Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Εργαστήριο Λειτουργικών Συστημάτων 2024 - 2025
2η εργαστηριακή άσκηση : Lunix - TNG

Ομάδα : 12
Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076
Οναματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Οδηγός Ασύρματου Δικτύου Αισθητήρων στο Λειτουργικό Σύστημα Linux

Στην 2η εργαστηριακή Άσκηση καλούμαστε, χρησιμοποιώντας το βοηθητικό κώδικα που μας δίνεται από το εργαστήριο, να υλοποιήσουμε οδηγό συσκευής χαρακτήρων στο Linux για ασύρματο δίκτυο αισθητήρων. Η διασύνδεση του δικτύου με το υπολογιστικό σύστημα μας θα γίνεται με κύκλωμα Serial over USB, για το οποίο ο πυρήνας διαθέτει ήδη ενσωματωμένους οδηγούς, και έτσι οι μετρήσεις των αισθητήρων (τάση της μπαταρίας τους, θερμοκρασία και φωτεινότητα του χώρου) θα εμφανίζονται σε εικονικές σειριακές θύρες. Η προσομοίωση της θύρας USB γίνεται "ακούγοντας" στι κατάλληλο port.

Τα πρώτα στάδια της διαδικασίας επικοινωνίας με το δίκτυο αισθητήρων έχουν ήδη υλοποιηθεί από το εργαστήριο. Συγκεκριμένα, η λήψη των μετρήσεων από το σταθμό βάσης είναι ήδη υλοποιημένη και μας δίνεται κώδικας ο οποίος είναι υπεύθυνος αρχικά για την προώθηση μέσω USB των μετρήσεων στο υπολογιστικό σύστημα μας, έπειτα για την υλοποίηση φίλτρου (Lunix Line Discipline) ώστε τα δεδομένα να προωθούνται σε ειδικό στρώμα που ερμηνεύει το περιεχόμενο τους και, τέλος, να το αποθηκεύει σε διαφορετικούς buffer ανα αισθητήρα.

Εμείς καλούμαστε τελικά να γράψουμε κώδικα ο οποίος να επιτρέπει την ανάκτηση των δεδομένων ανάλογα με το ποιό ειδικό αρχείο χρησιμοποιεί μία διεργασία στον χώρο χρήστη. Το βασικό πλεονέκτημα που προσφέρει η παραπάνω υλοποίηση, σε σχέση με το ήδη υλοποιημένο στον πυρήνα TTY στρώμα, είναι πως προσφέρεται η δυνατότητα ξεχωριστής πρόσβασης στα διαφορετικά ζευγάρια αισθητήρα - μέτρησης. Πιο συγκεκριμένα υλοποιήσαμε τις κατάλληλες συναρτήσεις στα αρχεία lunix-chrdev.h και lunix-chrdev.c αντίστοιχα. Παρακάτω φαίνεται ο κώδικας του πρώτου :

```
/*
 * lunix-chrdev.h
 *
 * Definition file for the
 * Lunix:TNG character device
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 */
```

```c
#ifndef _LUNIX_CHRDEV_H
#define _LUNIX_CHRDEV_H

/*
 * Lunix:TNG character device
 */
#define LUNIX_CHRDEV_MAJOR 60   /* Reserved for local / experimental use */
#define LUNIX_CHRDEV_BUFSZ 20   /* Buffer size used to hold textual info */

/* Compile-time parameters */

#ifdef __KERNEL__

#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/module.h>

#include "lunix.h"

/*
 * Private state for an open character device node
 */
struct lunix_chrdev_state_struct {
    enum lunix_msr_enum type;
    struct lunix_sensor_struct *sensor;

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

    struct semaphore lock;

    /*
     * Fixme: Any mode settings? e.g. blocking vs. non-blocking
```

```
     */
};

/*
 * Function prototypes
 */
int lunix_chrdev_init(void);
void lunix_chrdev_destroy(void);

#endif /* __KERNEL__ */

#include <linux/ioctl.h>

/*
 * Definition of ioctl commands
 */
#define LUNIX_IOC_MAGIC     LUNIX_CHRDEV_MAJOR
//#define LUNIX_IOC_EXAMPLE _IOR(LUNIX_IOC_MAGIC, 0, void *)

#define LUNIX_IOC_MAXNR 0

#endif /* _LUNIX_H */
```

