Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

2η Εργαστηριακή Άσκηση : Οδηγός Ασυρμάτου Δικτύου Αισθητήρων στο Λειτουργικό Σύστημα Linux

Εργαστήριο Λειτουργικών Συστημάτων

Ροη : Υ

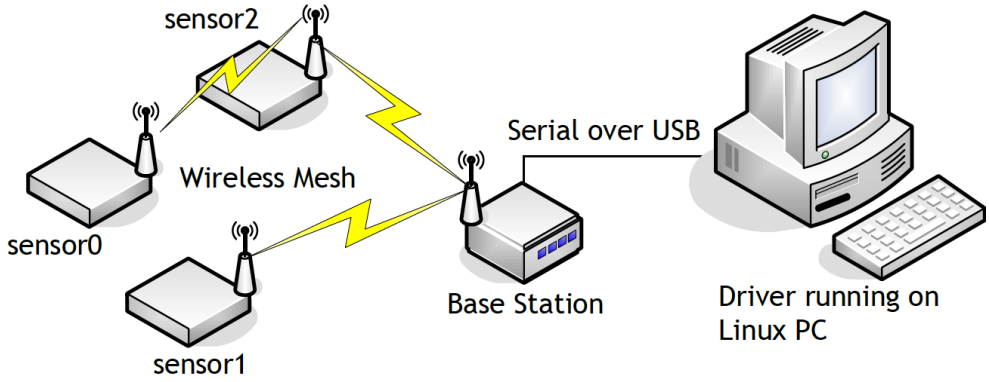
Ομάδα : **b04**

Ον/μο : **Βαβουλιώτης Γεώργιος**

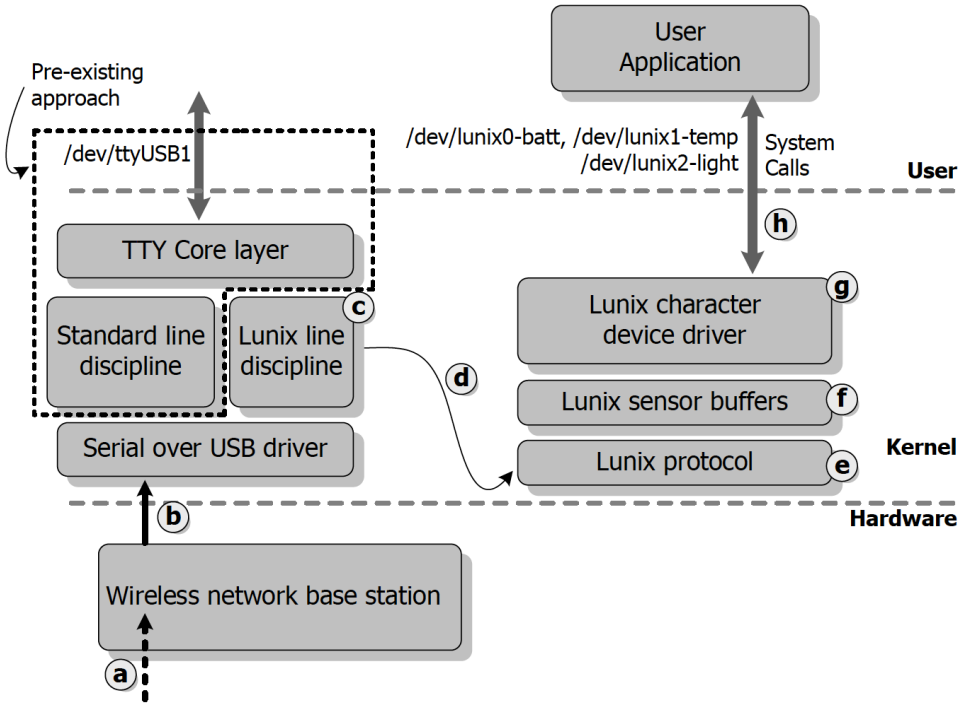
Α.Μ. : **03112083**

Εξάμηνο : **8**

Εισαγωγή - Σκοπός Άσκησης : Αντικείμενο της 2ης εργαστηριακής άσκησης είναι η υλοποίηση ενος οδηγού συσκεύης για ενα ασύρματο δίκτυο αισθητήρων κάτω απο το λειτουργικό σύστημα Linux. Το δίκτυο αυτό περιέχει ενα αριθμό απο αισθητήρες και ενα σταθμό βάσης, ο οποίος συνδέεται με USB με μηχάνημα που τρέχει Linux στο οποίο θα εκτελείται ο driver που καλούμαι να υλοποιήσω(το όνομα του driver είναι **Lunix:TNG**). Στη πράξη, καθένας απο τους αισθητήρες που έχουμε συλλέγει μετρήσεις για τάση,θερμοκρασία και φωτεινότητα και στέλνει τα δεδομένα που έχει συλλέξει στο σταθμό βάσης ο οποίος με τη σειρά του μέσω κυκλώματος serial over usb στέλνει-προωθεί τα data στο υπολογιστικό σύστημα στο οποίο τρέχει ο driver. Θα πρέπει να τονιστεί επίσης οτι οι αισθητήρες οργανώνονται σε ενα mesh και είναι εκείνοι υπεύθυνοι για το αν κάποιος βρίσκετε εκτός εμβέλειας, δηλαδή το δίκτυο καταφέρνει και προσαρμόζεται ακόμα και αν κάποιος-κάποιοι αισθητήρες είναι εκτός εμβέλειας.



