

*Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών
Υπολογιστών*



8ο εξάμηνο

2017-2018

Εργαστήριο Λειτουργικών Συστημάτων

*2^η Εργαστηριακή Άσκηση
Οδηγός Ασύρματου Δικτύου Αισθητήρων στο
λειτουργικό σύστημα Linux*

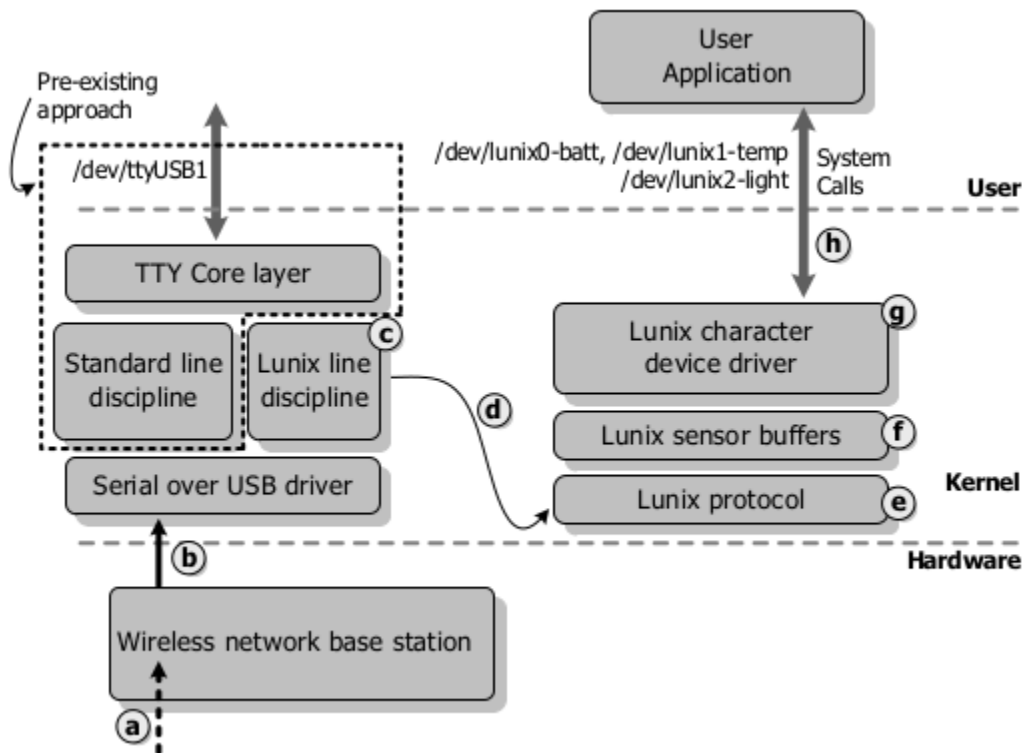
*Ομάδα 18
Μουζάκης Ανάργυρος-Γεώργιος
Μουζάκης Νικόλαος*

Εισαγωγή

Στα πλαίσια της παρούσας εργαστηριακής άσκησης ζητήθηκε η υλοποίηση ενός οδηγού συσκευής χαρακτήρων για το λειτουργικό σύστημα linux με όνομα *Lunix:TNG*. Συγκεκριμένα, ο οδηγός πρέπει να επεξεργάζεται και να απεικονίζει στο χώρο χρήστη δεδομένα που προέρχονται από έναν σταθμό βάσης, ο οποίος αλληλεπιδρά με ένα δίκτυο αισθητήρων (εν προκειμένω, μόνο ένας αισθητήρας ήταν λειτουργικός). Οι αισθητήρες στέλνουν πληροφορίες στον σταθμό βάσης που αφορούν στις συνθήκες περιβάλλοντος (θερμοκρασία, φωτεινότητα) και στην κατάστασή τους (στάθμη μπαταρίας). Θεωρητικά, ο σταθμός βάσης πρέπει να συνδέεται μέσω USB στον υπολογιστή που επεξεργάζεται τα δεδομένα. Στην πράξη, για τις ανάγκες της δικής μας υλοποίησης, όπου εργαζόμασταν σε remote vm της υπηρεσίας okeanos, δόθηκε το shell script *utopia.sh*. Αυτό, κατά την εκκίνηση μίας εικονικής μηχανής *QEMU-KVM* που λειτουργεί εντός του remote vm μας, φροντίζει για την απεικόνιση της θύρας TCP cerberus.cslab.ece.ntua.gr:49152 (απ' όπου λαμβάνονται οι μετρήσεις) στην πρώτη σειριακή θύρα της εικονικής μηχανής. Επομένως, ο οδηγός μας δέχεται δεδομένα από την πρώτη σειριακή θύρα και φροντίζει για την μορφοποίησή τους και, στη συνέχεια, για την απεικόνισή τους στο χώρο χρήστη.

