ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

OS Lab

Εαρινό Εξάμηνο 2015-2016

Αναφορά 2ης Εργαστηριακής Άσκησης Cgroups

Απρίλιος 2016

Καλλάς Κωσταντίνος 03112057 – Τζίνης Ευθύμιος 03112007

8ο Εξάμηνο, Σ.Η.Μ.Μ.Υ., Ε.Μ.Π.

Γενική Περιγραφή:

Σε αυτή την άσκηση ολοκληρώσαμε έναν driver για μια συσκευή χαρακτήρων. Ουσιαστικά υλοποιήσαμε το τμήμα του driver που είναι πιο κοντά στον χρήστη. Το τμήμα αυτό έχει την αρμοδιότητα για όλες τις κλήσεις συστήματος από τον χρήστη προς τη συσκευή.

Η συσκευή στην οποία αναφερόμαστε είναι ουσιαστικά μια συστοιχία από αισθητήρες που μετρούν θερμοκρασία και φωτεινότητα του χώρου που βρίσκονται καθώς και την τάση της μπαταρίας τους. Οι αισθητήρες δημιουργούν αυτόματα ένα δίκτυο μεταξύ τους και ένας κεντρικός κόμβος αναλαμβάνει να στέλνει όλα τα δεδομένα των αισθητήρων στο σύστημα του χρήστη. Ο κεντρικός κόμβος κανονικά συνδέεται με κάποιο σύστημα με την χρήση καλωδίου usb αλλά στην περίπτωση μας χρησιμοποιήθηκε ένας server που εξέπεμπε δεδομένα. Η σύνδεση με τον server έγινε με ένα script μέσω μιας σειριακής θύρας.

Οι κλήσεις συστήματος που υλοποιήσαμε είναι οι εξής:

- open
- release
- read
- ioctl
- mmap

Στον παρών κεφάλαιο θα εξηγηθούν σε γενικές γραμμές οι παραπάνω κλήσεις εκτός από την mmap() η οποία περιγράφεται σε ξεχωριστό κεφάλαιο παρακάτω. Εκτός από τις κλήσεις συστήματος υλοποιήθηκαν και βοηθητικές συναρτήσεις όπως (initialize(), update(), needs_refresh()).

open():

Αυτή η κλήση συστήματος καλείται από τον χρήστη για την έναρξη της “επικοινωνίας” του με τη συσκευή. Σε αυτή τη συνάρτηση βρίσκουμε σε ποιά μέτρηση ποιανού αισθητήρα θέλει να αποκτήσει πρόσβαση ο χρήστης, μέσω του minor number της συσκευής. Επίσης κάνουμε allocate όσο χώρο χρειαζόμαστε κατά τη διάρκεια της πρόσβασης του χρήστη στη συσκευή.

Ο κώδικας της open() βρίσκεται παρακάτω(1).

release():

Αυτή η κλήση συστήματος καλείται όταν τελειώνει η πρόσβαση του χρήστη στη συσκευή και σκοπός της είναι απλά να ελευθερώσει όσο χώρο έχει δεσμέυσει ο driver.

Ο κώδικας της release() βρίσκεται παρακάτω(2).

read():

Η read() είναι η βασική κλήση συστήματος με την οποία ο χρήστης αποκτάει πρόσβαση στην συσκευή χαρακτήρων μας. Όταν καλείται η read() ο driver ελέγχει αν υπάρχουν νέα δεδομένα για τον χρήστη (αν έχει τελειώσει με το διάβασμα των παλιών) και περιμένει να

τα λάβει περιμένοντας σε μια ουρά (εκτός από την περίπτωση που η συσκευή έχει ανοιχτή με O_NONBLOCK flag, κατά την οποία δεν περιμένει για νέα δεδομένα παρά επιστρέφει κατευθείαν στον χρήστη). Για την ανανέωση των δεδομένων καλείται η βοηθητική συνάρτηση update() η οποία εξηγείται αργότερα. Ύστερα η read() υπολογίζει πόσα bytes να επιστρέψει στον χρήστη και τα αντιγράφει σε ένα καθορισμένο buffer.