Θεωριτική Υλοποίηση : Αυτό που πρέπει να υλοποιήσω είναι ενας οδηγός συσκευής χαρακτήρων(char device driver) ο οποίος αυτό που θα κάνει είναι να λαμβάνει τα δεδομένα απο το δίκτυο των αισθητήρων και θα εξάγει στο χώρο χρήστη σε διαφορετικές συσκευές, ανάλογα με το είδος της μέτρησης και τον αισθητήρα. Για γίνει αντιληπτό το λογισμικό του συστήματος που εξετάζουμε παραθέτω παρακάτω ενα screenshot που το αναπαριστά για να μπορέσω να εξηγήσω καλύτερα την ιδέα υλοποίησης :



Απο το παραπάνω σχήμα έχω να υλοποιήσω τα (g) και (h) αφού τα υπόλοιπα δίνονται έτοιμα αλλά για να μπορέσω να τα υλοποιήσω θα πρέπει να γνωρίζω καλά το τρόπο με τον οποίο λειτουργεί το παραπάνω σύστημα. Αρχικά αφού ο σταθμός βάσης συλλέξει τα δεδομένα απο τους αισθητήρες και τα προωθήσει μέσω του κυκλώματος serial over usb στο υπολογιστικό μας σύστημα, το στρώμα line discipline του Linux θα παραλαμβάνει τα δεδομένα και θα προωθεί σε ενα στρώμα το οποίο θα ερμηνεύει τα δεδομένα, θα τα επεξεργαστεί και θα τα αποθηκεύει σε κατάλληλους buffers ανάλογα με το είδος της μέτρησης και τον αισθητήρα.

Υλοποίηση του Char Device Driver Lunix:TNG : Αρχικά απο το φάκελο που μας δίνεται με τον βοηθητικό κώδικα θα πρέπει να τονίσω οτι περισσότερα αρχεία δεν τα πείραξα αλλά μόνο στο **lunix-chrdev.c** έγραψα κώδικα για να μπορέσω να υλοποιήσω τις συναρτήσεις που υλοποιούν τις λειτουργίες που οφείλει να κάνει ο driver. Το **struct file_operations lunix_chrdev_fops** περιέχει τις συναρτήσεις που ο driver πρέπει να περιέχει. Επιπλέον υλοποίησα και άλλες συναρτήσεις οι οποίες φαίνονται στον κώδικα της άσκησης τον οποίο και παραθέτω παρακάτω(το περιεχόμενο του αρχείου **lunix-chrdev.c**):

```
/*
 * lunix-chrdev.c
 *
 * Implementation of character devices
 * for Lunix:TNG
 *
 * Giorgos Vavouliotis
 * A.M. : 03112083
 * e-mail : <nuovocominzio@hotmail.com>
 *
 */


#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/list.h>
#include <linux/cdev.h>
#include <linux/poll.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/ioctl.h>
#include <linux/types.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/mmzone.h>
#include <linux/vmalloc.h>
#include <linux/spinlock.h>

#include "lunix.h"
#include "lunix-chrdev.h"
#include "lunix-lookup.h"

/*
 * Global data
 */

struct cdev lunix_chrdev_cdev;

/*
 * 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;
    uint32_t sensor_timestamp_temp;

    /* see that the sensor exists and it is not NULL */
    WARN_ON (!(sensor = state->sensor));

    debug( "quick check for update without lock\n" );

    /* put in sensor_timestamp the last update */
    sensor_timestamp_temp = sensor->msr_data[state->type]->last_update;

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

    /* here i check if i have to make UPDATE */
    if (state->buf_timestamp < sensor_timestamp_temp){
        /* if returns 1 means that an update is needed */
        debug("I have to make an update guys\n");
        return 1;
    }
    else return 0;
}


static long convert( long index, enum lunix_msr_enum type )
{
    /* i can't have more than 65535 states */
    WARN_ON( index < 0 || index > 65535 );

    /* check the type of the data and take the value of the right lookup table */
    switch ( type )
    {
        case BATT:
            return lookup_voltage[ index ];
            break;
        case TEMP:
            return lookup_temperature[ index ];
            break;
        case LIGHT:
            return lookup_light[ index ];
            break;
        default:
            printk( KERN_ALERT "there is not lookup table to convert from!\n" );
            return 0;
            break;
    }
}



static int finalConvert(long val,char *buf)
{
    int counter; /* how many bytes snprintf writes */

    /* lookup tables can have also negative values */
    if (val>=0)
        counter = snprintf( buf, LUNIX_CHRDEV_BUFSZ,"%02ld.%03ld\n", val / 1000, val %
1000 );
    else
        counter = snprintf( buf, LUNIX_CHRDEV_BUFSZ,"-%02ld.%03ld\n", -val / 1000, -val %
1000 );

    /* WARN_ON is not neccesary here because
     * snprintf doesnt write more than LUNIX_CHRDEV_BUFSZ bytes
     * but i put it to be sure.
     */

    WARN_ON( !( counter < LUNIX_CHRDEV_BUFSZ ) );
    return counter;
}
```

