Εργαστήριο Λειτουργικών Συστημάτων

2η Εργαστηριακή Αναφορά

Παπαδόπουλος Χαράλαμπος 03120199

Στρουμπάκου Ειρήνη 03121183

Ομάδα 25

Εισαγωγή

Αντικείμενο της άσκησης είναι η υλοποίηση ενός character device driver για ένα δίκτυο αισθητήρων. Το δίκτυο διαθέτει έναν αριθμό από αισθητήρες (με ενδείξεις τάσης, θερμοκρασίας και φωτεινότητας) κα έναν σταθμό βάσης. Ο σταθμός βάσης συνδέεται μέσω USB με υπολογιστικό σύστημα Linux στο οποίο και θα εκτελείται ο εν λόγω οδηγός συσκευής.

Στο θεωρητικό υπόβαθρο, εξετάζονται οι βασικές αρχές του διαχωρισμού του χώρου πυρήνα και χρήστη στο Linux, καθώς και ο ρόλος των kernel modules. Οι οδηγοί συσκευών επιτρέπουν την επικοινωνία μεταξύ λογισμικού και υλικού, κρύβοντας την πολυπλοκότητα της διαχείρισης των συσκευών. Τα ειδικά αρχεία του συστήματος αρχείων, όπως αυτά στον κατάλογο /dev, λειτουργούν ως διεπαφές πρόσβασης για τις συσκευές.

Αρχιτεκτονική

Η υλοποίηση του οδηγού συσκευής χαρακτηρίζεται από δύο στάδια. Αρχικά, υλοποιούνται οι βασικές λειτουργίες των system calls, όπως open και read, με στόχο τη διασφάλιση της λειτουργικότητας της συσκευής. Στη συνέχεια, ενσωματώνονται οι buffers που περιέχουν πραγματικά δεδομένα από τους αισθητήρες, επιτρέποντας την ταυτόχρονη πρόσβαση από πολλές διεργασίες.

Το σύστημα οργανώνεται σε δύο κύρια μέρη: Το πρώτο αναλαμβάνει τη συλλογή και την επεξεργασία των δεδομένων και από το σταθμό βάσης και την επεξεργασία τους με συγκεκριμένο πρωτόκολλο, τα οποία αποθηκεύονται σε κατάλληλες δομές προσωρινής αποθήκευσης στη μνήμη. Το δεύτερο αναλαμβάνει την εξαγωγή των δεδομένων στον χώρο χρήστη.

Προεργασία

Το πρώτο μέρος που αναφέραμε προηγουμένως (συλλογή και επεξεργασία δεδομένων) μας δόθηκε υλοποιημένο και λειτουργεί κατά τον εξής τρόπο:

- **(a): Συντονισμός και Αρχικοποίηση Συστήματος**
  Αφορά την αρχικοποίηση και καθαρισμό του οδηγού.
  - **Αρχείο:** lunix-main.c
    - **lunix_module_init:** Αρχικοποιεί αισθητήρες, διάταξη γραμμής, και συσκευή χαρακτήρων.
    - **lunix_module_cleanup:** Καθαρίζει τους πόρους κατά την απενεργοποίηση του module.
  - **Αρχείο:** lunix.mod.c
    - Περιέχει τα metadata για την ενσωμάτωση του module στον πυρήνα (MODULE_INFO, init_module, cleanup_module).

- **(b): Προώθηση Δεδομένων μέσω USB**
  Αφορά τη σύνδεση του σταθμού βάσης με το ΛΣ μέσω Serial over USB driver.
  - **Αρχείο:** Δεν περιλαμβάνεται στον οδηγό, καθώς ο προϋπάρχων οδηγός USB αναλαμβάνει αυτήν την λειτουργία.

- **(c): Διάταξη γραμμής (Line Discipline)**
  Διαχειρίζεται την γραμμή TTY και την επικοινωνία με το /dev/ttyUSB1.
  - **Αρχείο:** lunix-ldisc.c
    - **lunix_ldisc_open:** Ανοίγει τη διάταξη γραμμής και την προσαρμόζει για Lunix.
    - **lunix_ldisc_close:** Κλείνει τη διάταξη γραμμής.
    - **lunix_ldisc_receive_buf:** Επεξεργάζεται εισερχόμενα δεδομένα και τα στέλνει στο lunix-protocol.

