Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Λειτουργικών Συστημάτων

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Οδηγός Ασύρματου Δικτύου Αισθητήρων στο λειτουργικό σύστημα Linux

Ημερομηνία Παράδοσης : 10/05/2018

Ομάδα : oslab38

Ονοματεπώνυμο	**Αριθμός Μητρώου**
Βασιλάκης Εμμανουήλ	03114167
Γιάννου Αγγελική	03114021

Εξάμηνο	**Ακαδημαϊκό Έτος**
8ο	2017-2018

1. Κώδικας

```
/*
 * lunix-chrdev.c
 *
 * Implementation of character devices
 * for Lunix:TNG
 *
 * < Angeliki Giannou, Emmanouil Vasilakis >
 *
 */

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

        if (state->buf_timestamp < sensor->msr_data[state->type]->last_update) return 1;
        else return 0;
}

/*
 * 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;
        uint16_t temp;                          //grab measurement without formatting in spinlock
        long num, akeraio, dekadiko;
        char sign;

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

        debug("updating\n");

        if (lunix_chrdev_state_needs_refresh(state) == 0) {
                debug("leaving without updating\n");
                return -EAGAIN;
        }

        spin_lock(&sensor->lock);               /* Why use spinlocks? See LDD3, p. 119 */
        state->buf_timestamp = sensor->msr_data[state->type]->last_update;
        temp = sensor->msr_data[state->type]->values[0];
        spin_unlock(&sensor->lock);

        if (state->type == BATT) num = lookup_voltage[temp];
        else if (state->type == TEMP) num = lookup_temperature[temp];
        else num = lookup_light[temp];

        if (num == 0) {
                state->buf_lim = sprintf(state->buf_data, "0\n");
                debug("leaving\n");
                return 0;
        }
        else if (num > 0) sign = '+';
        else {
                sign = '-';
                num *= -1;
        }

        dekadiko = num % 1000;                  // edo ta 3 psifia poy deixnoyn to float kommati
        akeraio = num / 1000;                   // edo ta 2 psifia poy deixnoyn to int kommati

        state->buf_lim = sprintf(state->buf_data, "%c%ld.%ld\n", sign, akeraio, dekadiko);

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

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

static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
        /* Declarations */
        unsigned int minor, type_no, sensor_no;
        struct lunix_chrdev_state_struct *state;
        int ret;

        debug("entering\n");
```

```
	ret = -ENODEV;
	if ((ret = nonseekable_open(inode, filp)) < 0)
		goto out;

	/*
	 * Associate this open file with the relevant sensor based on
	 * the minor number of the device node [/dev/sensor<NO>-<TYPE>]
	 */
	minor = iminor(inode);
	type_no = minor % 8;
	sensor_no = minor / 8;

	state = (struct lunix_chrdev_state_struct *) kmalloc(sizeof(struct lunix_chrdev_state_struct),
	                                                     GFP_KERNEL);
	if (state == NULL) goto out;

	state->buf_timestamp = 0;
	if (type_no == 0) state->type = BATT;
	else if (type_no == 1) state->type = TEMP;
	else if (type_no == 2) state->type = LIGHT;
	else goto out;
	sema_init(&state->lock, 1);
	state->sensor = &lunix_sensors[sensor_no];

	filp->private_data = state;

	/* Allocate a new Lunix character device private state structure */
	/* ? */
out:
	debug("leaving, with ret = %d\n", ret);
	return ret;
}

static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
	struct lunix_chrdev_state_struct *state;
	WARN_ON ( !(state = filp->private_data));

	state->sensor = NULL;
	kfree(state);
	return 0;
	/*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;

	struct lunix_sensor_struct *sensor;
	struct lunix_chrdev_state_struct *state;
```

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

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

	/* Corner Case */
	if (cnt == 0)
		return 0;