Η δομή του οδηγού

Η παρακάτω εικόνα, που προέρχεται από τον εργαστηριακό οδηγό που δόθηκε, παρουσιάζει την αρχιτεκτονική του οδηγού που κληθήκαμε να υλοποιήσουμε.



Ο οδηγός αποτελείται από πολλά τμήματα, καθένα εκ των οποίων υλοποιεί μία διαφορετική λειτουργία του οδηγού. Το **Lunix line discipline** (ο κώδικας δόθηκε στα αρχεία `lunix-ldisc.{c, h}`) αναλαμβάνει τη συλλογή δεδομένων, όταν αυτά φτάνουν στην πρώτη σειριακή θύρα και τη μεταβίβασή τους στο **Lunix protocol** (ο κώδικας δόθηκε στα αρχεία `lunix-protocol.{c, h}`). Αυτό, με τη σειρά του, αναλαμβάνει την εξαγωγή των απαραίτητων πληροφοριών (τύπος μέτρησης και τιμή αυτής, αισθητήρας προέλευσης κλπ). Με βάση αυτές τις τιμές γίνεται η αποθήκευση σε κατάλληλες δομές που αναπαριστούν ανά πάσα στιγμή την κατάσταση των αισθητήρων (`struct lunix_sensor_struct`, αναφέρεται στο σχήμα ως **Lunix sensor buffers**). Τέλος, υπάρχει το τμήμα του **interface**, το οποίο είναι υπεύθυνο για την αλληλεπίδραση του οδηγού με τις διεργασίες χώρου χρήστη. Συγκεκριμένα, οι συσκευές χαρακτηρών αναπαρίστανται ως **ειδικά αρχεία** στο **virtual file system** του υπολογιστή και υλοποιούν ένα interface που είναι παρεμφερές με αυτό των συμβατικών αρχείων. Όταν μία διεργασία χώρου χρήστη ανοίγει ένα τέτοιο ειδικό αρχείο, αναγνωρίζεται ο οδηγός συσκευής που το διαχειρίζεται

με βάση το *major number* του. Έτσι, κατά τη δημιουργία της σχετικής εγγραφής στο process control block της διεργασίας που θέλει να αποκτήσει πρόσβαση στο αρχείο, στο πεδίο *f_ops* της δομής *struct file* αποθηκεύεται η διεύθυνση της δομής *struct file_operations* που αντιστοιχεί στον οδηγό. Η *file_operations* έχει πεδία που είναι οι διευθύνσεις των συναρτήσεων όπου βρίσκονται οι ειδικές υλοποιήσεις των system calls για τα συγκεκριμένα αρχεία.

Στα πλαίσια της άσκησης, κληθήκαμε να υλοποιήσουμε 6 συναρτήσεις (όλες στο αρχείο *linux_chrdev.c*). Από αυτές, 3 αποτελούσαν τμήμα του interface. Αυτές ήταν οι *linux_chrdev_open*, *linux_chrdev_release*, *linux_chrdev_read*. Επιπλέον, άλλες 2 χρησίμευαν στην ανανέωση της δομής του *struct file* που περιγράφει την κατάσταση της εκάστοτε συσκευής (πεδίο *private_data*, το οποίο, εν προκειμένω, είναι συσχετισμένο με μία δομή τύπου *linux_chrdev_state_struct*). Αυτές ήταν οι *linux_chrdev_state_needs_refresh* και *linux_chrdev_state_update*. Τέλος, υλοποιήθηκε και η συνάρτηση *linux_chrdev_init*, η οποία εκτελείται κατά την ενσωμάτωση του *kernel module linux.ko* στον πυρήνα.

Στα επόμενα μέρη περιγράφουμε αναλυτικά την υλοποίηση της καθεμίας από τις προηγούμενες συναρτήσεις.

