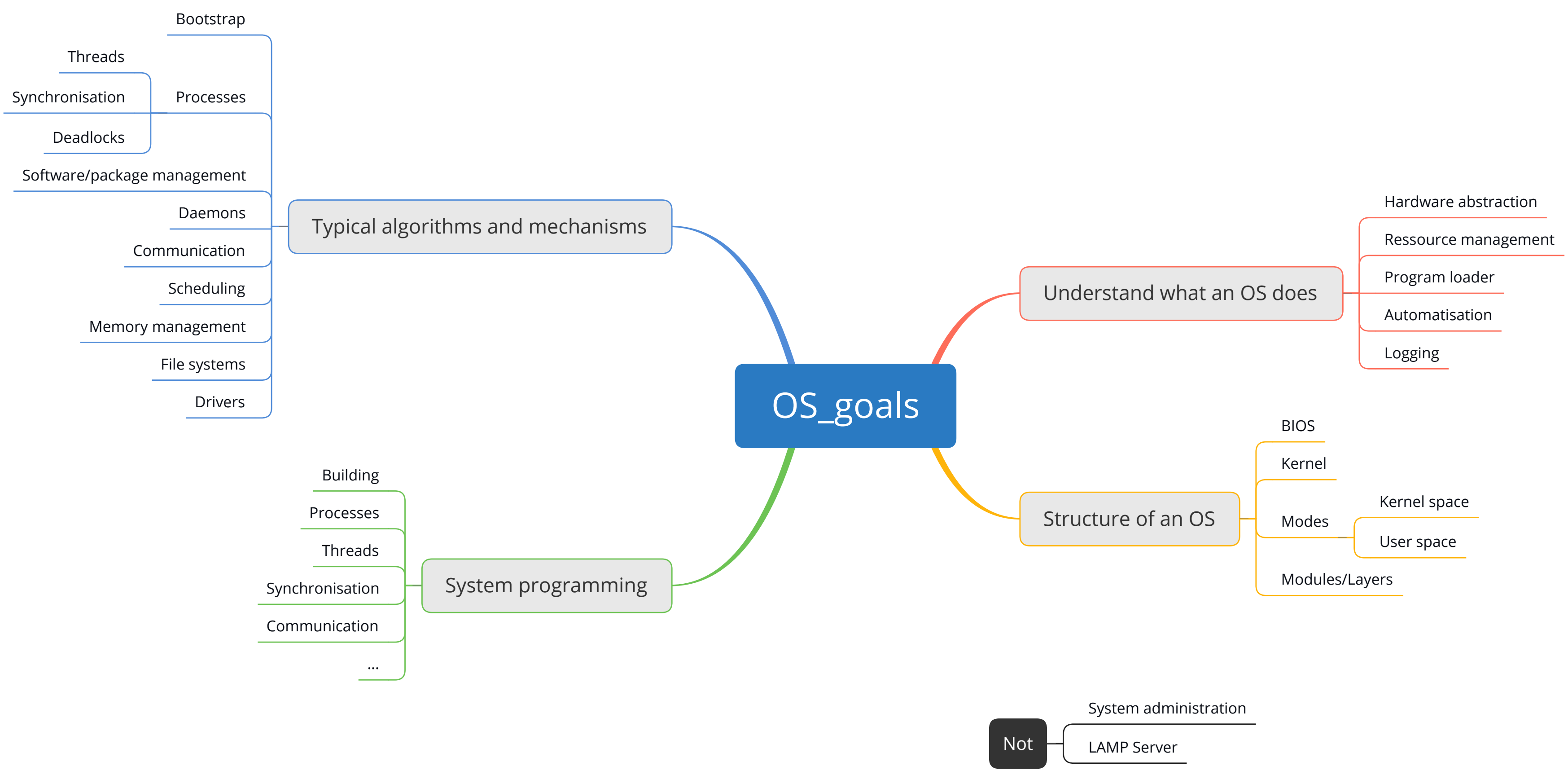
source: iconspng.com

The lecture is based on the work and the documents of Prof. Dr. Ludwig Frank

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal

OS::Drivers

- Design goals
- Structure of I/O systems
- Linux device files
- Kernel modules
- Linux device driver development

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intro

What is a **device driver**?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intro

A **device driver** is a piece of software that **controls** the **interaction** with an connected, built-in, or virtual device.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Static vs. dynamic driver

A **static** driver is **built-in** into the kernel.

