oslab2

- Παναγιώτα-Νικολέττα Μπάρμπα - el18604
- Ευαγγελάτος Ανδρέας - el18069

Parameter

Το lunix εχει parameter :

```
int lunix_sensor_cnt
```

το οποίο ρυθμίζουμε κατά το *insmod*:

```
> insmod lunix.ko lunix_sensor_cnt=<number>
```

Έχουμε k αισθητήρες οι οποίοι έχουν έως 8 διαφορετικές μετρήσεις. Αυτές εδώ είναι batt (battery) 0, temp (temperature) 1, light (brightness) 2, και κάθε μέτρηση πρέπει να εμφανίζεται ως διαφορετική συσκευή στο σύστημα.

Αυτές έχουν Major Number 60 και Minor Numbers (sensor_id * 8 + measurement_num).

---

init()

Από τον *lunix_sensor_cnt* που καθορίζει το *module.c* έχουμε τον αριθμό των συσκευών που αφορούν μετρήσεις από *lunix_sensor_cnt * 8* (αν και υποστηρίζουμε μόνο 3 ανα sensor). Οπότε με το *register_chrdev_region()* ορίζουμε ποιες συσκευές χειριζόμαστε **Major Number 60 και Minor Numbers (sensor_id * 8 + measurement_num)**.

Ο πυρήνας θέλει να ορίσουμε και μία δομή *cdev* με την οποία συσχετίζει τις κλήσεις συστήματος με τις δικές μας ορισμένες συναρτήσεις και για ποιες συσκευες αυτές αναφέρονται.

Η δομή όπου ορίζει την συσχέτιση των system calls στον χώρο χρήστη με τις συναρτήσεις του driver μας.

```
static struct file_operations lunix_chrdev_fops =
{
    .owner          = THIS_MODULE,
    .open           = lunix_chrdev_open,
    .release        = lunix_chrdev_release,
    .read           = lunix_chrdev_read,
    .unlocked_ioctl = lunix_chrdev_ioctl,
    .mmap           = lunix_chrdev_mmap
};
```

Η υλοποίηση της init :

```
int lunix_chrdev_init(void)
{
    /*
     * Register the character device with the kernel, asking for
     * a range of minor numbers (number of sensors * 8 measurements / sensor)
     * beginning with LINUX_CHRDEV_MAJOR:0
     */
    int ret;
    dev_t dev_no;
    unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;

    debug("initializing character device\n");
    cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
    lunix_chrdev_cdev.owner = THIS_MODULE;

    dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);

    /* register_chrdev_region? */
    ret = register_chrdev_region(dev_no, lunix_minor_cnt, "lunix");
    if (ret < 0) {
        debug("failed to register region, ret = %d\n", ret);
        goto out;
    }

    /* cdev_add? */

    ret = cdev_add(&lunix_chrdev_cdev, dev_no, lunix_minor_cnt);


    if (ret < 0) {
        debug("failed to add character device\n");
        goto out_with_chrdev_region;
    }
    debug("completed successfully\n");
    return 0;

out_with_chrdev_region:
    unregister_chrdev_region(dev_no, lunix_minor_cnt);
out:
    return ret;
}
```

---

exit()

Αφαιρούμε resources που αρχικοποιούνται στην init() ώστε να απελευθερωθούν κατά την αφαίρεση του module από τον πυρήνα.

Η υλοποίηση της exit():

```
void lunix_chrdev_destroy(void)
{
    dev_t dev_no;
    unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;

    debug("entering\n");
    dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
    cdev_del(&lunix_chrdev_cdev);
    unregister_chrdev_region(dev_no, lunix_minor_cnt);
    debug("leaving\n");
}
```

---

open()

```
    int open (struct inode *, struct file *)
```

Χρησιμοποιούμε

```
    unsigned int iminor(struct inode *inode);
```

ώστε να μπορούμε να ανακτήσουμε τον minor number του αρχείου, ώστε το ανοιχτό αρχείο που θα προκύψει να αφορά συγκεκριμένο αισθητήρα και μέτρηση του δικτύου. Ο πυρήνας αρχικοποιεί κάποια δομή **file** και την περνάει στην open. Αυτή έχει τα εξής πεδία που μας ενδιαφέρουν

```
    struct file{
        mode_t f_mode; //Πρόσβαση
        loff_t f_pos;  //Δείκτης εγγραφής/ανάγνωσης.
        unsigned short f_flags; //Flags O_NONBLOCK κ.α.
        struct file_operations *f_op; //Σύνδεση με το
        //file operations του driver. Το κάνει ήδη.
        void *private_data; //Pointer για δεδομένα μας
        //που θα δείχνει σε δομή
        //lunix_chrdev_state_struct
    };
```