Στη συνέχεια ξεκινήσαμε την υλοποίηση των παρακάτω συναρτήσεων:

lunix_chrdev_state_needs_refresh

Η συνάρτηση αυτή παίρνει ως όρισμα έναν δείκτη σε lunix_chrdev_state_struct που είναι μία δομή η οποία έχει οριστεί στο header file, και μέσω του private data δείκτη υπάρχει μία τέτοια δομή για κάθε ανοιχτό αρχείο (θα δούμε περισσότερα στην open). Η δομή αυτή περιλαμβάνει χρήσιμες πληροφορίες για το ανοιχτό αρχείο καθώς και buffer για δεδομένα. Αφού η συνάρτηση χειριστεί σωστά τυχόν λάθη

επιστρέφει 1 εφόσον το timestamp της δομής state του ορίσματος είναι παλαιότερο χρονικά από το last update του αισθητήρα (και συνεπώς απαιτείται update), ενώ 0 εάν δεν ισχύει το παραπάνω και συνεπώς δεν απαιτείται update. Ακολουθεί ο κώδικας :

```
/*
 * Just a quick [unlocked] check to see if the cached
 * chrdev state needs to be updated from sensor measurements.
 */

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

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


    if(state->buf_timestamp < (sensor->msr_data[state->type]->last_update)){
        debug("Refresh: needs to be updated, return 1");
        return 1;
    }
    else{
        debug("Refresh does not need to be updated, return 0");
        return 0;
    }
}
```

lunix_chrdev_state_update

Η συνάρτηση αυτή παίρνει επίσης ως όρισμα έναν δείκτη lunix_chrdev_state_struct, συγκεκριμένα του εκάστοτε ανοιχτού αρχείου και αρχικά τσεκάρει αν χρειάζεται update με την συνάρτηση lunix_chrdev_state_needs_refresh. Αν δεν χρειάζεται επιστρέφει την τιμή σφάλματος -EAGAIN, δηλαδή *"no data available right now , try again later"*.

Στην αντίθετη περίπτωση, πάει και λαμβάνει την νέα τιμή του αισθητήρα από τις κατάλληλες δομές. Εδώ προκύπτει η ύπαρξη κρίσιμου τμήματος(critical section) και

για αυτό πριν πάμε να διαβάσουμε από τις δομές του αισθητήρα, πρέπει να έχουμε αποκλειστική πρόσβαση σε αυτές. Το πρόβλημα με την χρήση σημαφόρου εδώ είναι ότι οι δομές του αισθητήρα μπορούν να προσπελαστούν και σε interrupt context, αφού όταν έρθουν νέες τιμές τα κατώτερα (πιο κοντά στο υλικό) μέρη του οδηγού θα στείλουν διακοπή σε κάποια ανώτερο επίπεδο. Σε interrupt context όμως δεν "υπάρχει κάποια διεργασία για να κοιμηθεί", και συνεπώς πρέπει αναγκαστικά να χρησιμοποιήσουμε spin lock.

Στην συνέχεια, ανάλογα με το είδος της μέτρησης,την μετατρέπουμε σε human readable μορφή, κάνοντας χρήση και των look up tables που μας δίνονται στον κώδικα του εργαστηρίου, και, τέλος, την γράφουμε στον buffer της δομής lunix_chrdev_state_struct που μας δίνεται ως όρισμα. Αν όλα έχουν γίνει σωστά επιστρέφουμε 0. Παρακάτω φαίνεται η υλοποίηση της:

```
/*
 * Updates the cached state of a character device
 * based on sensor data. Must be called with the
 * character device state lock held.
 */
static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
{
    struct lunix_sensor_struct *sensor;
    uint32_t new_value;
    long fixed_value;
    unsigned long cpu_flags;

    debug("entering update\n");

    /*If no refresh is needed then stop and return "try again" */

    if(!lunix_chrdev_state_needs_refresh(state)){
        return -EAGAIN;
    }

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