Init

Όπως αναφέρθηκε παραπάνω, η `init` εκτελείται κάθε φορά που το `kernel module` που αντιστοιχεί στον οδηγό ενσωματώνεται στον πυρήνα. Ο σκοπός της είναι να δημιουργήσει την κατάλληλη δομή του πυρήνα που αναπαριστά τον οδηγό συσκευής. Η δομή αυτή είναι η `linux_chrdev_cdev` και δημιουργείται με χρήση της εντολής `cdev_init`. Επιπλέον, πρέπει να δηλώσει ποιο είναι το εύρος των αριθμών συσκευών που διαχειρίζεται ο οδηγός και να τον ενσωματώσει στον πυρήνα. Αυτό επιτυγχάνεται μέσω των εντολών `register_chrdev_region` και `cdev_add`. Ακολουθεί ο πλήρης κώδικας:

```
int linux_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 linux_minor_cnt = linux_sensor_cnt << 3;

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

    dev_no = MKDEV(LINUX_CHRDEV_MAJOR, 0);
    ret = register_chrdev_region(dev_no, linux_minor_cnt, "Lunix:TNG");
    if (ret < 0) {
        debug("failed to register region, ret = %d\n", ret);
        goto out;
    }
    debug("device registered successfully with dev_no %d and range %d\n", (int) dev_no, (int)
linux_minor_cnt);
    ret = cdev_add(&linux_chrdev_cdev, dev_no, linux_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, linux_minor_cnt);
out:
    return ret;
}
```

needs_refresh

Η συνάρτηση ελέγχει αν τα δεδομένα που υπάρχουν στο state structure του αρχείου χρειάζονται ανανέωση. Αυτό επιτυγχάνεται συγκρίνοντας το πεδίο *timestamp* του με το αντίστοιχο του σχετικού sensor structure. Σημειώνεται πως τα timestamp είναι με ακρίβεια δευτερολέπτου (λόγω της *get_seconds()* που χρησιμοποιείται στη συνάρτηση *linux_sensor_update* του αρχείου *lunic-sensors.c*). Επομένως, αν έρθουν νέες μετρήσεις προτού περάσει ένα δευτερόλεπτο από όταν ήρθαν οι προηγούμενες, θα αγνοηθούν. Αυτό δεν είναι πρόβλημα, καθώς μετρήσεις που έρχονται το ίδιο δευτερόλεπτο συνήθως έχουν ίδιες τιμές. Ο κώδικας είναι:

```
/*  
 * Just a quick [unlocked] check to see if the cached  
 * chrdev state needs to be updated from sensor measurements.  
 */  
static int linux_chrdev_state_needs_refresh(struct linux_chrdev_state_struct *state)  
{  
    struct linux_sensor_struct *sensor;  
  
    WARN_ON ( !(sensor = state->sensor));  
    if (state->buf_timestamp != sensor->msr_data[state->type]->last_update) return 1;  
    return 0;  
}
```