Ο κώδικας της read() βρίσκεται παρακάτω(3) και κάποια συγκεκριμένα θέματα που αντιμετωπίστηκαν κατά την υλοποίηση της αναλύονται στο κεφάλαιο λεπτομέρειες υλοποίησης.

Ioctl():

TODO : Thymios

Ο κώδικας της ioctl() βρίσκεται παρακάτω(4).

Initialize():

TODO : Thymios

Ο κώδικας της initialize() βρίσκεται παρακάτω(5).

Update():

TODO: Thymios

Ο κώδικας της update() βρίσκεται παρακάτω(6).

Needs_refresh():

TODO : Thymios

Ο κώδικας της needs_refresh() βρίσκεται παρακάτω(7).

Λεπτομέρειες Υλοποίησης**:**

TODO : Templeta gia periexomena? ? Me links isws

TODO : Thymios

TODO : Grapse gia

- semaphores
- spinlocks

Σχεδιασμός και υλοποίηση της mmap:

Εκτός από την υλοποίηση της βασικής εντολής πρόσβασης σε μια συσκευή (read), αποφασίσαμε να υλοποιήσουμε και την κλήση συστήματος mmap() για τη συσκευή μας.

Η mmap() είναι χρήσιμη, αφού επιταχύνει την πρόσβαση σε δεδομένα της συσκευής από τον χρήστη (Αυτό συμβαίνει επειδή εξαλείφεται το overhead των συνεχόμενων κλήσεων συστήματος και της αντιγραφής δεδομένων από τον χώρο πυρήνα στον χώρο χρήστη. Επείτα από επιτυχή χρήση της mmap() ο χρήστης έχει πρόσβαση σε κάποιες σελίδες

μνήμης που έχει πρόσβαση και ο driver της συσκευής, οπότε η διαδικασία απόκτησης δεδομένων απλοποιείται σε ένα dereference.

Αποφασίσαμε να ακολουθήσουμε μια απλή υλοποίηση της mmap() η οποία θα δίνει στον χρήστη πρόσβαση σε μια μόνο σελίδα (αφού κάθε device χειρίζεται μνήμη μιας ακριβώς σελίδας). Ο κώδικας της mmap() παρατίθεται παρακάτω(8).

Παράλληλα υλοποιήσαμε και ένα πρόγραμμα σε user space με το οποίο ελέγξαμε την ορθή χρήση της κλήσης συστήματος. Ο κώδικας του δοκιμαστικού προγράμματος παρατίθεται παρακάτω(9) για λόγους πληρότητας.

Testing

TODO : Thymios

open()

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
        int ret;
        unsigned int dev_minor_no, sensor_no, sensor_data_type;
        struct lunix_sensor_struct *sensor;
        struct lunix_chrdev_state_struct *state;

        debug("entering\n");

        ret = -EACCES;
        if(filp->f_mode & FMODE_WRITE)
                goto out;

        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>]
         */
        dev_minor_no = iminor(inode);
        sensor_no = dev_minor_no / 8;
        sensor_data_type = dev_minor_no % 8;
        sensor = &lunix_sensors[sensor_no];

        /*
         * Allocate a new Lunix character device state structure
         * and initialize its parts
         */
        state = kzalloc(sizeof(*state), GFP_KERNEL);
```

```
        state->type = sensor_data_type;
        state->sensor = sensor;
        /* Buffer initialize */
        state->buf_lim = 0;
        state->raw_or_cooked = 1;
        sema_init(&state->lock, 1);

        filp->private_data = state;
        debug("Data for state allocated succesfully :)\n");

