Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Λειτουργικών Συστημάτων

2η Εργαστηριακή Άσκηση, Lunix:TNG
(Τελική Αναφορά)

ΟΜΑΔΑ: oslab10

Ασπρογέρακας Ιωάννης (03118942)

Κανατάς Άγγελος-Νικόλαος (03119169)

Οι λεπτομέρειες σχεδίασης και υλοποίησης του ζητούμενου οδηγού (***Lunix:TNG character device driver***) αναφέρονται στο δοθέν φυλλάδιο *«Οδηγός Ασύρματου Δικτύου Αισθητήρων στο Λειτουργικό Σύστημα Linux»*. Εδώ, θα δείξουμε και θα επεξηγήσουμε τον κώδικα που χρειάστηκε να συμπληρώσουμε εμείς για την σωστή λειτουργία του στρώματος συσκευής χαρακτήρων.

Αρχικά, τροποποιήσαμε το header file **lunix-chrdev.h**, με σκοπό να προσθέσουμε "έξτρα" λειτουργίες στην συσκευή χαρακτήρων, όπως *non-blocking I/O* και *"raw"/"cooked"* τρόπους λειτουργίας μέσω κλήσεων συστήματος `ioctl()`. Παρακάτω, φαίνεται ο πηγαίος κώδικας με επεξηγηματικά σχόλια:

```
/*
 * lunix-chrdev.h
 *
 * Definition file for the
 * Lunix:TNG character device
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 * Ioannis Asprogerakas <el18942@mail.ntua.gr>
 * Angelos-Nikolaos Kanatas <el19169@mail.ntua.gr>
 */

#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 <linux/types.h>

#include "lunix.h"

/*
 * Private state for an open character device node
 */
struct lunix_chrdev_state_struct {
        /*
         * Opened devices are associated with a sensor
         * and a type of measurement
         */
        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;

        /*
         * For mutual exclusion between threads with the same open file
         * description
         */
```

```
        struct semaphore lock;

        /* Blocking and nonblocking operation */
        int nonblock_mode;

        /* Raw / cooked mode (default is cooked) */
        atomic_t raw_mode;
};

/*
 * 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_RAWDATA               _IO(LUNIX_IOC_MAGIC, 0)
#define LUNIX_IOC_COOKEDDATA            _IO(LUNIX_IOC_MAGIC, 1)

#define LUNIX_IOC_MAXNR                 2

#endif  /* _LUNIX_CHRDEV_H */
```

Στο **lunix-chrdev.c** συμπληρώσαμε την `lunix_chrdev_init()` που αρχικοποιεί τον οδηγό συσκευής χαρακτήρων και υλοποιήσαμε τα ***file operations*** της συσκευής χαρακτήρων (`open()`, `release()`, `read()`, `ioctl()`, `mmap()`), καθώς και δύο βοηθητικές συναρτήσεις (`lunix_chrdev_state_needs_refresh()`, `lunix_chrdev_state_update()`), όπου η λειτουργία τους αναλύεται παρακάτω.

- lunix_chrdev_init()

Σε αυτή τη συνάρτηση, καλούμαστε να κάνουμε ***register*** και ***add*** το σύνολο των συσκευών που θα ζητηθεί να χρησιμοποιηθούν σε μελλοντικές κλήσεις συστήματος από τον χρήστη, αρχικοποιώντας τον οδηγό μας. Ο kernel χρησιμοποιεί δομές **cdev** για να αναπαριστά συσκευές χαρακτήρων, οι οποίες περιέχονται στο **struct inode** του αρχείου.

Η δομή **cdev** του πυρήνα:

```
struct cdev {
        struct kobject kobj;
        struct module *owner;
        const struct file_operations *ops;
        struct list_head list;
        dev_t dev;
        unsigned int count;
} __randomize_layout;
```

Η παραπάνω δομή έχει πεδία τα οποία θα επιτρέψουνε μελλοντικά στο πυρήνα να βρει τις μεθόδους που θα ορίσουμε μέσω του πεδίου **file_operations**.

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

Το struct file_operations περιέχει τους απαραίτητους function pointers στις συναρτήσεις που υλοποιήσαμε. Για να περάσουμε την παραπάνω δομή στο cdev struct που ορίσαμε globally (**static struct** cdev lunix_chrdev_cdev), καλούμε την `cdev_init()`, η οποία περνάει στα πεδία του cdev τα file operations.