update

Αυτή η συνάρτηση καλείται από τη *read* προκειμένου να ελεγχθεί αν υπάρχουν νέα δεδομένα για εμφάνιση. Αν δεν υπάρχουν, επιστρέφεται η τιμή *-EAGAIN*. Αν υπάρχουν, αντιγράφονται από το *sensor struct*. Δεδομένου ότι η δομή αυτή αλληλεπιδρά και με το *linux-protocol*, η λειτουργία του οποίου, όπως και του *linux discipline*, είναι *hardware initiated* πρέπει, προκειμένου να αποτραπεί το ενδεχόμενο να γραφτούν νέα δεδομένα ενώ κάποια διεργασία διαβάζει από αυτή, να χρησιμοποιηθεί το πεδίο *spinlock* της *sensor struct*. Ύστερα, πρέπει να ενημερωθούν τα σχετικά πεδία του *state structure* του αρχείου. Στα πλαίσια της ενημέρωσης ανατρέχουμε στα *lookup tables*, προκειμένου να βρούμε τις σωστές τιμές, καθώς αυτές που έχουν ληφθεί από το πρωτόκολλο είναι κωδικοποιημένες σε μορφή 16-bit που δεν είναι κατανοητή από το χρήστη. Ύστερα, εφόσον στους πίνακες οι αριθμοί που υπάρχουν είναι ακέραιοι (λόγω της δυσκολίας στην αναπαράσταση πραγματικών αριθμών σε επίπεδο πυρήνα), πρέπει να μετατρέψουμε την τιμή που έχουμε από τον πίνακα σε πραγματικό αριθμό με 3 δεκαδικά ψηφία και να τον αποθηκεύσουμε υπό μορφή συμβολοσειράς. Δεδομένου ότι ενδέχεται να υπάρχουν πολλά νήματα της ίδιας διεργασίας ή πολλές διεργασίες όπου να έχουμε σχέσεις γονέα-παιδιού και, άρα, να μοιράζονται την πρόσβαση στο αρχείο, πρέπει να χρησιμοποιηθεί ο **σημαφόρος** στη *state structure*. Η *update* καλείται μόνο από ένα σημείο στον κώδικα της *read* στο οποίο, όταν φτάνει μία διεργασία, έχει το σημαφόρο. Ως εκ τούτου, δεν χρειάζεται να καλέσουμε τη συνάρτηση *down* μέσα στον κώδικα της *update*.

```
static int linux_chrdev_state_update(struct linux_chrdev_state_struct *state)
{
    struct linux_sensor_struct *sensor;
    uint32_t value, timestamp;
    int i;
    long lookup_table_value, integer_part, decimal_part;
    unsigned long flags;

    debug("start of update function\n");

    if (!linux_chrdev_state_needs_refresh(state)) {
        debug("leaving update function, no new data\n");
        return -EAGAIN; // return with the value expected by read
    }

    debug("we start copying the new data from the sensor buffers\n");

    sensor = state->sensor;
```

```

WARN_ON(!sensor);

// we use the versions of the spinlock functions that disable interrupts temporarily
spin_lock_irqsave(&sensor->lock, flags);
value = sensor->msr_data[state->type]->values[0];
timestamp = sensor->msr_data[state->type]->last_update;
spin_unlock_irqrestore(&sensor->lock, flags);

if (state->type == 0) lookup_table_value = lookup_voltage[value];
else if (state->type == 1) lookup_table_value = lookup_temperature[value];
else lookup_table_value = lookup_light[value];
debug("the value we got from the lookup tables is %ld\n", lookup_table_value);

// start converting to user space form

state->buf_lim = 0;

// first, let's get the sign right

if (lookup_table_value < 0) {
    state->buf_data[state->buf_lim++] = '-';
    lookup_table_value = -lookup_table_value;
}

integer_part = lookup_table_value/1000;
decimal_part = lookup_table_value%1000;

if (integer_part == 0) {
    state->buf_data[state->buf_lim++] = '0';
    i = 0;
}
else {
    i = 1;
    while (integer_part >= i) i *= 10;
    i /= 10;
}
while (i != 0) {
    state->buf_data[state->buf_lim++] = (unsigned char) (integer_part/i + 48);
    integer_part %= i;
    i /= 10;
}

state->buf_data[state->buf_lim++] = '.'; // the decimal point

i = 1;
state->buf_lim += 3; // we have three positions for decimals
while (i <= 3) {
    state->buf_data[state->buf_lim - i] = (unsigned char) (decimal_part%10 + 48);
    decimal_part /= 10;
}

```



```
        i++;  
    }  
  
    state->buf_data[state->buf_lim++] = '\n';  
    state->buf_data[state->buf_lim] = '\0';  
    state->buf_timestamp = timestamp;  
  
    debug("leaving update with msr %s, %d\n", state->buf_data, state->buf_lim);  
    return 0;  
}
```

open

Στην υλοποίηση του linux για το συγκεκριμένο system call, με βάση τη σύμβαση για το *minor number* που περιγράφεται στην εκφώνηση προσδορίζουμε ποια είναι η δομή αισθητήρα που αντιστοιχεί στο αρχείο που έχουμε ανοίξει και ποια είναι η μέτρηση που μας ενδιαφέρει. Τέλος, δεσμεύουμε μία δομή *linux_chrdev_state_struct* και τη συσχετίζουμε με το πεδίο *private_data* του struct file, κάνοντας κάποιες αρχικοποιήσεις.

```
static int linux_chrdev_open(struct inode *inode, struct file *filp)
{
    int ret;
    unsigned int dev_minor_num, type;
    struct linux_sensor_struct *sensor;
    struct linux_chrdev_state_struct *state;

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

    dev_minor_num = iminor(inode);
    debug("the device has minor number %d, the corresponding sensor is %d\n",
dev_minor_num, dev_minor_num/8);
    sensor = &linux_sensors[dev_minor_num/8];
    type = dev_minor_num%8;

    /* Allocate a new Linux character device private state structure */
    state = (struct linux_chrdev_state_struct*) vmalloc(sizeof(struct
linux_chrdev_state_struct)); // allocate memory for a private state structure
    if (!state) {
        printk(KERN_ERR "Failed to allocate memory for Linux character device state
structure\n");
        ret = -ENOMEM;
        goto out;
    }
    state->sensor = sensor;
    state->type = type;
    state->buf_lim = 0;
    sema_init(&state->lock, 1);
    filp->private_data = state;

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

release

Αποδεσμεύεται η δομή *linux_chrdev_state_struct* που δεσμεύθηκε κατά την open.

