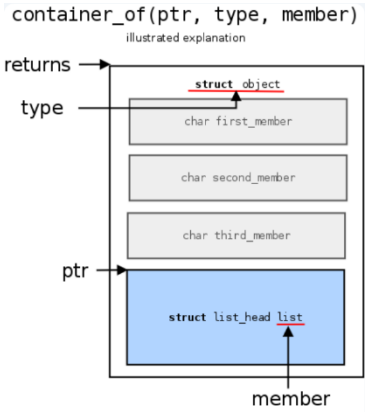
ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

Δραστηριότητα 5: Linux και Δημιουργία Οδηγητή (Driver)

ΑΣΚΗΣΗ 1

1. Στο αρχείο `include/linux/container_of.h` βρίσκεται το macro `container_of(ptr,type,member)`.
   Επιστέφει τη διεύθυνση του container structure

```
#define container_of(ptr, type, member) ({                  \
        const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
        (type *)( (char *)__mptr - offsetof(type,member) );})
```



Εικόνα 1
https://radek.io/posts/magical-container_of-macro/

`typeof(((type *)0)->member)`: Προσδιορίζει τον τύπο του member

`(char *)__mptr - offsetof(type, member)`: Αφαιρεί το offset του μέλους member από τη διεύθυνση του ptr, υπολογίζοντας έτσι τη διεύθυνση του container structure.

2. Στο αρχείο `include/linux/fs.h` βρίσκεται η δομή file που αναπαριστά ένα ανοικτό αρχείο και εμφανίζεται μόνο σε kernel κώδικα

```
struct file {
        atomic_long_t                   f_count;
        spinlock_t                      f_lock;
        fmode_t                         f_mode;
        const struct file_operations    *f_op;
        struct address_space            *f_mapping;
        void                            *private_data;
        struct inode                    *f_inode;
        unsigned int                    f_flags;
        unsigned int                    f_iocb_flags;
        const struct cred               *f_cred;
        /* --- cacheline 1 boundary (64 bytes) --- */
        struct path                     f_path;
        union {
                /* regular files (with FMODE_ATOMIC_POS) and directories */
                struct mutex            f_pos_lock;
                /* pipes */
                u64                     f_pipe;
        };
        loff_t                          f_pos;
```

3. Στο αρχείο `include/linux/fs.h` βρίσκεται η δομή file_operations που έχει ως μέλη function pointers ο καθένας από τους οποίους αντιστοιχεί σε ένα system call και η συνάρτηση στην οποία “δείχνει” εκτελείται όταν μία διεργασία εκτελέσει ένα system call πάνω στη συσκευή μας.

```
struct file_operations {
        struct module *owner;
        fop_flags_t fop_flags;
        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 *);
```

...

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

ΑΣΚΗΣΗ 2

Εκτελώντας την εντολή `sudo cat /dev/input/mice | hexdump # prettier output`, εμφανίζεται ένα stream από bytes κάθε φορά που μετακίνω το ποντίκι ή πατάω κάποιο πλήκτρο του.

```
nat@nataliarouska:~$ sudo cat /dev/input/mice | hexdump
[sudo] password for nat:
0000000 c218 087f 7f00 0008 284d fd00 0028 28ff
0000010 ff00 0028 28fd fd00 0028 28ff ff00 0028
0000020 28fe ff00 0028 28ff fe00 0a08 3800 fffd
0000030 0028 38fd fffd ff18 3800 ffff 0028 18fe
0000040 00fe ff38 18ff 00ff 0028 28ff fe00 fe18
0000050 3800 ffff ff18 2800 ff00 fe18 1800 00ff
0000060 0028 18fe 00ff fe18 2800 ff00 ff18 0800
0000070 0001 0208 0800 0002 0208 0800 0001 0108
0000080 0800 0002 0028 08ff 0001 0308 2800 fe00
0000090 0108 0800 0001 0208 0800 0001 0108 0800
00000a0 0002 0028 28ff ff00 0108 2800 fe00 0028
00000b0 08ff 0001 0228 28ff fe00 0128 08ff 0001
00000c0 0028 08ff 0002 0028 28fe ff00 0009 0800
00000d0 0000 000a 0800 0000 000c 0800 0000 000a
00000e0 0800 0000 000c 0800 0000 000c 0800 0000
00000f0 0009 0800 0000 fd38 38fd fcff fd38 28fe
0000100 fe00 0028 18ff 00ff fe38 28fd ff00 0009
^C
```

