

# ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

## Δραστηριότητα 5

ΜΠΑΣΑΓΙΑΝΝΗ ΓΕΩΡΓΙΑ 1084016

ΠΑΝΟΥΡΓΙΑΣ ΑΝΤΩΝΙΟΣ 1083996

ΣΤΕΡΓΙΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 1083861

### Άσκηση 1

Μέσω του εργαλείου αναζήτησης Linux LXR, εντοπίσαμε τους ορισμούς των παρακάτω

1. **container\_of:** Στο `/include/linux/container_of.h` βρίσκουμε τον ορισμό του macro `container_of`.

```
#define container_of(ptr, type, member) ({  
    void *__mptr = (void *)(ptr);  
    static_assert(__same_type(*(ptr), ((type *)0)->member) ||  
                  __same_type(*(ptr), void),  
                  "pointer type mismatch in container_of()");  
    ((type *)(__mptr - offsetof(type, member))); })
```

Η μακροεντολή αυτή, παίρνει σαν όρισμα τον δείκτη σε ένα member ενός struct, του τύπου `type`, το οποίο είναι και το δεύτερο όρισμα της, και σαν τελευταίο όρισμα παίρνει το όνομα του μέλους αυτού μέσα στο struct.

Αυτή λοιπόν, αναλαμβάνει να «πάρει» τον δείκτη σε member ενός struct και να επιστρέψει δείκτη στο ίδιο το struct.

2. **struct file:** Στο `/include/linux/fs.h` βρίσκουμε τον ορισμό του struct `file`:

```
struct file {  
    union {  
        struct llist_node    f_llist;  
        struct rcu_head      f_rcuhead;  
        unsigned int         f_iocb_flags;  
    };  
};
```

Στα λειτουργικά συστήματα βασισμένα σε Unix, όπως το Linux, υπάρχει η ιδέα «όλα είναι ένα αρχείο». Αυτό σημαίνει πως για την επικοινωνία με περιφερειακές συσκευές μπορούμε να φανταστούμε τις συσκευές αυτές σαν απλά αρχεία κειμένου, όπου το κείμενο σε αυτή την περίπτωση είναι τα δεδομένα της συσκευής. Για αυτό, η δομή `file` αναπαριστά ένα ανοιχτό αρχείο (τη συσκευή) και εμφανίζεται μόνο σε kernel κώδικα. Καταλαβαίνουμε ότι δεν έχει καμία σχέση το `file`, με την δομή `FILE` του user-space.

3. **struct file\_operations:**

Στο `/include/linux/fs.h` βρίσκουμε επίσης το `file_operations` struct:

```

1916 struct file_operations {
1917     struct module *owner;
1918     loff_t (*llseek) (struct file *, loff_t, int);
1919     ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
1920     ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
1921     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
1922     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
1923     int (*iopoll)(struct kiocb *kiocb, struct io_comp_batch *,
1924         unsigned int flags);
1925     int (*iterate_shared) (struct file *, struct dir_context *);
1926     __poll_t (*poll) (struct file *, struct poll_table_struct *);
1927     long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
1928     long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
1929     int (*mmap) (struct file *, struct vm_area_struct *);
1930     unsigned long mmap_supported_flags;
1931     int (*open) (struct inode *, struct file *);
1932     int (*flush) (struct file *, fl_owner_t id);
1933     int (*release) (struct inode *, struct file *);
1934     int (*fsync) (struct file *, loff_t, loff_t, int datasync);
1935     int (*fasync) (int, struct file *, int);
1936     int (*lock) (struct file *, int, struct file_lock *);
1937     unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
1938     int (*check_flags)(int);
1939     int (*flock) (struct file *, int, struct file_lock *);
1940     ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
1941     ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
1942     void (*splice_eof)(struct file *file);
1943     int (*setlease)(struct file *, int, struct file_lock **, void **);
1944     long (*fallocate)(struct file *file, int mode, loff_t offset,
1945         loff_t len);
1946     void (*show_fdinfo)(struct seq_file *m, struct file *f);

```

Η δομή `file_operations` έχει ως μέλη δείκτες σε συναρτήσεις που αντιστοιχούν σε system calls. Την χρησιμοποιούμε κατά την συγγραφή ενός driver συσκευής ώστε να ορίσουμε ποιες κλήσεις συστήματος θέλουμε να εκτελούνται στην ρουτίνα του driver που πραγματοποιεί την επικοινωνία του ΛΣ με την συσκευή και τα δεδομένα της.