A **dynamic** driver can be **loaded and unloaded at runtime**.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Device classes

- **Character device**
- **Block device**
- USB
- **Network**
- Bluetooth
- FireWire (IEEE1394)
- SCSI
- IrDA (Infrared Data Association)
- Cardbus and PCMCIA
- Parallelport
- I2C
- ...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Design goals

User programs should be device independent

The type of the device or the position shouldn't be visible to the user application.

- Different hardware vendors
- Different device types: hard drive, SSD, USB stick, network drive, ...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Design goals

Uniform device names

Devices are in the /dev folder and should be independent of hardware connection.

Example 1:

```
cat file > /dev/lp1  #sends file to printer on /dev/lp1
```

Example 2:

```
int main(void){
    int fd = open("/dev/device", O_RDWR, 0666);

    char buffer[LEN] = {"Hello\n"};
    ssize_t bytes_written = write(fd, buffer, LEN);
    ssize_t bytes_read    = read(fd, buffer, LEN);

    close(fd);

    return EXIT_SUCCESS;
}
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Design goals

Error handling

- Localisation and report of errors
- Error logging
- Correction of transient errors (e.g. read again)
- Automatic notification of the maintenance service

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Design goals

Support for synchronous and asynchronous data transfer

Synchronous
Process waits until the I/O operation completes.

Asynchronous
Process can proceed while the OS transfers the data.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Design goals

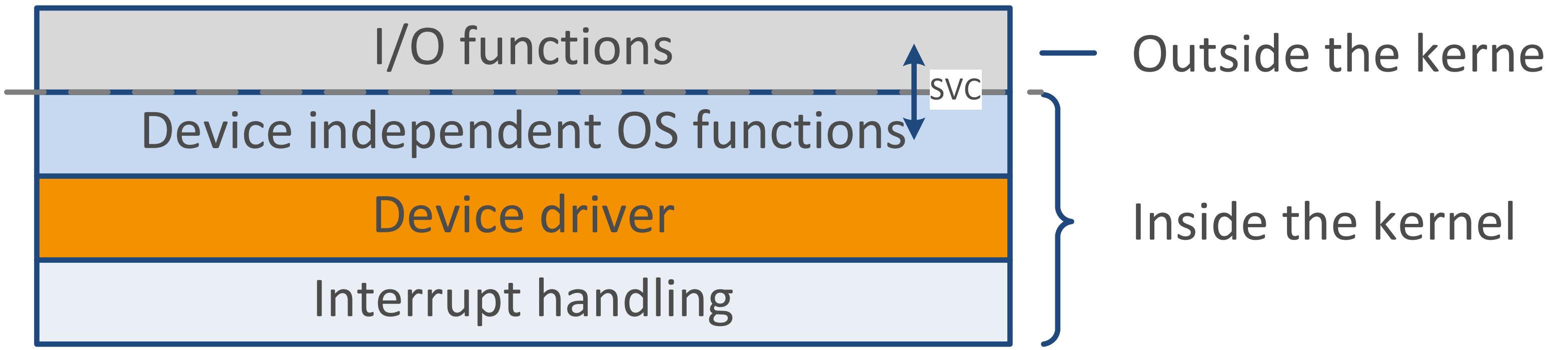
Device management

- Exclusive or shared access
- Locking mechanism (`fcntl()`)
- Deadlock monitoring (e.g. banker's algorithm)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Structure of I/O systems



Device independent OS functions

- Naming of device files (e.g. `/dev/fd0`)
- Access control
- Caching
- Locking mechanism
- Device allocation management
- Error handling
- Logging