Κρατάμε ξεχωριστό

```
    //enum lunix_msr_enum { BATT = 0, TEMP, LIGHT, N_LUNIX_MSR };
```

```
    /*
     * Private state for an open character device node
     */
struct lunix_chrdev_state_struct {
    enum lunix_msr_enum type; //Τι μέτρηση είναι
    struct lunix_sensor_struct *sensor; //Pointer
    //στον sensora που αφορά η μέτρηση

    /* A buffer used to hold cached textual info */
    int buf_lim;
    unsigned char buf_data[LUNIX_CHRDEV_BUFSZ];
    uint32_t buf_timestamp;

    //εσωτερικό lock.
    struct semaphore lock;

};
```

Για κάθε ανοιχτό αρχείο κάνουμε allocate ένα ξεχωριστό *lunix_chrdev_state_struct* και αρχικοποιούμε τα πεδία του.

Η υλοποίηση της open:

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
    /* Declarations */
    unsigned int mes_id, mes_type, sensor_id;

    int ret;

    debug("entering\n");
    ret = -ENODEV;
    if ((ret = nonseekable_open(inode, filp)) < 0)
        goto out;

    /*
     * Associate this open file with the relevant sensor based on
     * the minor number of the device node [/dev/sensor<NO>-<TYPE>]
     */

    mes_id = iminor(inode);
    sensor_id = mes_id >> 3;
    mes_type = mes_id & 0x7;

    /*
     * Check that the device is of the ones we handle.
    */
    if(sensor_id >= lunix_sensor_cnt || mes_type >= N_LUNIX_MSR){
        ret = -ENODEV;
        goto out;
    }
```

```
    //to lunix_sensor_struct * --> lunix_sensors[sensor_id];

    /* Allocate a new Lunix character device private state structure */
    struct lunix_chrdev_state_struct* new_chdev_state;

    new_chdev_state = (struct lunix_chrdev_state_struct* )kmalloc(sizeof(struct
lunix_chrdev_state_struct), GFP_KERNEL);

    if(!new_chdev_state){
        debug("failed to allocate memory for chrdev_state\n");
        goto out;
    }
    //Set state parameters
    new_chdev_state->type = mes_type;
    new_chdev_state->sensor = &lunix_sensors[sensor_id];
    new_chdev_state->buf_lim = 1;
    new_chdev_state->buf_data[0] = '\0';
    new_chdev_state->buf_timestamp = 0;

    //initialize semaphore to avoid race conditions.
    sema_init(&new_chdev_state->lock, 1);

    //pass it to private data
    filp->private_data = new_chdev_state;
    debug("successfuly opened\n");

out:
    debug("leaving, with ret = %d\n", ret);
    return ret;
}
```

---

read()

επιστρέφει έναν αριθμό από bytes που αναπαριστούν την πιο πρόσφατη τιμή του μετρούμενου μεγέθους, σε αναγνώσιμη μορφή, μορφοποιημένη ως δεκαδικό αριθμό *(xx.yyy)*

Αν δεν έχουν ακόμη παραληφθεί δεδομένα για τη συγκεκριμένη μέτρηση, ή αν γίνουν επόμενες προσπάθειες ανάγνωσης, ο οδηγός κοιμίζει τη διεργασία έως ότου παραληφθούν νέα δεδομένα.

Μπλοκάρισμα

Όταν δεν έχουν ληφθεί δεδομένα, ο οδηγός κοιμίζει τη διεργασία που τον κάλεσε. Αυτή μετακινείται σε χωριστή ουρά διεργασιών, ανάλογα με τον αισθητήρα απ' όπου περιμένει να φτάσουν δεδομένα.

Κάθε sensor struct έχει ένα πεδίο

```
    wait_queue_head_t wq;
```

Όπου κρατώνται οι διεργασίες που ζητούν ανάγνωση και δεν υπάρχουν δεδομένα, και άρα κοιμούνται με την **wait_event_interruptible()**. Οι διεργασίες συνεχίζουν την εκτέλεσή τους όταν μεταφερθούν νέα δεδομένα για τον συγκεκριμένο αισθητήρα και ξυπνήσουν από την ουρά με την κλήση **wake_up_interruptible(&s->wq)**. Όταν τα δεδομένα είναι, λοιπόν, έτοιμα παίρνονται από την read και αντιγράφονται στον χώρο χρήστη.

Η υλοποίηση της read και των συναρτήσεων που αυτή χρησιμοποιεί:

```
static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf, size_t
cnt, loff_t *f_pos)
{


    ssize_t ret = 0;

    struct lunix_sensor_struct *sensor;
    struct lunix_chrdev_state_struct *state;

    state = filp->private_data;
    WARN_ON(!state);

    sensor = state->sensor;
    WARN_ON(!sensor);

    /* Lock */


    if (down_interruptible(&state->lock))
        return -ERESTARTSYS;



    /*
     * If the cached character device state needs to be
     * updated by actual sensor data (i.e. we need to report
     * on a "fresh" measurement, do so
     */
    if (*f_pos == 0) {
        //if we enter here it means all data are consumed
        while (lunix_chrdev_state_update(state) == -EAGAIN) {

            up(&state->lock); //release the lock.
            /* The process needs to sleep */
            /* See LDD3, page 153 for a hint */

            //if no data.
            //mpes sthn oura tou sensor kai 9a se 3upnhsei to line_disciple
            if(wait_event_interruptible(sensor->wq,
lunix_chrdev_state_needs_refresh(state)))
                return -ERESTARTSYS;

            //reacquire lock
```

```
            if (down_interruptible(&state->lock))
                return -ERESTARTSYS;
        }
    }


    /* Determine the number of cached bytes to copy to userspace */

    if(*f_pos >= state->buf_lim){
        *f_pos = 0;
        goto out;

    }

    // if(*f_pos + cnt > state->buf_lim)
    //  cnt = state->buf_lim - *f_pos;
    cnt = min(cnt, state->buf_lim - *f_pos);

    debug("before copy_to_user (newchange)   cnt = %d, f_pos = %d\n", cnt,
*f_pos);
    //the following checks if the pointer is faulty.
    if(copy_to_user(usrbuf, state->buf_data + *f_pos, cnt)){
        debug("returned from copy_to_user with fault\n");
        ret = -EFAULT;
        goto out;
    }
    debug("returned from copy_to_user without problem\n");
    *f_pos += cnt;
    ret = cnt;
    /* Auto-rewind on EOF mode */
    if(*f_pos >= state->buf_lim)
        *f_pos = 0; //so that on next read we will block till new data arrive.


out:
    /* Unlock */
    up(&state->lock); //release the lock.
    return ret;
}

static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct
*state)
{
    struct lunix_sensor_struct *sensor;

    WARN_ON ( !(sensor = state->sensor));

    //0elei lock
    spin_lock(&sensor->lock);
    if(state->buf_timestamp == sensor->msr_data[state->type]->last_update)
    {
        spin_unlock(&sensor->lock);
        return 0;
    }
```

```
    spin_unlock(&sensor->lock);

    /* The following return is bogus, just for the stub to compile */
    return 1;
}


static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
{
    struct lunix_sensor_struct *sensor;
    uint32_t temp_val;
    debug("entered\n");
    sensor = state->sensor;
    /*
     * Grab the raw data quickly, hold the
     * spinlock for as little as possible.
     */


    if(lunix_chrdev_state_needs_refresh(state) == 0)
    {
        debug("leaving at needs -EAGAIN\n");
        return -EAGAIN;
    }

    // an den exei nea dedomena dwse  -EAGAIN
    /* Why use spinlocks? See LDD3, p. 119 */
    spin_lock(&sensor->lock);
    //SPLIN LOCK TO GET THE DATA FROM THE LINE DISCIPLE SENSOR_STRUCT
    //kai valta ston lunix_chrdev_state_struct.
    //kanoume copy to timestampt, kai ta dedomena ston buf_data
    state->buf_timestamp = sensor->msr_data[state->type]->last_update;
    temp_val = sensor->msr_data[state->type]->values[0];
    /*
     * Any new data available?
     */

    //SPIN UNLOCK
    spin_unlock(&sensor->lock);
    debug("YES THE LINE WORKS--------------------------\n\n\n\n");
    debug("%d\n", state->buf_timestamp);
    debug("temp_val read %u\n", temp_val);
    int dekadiko_meros, akeraio_meros;
    long whole;
    /*
     * Now we can take our time to format them,
     * holding only the private state semaphore
     */
    switch (state->type)
    {
    case BATT:
        whole = lookup_voltage[temp_val];
        break;
    case TEMP:
```