Στην εκφώνηση της Άσκησης βλέπουμε και ένα παράδειγμα του πώς ένας driver αξιοποιεί το `file_operations` struct για να ορίσει, ποιές κλήσεις συστήματος θα αξιοποιήσει (ή αλλιώς «υιοθετήσει») απο το `file_operations`.

```

const struct file_operations my_fops = {
    .owner = THIS_MODULE,
    .open = my_open,
    .read = my_read,
    .write = my_write,
    .release = my_release,
};

```

## Άσκηση 2

Η μετακίνηση του κέρσorra του ποντικιού στην οθόνη, είναι δουλειά του διαχειριστή παραθύρων, ο οποίος ωστόσο δεν γνωρίζει την θέση του ποντικιού και πρέπει να ρωτήσει το λειτουργικό. Το λειτουργικό όπως είπαμε και νωρίτερα μεταφράζει τις εισόδους των Input devices σε συνεχή ροή byte. Αυτά τα byte, ακολουθώντας ένα πρωτόκολλο, δείχνουν ουσιαστικά την σχετική κίνηση του ποντικιού.

Αυτά τα byte μπορούμε εμείς να τα δούμε μέσα από το αρχείο `mice` που βρίσκεται στο `/dev/input/mice` εκτελώντας τι κατάλληλες εντολές στο terminal του Linux

```

$ sudo cat /dev/input/mice
$ sudo cat /dev/input/mice | hexdump # prettier output

```

```
vboxuser@Debian-OS: ~  
0000d50 fc18 1802 02fa ff18 1800 02fb fc18 1801  
0000d60 01fe fe18 1800 00fc fc18 1802 01fe e518  
0000d70 1806 02f9 ff18 1800 01fb fd18 1801 00fe  
0000d80 fe18 1800 01ff ff18 1800 00ff ff18 1800  
0000d90 00fe ff18 1801 00ff ff18 1800 00ff ff18  
0000da0 1801 00ff ff18 1800 00ff fe18 1800 00ff  
0000db0 ff18 1800 00ff ff18 1800 00ff ff18 1800  
0000dc0 00ff ff18 1800 00ff fe18 1800 00fe ff18  
0000dd0 1800 00ff ff18 1800 00fe fc18 1800 00ff  
0000de0 ff18 1800 00fe ff18 1800 00ff ff18 1800  
0000df0 01fe ff18 1800 00fe fd18 1800 00ff ff18  
0000e00 1800 00ff ff18 1800 01ff ff18 1800 01ff  
0000e10 ff18 1800 00ff ff18 1800 00ff 0008 1801  
0000e20 00ff ff18 1800 01ff ff18 1800 01ff ff18  
0000e30 1801 01ff ff18 1800 03fd 0008 1801 01ff  
0000e40 ff18 1800 02ff ff18 0800 0100 ff18 0800  
0000e50 0100 ff18 1800 00ff 0008 1801 01ff ff18  
0000e60 0800 0100 fe18 0801 0100 ff18 1800 01ff  
0000e70 ff18 1801 01fe fd18 1802 02ff ff18 1801  
0000e80 00ff 0008 1801 00ff fa18 1805 01fe ff18  
0000e90 1801 00ff 0008 1802 00ff fe18 0800 0100  
0000ea0 ff18 1800 02fe ff18 0800 0100 fd18 1801  
0000eb0 01ff ff18 1800 01ff ff18 1801 00ff ff18
```

Παρατηρούμε εδώ ότι, με την κίνηση του ποντικιού μας, αλλάζουν οι τιμές των bytes που εκτυπώνονται στο terminal.