Έπειτα, καλούμε την `register_chrdev_region()`, όπου μέσω αυτής κάνουμε register το πεδίο των αριθμών συσκευών που θα εξυπηρετήσουμε, περνώντας ως ορίσματα τον συνολικό αριθμό τους, το πρώτο device με major και minor number που έχουμε θεωρήσει και το όνομα της συσκευής που σχετίζεται με αυτά τα ορίσματα (για να εμφανίζεται π.χ. ο driver μας στο */proc/devices*).

Το **dev_no** είναι ένα ειδικά διαμορφωμένο 32-bit integer που περιέχει μέσα του τους αριθμούς minor και major της συσκευής και μέσω χρήσης ειδικών MACROS μπορούμε να τους πάρουμε.

Τέλος, μετά την παραπάνω προετοιμασία καλούμε την συνάρτηση `cdev_add()`, η οποία κάνει γνωστό στον πυρήνα τον οδηγό μας. Εφόσον πετύχει η παραπάνω κλήση, ο πυρήνας δύναται να καλέσει τις μεθόδους που υλοποιήσαμε για τις συσκευές χαρακτήρων μας.

Αξίζει να σημειωθεί πως σε περίπτωση αποτυχίας του `cdev_add()` πρέπει να κληθεί η συνάρτηση `unregister_chrdev_region()`, δεδομένου πως πέτυχε το register, αποδεσμεύοντας το σύνολο των αριθμών των συσκευών χαρακτήρων μας.

```
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, "lunix");
        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;
}
```

Η `lunix_chrdev_destroy()` είναι ήδη υλοποιημένη και κάνει την αντίστροφη διαδικασία καλώντας `cdev_del()` και `unregister_chrdev_region()`, έτσι ώστε να αφαιρέσει τον οδηγό συσκευών χαρακτήρων μας και το πεδίο των αριθμών των συσκευών που δεσμεύσαμε προηγουμένως.

```
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()

Εδώ, αφού συσχετίσουμε το ανοιχτό αρχείο με τον sensor και το measurement που του αντιστοιχεί, δημιουργούμε μία private δομή ***chrdev_state***, όπου θα αποθηκεύουμε το current state της συσκευής χαρακτήρων για το συγκεκριμένο open file description. Συγκεκριμένα, αφού δεσμεύσουμε μνήμη με `kmalloc()`, αρχικοποιούμε την δομή όπως φαίνεται παρακάτω. Συγκεκριμένα, ορίζουμε τον τύπο του measurement και τον sensor που αντιστοιχεί στο open file, αρχικοποιούμε τον σημαφόρο όπου θα χρησιμοποιήσουμε για mutual exclusion μεταξύ threads που έχουν το ίδιο open file description, θέτουμε το non-bock mode ανάλογα με τα flags που έχουν δοθεί κατά την κλήση συστήματος `open()` από το userspace και τέλος ορίζουμε ότι by default ο τρόπος λειτουργίας μεταφοράς δεδομένων της συσκευής χαρακτήρων είναι *“cooked”* (μορφοποιημένες προσημασμένες δεκαδικές τιμές). Επιπλέον, η δομή αυτή περιέχει έναν buffer όπου εκεί θα αποθηκεύονται οι πιο πρόσφατες μορφοποιημένες (ή μη) μετρήσεις από τους sensors και είναι αυτός από όπου θα αντιγράφονται οι μετρήσεις για να τις στείλουμε στο user space όπως αναλύουμε παρακάτω. Μαζί με τον buffer έχουμε το timestamp του (πότε ενημερώθηκε για “φρέσκες” μετρήσεις), καθώς και το μήκος του, τα οποία τα αρχικοποιούμε στο 0 όπως φαίνεται παρακάτω. Τέλος, “αποθηκεύουμε” το state της συσκευής χαρακτήρων για το συγκεκριμένο ανοιχτό αρχείο στο struct file του, θέτοντας το private_data να δείχνει στο chrdev_state. Παρακάτω, φαίνεται ο πηγαίος κώδικας με επεξηγηματικά σχόλια:

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
        struct lunix_chrdev_state_struct *chrdev_state;
        unsigned int sensor, type;
        int ret;

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

        /* Device does not support seeking */
        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 = iminor(inode) / 8;
        type = iminor(inode) % 8;

        /* Allocate a new Lunix character device private state structure */
        if (!(chrdev_state = kmalloc(sizeof(struct lunix_chrdev_state_struct),
                                GFP_KERNEL))) {
                printk(KERN_ERR "Failed to allocate memory for character device
                                private state\n");
                ret = -ENOMEM;
                goto out;
        }

        /* Initialization of the device private state structure */
        chrdev_state->type = type;
        chrdev_state->sensor = &lunix_sensors[sensor];
        chrdev_state->buf_lim = 0;
        chrdev_state->buf_timestamp = 0;
        sema_init(&chrdev_state->lock, 1);
        chrdev_state->nonblock_mode = (filp->f_flags & O_NONBLOCK) ? 1 : 0;
        atomic_set(&chrdev_state->raw_mode, 0);

        /* Save the device state */
        filp->private_data = chrdev_state;
        debug("character device state initialized successfully\n");

        switch (chrdev_state->type) {
                case BATT:
                        debug("/dev/sensor%d-batt opened\n", sensor);
                        break;
                case TEMP:
                        debug("/dev/sensor%d-temp opened\n", sensor);
                        break;
                case LIGHT:
                        debug("/dev/sensor%d-light opened\n", sensor);
                        break;
                default:
                        /* Only battery, temperature and light measurements are
                         supported */
                        printk(KERN_ERR "This measurement isn't supported\n");
                        goto out;
        }

        ret = 0;
out:
        debug("leaving, with ret = %d\n", ret);
        return ret;
}
```

