Λειτουργικά Συστήματα Δραστηριότητα 5

2024-2025

AM: 1096060

Ονοματεπώνυμο: Μαρία-Νίκη Ζωγράφου

Περιεχόμενα

Ερώτημα 1

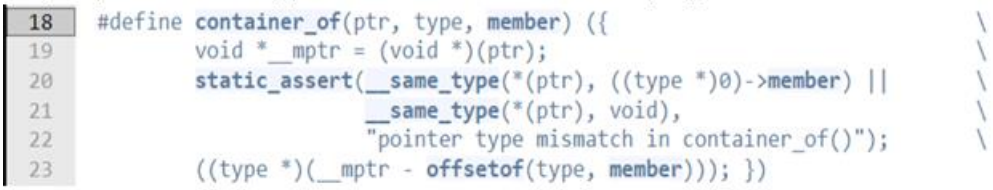
Άσκηση 1

Βρείτε στον πηγαίο κώδικα του Kernel τους ορισμούς των συμβόλων. Μπορείτε να χρησιμοποιήσετε το εργαλείο αναζήτησης LXR.

1. `container_of`
2. `struct file`
3. `struct file_operations`

Εξηγήστε τη σημασία τους.

1. Το `container_of` είναι ένα macro που στον πυρήνα του Linux με σκοπό να υπολογίσει τη διεύθυνση μιας δομής (structure) που "περιέχει" ένα μέλος της, όταν δίνουμε τη διεύθυνση του μέλους αυτού ως input.
2. Η **`struct file`** είναι μια δομή που χρησιμοποιείται στον πυρήνα του Linux για να αντιπροσωπεύει ένα **ανοικτό αρχείο**. Αυτή η δομή περιέχει όλες τις απαραίτητες



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

πληροφορίες για τη διαχείριση ενός αρχείου που έχει ανοιχτεί από έναν χρήστη ή ένα σύστημα.

```
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;
#ifdef CONFIG_SECURITY
    void                *f_security;
#endif
```

```
    /* --- cacheline 2 boundary (128 bytes) --- */
    struct fown_struct      *f_owner;
    errseq_t            f_wb_err;
    errseq_t            f_sb_err;
#ifdef CONFIG_EPOLL
    struct hlist_head       *f_ep;
#endif
    union {
        struct callback_head    f_task_work;
        struct llist_node   f_llist;
        struct file_ra_state    f_ra;
        freeptr_t       f_freeptr;
    };
    /* --- cacheline 3 boundary (192 bytes) --- */
} __randomize_layout
  __attribute__((aligned(4)));  /* lest something weird decides that
2 is OK */
```

Η **`struct file`** περιγράφει ένα συγκεκριμένο άνοιγμα του αρχείου (open instance) και δεν αναφέρεται στο ίδιο το αρχείο στο σύστημα αρχείων. Περιλαμβάνει το struct inode το οποίο αναφέρεται στο ίδιο το αρχείο στο σύστημα αρχείων. Περιέχει πληροφορίες όπως το μέγεθος, τα δικαιώματα, και το αναγνωριστικό (inode number).

3. Η **`struct file_operations`** είναι μια δομή δεδομένων στον πυρήνα του Linux που ορίζει ένα σύνολο λειτουργιών/συναρτήσεων που μπορούν να εκτελεστούν σε ένα αρχείο. Περιέχει δείκτες σε συναρτήσεις, οι οποίες χρησιμοποιούνται για τη διαχείριση του αρχείου, όπως το άνοιγμα, το κλείσιμο, η ανάγνωση και η εγγραφή.

```
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 *);
    int (*iopoll)(struct kiocb *kiocb, struct io_comp_batch *,
            unsigned int flags);
    int (*iterate_shared) (struct file *, struct dir_context *);
    __poll_t (*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 (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    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);
    void (*splice_eof)(struct file *file);
    int (*setlease)(struct file *, int, struct file_lease **, 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
    ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
            loff_t, size_t, unsigned int);
    loff_t (*remap_file_range)(struct file *file_in, loff_t pos_in,
                   struct file *file_out, loff_t pos_out,
                   loff_t len, unsigned int remap_flags);
    int (*fadvise)(struct file *, loff_t, loff_t, int);
    int (*uring_cmd)(struct io_uring_cmd *ioucmd, unsigned int
issue_flags);
    int (*uring_cmd_iopoll)(struct io_uring_cmd *, struct
io_comp_batch *,
                unsigned int poll_flags);
} __randomize_layout;
```

