CharacterDevice Drivers

Praktikum „Kernel Programming“

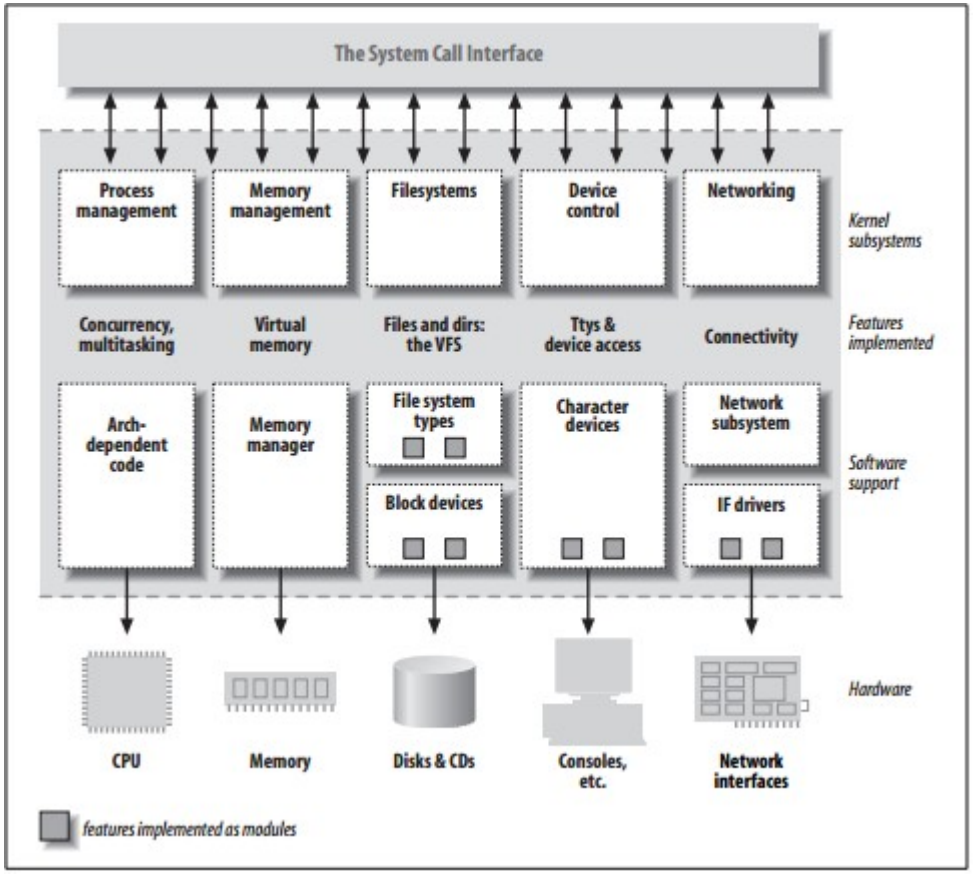
JohannesCoym

December2, 2015

Outline

- What are character device drivers
- Example of the connection between application and character
- Major and minor numbers
- File operations
- ioctl(Input/Output control)
- Blocking I/O
- Access control

What are character device drivers

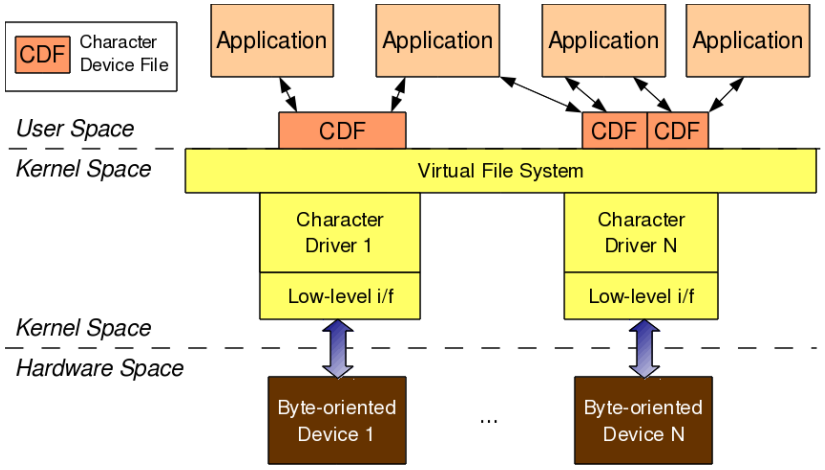


What are character device drivers

- Character devices can be accessed as a stream of bytes
- Character device drivers implement *open*, *close*, *read* and *writ*
  of the time and grant access to the data stream for the user s
- Examples for character devices:
  - Serial Ports (/dev/ttyS0)
  - Console (/dev/console)
  - Mouse (/dev/input/mouse0)
  - (all devices that are neither storage nor network devices)