- release()

Εδώ, το μόνο που έχουμε να κάνουμε είναι να αποδεσμεύσουμε τη μνήμη της private δομής ***chrdev_state*** που δημιουργήσαμε στην `open()`.

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
        WARN_ON(!filp->private_data);
        kfree(filp->private_data);
        debug("character device state memory released\n");
        return 0;
}
```

- read()

Ο σκοπός της read είναι να επιστρέψει με αξιόπιστο τρόπο τα πιο “φρέσκα” δεδομένα από τους σένσορες στο χρήστη, αντιγράφοντας τα από το sensor buffer στο chrdev_state buffer της συσκευής και έπειτα στον usrbuf μέσω της `copy_to_user()`. Αρχικά, λαμβάνουμε το lunix_chrdev_state_struct από το struct file το οποίο έχει αρχικοποιηθεί στην open (όπως αναλύσαμε προηγουμένως) και το struct lunix_sensor_struct που δείχνει τον sensor που ζητάμε να αντιγράψουμε τις κατάλληλες μετρήσεις του. Έπειτα, ξεκινάει το κρίσιμο τμήμα της συνάρτησης δεδομένου πως μπορεί να υπάρξει μόνο ένα thread σε αυτό με το ίδιο open file description. Στο πεδίο του lunix_chrdev_state_struct έχουμε ορίσει έναν semaphore που επιτρέπει μόνο σε ένα thread να εισέλθει στο κρίσιμο τμήμα (race conditions έχουμε μόνο για threads με το ίδιο ανοιχτό αρχείο και άρα το ίδιο private state). Εφόσον δεν είναι διαθέσιμος, η διεργασία που έκανε read θα κοιμηθεί μέχρι ωσότου να είναι. Μετά τα παραπάνω ήρθε η στιγμή να λάβουμε τα δεδομένα από τον sensor. Αρχικά, πρέπει να ελέγξουμε εάν έχουμε νέα δεδομένα και εφόσον υπάρχουν να ανανεώσουμε τον buffer (ελέγχουμε μόνο όταν το position μέσα στο file είναι στην αρχή του, καθώς διαφορετικά έχουμε bytes να διαβάσουμε). Την δουλειά αυτή αναλαμβάνουνε οι συναρτήσεις `lunix_chrdev_state_update()` και `lunix_chrdev_state_needs_refresh()`. Πιο συγκεκριμένα, η διεργασία που έχει κάνει κλήση read βρίσκεται σε ένα loop που ολοκληρώνεται όταν η lunix_chrdev_state_update επιστρέψει επιτυχώς. Στο μεταξύ κοιμάται μπαίνοντας σε ένα queue περιμένοντας για “φρέσκα” δεδομένα από τον sensor (πρέπει να κάνει release το lock και όταν ξυπνήσει το κάνει πάλι acquire). Αν ο χώρος χρήστη έχει ανοίξει την συσκευή με non-blocking mode τότε επιστρέφουμε στο errno -EAGAIN. Εφόσον κάνει update τα δεδομένα, τα αντιγράφει στο userspace με κλήση της `copy_to_user()`. Συγκεκριμένα, αφού υπολογίσει πόσα bytes υπάρχουν για να τα αντιγράψει στο user space σύμφωνα με το position στο file, τα bytes στον chrdev_state buffer και τον αριθμό των bytes που ζήτησε ο χώρος χρήστη, τα αντιγράφει με `copy_to_user()` και αφού κάνει update το position στο file, επιστρέφει στον χώρο χρήστη τον αριθμό των bytes που αντέγραψε. Όταν φτάσει στο τέλος του buffer (διάβασε όλη την μέτρηση) κάνει rewind στην αρχή του αρχείου. Τέλος, κάνουμε release το lock. Οι λεπτομέρειες υλοποίησης αναφέρονται παρακάτω στον πηγαίο κώδικα με επεξηγηματικά σχόλια.

