Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Λειτουργικών Συστημάτων 2021-2022

2η Εργαστηριακή Άσκηση

Ομάδα 51

Χρήστος Μαρρές, Α.Μ.: 03118912

Κωνσταντίνος Σιδέρης, Α.Μ.: 03118134

Για την υλοποίηση του character driver Lunix-TNG τροποποιήσαμε και προσθέσαμε κώδικα στο αρχείο lunix-chrdev.c. Ακολουθεί ο κώδικας των system calls που υλοποιήθηκαν με σχόλια:

➔ lunix_chrdev_state_needs_refresh

```
static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct
*state)
{
    struct lunix_sensor_struct *sensor;
    int ret = 0;

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

    if (sensor->msr_data[state->type]->last_update != state->buf_timestamp)
       {
      debug("State needs refreshing!\n");
      ret = 1; /* If the kernel buffer data timestamp is different from the user
      buffer data timestamp then the state needs to be refreshed */
       }

    return ret; /* Return 1 if state needs refresh else return 0 */
}
```

Το παραπάνω system call συγκρίνει το timestamp των δεδομένων που έχουν ληφθεί σε μορφή πακέτου στο kernel space και έχουν αποθηκευτεί στον buffer του κάθε σένσορα με το timestamp των δεδομένων που έχουν αποθηκευτεί στην κρυφή μνήμη (cache) του κάθε επιμέρους αισθητήρα, δηλαδή στο lunix_chrdev_state_struct (ορίζεται στο lunix-chrdev.h). Εάν τα δεδομένα στο struct με την κατάσταση του αισθητήρα πρέπει να ανανεωθούν επιστρέφει την τιμή 1 ενώ εάν έχουμε τα πιο πρόσφατα δεδομένα επιστρέφει την τιμή 0.

➔ lunix_chrdev_state_update

```
static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
{
    unsigned long flags;
    struct lunix_sensor_struct *sensor;
    sensor = state->sensor;
    long int result;

    uint16_t data;
    uint32_t timestamp;

    debug("entering\n");

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

```
     */
    spin_lock_irqsave(&sensor->lock, flags);
    data = sensor->msr_data[state->type]->values[0];
    /* Grab data from the kernel buffer */
    timestamp = sensor->msr_data[state->type]->last_update;
    /* Grab kernel buffer data timestamp */
    spin_unlock_irqrestore(&sensor->lock, flags);

    if(timestamp > state->buf_timestamp)
    { /* If kernel buffer data timestamp is different from the user
      buffer data timestamp update the sensor state */
       switch (state->type)
       { /* Find the sensor type and get the correct value
       from the corresponding lookup table */
             case BATT: result = lookup_voltage[data]; break;
             case TEMP: result = lookup_temperature[data]; break;
             case LIGHT: result = lookup_light[data]; break;
             default: return -EAGAIN; break;
       }