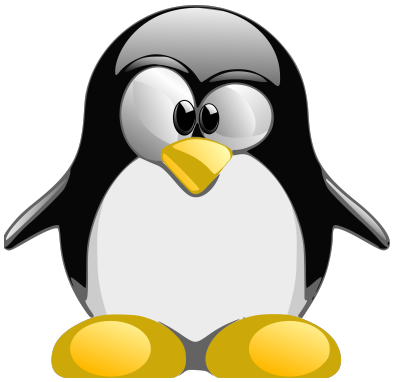
```c
/*
 * Updates the cached state of a character device
 * based on sensor data. Must be called with the
 * character device state lock held.
 */

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

    int ret;
    int need_update; /*the integer need_update informs us if an update is needed */

    uint32_t new_data; /*is the newly received data(index to a specific lookuptable) in
                        raw form*/
    uint32_t new_timestamp; /* is the time that the newly data has been received */

    long converted_data; /* this variable has the data after the conversion */

    unsigned long flags; /* Spinlock flags variable */

    debug("Update Starting\n");

    /* take my sensor so i have access to the struct to make the update */
    sensor = state->sensor ;
    /* check if an update is needed */
    need_update = lunix_chrdev_state_needs_refresh(state);

    if (need_update == 1){

        /* I lock the critical section with spinlock. I save interrupt flags,
         * disable interrupts, acquire the spinlock and finally
         * restore the interrupts. In this way i avoid deadlocks.
         */
        spin_lock_irqsave(&sensor->lock,flags);

        /* take the new data and the new timestamp */
        new_data = sensor->msr_data[state->type]->values[0];
        new_timestamp = sensor->msr_data[state->type]->last_update;

        /* i unlock because i leave from the critical section */
        spin_unlock_irqrestore( &sensor->lock, flags );

        converted_data = convert( new_data, state->type );
        state->buf_lim = finalConvert(converted_data, state->buf_data);
        state->buf_timestamp = new_timestamp; /* renew the timestamp */
        ret = 0;
    }
    else{
        debug("Update not needed\n");
        ret = -EAGAIN; //this means that i didn't make an update
    }

    debug("Leaving\n");
    return ret;
}


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

static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
    /* Declarations */
    /* We follow the C90 standard and declare everything at the top */
    unsigned sensor_minor; // The minor number passed file special character file
    unsigned sensor_type; // The sensor type is battery(0), themperature(1) or light(2)
    unsigned sensor_id; // Each sensor can measure many things (view type above)
    int ret; // Return value of this function
```

```
struct lunix_chrdev_state_struct *dev_state;

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>]
 */

debug("imajor is: %d, iminor is: %d\n", imajor( inode ), iminor( inode ) );

/* Obvious Check: Major number must match */
if (imajor(inode) != 60){
    printk(KERN_ERR "The driver is not suited for major number 60...What???\n");
    ret = -ENODEV;
    goto out;
}


sensor_minor = iminor(inode); /* get minor number */
sensor_type = sensor_minor%8; /* get the sensor type */
sensor_id = sensor_minor / 8; /* find which sensor is */

if (sensor_minor >= lunix_sensor_cnt*8){
    printk( KERN_ERR "up to %d sensors are supported,with 8 measurements each\n",
           lunix_sensor_cnt );
    ret = -ENODEV;
    goto out;
}

debug( "sensor type is: %d\n", sensor_type );

    if (sensor_type >= N_LUNIX_MSR){
            printk(KERN_ERR "no sensor for this purpose\n");
            ret = -ENODEV;
            goto out;
    }

/* Allocate a new Lunix character device private state structure */
dev_state = kmalloc (sizeof *dev_state, GFP_KERNEL );
if (!dev_state){
    printk( KERN_ERR "could not allocate memory for private field of pointer.\n" );
    ret = -ENOMEM;
    goto out;
}

/*Fill up private data -> save to my state all the necessary infos about my device*/
dev_state->type = sensor_type;

/* Note: dev_state sensor is a struct constaining
 * batt, temp, light together with spinlock and wait queue.
 */
dev_state->sensor = &lunix_sensors[ sensor_id ];

/* This variable holds the bytes in the buf_data buffer.
 * Thus, we do not need to zero out the actual buffer.
 */
dev_state->buf_lim = 0;

/* Initially the buffer was never updated */
dev_state->buf_timestamp = 0;

/* Initially the semaphore (used as mutex) is released.
 * We know that each open call creates a new file pointer. Why
 * bother to lock the private data? That is because threaded
 * applications share the same open file descriptors, unless
 * stated otherwise.
 */
sema_init (&dev_state->lock,1);
```

```
    /* Update the open file's private data */
    filp->private_data = dev_state;

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


static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
    /* release is called once every an effective
     * open call is made. e.g. a fork call does not
     * allocate memory if not needed, thus release does
     * not create memory leaks
     */

    debug( "Freeing the private data struct\n" );

    /* Need to be carefull to call kfree */
    WARN_ON ( !filp->private_data );
    kfree( filp->private_data );

    return 0;
}


static long lunix_chrdev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    /* Why? */
    return -EINVAL;
}


static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf, size_t cnt,
loff_t *f_pos)
{
    ssize_t ret;
    size_t count; /* has the number of bytes that the buffer contains */
    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) )
    {
        ret = -ERESTARTSYS;
        goto out;
    }

    /*
     * 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) {
        /* Issue a new read command */
        while (lunix_chrdev_state_update(state) == -EAGAIN) {
            /* EAGAIN is returned only if the cache is empty and new data are not
              available */
            up(&state->lock);
            debug("it is time to go for a sleep\n");

            /* The process needs to sleep */
            /* See LDD3, page 153 for a hint */
```

```
            if (wait_event_interruptible(sensor->wq,lunix_chrdev_state_needs_refresh(state))){
                ret = -ERESTARTSYS;
                goto out;
            }

            if (down_interruptible(&state->lock)){
                ret=-ERESTARTSYS;
                goto out;
            }
            debug("now it is time to wake up\n");
        }
    }

    /* Determine the number of cached bytes to copy to userspace */
    count = state->buf_lim - *f_pos;

    /* If userspace buffer does not have enough space,
     * then fill the buffer with cnt bytes
     */
    if (count > cnt ){
        count = cnt;
    }

    debug( "copying %ld chars starting at position %lu\n",(long) count, (unsigned
long)*f_pos );

    /* copy to user buffer */
    if (copy_to_user(usrbuf,state->buf_data+(*f_pos),count)){
        ret = -EFAULT;
        goto out_with_lock;
    }
    /* if count !=0 something is wrong */
    ret = count;
    /* the file pointer position is increased by the
     * same amount */
    *f_pos += count;

    /* Auto-rewind on EOF mode? */
    if (*f_pos >= state->buf_lim){
        *f_pos = 0;
    }

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


static int lunix_chrdev_mmap(struct file *filp, struct vm_area_struct *vma)
{
    return -EINVAL;
}


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
};
```

```
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");

    /* initialize the driver */
    cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
    lunix_chrdev_cdev.owner = THIS_MODULE;

    /* MKDEV does 20 left shifting and then OR with the minor number */
    dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);

    /* register the region that you want for device with name lunix */
    ret = register_chrdev_region(dev_no,lunix_minor_cnt,"lunix");
    if (ret < 0) {
        debug("failed to register region, ret = %d\n", ret);
        goto out;
    }

    /* add the device that is represented by
     * the 1st operand of the cdev_add,to the system
         */
    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;
}


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");
}
```



