Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλ. Μηχανικών & Μηχανικών Υπολογιστών

Εργαστήριο Λειτουργικών Συστημάτων

Ακαδημαϊκό έτος 2023-2024

Ομάδα oslab4

Κοκολάκης Γεώργιος 03114745

Πάντος Πάντος 03120408

Αναφορά 2ης Εργαστηριακής Άσκησης (lunix:TNG)

Οι προσθήκες που χρειάστηκαν στο αρχείο "lunix-chrdev.c", ώστε να λειτουργήσει σωστά η συσκευή φαίνονται στις παρακάτω εικόνες, αλλά τις αναφέρουμε επίσης στο κείμενο στη συνέχεια.

lunix_chrdev_state_needs_refresh():
Προστέθηκε κώδικας που συγκρίνει τη χρονοσφραγίδα της δομής κατάστασης της συσκευής χαρακτήρων με την τελευταία ενημέρωση των μετρήσεων του αισθητήρα, ώστε να μάθουμε αν χρειάζεται ενημέρωση η δομή κατάστασης

lunix_chrdev_state_update():
Γίνονται οι δηλώσεις των μεταβλητών που θα χρησιμοποιηθούν
struct lunix_sensor_struct __attribute__((unused)) * sensor μεταβλητή δομής αισθητήρα
int new_data μεταβλητή για την ύπαρξη νέων δεδομένων
data μεταβλητή για τα ωμά δεδομένα
long looked_up_data μεταβλητή για τη μετατροπή των δεδομένων σύμφωνα με τα lookup tables
int ret μεταβλητή επιστροφής συνάρτησης
int digit, int i μεταβλητές που χρησιμοποιούνται για την μετατροπή των lookedup δεδομένων στην τελική τους μορφή,
στη συνέχεια εντός περιστρεφόμενου κλειδώματος, ελέγχουμε αν η δομή κατάστασης της συσκευής χρειάζεται ανανέωση και εφόσον χρειάζεται, παίρνουμε τα νέα δεδομένα μέσα από τη δομή αισθητήρα , τα τοποθετούμε στη μεταβλητή data και ενημερώνουμε τη χρονοσφραγίδα της δομής κατάστασης και τη μεταβλητή new_data. Τέλος εκτός του κλειδώματος πλέον, μετατρέπουμε τα δεδομένα σε αναγνώσιμη μορφή μέσω των lookup tables και τέλος τους δίνουμε το απαιτούμενο format και τα τοποθετούμε στον buffer δεδομένων της δομής κατάστασης.

lunix_chrdev_open():
Γίνονται και πάλι οι δηλώσεις των μεταβλητών που θα χρησιμοποιηθούν
struct lunix_chrdev_state_struct *state μεταβλητή δομής κατάστασης
unsigned int minor_number μεταβλητή αποθήκευσης του minor number
unsigned int sensor_number μεταβλητή αποθήκευσης του sensor number
Προσδιορίζουμε τον minor αριθμό του ανοιχτού αρχείου μέσω της iminor συνάρτησης και μέσω του minor (/8) προσδιορίζουμε και τον τύπο του αισθητήρα. Κάνουμε allocate χώρο για τη δομή κατάστασης σε χώρο πυρήνα μέσω της kzalloc συνάρτησης και την κάνουμε προσβάσιμη από το πεδίο private data του file pointer. Τέλος ορίζουμε σωστό τύπο & αριθμό αισθητήρα στη δομή κατάστασης και αρχικοποιούμε το σημαφόρο.

lunix_chrdev_release():
Ελευθερώνουμε τη μνήμη που είναι δεσμευμένη για τη δομή κατάστασης, η οποία αντιστοιχεί στο αρχείο-όρισμα της lunix_chrdev_release(), με χρήση της kfree.

lunix_chrdev_read():
Ορίζουμε τη μεταβλητή **unsigned long bytes** στην οποία αποθηκεύουμε τον αριθμό των bytes που επιστρέφουμε στον χρήστη. Κλειδώνουμε το σημαφόρο της δομής κατάστασης με down_interruptible. Εάν δεν έχουμε ήδη δεδομένα στον buffer, δηλαδή *f_pos==0, τότε περιμένουμε νέα δεδομένα επιτρέποντας διακοπές από σήματα. Προσδιορίζουμε τον αριθμό των bytes που θα αντιγραφούν σε χώρο χρήστη, αντιγράφουμε με την copy_to_user, ξεκλειδώνουμε το σημαφόρο και επιστρέφουμε.