       state->buf_timestamp = timestamp;
       /* Format data as a number with 3 decimals digits */
       state->buf_lim = snprintf(state->buf_data, LUNIX_CHRDEV_BUFSZ, "%ld.%03ld\n",
result/1000, result%1000);
    }
    else
    { /* If kernel buffer data timestamp is not different from the user
      buffer data timestamp exit with error EAGAIN */
       debug("MSR Timestamp: %d\nbuf_timestamp: %d", timestamp, state->buf_timestamp);
       return -EAGAIN;
    }
    debug("leaving\n");
    return 0;
}
```

Το παραπάνω system call ανανεώνει τα δεδομένα που έχουν αποθηκευτεί στην κρυφή μνήμη (cache) του αισθητήρα. Αφού λάβουμε δεδομένα από τον buffer του σένσορα ελέγχουμε εάν το timestamp τον νέων δεδομένων που έχουμε λάβει είναι μεγαλύτερο από το timestamp των δεδομένων που βρίσκονται αυτή την στιγμή αποθηκευμένα στην κρυφή μνήμη (cache) αισθητήρα. Εάν δεν είναι, δηλαδή δεν έχουμε νέα δεδομένα, επιστρέφουμε το σφάλμα EAGAIN, διαφορετικά προχωράμε στην επεξεργασία τους. Στο kernel space δεν μπορούν να γίνουν πράξεις υποδιαστολής συνεπώς χρησιμοποιώντας την συνάρτηση snprintf με παράμετρο "%ld.%03ld\n" μορφοποιούμε τα δεδομένα, που αρχικά είναι σε μορφή ακεραίου, ώστε να πάρουν μορφή αριθμού με 3 δεκαδικά ψηφία. Χρησιμοποιούμε spinlocks, και συγκεκριμένα την spin_lock_irqsave η οποία απενεργοποιεί τις διακοπές, έτσι ώστε όταν λαμβάνουμε δεδομένα να μην έχουν πρόσβαση στον buffer του σένσορα δύο διεργασίες ταυτόχρονα. Έτσι αποτρέπουμε την περίπτωση του να λάβουμε νέο πακέτο δεδομένων ενώ κάνουμε ανανέωση και συνεπώς να διαβάσουμε λανθασμένα δεδομένα (κάποια δεδομένα από το παλιό πακέτο και κάποια από το καινούργιο ή δεδομένα με λάθος timestamp κτλ.).

➔ lunix_chrdev_open

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

    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); /*Find the minor nuber from the special /dev file inode*/
    type = minor%8; /* minor = sensor*8 + type => type = minor%8 */
    sensor = minor/8; /* minor = sensor*8 + type => sensor = minor/8 */
    debug("Minor number decoded. Sensor:%d and Type:%d\n", sensor, type);

    /* Allocate a new Lunix character device private state structure */
    state = kmalloc(sizeof(struct lunix_chrdev_state_struct), GFP_KERNEL);
    state->type = type;
    state->sensor = &lunix_sensors[sensor];

    state->buf_lim = 0; /* Initialise the sensor's state buffer info */
    state->buf_timestamp = 0;
    filp->private_data = state;

    sema_init(&state->lock,1); /* Initialise the semaphore */
       ret = 0;
out:
    debug("leaving, with ret = %d\n", ret);
    return ret;
}
```

Το παραπάνω system call ανοίγει το ειδικό αρχείο /dev του αισθητήρα και στην συνέχεια με την χρήση της μακροεντολής iminor ανακτά τον minor number του αρχείου. Γνωρίζουμε, από τον τρόπο δημιουργίας των αρχείων, που ορίζεται στο script lunix_dev_nodes.sh, πως το minor number των ειδικών αυτών αρχείων, είναι της μορφής minor = αισθητήρας* 8 + μέτρηση. Έτσι μπορούμε εύκολα να ανακτήσουμε τον τύπο του sensor, ως minor mod 8, και τον αριθμό του sensor, ως minor/8. Στην συνέχεια, χρησιμοποιώντας την kmalloc, με flag GFP_kernel (allocate normal kernel ram), δεσμεύουμε μνήμη από το kernel για το struct lunix_chrdev_state_struct (ορίζεται στο lunix-chdev.h) που περιγράφει την κατάσταση του αισθητήρα (δεδομένα, timestamp λήψης δεδομένων κτλ.). Έπειτα αρχικοποιούμε το struct με τις τιμές που έχουμε βρει για το type και το sensor, μηδενίζουμε τον counter του buffer (buf_lim) καθώς και το timestamp. Τέλος, αποθηκεύουμε στον δείκτη private_data πληροφορίες για την κατάσταση αισθητήρα στην δομή αρχείου (file structure) που τον περιγράφει και αρχικοποιούμε τον σεμαφόρο του σε ξεκλείδωτο.