- **(e): Ερμηνεία Πρωτοκόλλου**
  Επεξεργάζεται πακέτα δεδομένων και ενημερώνει τους αισθητήρες.
  - **Αρχείο:** lunix-protocol.c
    - **lunix_protocol_received_buf:** Διαχειρίζεται τα εισερχόμενα δεδομένα και καλεί τη lunix_protocol_update_sensors.
    - **lunix_protocol_update_sensors:** Ενημερώνει τις δομές αισθητήρων με τις νέες τιμές μέτρησης.

- **(f): Διαχείριση Buffer Αισθητήρων**
  Αποθηκεύει τις μετρήσεις αισθητήρων και συγχρονίζεται με τις διεργασίες.
  - **Αρχείο:** lunix-sensors.c
    - **lunix_sensor_init:** Δημιουργεί τις δομές δεδομένων για κάθε αισθητήρα.
    - **lunix_sensor_update:** Ενημερώνει τις τιμές μέτρησης και τα timestamps.
    - **lunix_sensor_destroy:** Καθαρίζει τους πόρους μνήμης.

Αξίζει σε αυτό το σημείο να σημειωθεί, πως στο qemu config έχει οριστεί η θύρα ttyS1 ως η επιλεγμένη σειριακή θύρα, ως αποτέλεσμα εμείς τρέχαμε ./lunix-attach /dev/ttyS1.

Τέλος, μας δόθηκαν και έτοιμες συναρτήσεις για δημιουργία συσκευών με τα minor και major numbers (mk-lunix-devs.sh) καθώς και ένα lookup table (lunix-lookup) καθώς το floating point arithmetic είναι «απαγορευμένο» στο kernel space, λόγω του σημαντικού overhead που θα προσέθετε η αποθήκευση των επιπλέον καταχωρητών στην στοίβα.

Υλοποίηση

Από εμάς ζητήθηκε να συμπληρώσουμε κομμάτια κώδικα για την υλοποίηση του κομματιού

- **(g): Συσκευή Χαρακτήρων**
  Παρέχει πρόσβαση στον χώρο χρήστη μέσω συσκευών χαρακτήρων
  - **Αρχείο:** lunix-chrdev.c

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

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

Η συνάρτηση αυτή είναι απλά βοηθητική και απλά συγκρίνει το timestamp των δεδομένων στον buffer του αισθητήρα με το timestamp του buffer δεδομένων. Πρακτικά, ελέγχει αν χρειάζονται ανανέωση τα δεδομένα.

```
/*
 * 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;
        unsigned long state_flags;

        long formatted_data, integer, fractional;
        uint32_t raw_data;

        /*
         * Grab the raw data quickly, hold the
         * spinlock for as little as possible.
         */

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

        spin_lock_irqsave(&sensor->lock, state_flags);


        /* Why use spinlocks? See LDD3, p. 119 */

        /*
         * Any new data available?
         */
        if(!(lunix_chrdev_state_needs_refresh(state))) {
                spin_unlock_irqrestore(&sensor->lock, state_flags);
                return -EAGAIN;
        }

        raw_data = sensor->msr_data[state->type]->values[0];
        state->buf_timestamp = sensor->msr_data[state->type]->last_update;
        spin_unlock_irqrestore(&sensor->lock, state_flags);


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

        switch (state->type) {
                case BATT:
                        formatted_data = lookup_voltage[raw_data];
                        formatted_data = (formatted_data * 100000) / 3500;
                        integer = formatted_data / 1000;
                        fractional = formatted_data % 100;
                        state->buf_lim = snprintf(state->buf_data,
LUNIX_CHRDEV_BUFSZ, "%ld.%02ld%%\n", integer, fractional);
                        break;
```

```
                case TEMP:
                        formatted_data = lookup_temperature[raw_data];
                        integer = formatted_data / 1000;
                        fractional = formatted_data % 1000;
                        state->buf_lim = snprintf(state->buf_data,
LUNIX_CHRDEV_BUFSZ, "%ld.%03ld\n", integer, fractional);
                        break;
                case LIGHT:
                        formatted_data = lookup_light[raw_data];
                        integer = formatted_data / 1000;
                        fractional = formatted_data % 1000;
                        state->buf_lim = snprintf(state->buf_data,
LUNIX_CHRDEV_BUFSZ, "%ld.%03ld\n", integer, fractional);
                        break;
                default:
                        return -EINVAL;
        }


        debug("leaving\n");

        return 0;
}
```

Πραγματοποιεί την λήψη των raw data από τους αισθητήρες και τα κάνει format ανάλογα την κατηγορία τους μέσω των lookup tables.