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

        struct lunix_sensor_struct *sensor;
```

```
        struct lunix_chrdev_state_struct *chrdev_state;

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

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

        debug("entering\n");
        /* Acquire the lock */
        if (down_interruptible(&chrdev_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) {
                while (lunix_chrdev_state_update(chrdev_state) == -EAGAIN) {

                        /* The process needs to sleep */
                        up(&chrdev_state->lock); //release the lock

                        if (chrdev_state->nonblock_mode) // check for non-
                                                         blocking i/o
                                return -EAGAIN;

                        /* Sleep */
                        if (wait_event_interruptible(sensor->wq,
                              lunix_chrdev_state_needs_refresh(chrdev_state)))
                                return -ERESTARTSYS;

                        if (down_interruptible(&chrdev_state->lock)) //acquire
                                                                     the lock
                                return -ERESTARTSYS;
                }
        }

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

        cnt = (cnt < remaining_bytes) ? cnt : remaining_bytes;

        if (copy_to_user(usrbuf, chrdev_state->buf_data + *f_pos, cnt)) {
                ret = -EFAULT;
                goto out;
        }

        /* Update the file offset in the open file description */
        *f_pos += cnt;

        /* Return the read number of bytes */
        ret = cnt;

        /* Auto-rewind on EOF mode */
        if (*f_pos == chrdev_state->buf_lim)
                *f_pos = 0;
out:
        /* Unlock */
        up(&chrdev_state->lock);
        debug("leaving\n");
        return ret;
}
```

- lunix_chrdev_state_needs_refresh()

Η συνάρτηση αυτή ελέγχει εάν έχουν έρθει “φρέσκα” δεδομένα στους sensors buffers, συγκρίνοντας το last_update με το buf_timestamp που ενημερώθηκε όταν κάναμε τελευταία φορά update τα δεδομένα στον chrdev_state buffer.

```
/*
 * 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
*chrdev_state)
{
        struct lunix_sensor_struct *sensor;

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

        return (sensor->msr_data[chrdev_state->type]->last_update >
                        chrdev_state->buf_timestamp);
}
```

- lunix_chrdev_state_update()