Τώρα θα αξιοποιήσουμε το δοθέν python script:

```
mouse.py  
  
import struct  
f = open( "/dev/input/mice", "rb" );  
  
# Open the file in the read-binary mode  
while True:  
    # Reads the 24 bytes  
    data = f.read(3)  
    # Unpack the bytes to integers  
    print(struct.unpack('bbb',data))
```

Παρατηρούμε το αποτέλεσμα:

```
vboxuser@Debian-OS: ~  
(24, -1, 1)  
(8, 0, 1)  
(24, -1, 0)  
(8, 0, 1)  
(24, -1, 1)  
(24, -1, 1)  
(24, -1, 1)  
(24, -2, 2)  
(24, -2, 1)  
(8, 0, 1)  
(24, -1, 1)  
(24, -2, 1)  
(24, -1, 1)  
(24, -1, 0)  
(24, -1, 1)  
(24, -2, 1)  
(24, -1, 0)  
(24, -1, 1)  
(24, -1, 0)  
(24, -1, 1)  
(24, -1, 0)  
(24, -1, 1)  
(24, -3, 0)  
(24, -3, 1)
```

Εδώ, οι τριάδες bytes που εκτυπώνονται αντιπροσωπεύουν τα bytes που υπάρχουν στο data-stream και δείχνουν την κατάσταση του ποντικιού σύμφωνα με κάποιο πρωτόκολλο, που ορίζει τι σημασία του κάθε bit. Υπάρχουν αρκετά διαφορετικά πρωτόκολλα.

### Άσκηση 3

Εδώ καλούμαστε να κάνουμε register μια συσκευή τύπου character στο σύστημα μας, δίνοντας της major 42 και minor 0 και όνομα mydevice. Αυτό το κάνουμε με τον παρακάτω τρόπο μέσω του terminal. Επίσης προσπαθήσαμε να γράψουμε σε αυτό μέσω του cat, πράγμα που δεν κατέστη δυνατό, επειδή δεν υπάρχει driver της συσκευής. Αυτός θα υλοποιηθεί στο επόμενο ερώτημα.



```
vboxuser@Debian-OS: ~  
root@Debian-OS:/home/vboxuser/Desktop/ask5# sudo mknod /dev/mydevice c 42 0  
root@Debian-OS:/home/vboxuser/Desktop/ask5# sudo cat /dev/mydevice  
cat: /dev/mydevice: No such device or address  
root@Debian-OS:/home/vboxuser/Desktop/ask5#
```

Τώρα θα προσπαθήσουμε να γράψουμε στην συσκευή, που για τον ίδιο λόγο με από πάνω δεν θα τα καταφέρουμε.



```
vboxuser@Debian-OS: ~  
root@Debian-OS:/home/vboxuser/Desktop/ask5# echo "HELLO BASA" > /dev/mydevice  
bash: /dev/mydevice: No such device or address  
root@Debian-OS:/home/vboxuser/Desktop/ask5#
```