Παρακάτω θα σχολιάσω κάθεμια απο τις συναρτήσεις που περιέχονται στο παραπάνω κώδικα λεπτομερώς και θα επισημάνω όλα τα κρίσιμα σημεία υλοποίησης(πρέπει να επισημάνω επίσης οτι ο κώδικας έχει πλήθος σχολίων τα οποία είναι πολύ βοηθητικά για την κατανόηση του). Συγκεκριμένα :

1. **Lunix_chrdev_init() :** Η συνάρτηση **lunix_chrdev_init()** είναι η πρώτη συνάρτηση που καλείται και αυτό που κάνει τελικά είναι να αρχικοποιεί τη συσκευή χαρακτήρων. Αρχικά καλείται η συνάρτηση **cdev_init()** με κατάλληλα ορίσματα ώστε να αρχικοποιηθεί η συσκευή χαρακτήρων με διαθέσιμές συναρτήσεις αυτές που ορίζονται μέσα στο **struct file_operations lunix_chrdev_fops**, το οποίο ανέφερα και παραπάνω. Στη συνέχεια καλείται η συνάρτηση **MKDEV()**, η οποία παίρνει σαν ορίσματα ενα major και ενα minor number και κάνοντας μια λογική πράξη(20 shift left major and after OR with minor) μεταξύ των ορισμάτων της δημιουργεί την νέα συσκεύη, η οποία είναι τύπου dev_t. Έπειτα καλώ την συνάρτηση **register_chrdev_region()** για να δεσμεύσω τους αριθμους τους οποίους χρειάζομαι για να αναφέρομαι στις συσκευές. Τέλος καλείται η συνάρτηση **cdev_add()** για να ενημερώσω τον πυρήνα οτι μια νέα συσκεύη δημιουργήθηκε ώστε να γνωρίζει πλεόν οτι υπάρχει. Προφανώς μετά την κλήση καθεμίας απο τις συναρτήσεις αυτές ελέγχω για πιθανό σφάλμα και αν υπάρχει ενημερώνω με κατάλληλο μήνυμα και κάνω την ανάλογη ενέργεια. Θα πρέπει να τονίσω επίσης πως όταν τρέχω την άσκηση η συνάρτηση **lunix_chrdev_init()** είναι αυτή που καλέιται πρώτη για να μπορέσει να γίνει η αρχικοποίηση της συσκευής.

2. **Lunix_chrdev_destroy() :** Η συνάρτηση **lunix_chrdev_destroy()** μας δίνεται έτοιμη ωστόσο κρίνω αναγκαίο να την εξηγήσω για πληρότητα. Αρχικά παίρνω το dev_number της συσκευής που θέλω να αφαιρέσω και έπειτα καλώ την συνάρτηση **cdev_del()** η οποία αφαιρεί τη συσκευή χαρακτήρων που της δίνω σαν όρισμα(στην πράξη της δίνω το cdev struct που πρέπει να αφαιρεθεί). Στο σημείο αυτό θα πρέπει να καλέσω και την συνάρτηση **unregister_chrdev_region()** για να αποδεσμεύσω τον χώρο που είχε δεσμευτεί για τους αριθμούς με τους οποίους θα γινόταν αναφορά στις αντίστοιχες συσκευές.

3. **Lunix_chrdev_open() :** Η συνάρτηση **lunix_chrdev_open()** είναι η συνάρτηση η οποία υλοποίησα για να ικανοποιεί το system call open(). Όταν γίνεται open() σε ένα ειδικό αρχείο τότε ο πυρήνας χρησιμοποιεί το major number του ειδικού αρχείου και κάνει πρόσβαση στο αντίστοιχο **struct file_operations lunix_chrdev_fops** και ικανοποιείται το system call μόνο αν έχει υλοποιηθεί απο εμένα η συνάρτηση **lunix_chrdev_open()**. Ας πάρουμε όμως τον κώδικα της **lunix_chrdev_open()** απο την αρχή. Αρχικά καλείται η **nonseekable_open()** για να γίνουν οι κατάλληλες αρχικοποιήσεις και να ενημερωθεί ο πυρήνας οτι η llseek λειτουργία δεν υποστηρίζεται. Έπειτα γίνονται οι απαραίτητοι έλεγχοι ώστε να είμαι σίγουρος οτι δεν έχει πάει κάτι στραβά(οι έλεγχοι είναι πολύ απλοί και δεν τους αναλύω διότι είναι προφανής και εξηγούνται στο κώδικα της άσκησης). Θα πρέπει όμως να επισημάνω οτι παίρνω το τύπο του κάθε αισθητήρα με χρήση του modulo αφού αν κάνω *minor_number % 8* παίρνω το τύπο του αισθητήρα διότι σε κάθε major number αντιστοιχούν 8 minor numbers. Στη συνέχεια αυτό που πρέπει να κάνω είναι να δεσμεύσω χώρο μνήμης με

χρήση της συνάρτησης **kmalloc()** για τη δομή dev_state. Έπειτα βάζω στα πεδία buf_lim και buff_timestamp του dev_state struct τη τιμή 0 αφού όταν ανοίγω το ειδικό αρχείο δεν έχει γίνει κάποιο update και ο buffer είναι αρχικά άδειος(τα πεδία αυτά θα τα χρησιμοποιήσω στις επόμενες συναρτήσεις και είναι πάρα πολύ σημαντικά για την σωστή υλοποίηση του char driver). Τέλος θα πρέπει να τονίσω οτι κάθε φορά που εκτελείται η open() ο πυρήνας δημιουργεί μια **δομή file** η οποία αναπαριστά ενα ανοιχτό αρχείο και μέσω αυτής έχει πρόσβαση στο **struct file_operations lunix_chrdev_fops**. Αυτό το υλοποιώ με την εντολή `fp->private_data = dev_state` δηλαδή θα έχω πρόσβαση στο dev_state μέσω του pointer `fp->private_data`.