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

ioctl():

```
static long lunix_chrdev_ioctl(struct file *filp, unsigned int cmd, unsigned long
arg)
{
        int ret = -EINVAL;
        struct lunix_chrdev_state_struct *state;

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

        switch(cmd) {

                /* Raw Data */
                case 0:
                        state->raw_or_cooked = 0;
                        ret = 0;
                        break;

                /* Cooked Data */
                case 1:
                        state->raw_or_cooked = 1;
                        ret = 0;
                        break;
        }
        return ret;
}
```

Initialize():

```
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);

        /*
         * Trying to register the character device sto that we get
         * minor and major numbers
         */
        ret = register_chrdev_region(dev_no, lunix_minor_cnt, "lunix_chrdev");
        if (ret < 0) {
                debug("failed to register region, ret = %d\n", ret);
                goto out;
        }

        /*
         * Trying to add the character device in 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;
}
```

update():

```
/*
 * 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)
{
        int ret = -EAGAIN;
        int value, timestamp;
        int t_light, t_temp, t_volt;
        int raw_or_cooked = state->raw_or_cooked;
        int t_ret;
        char *temp;
        struct lunix_sensor_struct *sensor = state->sensor;

        debug("Entering...\n");


        if(lunix_chrdev_state_needs_refresh(state)) {

                /*
                 * Try locking grab the data and unlock
                 * as fast as possible
                 */
                ret = spin_trylock(&sensor->lock);
                if(!ret) {
                        debug("Gamithike to lock!!\n");
                        goto out;
                }

                value = sensor->msr_data[state->type]->values[0];
                timestamp =  sensor->msr_data[state->type]->last_update;

                spin_unlock(&sensor->lock);

                debug("Value and timestamp: ( %d, %d )\n", value, timestamp);


                t_light = value;
                t_temp = value;
                t_volt = value;
                if(raw_or_cooked){
                        t_light = lookup_light[value];
                        t_temp = lookup_temperature[value];
                        t_volt = lookup_voltage[value];
                }
```

```
                switch(state->type) {

                   case LIGHT  :
                        temp = float2str(t_light, raw_or_cooked);
                        break;

                   case TEMP  :
                        temp = float2str(t_temp, raw_or_cooked);
                        break;

                   case BATT  :
                        temp = float2str(t_volt, raw_or_cooked);
                        break;

                   default :
                        ret = -1;
                        goto out;
                }
                ret = 0;
                state->buf_timestamp = timestamp;
                if( (t_ret = buf_add(state, temp)) < 0){
                        ret = t_ret;
                        goto out;
                }
        }


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

needs_refresh():

```
/*
 * 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)
{
        int ret;
        struct lunix_sensor_struct *sensor;

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

        debug("Buffer timestamp: %d and sensor_timestamp: %d\n",  state-
>buf_timestamp , sensor->msr_data[state->type]->last_update);
        ret = 0;
```

```
        if(state->buf_timestamp != sensor->msr_data[state->type]->last_update)
                ret = 1;

        return ret;
}
```

release():

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
        kfree(filp->private_data);
        debug("exiting release\n");
        return 0;
}
```

read()

```
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;

        size_t buf_length;
        char requested_str[20];

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

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

        debug("Entering...\n");

        /* Lock the state so that only one read at a time */
        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) {

                        /*
```

```
                         * 1) Release the semaphore so that process doesn't sleep
                         *    with the semaphore lock held
                         * 2) If the file is opened with nonblocking operation
return
                         * 3) Otherwise the process sleeps on the waiting queue
                         * 4) Re-grab the semaphore
                         */
                        up(&state->lock);
                        if (filp->f_flags & O_NONBLOCK)
                                return -EAGAIN;
                        debug("In loop\n");

                        if(wait_event_interruptible(sensor
>wq,lunix_chrdev_state_needs_refresh(state) )) {
                                return -ERESTARTSYS;
                        }

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

        }

        debug("Current string is: %s\n", state->buf_data);
        debug("Initial count is: %u\n",cnt);
        debug("F_pos was: %lld\n", *f_pos);
        debug("buf_length is: %u\n", buf_length);

        /*
         * Find out how many bytes will be copied
         */
        buf_length = strlen(state->buf_data);
        if(cnt + *f_pos < buf_length){
                strncpy(requested_str,(char *) ((state->buf_data) + *f_pos), cnt);
                *f_pos += cnt;
        }
        else{
                cnt = buf_length - *f_pos;
                strncpy(requested_str,(char *) ((state->buf_data) + *f_pos), cnt);
                *f_pos = 0;
        }
        debug("Later count is: %u\n", cnt);
        if (copy_to_user(usrbuf, &requested_str[0] , cnt)) {
                ret = -EFAULT;
                goto out;
        }
```

```
        debug("F_pos is: %lld\n", *f_pos);
        ret = cnt;
        debug("Ret = %d\n", ret);

out:
        up (&state->lock);
        debug("Exiting...\n");
        return ret;
}
```