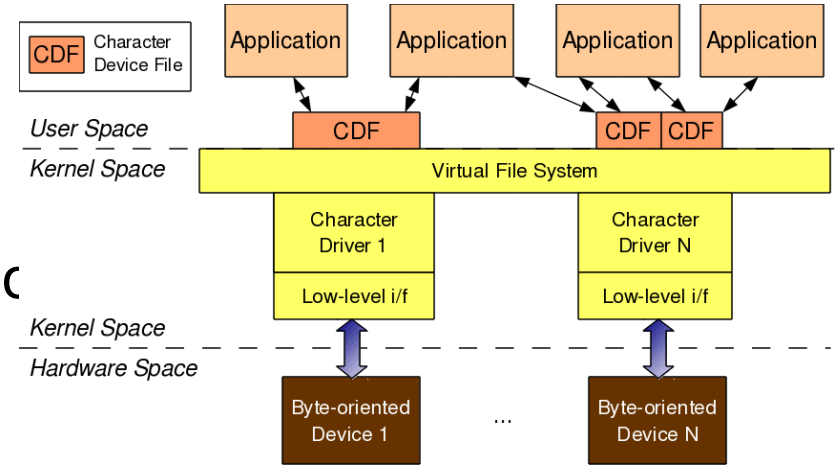
Whatarecharacterdevicedrivers

- Connection betweenapplication andthedevicein 4 steps:
  - Application
  - Characterdevicefile
  - Characterdevicedriver
  - Characterdevice



Example of the connection between application and character device

- The musicplayerwritesthe musictoplayintotheCDF
- The characterdevicedriver takesthemusicfromtheCDF andsendsitasa bytestreamtc thecharacterdevice



Major and minor numbers

- Access to device driver from user space through device file
- Kernel needs to know to which driver and which device the de
  file belongs
- Device files mapped by the kernel to a major and a minor nur
  - Major number refers top the driver, each driver has its own
  - Minor number refers to the device which is managed by the driver

Major andminor numbers

- Limit of 255 major and 255 minor numbers
- Each combination of major and minor number is unique and m
  to a device file
- Some functions need to know the major number
- In the Kernel the type dev_tcontains major and minor number
  device

Major andminor numbers

- Togetthemajororminor numberfroma dev_t:
  - `MAJOR(dev_tdev);`
  - `MINOR(dev_tdev);`
- Togeta dev_tfromthemajorandtheminor number:
  - `MKDEV(intmajor, intminor);`

Major andminor numbers

- Twotypesof majorandminor numberregionallocation:
  - intregister_chrdev_region(dev_tfirst,unsigned intcount,char *name);
    - Static allocation where it's not sure if you'll get the requested region
    - If the minor numbers exceed the 255 it will automatically assign the next major too, if it's fr
  - intalloc_chrdev_region(dev_t*dev,unsigned intfirstminor, unsigned intcount,char*name);
    - Dynamic allocation of the device numbers by the kernel
    - You will definitely get a free major number assigned
- Tofreetheassignedmajorandminor numbersin theexitfunction:
  - void unregister_chrdev_region(dev_tfirst,unsigned intcount);

Major andminor numbers

- Allocating only a major number with it’s full 256 minor numbe
  - intregister_chrdev(unsigned intmajor,constchar*
    name,conststructfile_operations* fops);
    - Will try to allocate the given major
    - Setting major to 0 will change the functions behavior to dynamically allocate a number
- Free the assigned major number:
  - voidunregister_chrdev(unsigned intmajor,constchar*
    name);

File Operations

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
        unsigned (*mmap_capabilities)(struct file *);
#endif
};
```

File Operations

- Structure defined in linux/fs.h
- Contains pointers to the common file operations by the driver
- Usage:

```
struct file_operations fops = {
        .read = device_read,
        .write = device_write,
        .open = device_open,
        .release = device_release
    };
```

File Operations–open/release

- `int (*open) (struct inode*, struct file *);`
- `int (*release) (struct inode*, struct file *);`
- Return value: 0 for success, negative numbers for failure
- Structinode* is a structdefined in linux/fs.hand includes information about the device
- Structfile * is a structdefined in linux/fs.hand references to the device file

File Operations–read/write

- `ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);`
- `ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);`
- Return value: the size read or written
- Structfile * is a structdefined in linux/fs.hand referencestothedevicefile
- Char __user * is the buffer we receive from user space
- Size_tis the size of the requested transfer
- Loff_tis the long offset type indicating the position in the file the user is accessing

File Operations-llseek

- loff_t(*llseek) (structfile *, loff_t, int);
- Return value: New positionin thefile
- Structfile * is a structdefined in linux/fs.hand referencestothe devicefile
- Loff_tisthevaluedefininghowmuchthepositionwill bechanged
- Intdefineswhereitshouldstart(0 frombeginning, 1 at current position, 2 at end)

ioctl(Input/Output control)

- Used for device control of the driver
- Can include software commands like receiving error logs
- Can also include hardware commands like opening a CD drive
- Some command-oriented character devices like terminals use commands instead of ioctl
  - It’s also possible to use only ioctlinstead of read and write, you just h implement the read and write operations as ioctlcommands

ioctl(Input/Output control)

- Prototype definition:
  - int ioctl(int fd, unsigned long cmd, ...);
  - ... stands for an optional argument, usually char *argp
- Each ioctl command is defined by one 8 bit Type number for th
  driver and an additional 8 bit Number for the actual command
- Should return –ENOTTY when an undefined ioctl command is c

ioctl(Input/Output control)

- Each module can define its own ioctlcommands
- The ioctlcommands should be defined in a header file in combination with the major number
  - Most of the time static major number allocation, when working with i
- The header file should be referenced by any programs using t commands

ioctl(Input/Output control)

- Usually with a switch-case
- Selecting in switch case which command was sent to him
- Default should just return –ENOTTY
- On success of one command 0 or an answer to the user space program should be returned
- Arguments can be given as a pointer or value and can be rece a return value or pointer

BlockingI/O

- If the driver gets a request which can‘t handle right now he pu
  process to sleep
- Reasons for the driver to be not able to handle the request:
  - Receiving a read request when there is no data to read available
  - Receiving a write request when the buffer is already full

BlockingI/O

- To send a process to sleep, we need a wait queue
  - Static initialized wait queue; initialized at compile time:

```
DECLARE_WAIT_QUEUE_HEAD (my_queue);
```

  - Dynamic initialized wait queue; intializedat runtime

```
wait_queue_head_t my_queue;
init_waitqueue_head(&my_queue);
```

BlockingI/O

- Several ways to send a process to sleep:
  - sleep_on(wait_queue_head_t*queue);
  - interruptible_sleep_on(wait_queue_head_t*queue);
  - sleep_on_timeout(wait_queue_head_t*queue,longtimeout);
  - interruptible_sleep_on_timeout(wait_queue_head_t*queue, longtimeout);
  - void wait_event(wait_queue_head_tqueue,intcondition);
  - intwait_event_interruptible(wait_queue_head_tqueue,int condition);

BlockingI/O

- All variants of sleep_oncan be woken up using this commands
  - `wake_up(wait_queue_head_t*queue);`
  - `wake_up_interruptible(wait_queue_head_t*queue);`
  - `wake_up_sync(wait_queue_head_t*queue);`
  - `wake_up_interruptible_sync(wait_queue_head_t*queue);`
- The wait_eventvariants don‘t need a wake_upcall, but wake up automatically on the condition

Access Control

- Single-open lock:
  - Device file can only be opened by one process at the same time
  - Usually implemented with an Integer which is 0 when no process is u
    driver and 1 when it‘s busy
- Single-user lock:
  - Device file can be opened by all processes owned by one user
  - Usually implemented using a field saving the owner of the first proce
    opening the file

Access Control

- „Blocking-open“:
  - Device file can be opened by any process at any time but if another p
    is using the device, thecallingprocess will have to wait
  - Usually implemented with a wait queue
- Cloning the device
  - When open is called by a process, it gets it‘s own copy of the device
    virtual device file

Literature

„Major andMinor Numbers“
http://www.makelinux.net/ldd3/chp-3-sect-2
Corbet, Rubini, Kroah-Hartmann(2005). „Linux Device Drivers“
https://static.lwn.net/images/pdf/LDD3/ch01.pdf
„Device Drivers, Part 4: Linux CharacterDrivers“
http://opensourceforu.efytimes.com/2011/02/linux-character-drivers
„CharacterDevice Files“
http://www.tldp.org/LDP/lkmpg/2.4/html/c577.htm
„Enhanced CharDriver Operations“
http://www.xml.com/ldd/chapter/book/ch05.html
„Writing a Linux Kernel Module —Part 2: A Character Device”
http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/