<u>Σημείωση:</u> Οι εντολές **`#ifndef`** και **`#endif`** στη C είναι μέρος του preprocessor, which processes the source code before it is compiled. ο οποίος επεξεργάζεται τον πηγαίο κώδικα πριν από τη μεταγλώττιση. Αυτές οι εντολές ελέγχουν το **conditional compilation** (υπό όρους μεταγλώττιση), επιτρέποντας να συμπεριληφθούν ή να αποκλειστούν συγκεκριμένα τμήματα του κώδικα ανάλογα με συγκεκριμένες συνθήκες. Η εντολή **`#ifndef`** σημαίνει "αν δεν έχει οριστεί" και ελέγχει αν μια συγκεκριμένη μακροεντολή ή σταθερά δεν έχει οριστεί. Αν η μακροεντολή δεν έχει οριστεί, ο κώδικας μεταξύ των **`#ifndef`** και **`#endif`** περιλαμβάνεται στη μεταγλώττιση. Αντίθετα, αν έχει οριστεί, ο κώδικας παραλείπεται. Η εντολή `#endif` σηματοδοτεί το τέλος του μπλοκ που ξεκινά με **`#ifndef`**, **`#ifdef`** ή **`#if`** .

Ερώτημα 2

Άσκηση 2

Μία εφαρμογή όπως ο διαχειριστής παραθύρων μας είναι υπεύθυνος για λειτουργίες όπως η εμφάνιση του κέρσορα του ποντικιού.

Όταν εμείς μετακινούμε τη φυσική συσκευή του ποντικιού βλέπουμε τον κέρσορα να μετακινείται.

Ο διαχειριστής παραθύρων δεν γνωρίζει στην πραγματικότητα τίποτα για το ποντίκι μας.

Κάθε ένα καθορισμένο χρονικό διάστημα η εφαρμογή αυτή ρωτάει το λειτουργικό σύστημα για να δει αν έχει μετακινηθεί το ποντίκι.

Το λειτουργικό σύστημα έχει μεταφράσει την είσοδο από τη συσκευή του ποντικιού σε μία ροή από bytes. Τα bytes αυτά ακολουθούν ένα συγκεκριμένο πρωτόκολλο μέσα από το οποίο περιγράφεται ουσιαστικά η κατεύθυνση στην οποία κινήθηκε το ποντίκι. Η ροή από bytes φαίνεται στο σύστημά σας ένα απλό αρχείο, το οποίο αν το διαβάσουμε θα δούμε μπορστά μας τη σειρά από bytes που περιγράφουν την κίνηση του ποντικιού μας ανα πάσα στιγμή.

Τελικά, αυτό το αρχείο το διαβάζει ο διαχειριστής παραθύρων και τελικά ζωγραφίζει το ποντίκι στη νέα θέση, όπως θα έπρεπε.

Μπορείτε και τώρα να δείτε αυτό το byte stream χρησιμοποιώντας την παρακάτω εντολή και έπειτα κουνώντας το ποντίκι σας. (Για πιο όμορφη έξοδο κατεβάστε την εφαρμογή hexdump εκτελώντας `sudo apt install hexdump`).

Τί παρατηρείται;

Έπειτα, εκτελέστε το παρακάτω python script το οποίο ερμηνεύει την έξοδο της συσκευής ποντικιού σαν τριάδες από bytes.

Ποια είναι η έξοδος; Ερμηνεύστε την.

Το `sudo apt install hexdump`: **E:** Unable to locate package hexdump. Επομένως χρησιμοποιήθηκε το `sudo apt-get install bsdmainutils`.