lunix_chrdev_init():
Κάνουμε εγγραφή περιοχής τιμών της συσκευής μας με τη register_chrdev_region κι έπειτα προσθέτουμε τη συσκευή στο σύστημα με τη cdev_add

```c
/*
 * lunix-chrdev.c
 *
 * Implementation of character devices
 * for Lunix:TNG
 *
 * < Pantos Pantos - Kokolakis Georgios >
 *
 */

#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.
 */
/*
 * Declare a prototype so we can define the "unused" attribute and keep
 * the compiler happy. This function is not yet used, because this helpcode
 * is a stub.
 */
```

```c
static int __attribute__((unused)) lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *);
static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *state)
{
    struct lunix_sensor_struct *sensor;

    // Assign the sensor of the state that is passed as an argument to the sensored declared above
    // Call WARN if unsuccesfull;
    WARN_ON(!(sensor = state->sensor));

    /* ? */ // TODO
    return state->buf_timestamp != sensor->msr_data[state->type]->last_update;

    /* The following return is bogus, just for the stub to compile */
    return 0; /* ? */ // TODO
}

/*
 * 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 __attribute__((unused)) * sensor;
    int new_data;
    uint32_t data;
    long looked_up_data;
    int ret = 0;
    int digit;
    int i;

    // debug("leaving\n");
    debug("pantos: Entering the update state method\n");
    sensor = state->sensor;
    new_data = 0;
    /*
     * Grab the raw data quickly, hold the
     * spinlock for as little as possible.
     */
    /* ? */
    spin_lock(&sensor->lock);
    /* Why use spinlocks? See LDD3, p. 119 */
```

```c
/*
 * Any new data available?
 */
/* ? */ // TODO
if (state->buf_timestamp != sensor->msr_data[state->type]->last_update)
{
    debug("pantos: bufer_timestamp: %d\n", state->buf_timestamp);
    new_data = 1;
    state->buf_timestamp = sensor->msr_data[state->type]->last_update;
    data = sensor->msr_data[state->type]->values[0];
    debug("pantos: Acquired (%d) type data: %d\n", state->type, data);
    debug("pantos: timestamp: %d\n", sensor->msr_data[state->type]->last_update);
}
else
{
    debug("pantos: no data to acquire \n");
    ret = -EAGAIN;
}

spin_unlock(&sensor->lock);

/*
 * Now we can take our time to format them,
 * holding only the private state semaphore
 */

/* ? */ // TODO
if (new_data)
{
    switch (state->type)
    {
    case BATT:
        looked_up_data = lookup_voltage[data];
        break;
    case TEMP:
        looked_up_data = lookup_temperature[data];
        break;
    case LIGHT:
        looked_up_data = lookup_light[data];
        break;
    default:
        // code should never reach here but it is needed to keep compiler happy
        return -1;
        break;
    }

    debug("pantos: Looked up data is: %ld\n", looked_up_data);
```

```
        // This loop just prints the looked up data in reverse order and ignoring sign
        i = 0;
        while (looked_up_data != 0 && i < LUNIX_CHRDEV_BUFSZ)
        {
            digit = looked_up_data % 10;
            state->buf_data[i++] = '0' + digit;
            looked_up_data /= 10;
        }
        state->buf_data[i++] = '\0';
        state->buf_lim = i;
    }

    debug("pantos: leaving update state\n");
    debug("pantos: created buffer: %s\n", state->buf_data);

    return ret;
}

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

static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
    /* Declarations */
    struct lunix_chrdev_state_struct *state;
    unsigned int minor_number;
    unsigned int sensor_number;
    /* ? */

    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>]
     */

    /* Allocate a new Lunix character device private state structure */

    // Get the minor number of the file openned
    minor_number = iminor(inode);
    sensor_number = minor_number >> 3;
```

```
    // Allocate a new state struct initialized to 0
    state = kzalloc(sizeof(*state), GFP_KERNEL);

    filp->private_data = state;
    // Set the fields of the struct to point at the corect sensor struct and correct type
    // Based on the minor number
    state->sensor = &lunix_sensors[sensor_number];
    state->type = minor_number % 8;
    debug("pantos: Opening sensor: %d-%d\n", sensor_number, state->type);

    // initialize semaphore
    sema_init(&state->lock, 1);

    /* ? */

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

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

static long lunix_chrdev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    /* Why? */
    // There is no reason to send any "commands" to the device (?)
    return -EINVAL;
}
```

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

    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? */
    // LDD3, page 112
    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 */
            /* ? */ // TODO
            // debug("pantos: Entering sleeping\n");

            up(&state->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 */
            // debug("pantos: Exiting sleeping\n");

            if (down_interruptible(&state->lock))
                return -ERESTARTSYS;
        }
    }
```

```
    /* Determine the number of cached bytes to copy to userspace */
    /* ? */
    if (cnt > state->buf_lim)
        bytes = state->buf_lim;
    else
        bytes = cnt;

    ret = copy_to_user(usrbuf, state->buf_data, bytes);
    if (ret)
    {
        debug("pantos: problem in copy to user\n");
        goto out;
    }
    ret = bytes;

    /* Auto-rewind on EOF mode? */
    /* ? */ // TODO

    /*
     * The next two lines  are just meant to suppress a compiler warning
     * for the "unused" out: label, and for the uninitialized "ret" value.
     * It's true, this helpcode is a stub, and doesn't use them properly.
     * Remove them when you've started working on this code.
     */
    // ret = -ENODEV;
    // goto out;
    ;
out:
    /* Unlock? */
    // LDD3, page 113
    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
```

```c
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);

    /* ? */
    // Done: Same as LDD3 page 48
    ret = register_chrdev_region(dev_no, lunix_minor_cnt, "lunix-tng");
    /* register_chrdev_region? */

    // /* Since this code is a stub, exit early */
    // return 0;

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

    /* ? */
    // LDD3 page 56
    // Not completely sure about 3rd parameter
    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");
}
```