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

	/*
	 * If the cached character device state needs to be
	 * updated by actual sensor data (i.e. we need to report
	 * on a "fresh" measurement, do so
	 */
	if (*f_pos == 0) {
		while (lunix_chrdev_state_update(state) == -EAGAIN) {
			/* The process needs to sleep */
			/* See LDD3, page 153 for a hint */
			up(&state->lock); /* release the lock */
			if (wait_event_interruptible(sensor->wq, lunix_chrdev_state_needs_refresh(state)
				== 1))
				return -ERESTARTSYS; /* signal: tell the fs layer to handle it */
			/* otherwise loop, but first reacquire the lock */
			if (down_interruptible(&state->lock))
				return -ERESTARTSYS;
		}
	}

	/* Determine the number of cached bytes to copy to userspace */
	if (cnt < ((sizeof(unsigned char) * state->buf_lim) - *f_pos)) ret = cnt;
	else ret = (sizeof(unsigned char) * state->buf_lim) - *f_pos;

	if (copy_to_user(usrbuf, state->buf_data + *f_pos, (unsigned long) ret)) {
		up (&state->lock);
		return -EFAULT;
	}

	/* Auto-rewind on EOF mode? */
	/* ? */
	*f_pos += ret;
	if (*f_pos == sizeof(unsigned char) * state->buf_lim) *f_pos = 0; /* wrapped */