## Άσκηση 4

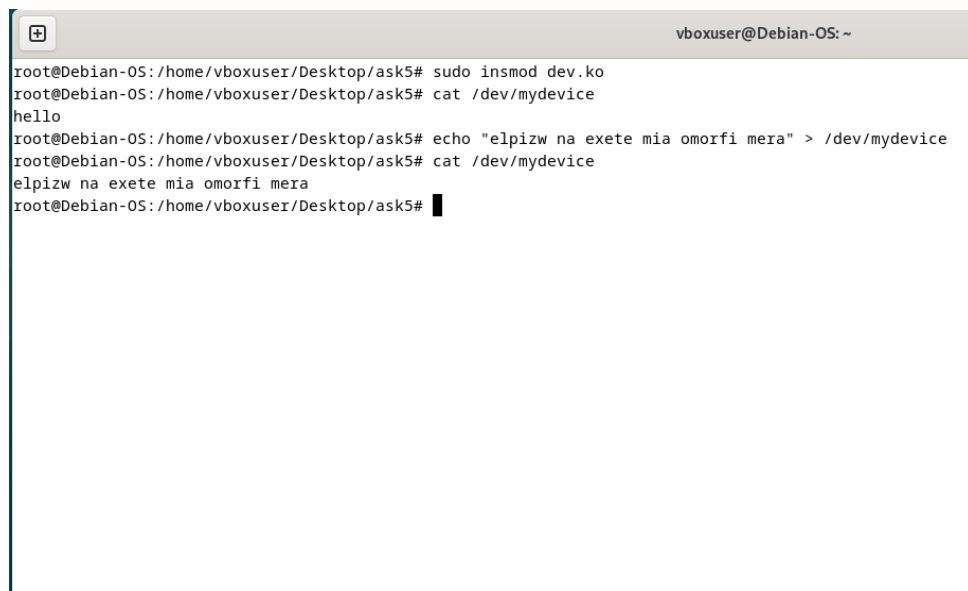
Στην άσκηση αυτή δημιουργούμε ένα kernel module το οποίο θα λειτουργήσει σαν driver του mydevice. Η συμπεριφορά της συσκευής θα είναι η εξής:

Όταν ο χρήστης διαβάζει από την συσκευή θα παίρνει σαν έξοδο ένα μήνυμα που είναι αποθηκευμένο σε μνήμη του kernel ("hello").

Όταν ο χρήστης γράφει αλλάζει το μήνυμα το οποίο τυπώνεται (άρα αλλάζει το τι υπάρχει αποθηκευμένο στην μνήμη του kernel).

Εφόσον έχουμε φορτώσει την συσκευή και έχουμε κάνει make (compile δηλαδή το dev.c) αρχείο μας, φορτώνουμε το module και μέσω της cat (read) και της echo (για write) παρατηρούμε την ορθή λειτουργία της συσκευής μας.

Κάνουμε πρώτα ένα read όπου βλέπουμε το default μήνυμα "hello" και έπειτα γράφουμε κάτι άλλο, συγκεκριμένα «elpizw na exete mia omorfi mera». Εκτελώντας λοιπόν ξανά read (cat) βλέπουμε το νέο μήνυμα.

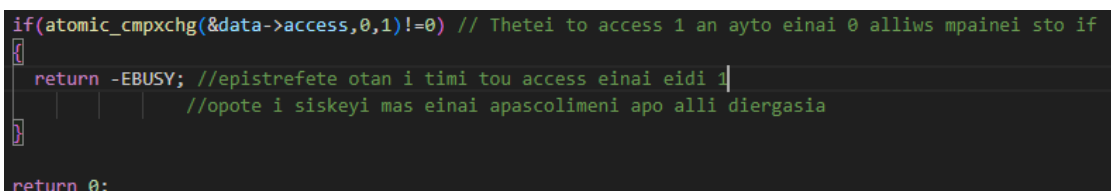


```
vboxuser@Debian-OS: ~
root@Debian-OS:/home/vboxuser/Desktop/ask5# sudo insmod dev.ko
root@Debian-OS:/home/vboxuser/Desktop/ask5# cat /dev/mydevice
hello
root@Debian-OS:/home/vboxuser/Desktop/ask5# echo "elpizw na exete mia omorfi mera" > /dev/mydevice
root@Debian-OS:/home/vboxuser/Desktop/ask5# cat /dev/mydevice
elpizw na exete mia omorfi mera
root@Debian-OS:/home/vboxuser/Desktop/ask5#
```

Ο τροποποιημένος κώδικας του dev.c με τις απαραίτητες αλλαγές που χρειάστηκαν για να λειτουργήσει όπως παραπάνω, καθώς και το Makefile, περιέχονται στο ίδιο .zip αρχείο, με αυτήν την αναφορά και συγκεκριμένα στο φάκελο src.

Οι τροποποιήσεις που κάναμε βασίστηκαν στους κώδικες-παραδείγματα της εκφώνησης της άσκησης.