Εκτελούμε την εντολή `sudo cat /dev/input/mice | hexdump`

Παίρνουμε σαν αποτέλεσμα το stream από bytes από την κίνηση του ποντικιού!!

```
root@debian:/home/manya# sudo cat /dev/input/mice | hexdump
0000000 9938 2881 cc00 0d08 0805 050a 0908 0804
0000010 0307 0408 0801 0103 0308 0802 0105 0508
0000020 0802 0205 0208 0801 0104 0108 0800 0101
0000030 0108 0800 0101 0108 0800 0001 0208 0800
0000040 0104 0a08 0800 0009 1708 0801 000e 1208
0000050 0801 0013 1608 0800 0010 1028 28fd fc08
0000060 0828 28fc ff04 0228 28fe ff02 0208 2800
0000070 ff00 0028 28ff fd00 fb38 38fc fbfe fa38
0000080 38fa fbf9 f738 38fb fcf4 ed38 38f8 faeb
0000090 ea38 38f9 f4de de38 38f5 f3cf c938 38f0
00000a0 f8db da38 38fb fee2 e238 18ff 00ed ed18
00000b0 1800 00f1 f318 1801 03f6 fb18 1804 04f7
00000c0 fa18 1806 0bfa f818 180d 13fa f618 1813
00000d0 13fb fe18 0818 0f00 0008 0812 1705 0808
00000e0 0813 110b 0a08 080d 0908 0908 0808 0608
00000f0 0608 0802 0406 0708 0801 0104 0308 0801
0000100 0002 0208 0800 0001 0108 0800 0002 0128
0000110 28ff ff01 0228 28fd ff02 0328 28fd fd01
0000120 0528 28fd fb04 0528 28fd f902 0528 28fc
```

Python script:

Το `f.read(3)` διαβάζει **3 bytes κάθε φορά** από τη συσκευή ποντικιού. Αυτά τα bytes περιέχουν πληροφορίες για τις κινήσεις και τις ενέργειες του ποντικιού.

Το struct.unpack('bbb', data) μετατρέπει τα 3 bytes σε υπογεγραμμένους ακέραιους (signed integers). Το αποτέλεσμα εκτυπώνεται ως τριάδα ακέραιων αριθμών.

Δομή των δεδομένων από το `/dev/input/mice`:

Πρώτο byte (flags):

Περιέχει πληροφορίες για τα κουμπιά του ποντικιού:
Bit 0: Αριστερό κουμπί (1 = πατημένο).
Bit 1: Δεξί κουμπί (1 = πατημένο).
Bit 2: Μεσαίο κουμπί (1 = πατημένο).
Περιλαμβάνει επίσης πληροφορίες για την κατεύθυνση της κίνησης.

Δεύτερο byte (x):

Δείχνει τη μετατόπιση στον άξονα X (αριστερά ή δεξιά) και είναι προσημασμένος ακέραιος αριθμός (π.χ., θετικός για δεξιά, αρνητικός για αριστερά).

Τρίτο byte (y):

Δείχνει τη μετατόπιση στον άξονα Y (πάνω ή κάτω) και είναι προσημασμένος ακέραιος αριθμός (π.χ., θετικός για κάτω, αρνητικός για πάνω).

```
manya@debian:~$ sudo python3 mouse.py
(40, 127, -124)
(8, 42, 0)
(8, 1, 7)
(24, -1, 8)
(24, -4, 8)
(24, -3, 7)
(24, -4, 8)
(24, -6, 9)
(24, -8, 9)
(24, -5, 8)
(24, -13, 12)
(24, -9, 10)
(24, -8, 9)
(24, -8, 8)
(24, -8, 6)
(24, -5, 4)
(24, -6, 2)
(24, -6, 3)
(24, -6, 1)
(24, -5, 0)
(24, -6, 0)
(56, -8, -5)
(56, -12, -8)
(56, -11, -9)
(56, -21, -18)
(56, -26, -26)
(56, -27, -26)
(56, -19, -17)
(56, -26, -29)
(56, -17, -19)
(56, -10, -14)
(56, -6, -13)
(56, -3, -12)
(56, -2, -10)
(40, 0, -6)
```

Ερώτημα 3

Άσκηση 3

Κάντε register ένα character device στο σύστημά σας χρησιμοποιώντας την εντολή `mknod`. Δώστε της major αριθμό 42, minor αριθμό 0 και όνομα `mydevice`.

Προσπαθήστε να γράψετε και να διαβάσετε στη συσκευή. Τί συμβαίνει;

```
manya@debian:~$ sudo mknod /dev/mydevice c 42 0
[sudo] password for manya:
manya@debian:~$ cat /dev/mydevice
cat: /dev/mydevice: No such device or address
manya@debian:~$ echo "test" > /dev/mydevice
bash: /dev/mydevice: Permission denied
manya@debian:~$
```

Επειδή δεν έχουμε υλοποιήσει ή συνδέσει κάποιον driver για το major αριθμό 42, το σύστημα δεν ξέρει πώς να διαχειριστεί αυτή τη συσκευή και εμφανίζει σφάλμα.