4. **Lunix_chrdev_needs_refresh() :** Η συνάρτηση **lunix_chrdev_needs_refresh()** είναι μια βοηθητική συνάρτηση η οποία χρησιμοποιείται απο τη συνάρτηση **lunix_chrdev_update()** για να γίνει έλεχος αν ήρθαν νέα δεδομένα και πρέπει να γίνει update στα πεδία του **struct lunix_chrdev_state_struct** . Στη πράξη αυτό που κάνει η συνάρτηση αυτή είναι να συγκρίνει την τιμή του buf_timestamp η οποία υπάρχει στη δομή του ανοιχτού αρχείου με την τιμή του πεδίου last_update που περιέχει η δομή του αισθητήρα. Αν το πεδίο last_update είναι μεγαλύτερο σημαίνει οτι τα δεδομένα του αισθητήρα έχουν ανανεωθεί και οτι θα πρέπει να γίνει update στα δεδομένα του ανοιχτού αρχείου. Επομένως αν χρειάζεται να γίνει update η συνάρτηση αυτή επιστρέφει 1 αλλίως επιστρέφει 0 και η τιμή αυτή χρησιμοποείται απο τη συνάρτηση **lunix_chrdev_update()** με τρόπο που θα εξηγηθεί παρακάτω.

5. **Lunix_chrdev_update():** Η συνάρτηση **lunix_chrdev_update()** είναι μια απλή συνάρτηση και αρχικά καλει την συνάρτηση **lunix_chrdev_needs_refresh()** και αν αυτή επιστρέψει 0 τότε επιστρέφει **-EAGAIN**(θα εξηγήσω στη συνάρτηση **lunix_chrdev_read()** τι ακριβώς σημαίνει αυτό) και τερματίζει. Αν όμως η **lunix_chrdev_needs_refresh()** επιστρέψει 1 σημαίνει οτι έχουν έρθει νέα δεδομένα και θα πρέπει να γίνει update. Στη περίπτωση αυτή θα μπω σε critical section και για το λόγο αυτό θα χρησιμοποιήσω spinlock για να κλειδώσω ώστε να διασφαλίσω οτι τα δεδομένα που θα αντιγραφούν θα είναι έγκυρα και δεν θα γίνει κάποιο λάθος. Αυτό που κάνω όταν είμαι στο κρίσιμο τμήμα είναι να ενημερώσω σύμφωνα με τα νέα δεδομένα και να αλλάξω την τιμή του buff_timestamp για να έχω πλέον τη νέα χρονική τιμή που έγινε update και να μπορώ να ελέγχω ορθά στη συνέχεια αν πρέπει να γίνει κάποιο άλλο update ή οχι. Επίσης μέσα στη συνάρτηση αυτή καλώ και δυο άλλες συναρτήσεις τις οποίες υλοποίησα εγώ, με ονόματα **convert()** και **finalconvert()**. Ο λόγος που υλοποιώ τις συναρτήσεις αυτές είναι οτι ο πυρήνας του Linux δεν υποστηρίζει floating point πράξεις διότι η FPU **δεν** σώζεται αν πάω απο user space σε kernel space(υπάρχει και ενας πιο σπάνιος λόγος ο οποίος είναι κάποια αρχιτεκτονική να μην υποστηρίζει floating point εντολές και το αναφέρω απλά για πληρότητα). Για να μπορέσω να κάνω το κατάλληλο conversion αρχικά καλώ τη συνάρτηση **convert()** η οποία επιστρέφει τη τιμή του πεδίο του κατάλληλου lookup_table που ορίζεται απο το δείκτη με όνομα index που δέχεται σαν όρισμα αφού πρώτα ελέγξει σε ποια απο τις τρεις μετρήσεις αναφερόμαστε. Για πληρότητα αναφέρω οτι τα lookup_tables είναι ενας τρόπος απεικόνισης των 16-bit τιμών(μπορεί να προκύψουν μέχρι 65536 καταστάσεις) που δεχόμαστε σε αναγνώσιμες τιμές τα οποία υλοποιούνται στο αρχείο

mk_lookup_tables.c. Για καθεμία απο τις μετρήσεις(φωτεινότητα, μπαταρία,θερμοκρασία) έχει υλοποιηθεί ενα look_up_table με αντίστοιχο όνομα για να γίνεται εύκολα η διάκριση και οι μετρήσεις αποθηκεύονται στο αρχείο **lunix-lookup.h**. Έπειτα η τιμή που πήραμε απο το κατάλληλο lookup_table στέλνετε στη συνάρτηση **finalConvert()** η οποία επιστρέφει πόσα bytes κατάφερε να γράψει η **snprintf()** στο buffer. Χρησιμοποιώ την snprintf() διότι είναι υλοποιημένη ώστε να γράφει στο buffer μέχρι **LUNIX_CHRDEV_BUFSZ** bytes και ποτέ περισσότερα(μπορεί να γράψει και λιγότερα αλλά ποτέ περισσότερα). Αφού βάλω στο πεδίο buf_lim τη τιμή που επέστρεψε η συνάρτηση **finalConvert()** ενημερώνω όπως είπα και πριν το πεδίο buf_timestamp. Μετά το σωστό conversion των δεδομένων και την αποθήκευσή τους στο buffer πλέον μπορούν να εμφανιστούν στο χρήστη στη μορφή που επιθυμούμε. Τέλος κρίνεται αναγκαίο να εξηγήσω το λόγο για τον οποίο επέλεξα να χρησιμοποιήσω spinlocks για να κάνω το κλείδωμα που ανέφερα παραπάνω. Ο λόγος είναι οτι τα spinlocks όταν ενα τμήμα κώδικα δεν μπορεί να πάρει το κλείδωμα επειδή είναι κατειλμμένο, δεν θα αφήσουν τη cpu αλλά θα προσπαθούν συνέχεια μέχρι να τα καταφέρουν, δηλαδή θα κάνουν busy-wait. Εμένα με βολεύει αυτό διότι η συνάρτηση που ανανεώνει τα δεδομένα τρέχει σε interupt context όταν πάρει νεά δεδομένα γεγονός το οποίο καθιστά αναγκαία τα spinlocks για να κάνω το κλειδώμα διότι αν έκανα χρήση κάποιου mutex είναι πολυ πιθανό να έπεφτα σε deadlock απο το οποίο δεν θα έφευγα ποτέ.