```
            whole = lookup_temperature[temp_val];
            break;
        case LIGHT:
            whole = lookup_light[temp_val];
            break;

        default:
            whole = 0;
            break;
        }
        debug("WHOLE =  %ld\n", whole);
        akeraio_meros = whole / 1000;
        dekadiko_meros = whole % 1000;
        state->buf_lim = snprintf(state->buf_data, LUNIX_CHRDEV_BUFSZ, "%d.%d\n",
    akeraio_meros, dekadiko_meros);


        debug("buf data = %s\n", state->buf_data);
        debug("leaving\n");
        return 0;
    }
```

---

close/release()

Deallocate memory που αφορά την *lunix_chrdev_state_struct* του κάθε ανοιχτού αρχείου.

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
    /* Gets called only 0 reference count. */
    kfree((filp->private_data));
    return 0;
}
```

Δομή

Το lunix_line_desciple παίρνει raw data από τους αισθητήρες και τα βάζει σε έναν buffer και καλεί τις συναρτήσεις στο lunix_protocol όπου γίνεται parsing των δεδομένων από την φόρμα με την οποία μεταφέρονται. Μετά καλεί μία συνάρτηση στο lunix-sensors.c που διαχειρίζεται την μεταφορά των parsed δεδομένων στις structures που υπάρχουν για τον κάθε αισθητήτα

```
    struct lunix_sensor_struct { //1 gia ka9e sensora
    /*
     * A number of pages, one for each measurement.
```

```
     * They can be mapped to userspace.
     */
    struct lunix_msr_data_struct *msr_data[N_LUNIX_MSR];

    /*
     * Spinlock used to assert mutual exclusion between
     * the serial line discipline and the character device driver
     */
    spinlock_t lock;

    /*
     * A list of processes waiting to be woken up
     * when this sensor has been updated with new data
     */
    wait_queue_head_t wq;
};
```

Στο

```
    struct lunix_msr_data_struct *msr_data[];
```

υπάρχουν τα δεδομένα των μετρήσεων στην παρακάτω μορφή :

```
    struct lunix_msr_data_struct {
        uint32_t magic;
        uint32_t last_update;
        uint32_t values[];
};
```

Τα δεδομένα αυτά ανανεώνονται με την χρήση ενός spinlock_t που ορίζεται για κάθε αισθητήρα ώστε να μην υπάρχει διένεξη με το ανώτερο επίπεδο του *lunix_chrdev.c*.

Όταν θέλουμε να διαβάσουμε δεδομένα και έχει ήδη το line_discpl το lock, βάζουμε την διεργασία σε ύπνο στο wait_queue_head_t του κάθε αισθητήρα μέχρι να ξεκλειδώσει **(εδώ αποκλείονται με *spinlock* όσες είναι για τον ίδιο αισθητήρα.)**

Παίρνουμε τα δεδομένα από τον sensor και τα βάζουμε στο αντίστοιχο lunix_chrdev_state_struct όπου πλέον τα δεδομένα διακρίνονται με βάση την μέτρηση.

Στην συνέχεια, τα δεδομένα μεταφέρονται κατά την ζήτηση στον χώρο χρήστη από την lunix_chrdev_state_struct κατά την κλήση της read. Φροντίζονται παράλληλα οι διενέξεις με όσους έχουν το ίδιο file struct (διεργασίες παιδιά ή threads ίδιας διεργασίας) με την χρήση semaphores και φροντίζεται η αναμονή των διεργασιών όταν δεν υπάρχουν νέα δεδομένα.

User Space Program

Το ακόλουθο αποτελεί ένα πρόγραμμα σε χώρο χρήστη που χρησιμοποιήθηκε για να ελέγξει την σωστή λειτουργία του οδηγού καλώντας σε διαφορετικές περιστάσεις τις λειτουργίες του όπως σε race conditions από διαφορετικές ή θυγατρικές διεργασίες και τυπώνει τα αποτελέσματα σε ένα logfile με το όνομα *lunix-logfile*.