```
    debug("Spinlock not locked yet\n");

    spin_lock_irqsave(&sensor->lock, cpu_flags);

    new_value = sensor->msr_data[state->type]->values[0];

    spin_unlock_irqrestore(&sensor->lock, cpu_flags);
    debug("Spinlock just unlocked\n");

    /*Whats our measurement type? Use lookup tables to make it decimal human
readable*/

    switch (state->type){
    case BATT:
        fixed_value = lookup_voltage[new_value];
        break;
    case TEMP:
        fixed_value = lookup_temperature[new_value];
        break;
    case LIGHT:
        fixed_value = lookup_light[new_value];
        break;
    default:
        debug("lookup : rubbish");
    }

    /*Split integers and decimals*/

    debug("Fixed value = %ld\n", fixed_value);

    int integer_part = fixed_value / 1000;
    int decimal_part = fixed_value % 1000;

    debug("Final value = %d.%d\n", integer_part, decimal_part);
```

```
    /*Copy to buffer with respect to its capacity*/
    state->buf_lim = snprintf(state->buf_data, LUNIX_CHRDEV_BUFSZ, "%d.%d\n",
integer_part, decimal_part);


    if(state->buf_lim >= LUNIX_CHRDEV_BUFSZ){
        debug("buffer length exceeded: snprintf truncated string\n");
    }

    /*Update new timestamp*/
    state->buf_timestamp = ktime_get_real_seconds();

    debug("leaving\n");
    return 0;
}
```

lunix_chrdev_open

Η συνάρτηση open παίρνει ως ορίσματα 2 δείκτες, έναν προς το inode του αρχείου που θα ανοίξουμε και έναν pointer στη δομή file (filp) που χρησιμοποιέι ο πυρήνας για να αναπαραστήσει το αρχείο. Καταρχάς ανοίγουμε με την **nonseekable_open** το ζητούμενο αρχείο και αν δεν το βρούμε, η open επιστρέφει -ENODEV, δηλαδή *"no device"*.

Έπειτα, εξάγουμε από το minor number του αρχείου το sensor_id και τον τύπο μέτρησης (φροντίζοντας βέβαια να είναι έγκυρος).

Μετά, χρησιμοποιώντας την kmalloc (kernel malloc) κάνουμε δυναμικά allocate χώρο στην μνήμη για την δομή lunix_chrdev_state_struct του αρχείου που ανοίγουμε, και αρχικοποιούμε τα πεδία του (συμπεριλαμβανομένου του σημαφόρου του σε τιμή 1). Τέλος, βάζουμε τον δείκτη private data της δομής file να δείχνει στην newly allocated περιοχή της μνήμης) και επιστρέφουμε 0, εφόσον πήγαν όλα καλά. Εδώ βλέπουμε τον κώδικα της συνάρτησης :

```
/*********************************
 * Implementation of file operations
 * for the Lunix character device
 *********************************/
```

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
    /* Declarations */
    struct lunix_chrdev_state_struct *new_state;
    int type;
    int sensor_id;
    int ret;

    debug("fez: entering\n");
    ret = -ENODEV;

    /*Check if we have a correct device outherwise go to out*/
    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>]
     */

    sensor_id = iminor(inode) >> 3;
    type = iminor(inode) % 8;
    debug("fez: sensorId is  = %d\n", sensor_id);
    debug("fez: type is  = %d\n", type);

    /*if we have bigger sensor number than allowed then return no device*/
    if(type >= N_LUNIX_MSR){
        ret = -ENODEV;
        goto out;
    }

    /* Allocate a new Lunix character device private state structure */
    new_state = kmalloc(sizeof(struct lunix_chrdev_state_struct),
GFP_KERNEL);
```

```
        /* Error handling */
    if(!new_state){
        printk(KERN_ERR "fez: Lunix_Open: Error with allocationg private
state structure\n");
        /* return bad-address */
        ret = -EFAULT;
        goto out;
    }

    /*Define our new state stuct*/

    new_state->type = type;
    new_state->sensor = &lunix_sensors[sensor_id];
    new_state->buf_lim = 0;
    new_state->buf_timestamp = 0;

    sema_init(&new_state->lock, 1);

    filp->private_data = new_state;

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

lunix_chrdev_release

Η συνάρτηση release είναι η αντίστοιχη της close(), παίρνει αντίστοιχα ορίσματα με την open, απελευθερώνει την δεσμευμένη περιοχή μνήμης που δείχνουν τα private_data του ανοιχτού αρχείου με την χρήση της kfree (kernel free) και επιστρέφει 0.

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
    kfree(filp->private_data);
    return 0;
}
```