6. **Lunix_chrdev_read() :** Η συνάρτηση **lunix_chrdev_read()** αυτό που κάνει αρχικά είναι να ανακτά τη δομή **lunix_chrdev_state_struct** και κάνει τους απαραίτητους ελέγχους εγκυρότητας. Θα πρέπει να τονιστεί οτι η δομή είναι κοινή για διεργασίες που έχουν προκύψει απο κάποιο **fork()** και για το λόγο αυτό θα κάνω χρήση σημαφόρου για να κλειδώσω όταν μια διεργασία μπεί στο κρίσιμο τμήμα. Συγκεκριμένα όταν γίνει πρόσβαση στο κρίσιμο τμήμα εκτελείται η συνάρτηση **down_interruptible()** η οποία θα δεσμεύσει το σημαφόρο του αντίστοιχου ανοιχτού αρχείου και όλες οι άλλες διεργασίες που επιθυμούν να μπουν κλειδώνονται έξω δηλαδή μπλοκάρουν. Για το λόγο αυτό είναι απαραίτητη η χρήση σεμαφόρου. Θα πρέπει να τονίσω οτι είναι πιθανό μια διεργασία που έχει μπει στο κρίσιμο τμήμα να 'φάει' κάποιο σήμα και τότε η εκτέλεση θα κολλήσει αφού θα πέσω σε deadlock απο το οποίο δεν θα μπορώ να βγω. Για να αντιμετωπίσω αυτό το φαινόμενο κάνω χρήση της συνάρτησης **down_interruptible()** και καταφέρνω να αποδευσμεύσω το σεμαφορό ακόμα και αν συμβεί αυτό που ανέφερα παραπάνω και να συνεχιστεί ομάλα η εκτέλεση. Στη συνέχεια ελέγχω αν έχουν έρθει νέα δεδομένα. Αν δεν υπάρχουν διαθέσιμα νέα δεδομένα τότε η συνάρτηση **lunix_chrdev_update()** επιστρέφει **-EAGAIN** και τότε ο σημαφόρος ξεκλειδώνει, η διεργασία 'πέφτει για ύπνο' μέχρι να έρθουν νέα δεδομένα και μπαίνει σε μια ουρά προτεραιότητας(αυτό γίνεται με τη συνάρτηση **wait_event_interruptible()** που μας δίνεται έτοιμη). Αν τώρα έχει γίνει update των δεδομένων και είναι πλέον σε κατάλληλη μορφή για να μεταφερθούν στο user space, δηλαδή στο user space buffer ελέγχω αρχικά αν ο χρήστης ζήτησε λιγότερα δεδομένα απο αυτά που έχω διαθέσιμα και βάζω τον κατάλληλο αριθμό στη μεταβλητή count(επιτρέπω στο χρήστη να ζητήσει και λιγότερα δεδομένα απο τα διαθέσιμα). Η μεταφορά των ζητούμενων δεδομένων στο user space buffer γίνεται με κλήση της συνάρτησης **copy_to_user()**. Η συνάρτηση αυτή

αν δεν καταφέρει τελικά να γράψει στο user space buffer τα δεδομένα που ζητήθηκαν τότε επιστρέφει θετική τιμή, η οποία ισούται με τον αριθμό των bytes που δεν κατάφερε να αντιγράψει και τερματίζει με **-EFAULT**. Αν όμως η μεταφορά των ζητούμενων δεδομένων έγινε επιτυχώς, η **copy_to_user()** επιστρέφει μηδέν και στο σημείο αυτό θα πρέπει να γίνει και ανανέωση του περιεχομένου της μεταβλητής f_pos(αν έχω φτάσει στο τέλος του buffer πάω απο την αρχή ξανά όπως φαίνεται και στο κώδικα). Στο τέλος αποδεσμεύεται ο σημαφόρος αφού πλέον η διεργασία έχει εξέλθει απο το κρίσιμο τμήμα.

7. **Lunix_chrdev_release() :** Η συνάρτηση **lunix_chrdev_release()** είναι μια συνάρτηση η οποία καλείται μόνο σε περίπτωση που όλες οι διεργασίες που έχουν πρόσβαση στη δομή του ανοιχτού αρχείου κάνουν close() (εκτελέσουν το system call close()). Θα πρέπει να επισημάνω οτι η δομή file(δομή ανοιχτού αρχείου) είναι αυτή που ανέφερα σε προηγούμενη εξήγηση και είναι εκείνη που δημιουργείται απο τον πυρήνα όταν γίνει κάποιο open().

Πίνακας Χρωμάτων

Σκούρο Μπλε : source files και header files.
Μωβ : συναρτήσεις που υλοποίησα εγώ για τον driver.
Πορτοκαλί : έτοιμες συναρτήσεις που χρησιμοποιούνται.
Κόκκινο : δομές - structs που χρησιμοποιούνται.