Το αρχείο συσκευής που δημιουργήσαμε με την εντολή `mknod` δεν έχει συσχετιστεί με κάποιον υπαρκτό driver. Όταν εκτελούμε την εντολή `mknod /dev/mydevice c 42 0`, δημιουργούμε ένα **character device file** με major αριθμό 42 και minor αριθμό 0. Ωστόσο, το λειτουργικό σύστημα αναμένει ότι υπάρχει ένας driver καταχωρημένος στον πυρήνα για τη διαχείριση συσκευών με τον συγκεκριμένο major αριθμό. Επειδή δεν έχουμε υλοποιήσει και φορτώσει έναν τέτοιο driver, οι λειτουργίες ανάγνωσης (`read`) και εγγραφής (`write`) δεν μπορούν να πραγματοποιηθούν, με αποτέλεσμα να εμφανίζονται σφάλματα όπως `Permission Denied` ή `No such device or address`.

Ερώτημα 4

Άσκηση 4

Φτιάξτε ένα kernel module το οποίο θα λειτουργήσει σαν driver της συσκευής που ορίσατε. Ο χρήστης θα μπορεί να διαβάσει από και να γράψει στη συσκευή. Συγκεκριμένα, όταν ο χρήστης διαβάζει από τη συσκευή θα παίρνει σαν έξοδο ένα μήνυμα που είναι αποθηκευμένο σε μνήμη που ανήκει στον πυρήνα. Όταν γράφει ο χρήστης στη συσκευή θα μπορεί να αλλάξει το μήνυμα το οποίο τυπώνεται.

Βασιστείτε στον σκελετό που δίνεται στο αρχείο **`mychardev.c`**.

Info: Μόλις συμπληρώσετε τον κώδικα. Κάντε τον compile και φορτώστε το module `chardev.ko` στον πυρήνα σας. Πλέον, θα μπορείτε να αλληλεπιδράσετε με την εικονική συσκευή που φτιάξατε στο προηγούμενο ερώτημα μέσω του driver.

Έπειτα, φτιάξτε ένα αρχείο σε όποια γλώσσα προγραμματισμού θέλετε που να επιδεικνύει τη λειτουργία της συσκευής σας.

Ο κώδικας που παρέχεται στο αρχείο dev.c είναι ο σκελετός για τη δημιουργία ενός character device driver. Ο driver αυτός θα επιτρέπει στον χρήστη να διαβάζει και να γράφει δεδομένα στη συσκευή, αποθηκεύοντας το μήνυμα στον buffer που ανήκει στη δομή του driver.

Συμπληρώσεις στον κώδικα του chardev.c:

1. Προσθήκη του μέλους `cdev` και του buffer στη δομή `my_device_data`: Στη δομή `my_device_data`, προσθέτουμε:

```
struct cdev cdev; // Χειριστής character device
char buffer[BUFFER_SIZE]; // Buffer αποθήκευσης δεδομένων
```

2. Χρήση του `container_of` για απόκτηση του δείκτη στη δομή: Στη συνάρτηση `my_cdev_open`, συμπληρώνουμε:

```
data = container_of(inode->i_cdev, struct my_device_data, cdev);
```

3. Διαχείριση της μεταβλητής `access`: Στη συνάρτηση `my_cdev_open`, ελέγχουμε την πρόσβαση:

```
if (atomic_cmpxchg(&data->access, 0, 1) != 0) {
    return -EBUSY; // Συσκευή είναι ήδη σε χρήση
}
```

τη συνάρτηση `my_cdev_release`, επαναφέρουμε την πρόσβαση:

```
atomic_set(&data->access, 0);
```

4. Αντιγραφή δεδομένων από και προς τον buffer:

Στη `my_cdev_read`:

```
if (copy_to_user(user_buffer, data->buffer + *offset, to_read)) {
    return -EFAULT; // Σφάλμα κατά την αντιγραφή
}
```

Στη `my_cdev_write`:

```
if (copy_from_user(data->buffer + *offset, user_buffer, size)) {
    return -EFAULT; // Σφάλμα κατά την αντιγραφή
}
```

5. Προσθήκη συναρτήσεων στη δομή `my_fops`:

.open = my_cdev_open,
.release = my_cdev_release,
.read = my_cdev_read,
.write = my_cdev_write,

6. Προσθήκη συναρτήσεων στη δομή `my_fops`:

Στη συνάρτηση `my_init`:

```
err = register_chrdev_region(MKDEV(MY_MAJOR, MY_MINOR), NUM_MINORS,
MODULE_NAME);
if (err != 0) {
```

```
    pr_info("Register character device failed.");
    return err;
}

for (i = 0; i < NUM_MINORS; i++) {
    cdev_init(&devs[i].cdev, &my_fops);
    devs[i].cdev.owner = THIS_MODULE;
    err = cdev_add(&devs[i].cdev, MKDEV(MY_MAJOR, MY_MINOR + i), 1);
    if (err) {
        pr_info("Add cdev failed.");
        return err;
    }
}
```