Αυτή η βοηθητική συνάρτηση είναι αυτή που μεταφέρει τα δεδομένα από τους sensor buffers στο chrdev_state buffer. Χρησιμοποιεί και αυτή την `lunix_chrdev_state_needs_refresh()` για να βεβαιωθεί πως χρειάζεται να ανανεώσει τον chrdev_state buffer (σε άλλη περίπτωση επιστρέφει -EAGAIN και η διεργασία κάνει sleep). Για να κάνουμε access τους sensor buffers χρησιμοποιούμε spinlocks, έτσι ώστε να εξαλείψουμε τα race conditions, αφού η διαδικασία διαβάσματος ανταγωνίζεται κώδικα σε interrupt context. Όσο διαβάζουμε από τους sensor buffers μπορεί να υπάρξει διακοπή για ανανέωση των τιμών του συγκεκριμένου sensor στον ίδιο CPU core που τρέχουμε. Έτσι, χρησιμοποιούμε macros για το κλείσιμο τα οποία πριν γίνει acquire το lock κλείνουνε τα interrupts στο CPU core που τρέχουμε και έτσι διακοπή για την ανανέωση του συγκεκριμένου sensor δεν θα οδηγήσει σε deadlock το σύστημα. Έπειτα, βάση του raw mode που ρυθμίζεται από κλήσεις συστήματος `ioctl()` από το userspace, περνάμε τη φρέσκια μέτρηση από το lookup table για να τη κάνουμε convert και φορμάροντας τη κατάλληλα σε προσημασμένη δεκαδική την γράφουμε στο chrdev_state buffer επιστρέφοντας στην read για να στείλουμε τα δεδομένα στον χώρο χρήστη, όπως αναλύσαμε παραπάνω. Ένα switch statement μας επιτρέπει να βρούμε το κατάλληλο table βάση της μέτρησης που συσχετίζεται με το open file description. Τέλος, κάνουμε update το buf_timestamp με αυτό του sensor.

```
/*
 * 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
*chrdev_state)
{
        int ret;
```

```
struct lunix_sensor_struct *sensor;
uint32_t raw_data;
uint32_t current_timestamp;
long measurement;
unsigned char sign;

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

/*
 * Any new data available?
 */
if (lunix_chrdev_state_needs_refresh(chrdev_state)) {
        /*
        * Grab the raw data quickly, hold the
        * spinlock for as little as possible.
        * We must disable the interrupts in running CPU to prevent
          deadlocks!
        * We use spinlocks because we compete code that runs on
          interrupt context.
        */
        spin_lock_irq(&sensor->lock);
        raw_data = sensor->msr_data[chrdev_state->type]->values[0];
        current_timestamp = sensor→msr_data[chrdev_state→type]
                                                ->last_update;
        spin_unlock_irq(&sensor->lock);

        /*
        * Now we can take our time to format them,
        * holding only the private state semaphore
        */
        if(!atomic_read(&chrdev_state->raw_mode)) {
                /* COOKED MODE */
                switch (chrdev_state->type) {
                        case BATT:
                                measurement = lookup_voltage[raw_data];
                                break;
                        case TEMP:
                                measurement =
                                        lookup_temperature[raw_data];
                                break;
                        case LIGHT:
                                measurement = lookup_light[raw_data];
                                break;
                        default: //we have checked this in
                                 lunix_chrdev_open()
                                goto out;
                }

                sign = (measurement >= 0) ? ' ' : '-';
                measurement = (measurement >= 0) ? measurement : -
                                        measurement;

                chrdev_state->buf_lim = snprintf(chrdev_state->buf_data,
                           LUNIX_CHRDEV_BUFSZ, "  %c%ld.%03ld", sign,
                           measurement / 1000, measurement % 1000);

        } else
                /* RAW MODE */
                chrdev_state->buf_lim = snprintf(chrdev_state->buf_data,
                                 LUNIX_CHRDEV_BUFSZ, "%u\n", raw_data);

        /* Update the timestamp */
        chrdev_state->buf_timestamp = current_timestamp;
```

```
                ret = 0;
        }
        else
                ret = -EAGAIN;
out:
        return ret;
}
```

- ioctl()

Υλοποιώντας την `ioctl()`, μπορούμε να μεταβάλλουμε τον τρόπο λειτουργίας της συσκευής χαρακτήρων (**“raw”/“cooked”**). Ο “cooked” τρόπος θα περνάει στο userspace μορφοποιημένες τις μετρούμενες τιμές σε προσημασμένες δεκαδικές, ενώ ο “raw” τρόπος θα περνάει χωρίς καμία μεταβολή τις 16-bit ποσότητες που επιστρέφονται από τους αισθητήρες (έτσι, δίνεται η δυνατότητα η απεικόνιση σε δεκαδικές ποσότητες να γίνεται από κατάλληλο πρόγραμμα χώρου χρήστη). Παρακάτω φαίνεται ο πηγαίος κώδικας με επεξηγηματικά σχόλια:

```
static long lunix_chrdev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
        long ret;
        struct lunix_chrdev_state_struct *chrdev_state;

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

        debug("entering\n");
        ret = -ENOTTY;

        if (_IOC_TYPE(cmd) != LUNIX_IOC_MAGIC)
                goto out;
        if (_IOC_NR(cmd) >= LUNIX_IOC_MAXNR)
                goto out;

        switch(cmd) {
                /* We use atomic variables instead of semaphores */
                case LUNIX_IOC_RAWDATA:
                        atomic_set(&chrdev_state->raw_mode, 1);
                        break;
                case LUNIX_IOC_COOKEDDATA:
                        atomic_set(&chrdev_state->raw_mode, 0);
                        break;
                default:
                        goto out;
        }

        ret = 0;
out:
        debug("leaving\n");
        return ret;
}
```