Παραθέτουμε μια προσθήκη στον κώδικα, με τα κατάλληλα επεξηγηματικά σχόλια, η οποία δεν βασίστηκε στην εκφώνηση της άσκησης.

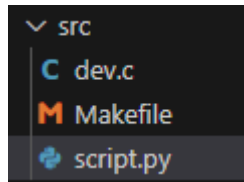


```
if(atomic_cmpxchg(&data->access,0,1)!=0) // Thetei to access 1 an ayto einai 0 alliws mpainei sto if
{
    return -EBUSY; //epistrefete otan i timi tou access einai eidi 1
                //opote i siskeyi mas einai apascolimeni apo alli diergasia
}

return 0;
```

## DEMO

Φτιάξαμε λοιπόν τώρα ένα πρόγραμμα σε python (script.py) το οποίο τοποθετούμε μαζί με το αρχείο του driver μας (dev.c) και το Makefile στο ίδιο folder.



```
script.py
1 import subprocess
2
3 def register_device(device_name):
4     register_device=f"sudo mknod /dev/{device_name} c 42 0"
5     subprocess.run(register_device,shell=True)
6     make=f"make"
7     subprocess.run(make)
8     load_module=f"sudo insmod dev.ko"
9     subprocess.run(load_module,shell=True)
10
11 def remove_device(device_name):
12     remove_device=f"sudo rm /dev/{device_name}"
13     subprocess.run(remove_device,shell=True)
14     remove_module=f"sudo rmmmod dev"
15     subprocess.run(remove_module,shell=True)
16
17 def write_device(device_name,msg):
18     write_device=f"echo {msg} > /dev/{device_name}"
19     subprocess.run(write_device,shell=True)
20
21
22 def read_device(device_name):
23     read_device=f"cat /dev/{device_name}"
24     subprocess.run(read_device,shell=True)
25
26 register_device("mydevice")
27 while 1:
28     read_device("mydevice")
29     newmsg=str(input("Enter the new msg for the device (0 for exit): "))
30     if(newmsg=="0"):
31         break
32     write_device("mydevice",newmsg)
33
34 remove_device("mydevice")
35 print(["Bye Bye"])
```

Εδώ πέρα όταν τρέχει το πρόγραμμα κάνει register την συσκευή, compile το dev.c, σύμφωνα με το Makefile, και φορτώνει το module dev.ko (driver της συσκευής). Έπειτα δείχνει στον χρήστη το μήνυμα της και του δίνει την δυνατότητα να το αλλάξει. Τώρα, τυπώνει το νέο μήνυμα που έχει πλέον αποθηκευμένο η συσκευή στο kernel memory. Όταν ο χρήστης, δώσει σαν input το 0, το loop τερματίζεται, το module αφαιρείται και η συσκευή διαγράφεται. Έτσι έχουμε αποδείξει την ορθή λειτουργία της συσκευής μας.

Παράδειγμα εκτέλεσης:

```
vboxuser@Debian-OS: ~
root@Debian-OS: /home/vboxuser/Desktop/ask5# sudo python3 script.py
make -C /lib/modules/6.1.0-13-amd64/build M=/home/vboxuser/Desktop/ask5 modules

make[1]: Entering directory '/usr/src/linux-headers-6.1.0-13-amd64'
CC [M] /home/vboxuser/Desktop/ask5/dev.o
MODPOST /home/vboxuser/Desktop/ask5/Module.symvers
CC [M] /home/vboxuser/Desktop/ask5/dev.mod.o
LD [M] /home/vboxuser/Desktop/ask5/dev.ko
BTF [M] /home/vboxuser/Desktop/ask5/dev.ko
Skipping BTF generation for /home/vboxuser/Desktop/ask5/dev.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-13-amd64'
hello
Enter the new msg for the device (0 for exit): kalispera
kalispera
Enter the new msg for the device (0 for exit): geia sas
geia sas
Enter the new msg for the device (0 for exit): 0
Bye Bye
root@Debian-OS: /home/vboxuser/Desktop/ask5#
```