Εκτελώντας το script με την εντολή `sudo python mouse.py`, εμφανίζονται τριάδες από bytes, όπως αυτές περιγράφονται στο PS/2 mouse protocol

Yovfl	Xovfl	dy8	dx8	1	Middle Btn	Right Btn	Left Btn
dx7	dx6	dx5	dx4	dx3	dx2	dx1	dx0
dy7	dy6	dy5	dy4	dy3	dy2	dy1	dy0

```
nat@nataliarouska:/media/sf_shared_folder_vm$ sudo python3 mouse.py
(56, -28, -8)
(8, 0, 2)
(8, 0, 1)
(8, 0, 1)
(8, 0, 2)
(8, 0, 1)
(40, 0, -1)
(40, 0, -2)
(40, 0, -1)
(40, 1, -1)
(40, 0, -2)
(24, -1, 0)
(24, -3, 0)
(24, -3, 0)
(24, -2, 0)
(24, -3, 0)
(24, -1, 0)
(24, -2, 0)
(24, -1, 0)
(9, 0, 0)
(8, 0, 0)
(12, 0, 0)
(8, 0, 0)
```

Η κάθε τριάδα από bytes έχει την μορφή

(button states, movement in x-axis, movement in y-axis)

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

ΑΣΚΗΣΗ 3

Καταγράφω ένα νέο character device file στο σύστημά μέσω της εντολής

`sudo mknod /dev/mydevice c 42 0.`

Τα νούμερα 42 και 0 είναι ο major και ο minor identifier αντίστοιχα του device file. Το major αριθμός παραπέμπει στον τύπο της συσκευής ενώ ο minor αντιπροσωπεύει την ίδια τη συσκευή.

```
nat@nataliarouska:~$ sudo mknod /dev/mydevice c 0 42
mknod: /dev/mydevice: File exists
nat@nataliarouska:~$ sudo chmod 666 "hello" > /dev/mydevice
bash: /dev/mydevice: Permission denied
nat@nataliarouska:~$ cat /dev/mydevice
cat: /dev/mydevice: No such device or address
nat@nataliarouska:~$
```

Το αρχείο συσκευής που δημιουργήσα δεν έχει συνδεθεί με κάποιον driver, οπότε οι λειτουργίες read, write στο device file δεν είναι δυνατές χωρίς την επικοινωνία μέσω driver.

ΑΣΚΗΣΗ 4

Συμπληρώνω τον κώδικα στα σημεία TODO και αφού κάνω compile(make), φορτώνω το module chardev.ko

Πλέον μπορώ να γράψω και να διαβάσω από τη συσκευή mydevice μέσω του driver που φορτώνεται ως kernel module

```
nat@nataliarouska:/media/sf_shared_folder_vm$ cat /dev/mydevice
hello
nat@nataliarouska:/media/sf_shared_folder_vm$ sudo chmod 666 /dev/mydevice
nat@nataliarouska:/media/sf_shared_folder_vm$ echo "test test" >/dev/mydevice
nat@nataliarouska:/media/sf_shared_folder_vm$ cat /dev/mydevice
test test
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/sched.h>
#include <linux/wait.h>

#define MY_MAJOR 42
#define MY_MINOR 0
#define NUM_MINORS 2 /* Allow up to two different devices with major MY_MAJOR to exist in the system at a time. */
#define MODULE_NAME "MyDevice"
#define MESSAGE "hello\n"

#define BUFFER_SIZE     4096

MODULE_DESCRIPTION("Character Device Driver");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

/* This is struct that holds all the information related to your character device. */

struct my_device_data {
    /* TODO Add the cdev member variable. */
        struct cdev cdev; //character device in kernel
    /* TODO Add a buffer with BUFFER_SIZE elements member variable. */
        char buffer[BUFFER_SIZE];
        size_t size; /* "size" variable holds the size of the buffer. */
        atomic_t access; /* "access" variable is used so that this device can only be accesses by a single process at a time. */
};

/* Your machine will be able to support at most NUM_MINORS character devices with major number MY_MAJOR. */
struct my_device_data devs[NUM_MINORS];
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

```c
/* This function will get called when an application tries to open the device. */
static int my_cdev_open(struct inode *inode, struct file *file)
{
        struct my_device_data *data;

        /* TODO Print message when the device file is opened. */
        printk("Device file is opened\n");
    /* TODO inode->i_cdev contains our cdev struct, use container_of to obtain a pointer to my_device_data */
        data = container_of(inode->i_cdev, struct my_device_data, cdev);