//out:
	/* Unlock? */
	up (&state->lock);
	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");
        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:TNG");
        /* register_chrdev_region? */
        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);
        /* cdev_add? */
        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");
}
```

2. Συνοπτική Περιγραφή

Σε αυτή την άσκηση, κληθήκαμε να υλοποιήσουμε κομμάτι κώδικα για τον driver κάποιων character devices. Οι character devices που χρησιμοποιούν αυτόν τον driver είναι κάποιοι αισθητήρες που αποστέλλουν πληροφορίες σχετικά με το επίπεδο φωτεινότητας, την θερμοκρασία και τέλος την τάση της μπαταρίας τους. Κάθε αισθητήρας αναπαρίσταται με μία δομή sensor που περιέχει τις μετρήσεις του αλλά και το πότε έγινε η τελευταία ενημέρωση. Κάθε αισθητήρας έχει τρία διαφορετικά αρχεία στο /dev/, κάθε ένα από τα οποία μας επιτρέπει την πρόσβαση σε μία από τις μετρήσεις του. Σκοπός της άσκησης ήταν να υλοποιήσουμε κάποιες από τις κλήσεις συστήματος που χρησιμοποιεί ο driver.

- ***init()***

  Η κλήση αυτής της συνάρτησης, γίνεται κατά την προσάρτηση του driver στον πυρήνα του συστήματος. Αυτή αναλαμβάνει την δέσμευση και την καταχώρηση device_number στις διάφορες συσκευές που δημιουργούμε για να πάρουμε τις μετρήσεις μας.

- ***open()***

  Η συνάρτηση αυτή καλείται κάθε φορά που κάποια διεργασία κάνει open ένα device που ελέγχεται με τον driver μας. Εδώ αναλαμβάνουμε την δημιουργία και αρχικοποίηση μιας δομής state που συνδέεται πλέον με το συγκεκριμένο ανοιχτό αρχείο (μέσω του δείκτη filp->private_data), και από την οποία δομή θα παίρνουμε τις πληροφορίες που χρειαζόμαστε. Από το minor number της συσκευής που ανοίγουμε, μπορούμε να καταλάβουμε σε ποιο sensor αναφερόμαστε, αλλά και τι τύπου μέτρηση θέλουμε, πληροφορίες που εντάσσουμε στην δομή state.

- ***release()***

  Κάθε ανοιχτό αρχείο δεσμεύει μνήμη για το state του. Επομένως, κάθε φορά που κλείνουμε ένα ανοιχτό αρχείο, πρέπει να ελευθερώνουμε την μνήμη που αυτό δέσμευε.

- ***read()***

  Σκοπός της read() είναι η επιστροφή στην διεργασία που την κάλεσε των δεδομένων που ζήτησε. Μέσω του πεδίου private_data του ανοιχτού αρχείου filp, έχουμε πρόσβαση στο struct state που αντιστοιχίζεται στο ανοιχτό αρχείο και το οποίο μας παρέχει τις ζητούμενες πληροφορίες. Σημειώνεται ότι στην read() γίνεται χρήση σημαφόρων ώστε να κλειδώσει το state, που αντιστοιχεί στο ανοιχτό αρχείο από το οποίο η διεργασία θέλει να διαβάσει. Η χρήση κλειδώματος για το state είναι αναγκαία, καθώς διεργασίες που έχουν προκύψει μέσω threads ή fork, μοιράζονται το ίδιο state. Αρχικά, διακρίνουμε δύο βήματα :

  1. Αν πρόκειται για νέα μέτρηση (*f_pos = 0), περιμένουμε μέχρι να έρθουν νέα δεδομένα στον αισθητήρα μέσω της needs_refresh. Μέχρι να έρθουν αυτά τα δεδομένα η διεργασία "κοιμάται" στην ουρά διεργασιών του αντίστοιχου sensor (κλήση της wait_event_interruptible). Μόλις γίνει ανανέωση του sensor, η sensor_update (αρχείο sensors.c), ξυπνάει όλες τις διεργασίες που περιμένουν νέα δεδομένα.

  2. Σε περίπτωση που δεν πρόκειται για νέα μέτρηση πάμε απευθείας στο βήμα 2. Τα δεδομένα, τα οποία βρίσκονται στο πεδίο buf_data του state, αντιγράφονται στους απομωνοτές του χώρου χρήστη (κλήση της copy_to_user). Πριν την μεταφορά των δεδομένων βρίσκουμε το πλήθος των δεδομένων που πρέπει να μεταφέρουμε (μεταβλητή ret).

- ***update()***

  Η συνάρτηση αυτή σχετίζεται με την ενημέρωση των δεδομένων στο state, το οποίο παίρνει σαν όρισμα. Πιο συγκεκριμένα, αρχικά καλείται η needs_refresh, και αν η επιστρεφόμενη τιμή είναι 0 τότε η update απλώς επιστρέφει, ενώ σε αντίθετη περίπτωση, κλειδώνεται ο sensor από τον οποίο θέλουμε να πάρουμε τα δεδομένα. Αφού τα δεδομένα αυτά αποθηκευτούν σε προσωρινές μεταβλητές, ξεκλειδώνεται ο sensor, ενώ στη συνέχεια μορφοποιούμε (χρήση της sprintf) και αποθηκεύουμε τα δεδομένα στο πεδίο buf_data του state. Σημειώνεται ότι, για το κλείδωμα του sensor χρησιμοποιούνται spinlocks, καθώς οι sensor χρησιμοποιούνται και σε interrupt context, όπου και γίνεται χρήση spinlocks.

- ***needs_refresh()***

  Αυτή η συνάρτηση καλείται μέσα από την update, και κάνει έναν γρήγορο έλεγχο για το κατά πόσο υπάρχει νέα τιμή στον αισθητήρα που δεν έχει περαστεί στο state. Ο έλεγχος επιτυγχάνεται με την σύγκριση των χρόνων τελευταίας ενημέρωσης του state και του sensor αντίστοιχα.

3. Παράδειγμα Εκτέλεσης

Παρακάτω παραθέτουμε ενδεικτικά αποτελέσματα που επιβεβαιώνουν την λειτουργία του driver.

```
user@utopia:~$ cat /dev/lunix0-batt
+3.354
+3.354
+3.354
+3.354
^C
user@utopia:~$ cat /dev/lunix0-temp
+24.295
+24.295
+24.295
+24.295
+24.295
+24.295
+24.295
^C
user@utopia:~$ cat /dev/lunix0-light
+0.457
+0.457
+0.457
+0.457
+0.457
+0.457
+0.457
^C
```