Device driver

- Knows the HW properties of the devices
- Provides a HW independent interface

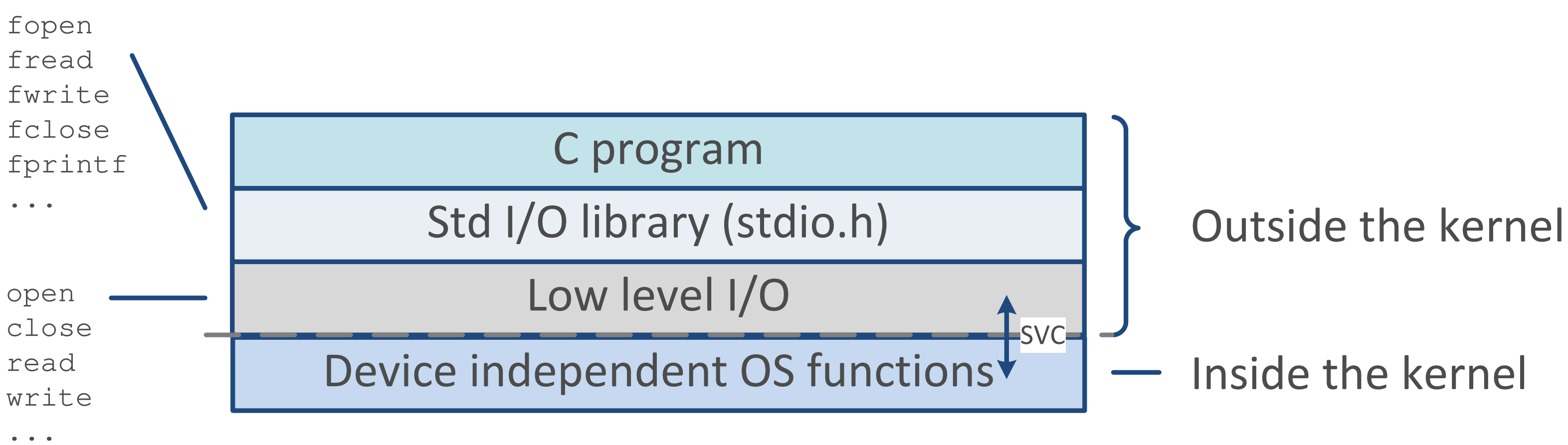
Interrupt handling

- Handling of interrupts
- Assignment to the waiting process

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

I/O layers



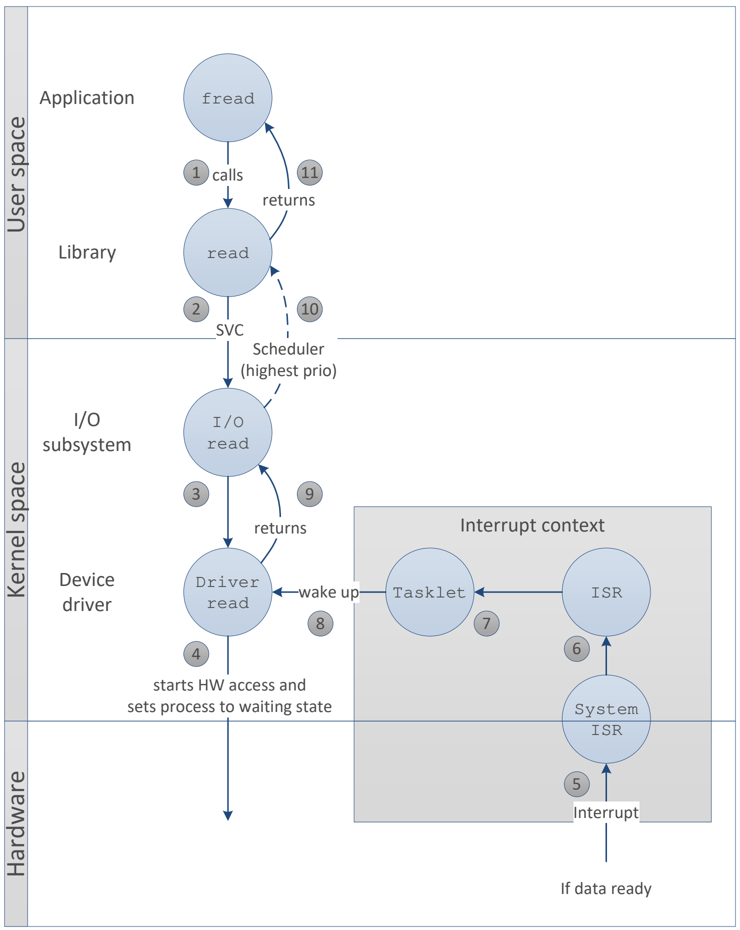
Functions of `stdio.h`

- Buffering
- Formatting
- Format conversion
- Higher level of abstraction (easier to use)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example operation: read data



1. Application calls `fread()`
2. `read()` starts a SVC (supervisor call)
3. The HW independent I/O read calls the driver
4. The driver starts the read on the HW and sets the calling process into waiting state.
5. The HW triggers an interrupt if the data is ready. The kernel first handles the interrupt in its system ISR (interrupt service routine)
6. The system ISR calls the ISR of the driver
7. The ISR of the driver starts a tasklet (its like a thread in the kernel but more lightweight)
8. The tasklet wakes up the driver
9. The driver read function continuous: fetches and copies the data into user space.
10. The I/O read function continuous and the SVC is now finished. If the process has the highest prio it can run on the CPU.
11. The data is processed in `fread()` and finally returned to the application.

CAMPUS Rosenheim
Computer Science

Linux device files

- One **device file** per device
- **Device examples**: mouse, harddisk, keyboard, display, terminal, ...
- **Device file examples**: `/dev/sda`, `/dev/tty0`, `/dev/cdrom`
- Talk to device → `open()/read()/write()` with device file
- Application uses the device file to communicate with a device
- Device files only have some meta data (inode) and does not require disk space.

Device types

Device type	File type	Example	Description
character device	c	Terminal, printers, ...	Sending and receiving single characters (bytes).
block device	b	Harddisks, USB, CD, ...	Sending and receiving of entire data blocks.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Create a Linux device files

Create a device file