Για να καταφέρω να ελέγξω αν όντως ο driver μου κάνει σωστά τις απαιτούμενες λειτουργίες και δίνει σωστά αποτελέσματα θα πρέπει να κάνω τις εξής ενέργειες:

- Συνδέομαι στο vm με ssh , κάνω sshfs για να είμαι σίγουρος για τα αρχεία μου.

```
gvavou5@linas ~/Desktop $ ssh -p 22223 root@localhost
root@localhost's password:
Linux utopia 3.16.0-0.bpo.4-amd64 #1 SMP Debian 3.16.7-ckt4-3~bpo70+1 (2015-02-12) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 21 04:52:36 2016 from 10.0.2.2
root@utopia:~# sshfs -o allow_other gvavou5@10.0.2.2:/home/gvavou5 /home/user/host
gvavou5@10.0.2.2's password:
root@utopia:~# cd /home/user/host/utopia/final_code/
root@utopia:/home/user/host/utopia/final_code#
```

- Πηγαίνω στο directory που έχω των κώδικα της άσκησης και κάνω compile με τη βοήθεια του δοσμένου Makefile.

```
root@utopia:/home/user/host/utopia/final_code# ls
lunix-attach    lunix-chrdev_karaman.c  lunix.ko       lunix-lookup.h  lunix-module.o    lunix-protocol.o  Makefile           Module.symvers
lunix-attach.c  lunix-chrdev.o          lunix-ldisc.c  lunix.mod.c     lunix.o           lunix-sensors.c   mk_lookup_tables
lunix-chrdev.c  lunix_dev_nodes.sh      lunix-ldisc.h  lunix.mod.o     lunix-protocol.c  lunix-sensors.o   mk_lookup_tables.c
lunix-chrdev.h  lunix.h                 lunix-ldisc.o  lunix-module.c  lunix-protocol.h  lunix-tcp.sh      modules.order
root@utopia:/home/user/host/utopia/final_code# make
make -C /lib/modules/3.16.0-0.bpo.4-amd64/build M=/home/user/host/utopia/final_code 'V=1'  modules
make[1]: Entering directory `/usr/src/linux-headers-3.16.0-0.bpo.4-amd64'
make[1]: Entering directory `/usr/src/linux-headers-3.16.0-0.bpo.4-amd64'
make -C /usr/src/linux-headers-3.16.0-0.bpo.4-amd64 \
        KBUILD_SRC=/usr/src/linux-headers-3.16.0-0.bpo.4-common \
        KBUILD_EXTMOD="/home/user/host/utopia/final_code" -f /usr/src/linux-headers-3.16.0-0.bpo.4-common/Makefile \
        modules
test -e include/generated/autoconf.h -a -e include/config/auto.conf || (               \
        echo >&2;                                                       \
        echo >&2 "  ERROR: Kernel configuration is invalid.";           \
        echo >&2 "         include/generated/autoconf.h or include/config/auto.conf are missing.";\
        echo >&2 "         Run 'make oldconfig && make prepare' on kernel src to fix it.";      \
        echo >&2 ;                                                      \
        /bin/false)
mkdir -p /home/user/host/utopia/final_code/.tmp_versions ; rm -f /home/user/host/utopia/final_code/.tmp_versions/*
make -f /usr/src/linux-headers-3.16.0-0.bpo.4-common/scripts/Makefile.build obj=/home/user/host/utopia/final_code
(cat /dev/null;   echo kernel//home/user/host/utopia/final_code/lunix.ko;) > /home/user/host/utopia/final_code/modules.order
make -f /usr/src/linux-headers-3.16.0-0.bpo.4-common/scripts/Makefile.modpost
  find /home/user/host/utopia/final_code/.tmp_versions -name '*.mod' | xargs -r grep -h '\.ko$' | sort -u | sed 's/\.ko$/.o/' | scripts/mod/modpost -m  -i ./Module.symv
ers -I /home/user/host/utopia/final_code/Module.symvers  -o /home/user/host/utopia/final_code/Module.symvers -S -w  -s -T -
make[1]: Leaving directory `/usr/src/linux-headers-3.16.0-0.bpo.4-amd64'
root@utopia:/home/user/host/utopia/final_code#
```

- Δημιουργώ τους κόμβους(nodes) για καθένα απο τους αισθητήρες με χρήση της εντολής **`./lunix_dev_nodes.sh`**(αρκεί να τους δημιουργήσω μια φορά δεν χρειάζεται να το κάνω κάθε φορά, γι'αυτό γράφει File exists).

```
root@utopia:/home/user/host/utopia/final_code# ./lunix_dev_nodes.sh
mknod: `/dev/ttyS0': File exists
root@utopia:/home/user/host/utopia/final_code#
```

- Εκτελώ την εντολή **`insmod /lunix.ko`** και μετά την **`./lunix-attach /dev/ttyS0`** και καταφέρνω να παίρνω δεδομένα απο τους αισθητήρες τα οποία θα εξηγήσω στη συνέχεια πως μπορώ να τα δώ.

```
root@utopia:/home/user/host/utopia/final_code# insmod ./lunix.ko
root@utopia:/home/user/host/utopia/final_code# ./lunix-attach /dev/ttyS0
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...
```

- Πλέον δέχομαι δεδομένα απο τους αισθητήρες και μπορώ πολύ εύκολα να τα δώ και να ελέγξω αν όντως παίρνω σωστά αποτελέσματα. Αυτό που πρέπει να κάνω είναι ενα ανοίξω ενα άλλο terminal, συνδέομαι με ssh πάλι και κάνω το εξής:

```
root@utopia:~# cat /dev/lunix0-temp
23.814
23.814
23.814
^C
root@utopia:~# cat /dev/lunix0-light
00.152
00.152
00.152
^C
root@utopia:~# cat /dev/lunix0-batt
03.354
03.354
^C
root@utopia:~# cat /dev/lunix1-temp
31.477
31.477
^C
root@utopia:~# cat /dev/lunix1-batt
03.301
^C
root@utopia:~# cat /dev/lunix1-light
23.575
23.041
^C
root@utopia:~#
```

Πλέον βλέπω τις μετρήσεις τις οποίες στέλνει κάποιος αισθητήρας στο σταθμό βάσης για κάθεμία απο τις τιμές μπαταρία, θερμοκρασία και φωτεινότητα και οι τιμές αυτές είναι σωστές όπως έλεγξε και ο βοηθός του εργαστηρίου.