lunix_chrdev_read

Η σημαντικότερη ίσως συνάρτηση του οδηγού που υλοποιούμε είναι η read, η οποία παίρνει τα εξής ορίσματα:

- Έναν δείκτη στην δομή file του πυρήνα για το ανοιχτό αρχείο από το οποίο θέλουμε να διαβάσουμε
- Έναν δείκτη σε buffer σε user space
- Τον αριθμό των bytes που θέλει να διαβάσει η διεργασία χώρου χρήστη
- Το file offset στο ήδη ανοιχτό αρχείο

Στο συγκεκριμένο σημείο επιθυμούμε να διαβάσουμε χαρακτήρες από τον buffer που βρίσκεται στο state του ανοιχτού αρχείου. Επειδή όμως υπάρχει πιθανότητα να υπάρχουν και άλλες διεργασίες, εν προκειμένω διεργασίες παιδία οι οποίες "κληρονομούν" το ανοιχτό αρχείο, που προσπαθούν να διαβάσουν από αυτόν τον buffer, πρέπει να διαχειριστούμε το κρίσιμο τμήμα κατάλληλα. Επειδή η read καλείται μόνο από process context και επειδή τον buffer αυτόν θα τον προσπελάσουμε μόνο από process context, ο σημαφόρος είναι μία πλήρως ταιραστή λύση (και για αυτό τον έχουμε ορίσει στην δομή lunix_chrdev_state_struct και ήδη αρχικοποιήσει στην open).

Έτσι μειώνουμε την τιμή του σημαφόρου με την down_interruptible, και έτσι θα μπούμε στο παρακάτω κομμάτι κώδικα μόνο όταν έχουμε απόλυτη και μοναδική πρόσβαση σε έναν επεξεργαστή.

Στην συνέχεια ελέγχουμε το offset στο αρχείο με τη χρήση του f_pos και εφόσον πληροί τις προϋποθέσεις για ενημέρωση, θα ελέγχξουμε αν χρειάζεται update το state του αρχείου, με την συνάρτηση lunix_chrdev_state_update. Όσο γίνεται αυτό θα απελευθερώνουμε τον κοινό πόρο (με up στον σημαφόρο) και θα χρησιμοποιούμε την wait_event_interruptible ώστε η διεργασία που κάλεσε την open να κοιμηθεί μέχρι να απαιτείται update στο state του ανοιχτού αρχείου. Έτσι ουσιαστικά υλοποιούμε την απαίτηση να κοιμίζεται η διεργασία όσο δεν υπάρχουν νέα δεδομένα.

Έπειτα, αφού ξυπνήσει η διεργασία, θα ξαναπροσπαθήσει με την down_interruptible να πάρει απόλυτη πρόσβαση του state του ανοιχτού αρχείου. Έχοντας λοιπόν την απαραίτητη πρόσβαση, αφού χειριστεί κατάλληλα τις περιπτώσεις να μην υπάρχουν δεδομένα ή να έχει ζητηθεί να διαβαστεί μεγαλύτερο αριθμός byte από τον διαθέσιμο, ο κώδικας μας αντιγράφει τα δεδομένα στον buffer χώρου χρήστη, με την συνάρτηση copy_to_user.