Χρησιμοποιούμε spinlocks για την αποφυγή race conditions με τα δεδομένα του αισθητήρα. Η επιλογή τους έναντι των semaphores έγινε επειδή εργαζόμαστε σε interrupt context και επιθυμούμε βέλτιστη ταχύτητα, δύο τομείς στους οποίους τα semaphores υστερούν (για την ακρίβεια είναι αδύνατον να χρησιμοποιηθούν semaphores λόγω του interrupt context).

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
        /* Declarations */
        int ret;
        int minor = iminor(inode);

         struct lunix_chrdev_state_struct *state;

        debug("entering\n");

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

        state = kzalloc(sizeof(*state), GFP_KERNEL);
```

```
        if(!state) {
                ret = -ENOMEM;
                goto out;
        }

        /*
         * Associate this open file with the relevant sensor based on
         * the minor number of the device node [/dev/sensor<NO>-<TYPE>]
         */
        state->sensor = &lunix_sensors[minor / 8];
        state->type = minor % 8;
        state->buf_data[LUNIX_CHRDEV_BUFSZ - 1]='\0';
        state->buf_lim = strnlen(state->buf_data, LUNIX_CHRDEV_BUFSZ);
        state->buf_timestamp = (uint32_t)(ktime_get_ns()/1000000);

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


        /* Allocate a new Lunix character device private state structure */
        filp->private_data = state;

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

Στην πραγματικότητα δεν χρειαζόταν να ορίσουμε την μέθοδο open(), καθώς ο πυρήνας διαχειρίζεται από μόνος του την δημιουργία μια δομής file και την αντιστοίχιση με την αντίστοιχη δομή file operations.

Παρ' όλα αυτά, εμείς θέλαμε να ενημερώνεται ο οδηγός μας για το άνοιγμα της συσκευής, ώστε να μας δίνεται η ευκαιρία να κάνουμε αρχικοποιήσεις σε δομές δεδομένων του οδηγού.

Συγκεκριμένα, δεσμεύει χώρο για τη δομή τύπου lunix_chrdev_state_struct που περιγράφει την τρέχουσα κατάσταση της συσκευής και τη συνδέει με τη δομή file μέσω του δείκτη private_data.

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
        struct lunix_chrdev_state_struct *state;
        state = filp->private_data;
        kfree(state);
        return 0;
}
```

Καλείται όταν μια διεργασία εκτελέσει το system call close() και απελευθερώνει τον χώρο που καταλάμβανε η δομή δεδομένων state.

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

        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))
                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);  //unlock before sleeping
                        if (wait_event_interruptible(sensor->wq,
lunix_chrdev_state_needs_refresh(state))) {
                                return -ERESTARTSYS;
                        }
                        //loop but first reacquire the lock
                        if (down_interruptible(&state->lock)){
                                return -ERESTARTSYS;
                        }
                }
        }
        /* End of file */
        available = state->buf_lim - *f_pos;

        /* Determine the number of cached bytes to copy to userspace */

        cnt = min(cnt, available);

        if(copy_to_user(usrbuf, (state->buf_data + *f_pos), cnt)) {
                ret = -EFAULT;
                up(&state->lock);
                goto out;
        }
```

```
        *f_pos += cnt;
        ret = cnt;
        /* Auto-rewind on EOF mode? */
        if(*f_pos == state->buf_lim)
                *f_pos = 0;

        debug("Read %li bytes \n", (long)cnt);

out:
        up(&state->lock);
        return ret;
}
```

Χρησιμοποιείται για να διαβάζει δεδομένα από τη συσκευή. Μια μη αρνητική επιστρεφόμενη τιμή φανερώνει το πλήθος των bytes που διαβάστηκαν επιτυχώς, ενώ η τιμή 0 υποδηλώνει ότι η ανάγνωση έφτασε στο τέλος του αρχείου (EOF).

Λαμβάνει δεδομένα από τους sensor buffers, φροντίζει για τη σωστή μορφοποίησή τους και τα περνά στο χώρο χρήστη.

```
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);
        /* register_chrdev_region? */
        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;
        }
        /* cdev_add? */
        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;
}
```

Καταχωρεί μία συσκευή χαρακτήρων στον πυρήνα. Αρχικά, ορίζει τη δομή του χαρακτήρα συσκευής (cdev) και την αντιστοιχεί στη δομή file_operations μέσω της cdev_init(). Στη συνέχεια, καταχωρεί την περιοχή αριθμών συσκευών (register_chrdev_region()), ξεκινώντας από το καθορισμένο LUNIX_CHRDEV_MAJOR και υπολογίζοντας τον αριθμό των μικρότερων αριθμών για κάθε αισθητήρα. Αφού καταχωρηθεί η περιοχή, προσθέτει την συσκευή στο σύστημα με την cdev_add().