Στη συνάρτηση `my_exit`:

```
for (i = 0; i < NUM_MINORS; i++) {
    cdev_del(&devs[i].cdev);
}
unregister_chrdev_region(MKDEV(MY_MAJOR, MY_MINOR), NUM_MINORS);
```

Επιβεβαίωση ότι το module μας λειτουργεί ορθά:

```
root@debian:/media/sf_OS3# sudo insmod dev.ko
root@debian:/media/sf_OS3# lsmod | grep dev
dev                    24576  0
snd_seq_device         16384  1 snd_seq
snd                                                                    126976         10
snd_seq,snd_seq_device,snd_intel8x0,snd_timer,snd_ac97_codec,snd_pcm
joydev                 28672  0
evdev                  28672  10
ppdev                  24576  0
parport                73728  3 parport_pc,lp,ppdev
root@debian:/media/sf_OS3# sudo mknod /dev/mydevice c 42 0
mknod: /dev/mydevice: File exists
root@debian:/media/sf_OS3# sudo chmod 666 /dev/mydevice
root@debian:/media/sf_OS3# echo "Νέο μήνυμα!" > /dev/mydevice
root@debian:/media/sf_OS3# cat /dev/mydevice
Νέο μήνυμα!
root@debian:/media/sf_OS3# sudo rmmod dev
```

Ο driver μας είναι λειτουργικός και μπορούμε να γράψουμε και να διαβάσουμε από την συσκευή. Θα το επιβεβαιώσουμε και με εφαρμογή σε python.

Δημιουργία εφαρμογής χρήστη:

```
import os

DEVICE_PATH = "/dev/mydevice"

def write_to_device(message):
    """Γράφει ένα μήνυμα στη συσκευή."""
    try:
        with open(DEVICE_PATH, "w") as device:
            device.write(message)
        print(f"Γράφτηκε το μήνυμα: '{message}' στη συσκευή.")
    except Exception as e:
        print(f"Σφάλμα κατά την εγγραφή: {e}")

def read_from_device():
    """Διαβάζει δεδομένα από τη συσκευή."""
    try:
        with open(DEVICE_PATH, "r") as device:
            data = device.read()
        print(f"Διαβάστηκε από τη συσκευή: '{data}'")
    except Exception as e:
        print(f"Σφάλμα κατά την ανάγνωση: {e}")

if __name__ == "__main__":
    print("1. Γράψτε δεδομένα στη συσκευή.")
    print("2. Διαβάστε δεδομένα από τη συσκευή.")
    choice = input("Δώστε επιλογή (1/2): ")

    if choice == "1":
        message = input("Δώστε το μήνυμα που θέλετε να γράψετε στη
συσκευή: ")
        write_to_device(message)
    elif choice == "2":
        read_from_device()
    else:
        print("Μη έγκυρη επιλογή.")
```

Τρέχουμε την εφαρμογή και επιβεβαιώνουμε ότι λειτουργεί σωστά:

```
root@debian:/media/sf_OS3# sudo insmod dev.ko
root@debian:/media/sf_OS3# lsmod | grep dev
dev                    24576  0
snd_seq_device         16384  1 snd_seq
snd                   126976  10 snd_seq,snd_seq_device,snd_intel8x0,snd_timer,s
nd_ac97_codec,snd_pcm
joydev                 28672  0
evdev                  28672  10
ppdev                  24576  0
parport                73728  3 parport_pc,lp,ppdev
root@debian:/media/sf_OS3# python3 tapp.py
1. Γράψτε δεδομένα στη συσκευή.
2. Διαβάστε δεδομένα από τη συσκευή.
Δώστε επιλογή (1/2): 1
Δώστε το μήνυμα που θέλετε να γράψετε στη συσκευή: I love spaghetti!
Γράφτηκε το μήνυμα: 'I love spaghetti!' στη συσκευή.
root@debian:/media/sf_OS3# python3 tapp.py
1. Γράψτε δεδομένα στη συσκευή.
2. Διαβάστε δεδομένα από τη συσκευή.
Δώστε επιλογή (1/2): 2
Διαβάστηκε από τη συσκευή: 'I love spaghetti!'
root@debian:/media/sf_OS3#
```