```
mknod /dev/device_file TYPE MAJOR MINOR

#TYPE  = c (character) or b (block)
#MAJOR = device type and the corresponding driver
#MINOR = device within the driver (major_device_number)
```

Delete a device file

```
#delete device file
rm /dev/device_file
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Inode device file example

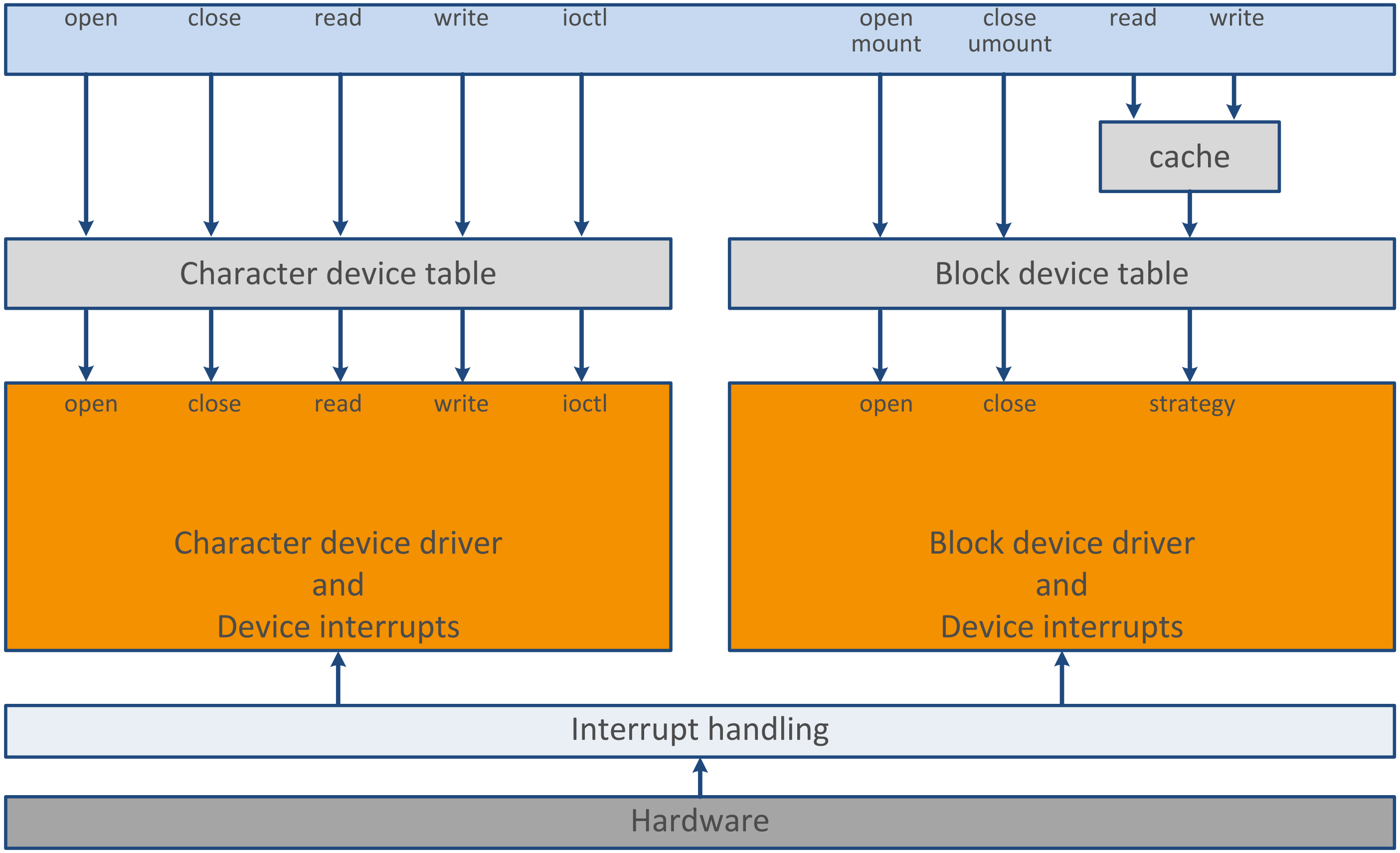
An inode on a device files contains the device type (c/b), **major**, and the **minor** number.

```
ls -l -i /dev
    type                           major  minor
328 brw-rw----  1 root disk     259,    1       Jan 14 22:37 nvme0n1p1
329 brw-rw----  1 root disk     259,    2       Jan 14 22:37 nvme0n1p2
330 brw-rw----  1 root disk     259,    3       Jan 14 22:37 nvme0n1p3
...
  6 crw-rw-rw-  1 root root       1,    3       Dez 27 10:42 null
...
 15 crw--w----  1 root tty        4,    0       Dez 27 10:42 tty0
 20 crw--w----  1 root tty        4,    1       Dez 27 10:42 tty1
...
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Integration of the device drivers



CAMPUS Rosenheim
Computer Science

Linux device driver tables

Character device table*

MAJOR	open	close	read	write	ioctl	Example
1	ram_open	ram_close	ram_read	ram_write	ram_ioctl	Memory
3	ser_open	ser_close	ser_read	ser_write	ser_ioctl	Serial
4	tty_open	tty_close	tty_read	tty_write	tty_ioctl	Terminal
...						

Block device table*

MAJOR	open	close	strategy	Example
1	hd_open	hd_close	hd_strategy	Harddisk
2	fd_open	fd_close	fd_strategy	Floppy disk
...				

* Simplified visualisation of device table with sample data.

List of MAJOR device numbers: `https://www.kernel.org/doc/html/latest/admin-guide/devices.html`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kernel modules

A **kernel module** is a piece of compiled code that is dynamically loaded into the kernel.

- Filetype: `*.ko` (kernel object)
- Place: `/lib/modules` btw. `/lib/modules/$(uname -r)/`
- Executed in the context of the kernel (kernel space).
- Can be loaded and unloaded at runtime (without reboot).

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kernel module: C template

```c
#include <linux/module.h>   //needed by all modules
#include <linux/kernel.h>   //needed for KERN_INFO
#include <linux/init.h>     //needed for the macros