- mmap()

Εδώ, υλοποιούμε ***memory-mapped I/O***, απεικονίζοντας την κατάλληλη μέτρηση από τους sensor buffers απευθείας στον χώρο εικονικών διευθύνσεων μιας διεργασίας μόνο για ανάγνωση με χρήση της κλήσης συστήματος `mmap()`. Με αυτόν τον τρόπο, η διεργασία θα μπορεί να διαβάζει απευθείας τις τιμές που χρειάζεται από τους sensor buffers που βρίσκονται μέσα στον πυρήνα και να έχει πρόσβαση στις πιο πρόσφατες μετρήσεις, στην αρχική, μη μορφοποιημένη εκδοχή τους. Παρακάτω φαίνεται ο πηγαίος κώδικας με επεξηγηματικά σχόλια:

```
static int lunix_chrdev_mmap(struct file *filp, struct vm_area_struct *vma)
{
        unsigned long addr;
        struct page *page;

        struct lunix_chrdev_state_struct *chrdev_state;
        struct lunix_sensor_struct *sensor;

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

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

        debug("entering\n");
        /* Associate kernel logical address to struct page */
        /* We map the measurement of the sensor that is associated with this
            open file */
        page = virt_to_page(sensor->msr_data[chrdev_state->type]->values);

        /* Set the PG_reserved flag (it is needed by remap_pfn_range()) */
        page->flags |= PG_reserved;

        /* Virtual address of the page */
        addr = (unsigned long) page_address(page);

        if (vma->vm_end - vma->vm_start < PAGE_SIZE)
                return -EINVAL;

        /* Build the new page tables */
        if (remap_pfn_range(vma, vma->vm_start, __pa(addr) >> PAGE_SHIFT,
                PAGE_SIZE, vma->vm_page_prot))
                return -EAGAIN;

        /* FIXME: try using vm_insert_page() instead of remap_pfn_range() */

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

Αξίζει, επίσης, να συγκρίνουμε τις επιδόσεις των δύο διαφορετικών μεθόδων (**read() vs. mmap()**).

Τέλος, έχουμε δημιουργήσει κάποια userspace προγράμματα που “τεστάρουν” την λειτουργία του driver μας. Ο πηγαίος κώδικας και η λειτουργία τους φαίνεται παρακάτω:

- fork_test

```c
/*
 * fork_test.c
 *
 * A userspace program to test the functionality of the
 * lunix character device driver using multiple forks.
 *
 * Angelos-Nikolaos Kanatas <el19169@mail.ntua.gr>
 * Ioannis Asprogerakas <el18942@mail.ntua.gr>
 *
 */

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

#include "lunix-chrdev.h"

int safe_atoi(char *s, int *val)
{
        long l;
        char *endp;

        l = strtol(s, &endp, 10);
        if (s != endp && *endp == '\0') {
                *val = l;
                return 0;
        } else
                return -1;
}

ssize_t insist_write(int fd, const char *buf, size_t count)
{
        ssize_t ret;
        size_t orig_count = count;

        while (count > 0) {
                ret = write(fd, buf, count);
                if (ret < 0)
                        return ret;
                buf += ret;
                count -= ret;
        }
        return orig_count;
}

int main(int argc, char **argv)
{
        int procnt;

        if ((argc != 3) || (safe_atoi(argv[2], &procnt) < 0) || (procnt <=0)) {
                fprintf(stderr, "Usage: %s <lunix-chrdev> <number of processes>\
                        n\n", argv[0]);
                exit(1);
        }