Τέλος, ενημερώνει κατάλληλα τον δείκτη f_pos για το file offset, αφήνει τον κοινό πόρο (με up στον σημαφόρο) και επιστρέφει τον αριθμό των bytes που διαβάστηκαν επιτυχώς. Παρακάτω ακολουθεί ο κώδικας :

```
static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf,
                                 size_t cnt, loff_t *f_pos)
{
    ssize_t ret;
    struct lunix_sensor_struct *sensor;
    struct lunix_chrdev_state_struct *state;


    /*Warnings on differences in state and sensor between file and cached
data*/
    state = filp->private_data;
    WARN_ON(!state);

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

    debug("Fez is in read\n");

    /* Lock */
    if(down_interruptible(&state->lock)){
        return -ERESTARTSYS;
    }

    /*Utilise f_pos to discover if we need to print to user*/
    if (*f_pos == 0) {
        while (lunix_chrdev_state_update(state) == -EAGAIN) {
            up(&state->lock);
            debug(" Read : No new data, i'm going to sleep!!!\n");

            if(wait_event_interruptible(sensor->wq,
lunix_chrdev_state_needs_refresh(state))){
                debug("Wake up from interrupt\n");
                return -ERESTARTSYS;
```

```
            }

            if (down_interruptible(&state->lock)){
                    return -ERESTARTSYS;    /* Lock because other procs may
have the same state with you */

            }
        }
    }

    /* End of file */

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

    /*No bytes read go print 0*/
    if(state->buf_lim == 0){
        ret = 0;
        goto out;
    }

    /*If our buf lim is smaller than count then print only the ones we can
print*/
    int temp = state->buf_lim - *f_pos;
    if(temp < cnt) {
        cnt = temp;
    }

    debug("Start printing to user\n");

    /*Use copy_t0_user to send the data to user space buffer*/
    if(copy_to_user(usrbuf, state->buf_data + *f_pos, cnt)){
        ret = -EFAULT;
        goto out;
    }
    debug("completed print to user\n");
```

```
    /*Update new f_pos after printing*/
    *f_pos += cnt;

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

    /*Return amount of bytes read*/
    ret = cnt;

out:
    /* Unlock? */
    up(&state->lock);
    return ret;
}
```

lunix_chrdev_init

Η συνάρτηση init θα κληθεί από κατώτερο (πιο κοντά στο υλικό) τμήμα του οδηγού για την αρχικοποίηση.

Καταρχάς, αρχικοποιεί την δομή του οδηγού συσκευής χαρακτήρων με τα file operations, έπειτα δημιουργεί το device number και καταχωρεί την περιοχή των συσκευών, μετά προσθέτει τον οδηγό στον πυρήνα και τέλος, εφόσον όλες οι προηγούμενες ενέργειες είναι επιτυχείς, επιστρέφει 0. Ο κώδικας ακολουθεί παρακάτω :

```
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);
;
    ret = register_chrdev_region(dev_no, lunix_minor_cnt, "lunxTNG");

    if (ret < 0) {
        debug("failed to register region, ret = %d\n", ret);
        goto out;
    }


    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;
}
```