```
static int linux_chrdev_release(struct inode *inode, struct file *filp)
{
    vfree(filp->private_data); // since vmalloc was used, we have to use vfree
    debug("closing down (release)\n");
    return 0;
}
```

read

Καταρχάς, προτού μία διεργασία ή ένα νήμα αρχίσει να διαβάσει, πρέπει να διεκδικήσει το σημαφόρο του αρχείου. Ύστερα, παρατηρούμε το εξής: εφόσον δεν πρόκειται για ένα συμβατικό αρχείο αλλά για μία συσκευή χαρακτήρων που λαμβάνει μετρήσεις ανά τακτά χρονικά διαστήματα, δεν υφίσταται ούτε αρχή ούτε τέλος του αρχείου με τη σημασία που έχουν συμβατικά αυτοί οι όροι. Επομένως, θεωρούμε πως βρισκόμαστε στην αρχή του αρχείου (**f_pos = 0*), όταν βρισκόμαστε στην αρχή μίας μέτρησης και στο τέλος του αρχείου (**f_pos = state->buf_lim*sizeof(unsigned char)*) όταν έχουμε διαβάσει όλα τα bytes μίας προηγούμενης. Στην μεν πρώτη περίπτωση πρέπει να γίνει έλεγχος αν έχουν φτάσει νέες μετρήσεις. Αν δεν υπάρχουν νέα δεδομένα, η διεργασία πρέπει να αφήσει το σημαφόρο και να κοιμηθεί (ή να επανέλθει ο έλεγχος στον χρήστη, αν έχουμε *non-blocking I/O*). Αλλιώς, η εκτέλεση συνεχίζει κανονικά. Στην δε δεύτερη, πρέπει να μηδενίσουμε την τιμή του *f_pos*. Επιπλέον, κατά το διάβασμα πρέπει πάντοτε να γίνεται έλεγχος αν τα ζητούμενα bytes υπερβαίνουν τα διαθέσιμα, ώστε να μην ξεφύγουμε από τα όρια του *buf_data*. Τέλος, προκειμένου να μην κάνουμε dereference διευθύνσεις του χρήστη μέσα στον κώδικα του πυρήνα, χρησιμοποιείται η *copy_to_user*.

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

    struct linux_sensor_struct *sensor;
    struct linux_chrdev_state_struct *state;

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

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

    debug("about to read from sensor\n");
    if (down_interruptible(&state->lock)){
        return -ERESTARTSYS;
    }

    if (*f_pos == 0) {
        while (linux_chrdev_state_update(state) == -EAGAIN) {
            up(&state->lock);
            if (filp->f_flags & O_NONBLOCK) return -EAGAIN;
            debug("\n%s\n" reading: going to sleep\n", current->comm);
            if (wait_event_interruptible(sensor->wq, linux_chrdev_state_needs_refresh(state))) {
```

```

        debug("restart read by ctrl+c");
        return -ERESTARTSYS;
    }
    debug("\ "%s\ " has woken up and reached this point of execution, new data must be
available!\n", current->comm);
    if (down_interruptible(&state->lock)) return -ERESTARTSYS;
}
}

if (*f_pos >= state->buf_lim*sizeof(unsigned char)) { //under normal circumstances, this is
unreachable
    debug("reached eof, nothing to copy");
    *f_pos = 0;
    ret = 0;
    goto out;
}

if (*f_pos + cnt > state->buf_lim*sizeof(unsigned char)) {
    cnt = state->buf_lim*sizeof(unsigned char) - *f_pos;
    debug("more bytes requested than existing, new cnt is %d", (int) cnt);
}

debug("about to copy %d bytes", (int) cnt);
if (copy_to_user(usrbuf, state->buf_data + *f_pos, cnt)) {
    debug("user gave invalid adress");
    ret = -EFAULT;
    goto out;
}

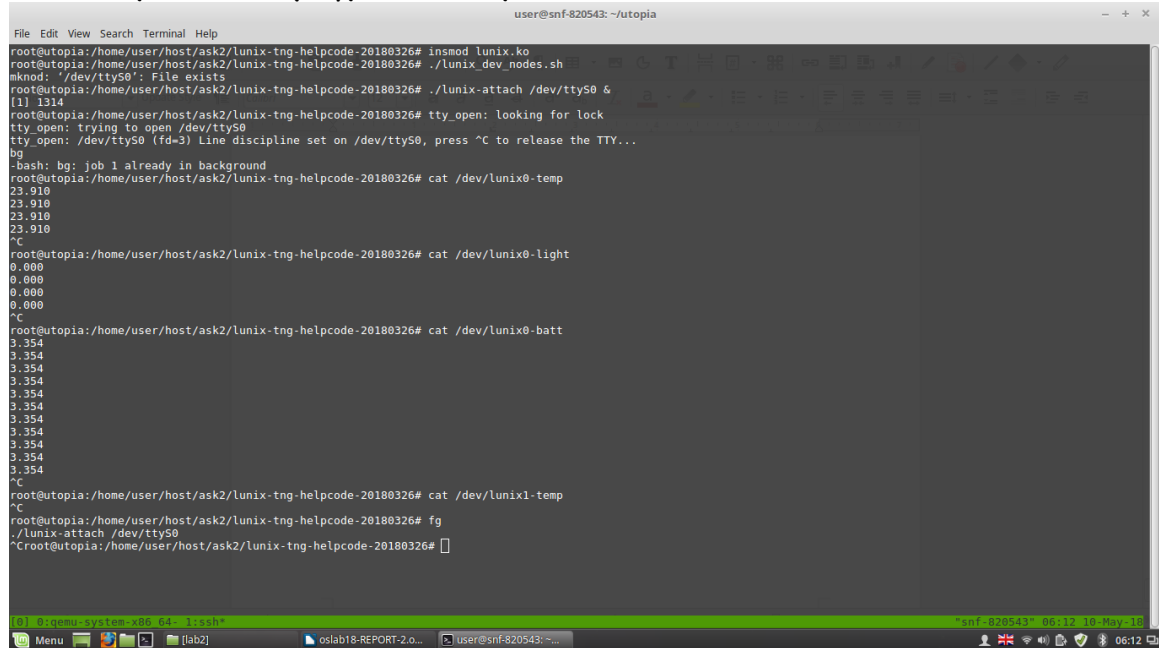
*f_pos += cnt;
ret = cnt;
if (*f_pos == state->buf_lim*sizeof(unsigned char)) {
    debug("reached eof, rewinding");
    *f_pos = 0;
}

out:
    up(&state->lock);
    return ret;
}