        int fd;
        fd = open(argv[1], O_RDONLY);
        if (fd == -1) {
```

```
                perror("open");
                exit(1);
        }

        pid_t pid;
        for(int i=0; i<procnt; i++) {
                pid = fork();
                if (pid < 0) {
                        perror("fork");
                        exit(1);
                }
                if (pid == 0) {
                        char buff[LUNIX_CHRDEV_BUFSZ];
                        char output[30];
                        int output_size;
                        for(;;) {
                                if (read(fd, buff, sizeof(buff)) == -1) {
                                        perror("read");
                                        close(fd);
                                        exit(1);
                                }
                                output_size = snprintf(output, 30, "[PID %d]:%s\
                                                n", getpid(), buff);
                                insist_write(1, output, (size_t) output_size);
                        }
                }
        }

        for(int i=0; i<procnt; i++) {
                if ((pid = wait(NULL)) == -1) {
                        perror("wait");
                        exit(1);
                }
        }

        /* Unreachable */
        return 100;
}
```

- ioctl_test

```
/*
 * ioctl_test.c
 *
 * A userspace program to test the use of ioctl
 * syscall and set the mode of data tranferring from
 * the character device.
 *
 * Angelos-Nikolaos Kanatas <el19169@mail.ntua.gr>
 * Ioannis Asprogerakas <el18942@mail.ntua.gr>
 *
 */

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

#include "lunix-chrdev.h"

ssize_t insist_write(int fd, const char *buf, size_t count)
```

```
{
        ssize_t ret;
        size_t orig_count = count;

        while (count > 0) {
                ret = write(fd, buf, count);
                if (ret < 0)
                        return ret;
                buf += ret;
                count -= ret;
        }
        return orig_count;
}

int main(int argc, char **argv)
{
        if ((argc != 3) || (strcmp(argv[2], "raw") && strcmp(argv[2],
                                "cooked"))) {
                fprintf(stderr, "Usage: %s <lunix-chrdev> <raw/cooked>\n\n",
                                argv[0]);
                exit(1);
        }

        int fd;
        fd = open(argv[1], O_RDONLY);
        if (fd == -1) {
                perror("open");
                exit(1);
        }

        if (!strcmp(argv[2], "raw")) {
                if (ioctl(fd, LUNIX_IOC_RAWDATA) == -1) {
                        perror("ioctl");
                        exit(1);
                }
        }

        ssize_t rcnt;
        char buff[LUNIX_CHRDEV_BUFSZ];

        for(;;) {
                rcnt = read(fd, buff, sizeof(buff));
                if (rcnt == -1) {
                        perror("read");
                        close(fd);
                        exit(1);
                }
                insist_write(1, buff, rcnt);
        }

        /* Unreachable */
        return 100;
}
```

- mmap_test

```
/*
 * mmap_test.c
 *
 * A userspace program to test the functionality of the
 * lunix character device driver using mmap() (memory-mapped I/O).
 *
```

```
 * Angelos-Nikolaos Kanatas <el19169@mail.ntua.gr>
 * Ioannis Asprogerakas <el18942@mail.ntua.gr>
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>

#include "lunix.h"

int main(int argc, char **argv)
{
        struct lunix_msr_data_struct *addr;
        uint32_t measurement;
        uint32_t buf_timestamp = 0;
        size_t len;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s <lunix-chrdev>\n\n", argv[0]);
                exit(1);
        }

        if ((len = sysconf(_SC_PAGESIZE)) == -1) {
                perror("sysconf(_SC_PAGESIZE)");
                exit(1);
        }

        int fd;
        fd = open(argv[1], O_RDONLY);
        if (fd == -1) {
                perror("open");
                exit(1);
        }

        addr = mmap(NULL, len, PROT_READ, MAP_PRIVATE, fd, 0);
        if (addr == MAP_FAILED) {
                perror("mmap");
                exit(1);
        }

        while(1) {
                if (buf_timestamp != addr->last_update) {
                        buf_timestamp = addr->last_update;
                        measurement = addr->values[0];
                        printf("%u\n", measurement);
                }
                /* FIXME */
                usleep(200000); //sleep until new data arrives
        }

        /* FIXME: convert raw data to floating point values specific to the
            measurement */

        /* Unreachable */
        return 100;
}
```