mmap()

```
static int lunix_chrdev_mmap(struct file *filp, struct vm_area_struct *vma)
{
        int i;
        struct lunix_sensor_struct *sensor;
        struct lunix_chrdev_state_struct *state;
        struct page * temp_page;
        unsigned long long addr;

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

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

        debug("Entering...\n");

#if LUNIX_DEBUG
        for(i=0;i<3;i++) {
                temp_page = virt_to_page(sensor->msr_data[i]->values);
                addr = page_address(temp_page);
                debug("Buffer address: %llu, and buffer page address%llu\n", sensor-
>msr_data[i]->values ,addr );
        }
#endif

        /*
         * Gives back the page address of the struct
         */
        temp_page = virt_to_page(sensor->msr_data[state->type]->values);

        addr = page_address(temp_page);

        debug("Buffer address: %llu, and buffer page address%llu\n",sensor-
>msr_data[i]->values,addr);
        debug("Start: %llu, end: %llu\n", vma->vm_start ,vma->vm_end);
        debug("Logical add: %llu, Physical address: %llu\n", addr, __pa(addr));
        debug("Page number: %llu, PAGE_SHIFT: %llu\n", __pa(addr) >> PAGE_SHIFT,1 <<
```

```
PAGE_SHIFT);

        /*
         * Map the page only if he asks for one page or more
         */
        if(vma->vm_end - vma->vm_start < 1<<PAGE_SHIFT) {
                return -EAGAIN;
        }
        if (remap_pfn_range(vma, vma->vm_start, __pa(addr) >> PAGE_SHIFT,
            1 << PAGE_SHIFT,  vma->vm_page_prot))
                return -EAGAIN;

        vma->vm_ops = &simple_remap_vm_ops;
        simple_vma_open(vma);

        debug("Exiting...\n");
        return 0;
}
```

try_mmap.c

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

struct lunix_msr_data_struct {
        uint32_t magic;
        uint32_t last_update;
        uint32_t values[];
};



int main(void){

        struct lunix_msr_data_struct * addr;
        int fd = open("/dev/lunix1-temp", O_RDONLY);
        off_t offset, pa_offset;
        int length;
        ssize_t s;
        unsigned int buf_timestamp, our_timestamp = 0;
        unsigned int value;

        if(fd <= 0)
                return -1;
```

```
        printf("File descriptor: %d\n", fd);

        offset = 0;
        length = 100;
        pa_offset = offset & ~(sysconf(_SC_PAGE_SIZE) - 1);

        addr = mmap(NULL, length - pa_offset, PROT_READ,
                MAP_PRIVATE, fd, pa_offset);

        if (addr == MAP_FAILED)
                return -1;

        printf("I LIKE %x\n",addr->magic);
        usleep(700000);
        while(1){
                buf_timestamp = addr->last_update;
                printf("Our: %u, their: %u\n", our_timestamp, buf_timestamp);
                if(buf_timestamp != our_timestamp) {
                        our_timestamp = buf_timestamp;
                        value = addr->values[0];
                        printf("Number read is: %u\n", value);
                }
                usleep(300000);
        }
        return 0;
}
```