- Αν πατήσω την εντολή **`cat /proc/devices`** θα δω οτι ο driver μου υπάρχει μέσα στο αρχείο devices με major number 60 και όνομα lunix :

```
^Croot@utopia:/home/user/host/utopia/final_code# cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 60 lunix
128 ptm
136 pts
226 drm
252 bsg
253 watchdog
254 rtc

Block devices:
  2 fd
259 blkext
  7 loop
 11 sr
254 virtblk
root@utopia:/home/user/host/utopia/final_code#
```

- Μπορώ επίσης αν θέλω να διαγράψω το module. Αυτό μπορώ να το κάνω με την εντολή **`rmmod lunix`**, όπως φαίνεται και παρακάτω, αφού μετά την εντολή αυτή βλέπω το περιεχόμενο του φακέλου devices και λείπει ο driver που υλοποίησα:

```
root@utopia:/home/user/host/utopia/final_code# rmmod lunix
root@utopia:/home/user/host/utopia/final_code# cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
128 ptm
136 pts
226 drm
252 bsg
253 watchdog
254 rtc

Block devices:
  2 fd
259 blkext
  7 loop
 11 sr
254 virtblk
root@utopia:/home/user/host/utopia/final_code#
```

Χρήσιμα Εργαλεία : Για να καταφέρω να υλοποιήσω την άσκηση χρησιμοποίησα αρκετές φορές την εντολή **`dmesg`** για να ελέγξω αν αυτά που κάνω είναι σωστά και είμαι στη σωστή κατεύθυνση. Αν πατήσω την εντολή dmesg παίρνω αρκετές πληροφορίες αφού μου τυπώνει τα μηνύματα που πυρήνα. Ενα στιγμιότυπο απο την εκτέλεση της εντολής dmesg φαίνεται παρακάτω, απλά για πληρότητα:

```
[  317.236078] lunix_ldisc_receive called
[  317.276964] lunix_ldisc_receive: called, 25 characters have been received. Data at *cp: { 0x00, 0x33, 0x91, 0x81, 0x00, 0x00, 0x7b, 0x01, 0x37, 0x02, 0x36, 0x01, 0x6
6, 0x01, 0x68, 0x01, 0x80, 0x01, 0x76, 0x01, 0x7a, 0x01, 0xda, 0x8c, 0x7e }
[  317.277001] lunix_ldisc_receive called
[  317.754534] lunix_ldisc_receive: called, 2 characters have been received. Data at *cp: { 0x7e, 0x42 }
[  317.754549] lunix_ldisc_receive called
[  317.789010] lunix_ldisc_receive: called, 36 characters have been received. Data at *cp: { 0x7d, 0x5e, 0x00, 0x0b, 0x81, 0x1b, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x3
3, 0x91, 0x81, 0x00, 0x00, 0x75, 0x01, 0xe9, 0x01, 0x02, 0x00, 0x20, 0x01, 0x72, 0x01, 0x86, 0x01, 0x8d, 0x01, 0x59, 0x01, 0xa1, 0x78, 0x7e }
[  317.789085] lunix_ldisc_receive called
[  317.789205] lunix_chrdev_state_needs_refresh: quick check for update without lock
[  317.789212] lunix_chrdev_state_needs_refresh: I have to make an update guys
[  317.789215] lunix_chrdev_read: now it is time to wake up
[  317.789218] lunix_chrdev_state_update: Update Starting
[  317.789220] lunix_chrdev_state_needs_refresh: quick check for update without lock
[  317.789223] lunix_chrdev_state_needs_refresh: I have to make an update guys
[  317.789226] lunix_chrdev_state_update: Leaving
[  317.789230] lunix_chrdev_read: copying 7 chars starting at position 0
[  317.789262] lunix_chrdev_state_update: Update Starting
[  317.789265] lunix_chrdev_state_needs_refresh: quick check for update without lock
[  317.789267] lunix_chrdev_state_update: Update not needed
[  317.789270] lunix_chrdev_state_update: Leaving
[  317.789272] lunix_chrdev_read: it is time to go for a sleep
[  317.789275] lunix_chrdev_state_needs_refresh: quick check for update without lock
[  317.789277] lunix_chrdev_state_needs_refresh: quick check for update without lock
[  318.070836] lunix_chrdev_state_needs_refresh: quick check for update without lock
[  318.071003] lunix_chrdev_release: Freeing the private data struct
[  319.040383] lunix_ldisc_receive: called, 8 characters have been received. Data at *cp: { 0x7e, 0x42, 0x7d, 0x5e, 0x00, 0x0b, 0x81, 0x1b }
[  319.040405] lunix_ldisc_receive called
[  319.074657] lunix_ldisc_receive: called, 30 characters have been received. Data at *cp: { 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x33, 0x91, 0x81, 0x00, 0x00, 0x7b, 0x0
1, 0x37, 0x02, 0x38, 0x01, 0x67, 0x01, 0x68, 0x01, 0x7f, 0x01, 0x75, 0x01, 0x79, 0x01, 0x39, 0x69, 0x7e }
[  319.074704] lunix_ldisc_receive called
[  319.547569] lunix_ldisc_receive: called, 6 characters have been received. Data at *cp: { 0x7e, 0x42, 0x7d, 0x5e, 0x00, 0x0b }
[  319.547588] lunix_ldisc_receive called
[  319.580449] lunix_ldisc_receive: called, 24 characters have been received. Data at *cp: { 0x81, 0x1b, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x33, 0x91, 0x81, 0x00, 0x0
0, 0x74, 0x01, 0xe9, 0x01, 0x02, 0x00, 0x1f, 0x01, 0x6f, 0x01, 0x77 }
[  319.580485] lunix_ldisc_receive called
[  319.580492] lunix_ldisc_receive: called, 8 characters have been received. Data at *cp: { 0x01, 0x79, 0x01, 0x48, 0x01, 0x9d, 0xf1, 0x7e }
[  319.580504] lunix_ldisc_receive called
[  319.611202] lunix_ldisc_receive: called, 13 characters have been received. Data at *cp: { 0x7e, 0x42, 0x7d, 0x5e, 0x00, 0xfd, 0x81, 0x02, 0x9a, 0xb3, 0xc7, 0x74, 0x7
e }
[  319.611226] lunix_ldisc_receive called
```