lunix_chrdev_destroy

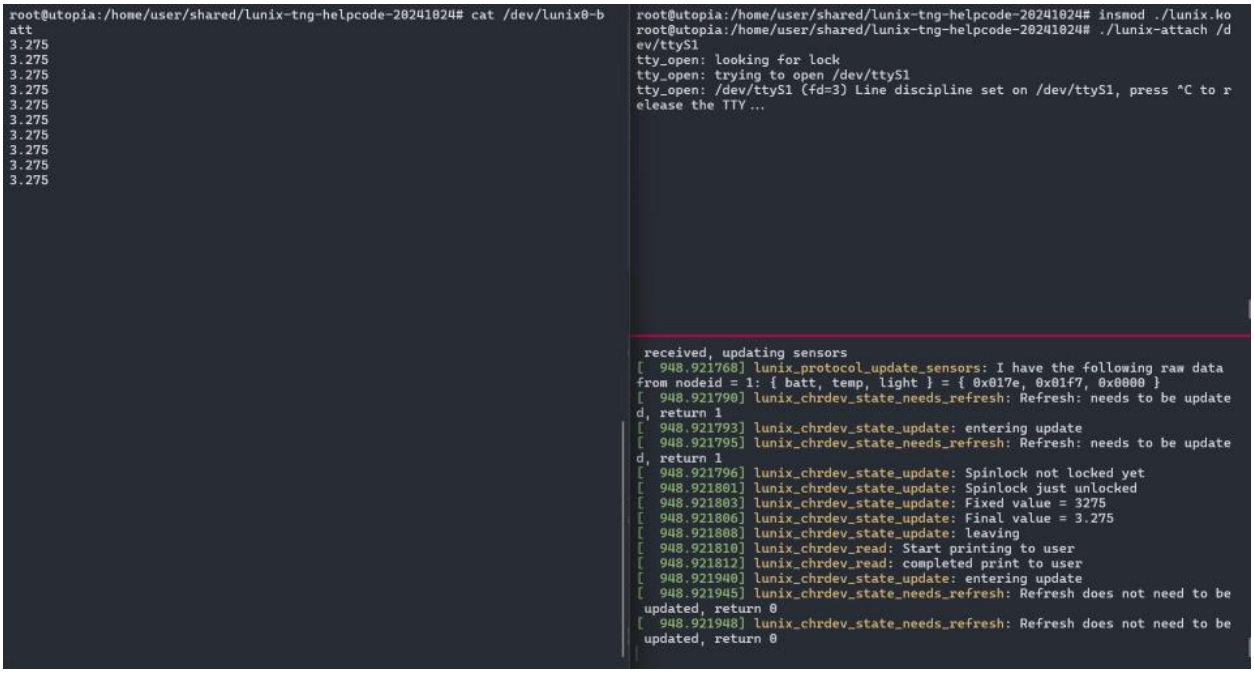
Η συνάρτηση destroy υπάρχει ώστε να κληθεί εάν ποτέ θελήσουμε να αφαιρέσουμε τον οδηγό από τον πυρήνα, η οποία πρώτα σβήνει τον οδηγό και μετά "καταστρέφει" (unregister) την περιοχή των συσκευών.Ο κώδικας της :

```
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");
}
```

Εισαγωγή Module και λήψη μετρήσεων

Έχοντας υλοποιήσει όλα τα παραπάνω το μόνο που απομένει είναι η εισαγωγή του Module μας στον πυρήνα και η λήψη των πρώτων μετρήσεων μας. Αρχικά εκτελούμε το script που μας δόθηκε το οποίο δημιουργέι τα κατάλληλα device nodes με την εντολή ./**mk-lunix-devs.sh**. Στη συνέχεια εισάγουμε το module του πυρήνα που μεταγλωτίσσαμε με την εντολη **insmod ./lunix.ko**. Επισυνάπτουμε το Lunix line discipline με την εντολή **./lunix-attach /dev/ttyS1** και ανοίγουμε και ένα τερματικό για να παρακολουθούμε τα logs του πυρήνα χρησιμοποιώντας την εντολή **dmesg**. Όπως φαίνεται και στον κώδικα μας παραπάνω, υπάρχουν αρκετά σημεία στις συναρτήσεις μας τα οποία χρησιμοποιούν την εντολή **debug**, η οποία επιτρέπει να χρησιμοποιούμε τα logs του πυρήνα για να παρακολουθούμε ολόκληρη τη ροή του προγράμματος μας και τις συναρτήσεις που καλούνται κάθε φορά. Τέλος για να διαβάσουμε την τιμή κάποιας μέτρησης εκτελούμε **cat** στα ειδικά αρχεία των αισθητήρων. Ένα παράδειγμα του περιβάλλοντος εργασίας μας όπου φάινονται όλα αυτά που αναφέρθηκαν φαίνεται εδώ :



Όπως φάινεται στο αριστερό τερματικό διαβάζουμε την τιμή της μπαταρίας για τον πρώτο αισθητήρα, κάτω δεξία βλέπουμε τα logs του πυρήνα και πάνω δεξία τις εντολές για να εισάγουμε το module και το line discipline. Για να αφαιρέσουμε το lunix module από τον πυρήνα εκτελούμε **rmmod ./lunix.ko**.