    /* We set the file's private_data member equal to our device's data so we can access it later
       in the my_cdev_release, my_cdev_read, my_cdev_write functions. */
        file->private_data = data;

        /* TODO Return immediately if access isn't 0, use atomic_cmpxchg */
        if (atomic_cmpxchg(&data->access, 0, 1) != 0) {
                printk( "Device is already open!\n");
                return -EBUSY;
        }

        return 0;
}

static int my_cdev_release(struct inode *inode, struct file *file)
{

        struct my_device_data *data = (struct my_device_data *) file->private_data;

        printk("Close called!\n");

        /* TODO Reset access variable to 0, use atomic_set */
        atomic_set(&data->access,0);
        return 0;
}
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

```c
static ssize_t my_cdev_read(struct file *file, char __user *user_buffer, size_t size, loff_t *offset)
{
        struct my_device_data *data =
                (struct my_device_data *) file->private_data;

        size_t to_read;

        to_read = (size > data->size - *offset) ? (data->size - *offset) : size;

    printk("[READ] Size: %lu Offset: %lld", size, *offset);

    /* TODO :  Copy data from data->buffer to user buffer. Use copy_to_user. Make sure to include the offset! */
        if (copy_to_user(user_buffer , data ->buffer + *offset , to_read) != 0) return -EFAULT;
    /* move the offset by how many bytes we have read. */
        *offset += to_read;

        return to_read;
}

static ssize_t my_cdev_write(struct file *file, const char __user *user_buffer, size_t size, loff_t *offset)
{
    /* This is the my_device_data* we set in the my_cdev_open function. */
        struct my_device_data *data =
                (struct my_device_data *) file->private_data;

    printk("[WRITE] Size: %lu Offset: %lld", size, *offset);


        size = (*offset + size > BUFFER_SIZE) ? (BUFFER_SIZE - *offset) : size;
        /* TODO Copy user_buffer to data->buffer. Use copy_from_user. Make sure to include the offset! */
        if (copy_from_user(data ->buffer + *offset , user_buffer , size) != 0) return -EFAULT;

    /* move the offset by how many bytes we have written. */
    *offset += size;
        data->size = *offset;

    return size;
}
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025

```c
/* This struct will hold pointers to the functions that will be called for each system call that is executed on the device. */
static const struct file_operations my_fops = {
        .owner = THIS_MODULE,
/* TODO Add open function. */
        .open=my_cdev_open,
/* TODO Add release function. */
        .release=my_cdev_release,
/* TODO Add read function */
        .read=my_cdev_read,
/* TODO Add write function */
        .write=my_cdev_write
};
static int my_init(void)
{
        int err;
        int i;
    /* TODO register char device region for MY_MAJOR and NUM_MINORS starting at MY_MINOR.
        Use the register_chrdev_region function */
        err = register_chrdev_region(MKDEV(MY_MAJOR , 0), NUM_MINORS ,MODULE_NAME);
        if (err != 0) {
                pr_info("Register character device.");
                return err;
        }
        for (i = 0; i < NUM_MINORS; i++) {
                /* Initializing buffer with MESSAGE string */
                memcpy(devs[i].buffer, MESSAGE, sizeof(MESSAGE));
                devs[i].size = sizeof(MESSAGE);

                /* We set the access variable to 0 using atomic_set */
                atomic_set(&devs[i].access, 0);

                /* TODO init and add cdev to kernel core. Use cdev_init and cdev_add */
                cdev_init (&devs[i].cdev , &my_fops );
                cdev_add (&devs[i].cdev , MKDEV(MY_MAJOR , i), 1);
                if (err) {
            pr_info("Failed to add cdev for minor %d\n", i);
            unregister_chrdev_region(MKDEV(MY_MAJOR, 0), NUM_MINORS);
            return err;
        }
        }
        return 0;
}
```

```c
static void my_exit(void)
{
        int i;

        for (i = 0; i < NUM_MINORS; i++) {
                /* TODO delete cdev from kernel core using cdev_del */
                cdev_del (&devs[i].cdev);


        }

        /* TODO Unregister char device region, for MY_MAJOR and NUM_MINORS starting at MY_MINOR. Use unregister_chrdev_region */
        unregister_chrdev_region(MKDEV(MY_MAJOR , 0), NUM_MINORS );
}

module_init(my_init);
module_exit(my_exit);
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
16/1/2025