//this is called when the module is loaded
static int __init exmod_init(void) {
    printk(KERN_INFO "Hello, exmod\n");
    //register functions and initialise the module
    return 0;
}
//this is called when the module is unloaded
static void __exit exmod_exit(void) {
    printk(KERN_INFO "Goodbye, exmod\n");
    //unregister functions and cleanup the module
}

module_init(exmod_init); //register the exmod_init to be called on load
module_exit(exmod_exit); //register the exmod_exit to be called on unload

//module meta data
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Florian Künzner");
MODULE_DESCRIPTION("Exmod is an example module");
MODULE_VERSION("1.0");
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kernel modules: Linux kernel API

Inside the kernel you can **only** use the **Linux kernel API**. The `glibc` is not available there.

Linux kernel API examples:

Function	Description
`printk`	Print into the kernel log `/var/log/kern.log`.
`strcpy, strncpy, ...`	String functions from `include/linux/string.h`
`kmalloc/kfree`	Allocates and frees memory in the kernel space.
...	

Linux kernel API `https://linux-kernel-labs.github.io/master/labs/kernel_api.html#linux-kernel-api`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Compile a kernel module

Makefile

```
obj-m += module_name.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
```

On shell:

```
dev@dev ~/module$  make
```

Kernel Makefile doc: `https://elixir.bootlin.com/linux/v4.20.2/source/Documentation/kbuild/makefiles.txt`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kernel modules: Linux commands

Command	Description
`lsmod`	**List** the loaded modules (`/proc/modules`).
`modinfo module`	**Show** information about a module.
`insmod module.ko`	**Load** a `module.ko` file into the kernel.
`modprobe module`	**Load** a module from `/lib/module/$(uname -r)/` into the kernel and handle dependencies. Requires an up to date `modules.dep`.
`depmod`	**Updates** `/lib/module/$(uname -r)/modules.dep`
`rmmod module`	**Unloads** a module
`modprobe -r module`	**Unloads** a module and handle dependencies.

CAMPUS Rosenheim
Computer Science

Device driver development

The developer has to implement

- a kernel module
- register itself at the I/O management in the OS
- handle application requests

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Functions that have to be implemented

Functions to load the module into the kernel

- `module_init()` → `driver_init()`
- `module_exit()` → `driver_exit()`

Functions triggered through applications

- `open()` → `driver_open()`/`close()` → `driver_release()`
- `read()` → `driver_read()`/`write()` → `driver_write()`
- ...

Functions triggered through OS or hardware

- ISR (Interrupt Service Routine)
- Kernel threads
- Tasklet

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Device driver: C template

```c
#define MAJOR_NR 236
static int __init driver_init(void) {
    register_chrdev(MAJOR_NR, "driver" , &fops);
    //...
    return 0;
}
static void __exit driver_exit(void) {
    unregister_chrdev(MAJOR_NR ,NAME);
}
module_init(driver_init);
module_exit(driver_exit);

static struct file_operations fops = {
    .open    = driver_open,
    .release = driver_release,
    .read    = driver_read,
    .write   = driver_write,
};
static int driver_open(...){                //called on open()
    bool ok = try_module_get(THIS_MODULE); //increment driver usage
    //...
}
static int driver_release(...){             //called on close()
    module_put(THIS_MODULE);               //decrement driver usage
    //...
}
static ssize_t driver_read(...){            //called on read()
    //...
}
static ssize_t driver_write(...) {          //called on write()
    //...
}
//module meta data...
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Low level function usage

User space program:

```c
#include <stdlib.h> //EXIT_SUCCESS
#include <fcntl.h>  //flags O_RDWR, ...
#include <unistd.h> //open, close, read, write...
#include <stdio.h>  //printf

#define LEN 10

int main(void){
    int fd = open("/dev/device", O_RDWR, 0666);

    char buffer[LEN] = {"Hello\n"};
    ssize_t bytes_written = write(fd, buffer, LEN);
    ssize_t bytes_read    = read(fd, buffer, LEN);

    close(fd);

    return EXIT_SUCCESS;
}
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Driver example

C code example.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary and outlook

Summary

- Design goals
- Structure of I/O systems
- Linux device files
- Kernel modules
- Linux device driver development