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <string.h>

int batts[2], temps[2], lights[2];
void closefds(){
    for(int i = 0; i < 2; i ++){
        close(batts[i]);
        close(temps[i]);
        close(lights[i]);
    }
}

int main(int argc, char** argv){
    if(argc > 2){
        printf("Usage is ./test [<random_word>]\nif random word is given then we
run it with a child doing the same things\n");
        exit(EXIT_FAILURE);
    }

    char buf[4096];
    const int father = getpid();
    int cid;




    for(int i = 0 ; i < 2; i ++){
        sprintf(buf, "/dev/lunix%d-batt", i);
        batts[i] = open(buf, 0);
        if(batts[i] < 0){
            printf("failed to open battery at linux%d\n", i);
            exit(EXIT_FAILURE);
        }

        sprintf(buf, "/dev/lunix%d-temp", i);
        temps[i] = open(buf, 0);
        if(temps[i] < 0){
            printf("failed to open temp at linux%d\n", i);
            exit(EXIT_FAILURE);
        }

        sprintf(buf, "/dev/lunix%d-light", i);
        lights[i] = open(buf, 0);
```

```
        if(lights[i] < 0){
            printf("failed to open battery at linux%d\n", i);
            exit(EXIT_FAILURE);
        }
    }

    if(argc == 2){
        //make the child only if requested

        cid = fork();
        if(cid < 0){
            printf("fork failed\n");
            exit(EXIT_FAILURE);
        }
    }


    int rv, wv;
    char helperbuf[BUFSIZ];
    for(int i = 0; i < BUFSIZ; i ++) helperbuf[i] = '\0';
    int logfile = open("lunix-logfile", O_CREAT | O_RDWR | O_APPEND, S_IRWXU);
    if(logfile < 0){
        printf("opening logfile failed\n");
        exit(EXIT_FAILURE);
    }
    for(int i = 0 ; i < 2; i ++){
        for(int j = 0; j < 2; j ++){
            rv = read(batts[j], buf, sizeof(buf));
            if(rv < 0){
                printf("read from batt of lunix%d failed\n", j);
                exit(EXIT_FAILURE);
            }

            // if(j == 0)
            //     printf("read returned for %d with rv = %d with buf =
%s\n",getpid(), rv, buf);

            buf[rv] = '\0';
            sprintf(helperbuf, "[PID = %d] lunix%d-batt %s\n\0", getpid(), j,
buf);
            wv = write(logfile, helperbuf, strlen(helperbuf));
            if(wv != strlen(helperbuf)){
                printf("writing value from lunix%d-batt less than requested bytes
= %d\n", j, wv);
            }

            rv = read(temps[j], buf, sizeof(buf));
            if(rv < 0){
                printf("read from temp of lunix%d failed\n", j);
                exit(EXIT_FAILURE);
            }
            buf[rv] = '\0';
            for(int i = 0; i < BUFSIZ; i ++) helperbuf[i] = '\0';
            sprintf(helperbuf, "[PID = %d] lunix%d-temp %s\n\0", getpid(), j,
```

```
buf);
            wv = write(logfile, helperbuf, strlen(helperbuf));
            if(wv != strlen(helperbuf)){
                printf("writing value from lunix%d-temp less than requested bytes
= %d\n", j, wv);
            }



            rv = read(lights[j], buf, sizeof(buf));
            if(rv < 0){
                printf("read from light of lunix%d failed\n", j);
                exit(EXIT_FAILURE);
            }
            buf[rv] = '\0';
            for(int i = 0; i < BUFSIZ; i ++) helperbuf[i] = '\0';
            sprintf(helperbuf, "[PID = %d] lunix%d-light %s\n\0", getpid(), j,
buf);
            wv = write(logfile, helperbuf, strlen(helperbuf));
            if(wv != strlen(helperbuf)){
                printf("writing value from lunix%d-light less than requested bytes
= %d\n", j, wv);
            }

        }


        sleep(1);
    }
    if(argc == 2){
        if(cid > 0){
            //father
            closefds();
            int ret, status;
            ret = wait(NULL);
            if(ret < 0){
                printf("Wait failed\n");
                exit(EXIT_FAILURE);
            }
            return 0;
        }
        else {
            closefds();
            return 0;
        }
    }
    else {
        closefds();
        return 0;
    }
}
```

Το shell script που τρέχει το παραπάνω πρόγραμμα για έλεγχο των race conditions:

```
#!/bin/bash
make test
echo "running with $1"
if [ $1 == "fork" ]
then

    ./test random
else
    if [ $1 == "sep" ]
    then
        for x in 1 2
        do
            ./test
        done
        else
            echo "usage : ./tester.sh -opt where opt is fork or sep"
    fi

fi
```