```

Επίδειξη λειτουργίας

Ακολουθεί ένα screenshot που δείχνει τη λειτουργία του οδηγού. Πρώτα παίρνουμε διάφορες μετρήσεις από ενεργό αισθητήρα και μετά δοκιμάζουμε να πάρουμε μετρήσεις από ανενεργό αισθητήρα. Φαίνονται όλα τα στάδια των δοκιμών που πραγματοποιούμε.



```
File Edit View Search Terminal Help
user@snf-820543: ~/utopia

root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# insmod linux.ko
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# ./linux_dev_nodes.sh
mkmod: /dev/ttyS0: File exists
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# ./linux-attach /dev/ttyS0 &
[1] 1314
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# tty_open: looking for lock
tty_open: trying to open /dev/ttyS0
tty_open: /dev/ttyS0 (fd=3) Line discipline set on /dev/ttyS0, press ^C to release the TTY...
bg
-bash: bg: job 1 already in background
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# cat /dev/lunix0-temp
23.910
23.910
23.910
23.910
^C
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# cat /dev/lunix0-light
0.000
0.000
0.000
0.000
^C
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# cat /dev/lunix0-batt
3.354
3.354
3.354
3.354
3.354
3.354
3.354
3.354
3.354
3.354
3.354
^C
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# cat /dev/lunix1-temp
^C
root@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326# fg
./linux-attach /dev/ttyS0
^Croot@utopia:/home/user/host/ask2/linux-tng-helpcode-20180326#
```