➔ lunix_chrdev_realease

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
    debug("Releasing allocated memory!\n");
       kfree(filp->private_data); /*Release allocated memory*/
       debug("Done releasing memory!");
    return 0;
}
```

Το παραπάνω system call απελευθερώνει τον χώρο που έχουμε δεσμεύσει για το struct lunix_chrdev_state_struct που περιγράφει την κατάσταση του αισθητήρα. χρησιμοποιώντας την kfree.

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

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

    debug("Inside read!");
    if (down_interruptible(&state->lock))
    { /* Lock semaphore to prevent race conditions with father-child proccesses */
       ret = -ERESTARTSYS;
       goto out;
    }

    /* 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)
       {
                        up(&state->lock);
                        if (filp->f_flags & O_NONBLOCK)
                    return -EAGAIN; /* If we called lunix_chrdev_read with O_NONBLOCK
flag return -EAGAIN*/
                        if (wait_event_interruptible(sensor->wq,
lunix_chrdev_state_needs_refresh(state)))
                    return -ERESTARTSYS; /* Make proccess sleep until it is woken up
by lunix_chrdev_state_needs_refresh */
```

```
                        if (down_interruptible(&state->lock))
                    return -ERESTARTSYS; /* When the proccess wakes up lock the
semaphore */
        }
    }

    debug("After update, start reading");
    /* Determine the number of cached bytes to copy to userspace */
    if (*f_pos + cnt >= state->buf_lim)
        cnt = state->buf_lim - *f_pos;

    /* Copy to user space */
    if (copy_to_user(usrbuf, state->buf_data + *f_pos, cnt))
    {
            ret = -EFAULT; /* If unsuccesful exit with error EFAULT */
            goto out;
        }

    *f_pos += cnt; /* Update the buffer offset */
    ret = cnt;

    if (*f_pos == state->buf_lim)
      *f_pos = 0; /* Auto-rewind on EOF mode */
out:
    up(&state->lock); /* Unlock semaphore */
    debug("Done reading, returning");
    return ret; /* Return number of bytes read */
}
```

Το παραπάνω system call διαβάζει τα δεδομένα από τον buffer του struct lunix_chrdev_state_struct που περιγράφει την κατάσταση του αισθητήρα, και τα αντιγράφει στον χώρο χρήστη. Αρχικά κλειδώνουμε τον σεμαφόρο του struct. Στην συνέχεια όσο το system call lunix_chrdev_state_update επιστρέφει σφάλμα EAGAIN, δηλαδή δεν έχουμε νέες μετρήσεις, η διεργασία ξεκλειδώνει τον σεμαφόρο και εάν η lunix_chrdev_read έχει κληθεί με O_NONBLOCK flag τερματίζεται με σφάλμα EAGAIN αφού δεν διάβασε επιτυχώς. Διαφορετικά, αφού η read είναι blocking system call, δηλαδή περιμένει μέχρι να υπάρχει κάτι για να διαβάσει, καλούμε την wait_event_interruptible με condition την lunix_chrdev_state_needs_refresh έτσι ώστε η διεργασία να ξυπνήσει όταν θα υπάρχουν νέες τιμές για να διαβάσει. Αφού ξυπνήσει ξανά κλειδώνει τον σεμαφόρο και συνεχίζει να διαβάζει. Στην συνέχεια υπολογίζουμε τα αποθηκευμένα bytes που πρέπει να αντιγραφούν στο χώρο χρήστη και τα αντιγράφουμε με την copy_to_user. Έπειτα, ανανεώνουμε το offset του buffer (*f_pos). Αν έχουμε φτάσει στο τέλος του buffer το μηδενίζουμε, διαφορετικά προσθέτουμε σε αυτό τον αριθμό bytes που διαβάσαμε. Τέλος ξεκλειδώνουμε τον σεμαφόρο και επιστρέφουμε τον αριθμό των bytes που διαβάστηκαν. Ο λόγος που χρησιμοποιούμε κλειδώματα είναι διότι σε περίπτωση που διαβάζουν δύο διεργασίες πατέρας και παιδί θα δημιουργούταν race condition αφού οι δύο διεργασίες μοιράζονται το ίδιο state struct για τον αισθητήρα. Η διεργασία που διαβάζει πρώτη θα διάβαζε δεδομένα και θα άλλαζε το offset του buffer με συνέπεια η διεργασία που διαβάζει δεύτερη να διαβάσει λάθος δεδομένα.

➔ lunix_chrdev_init

```
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 a range of device numbers  */
    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);
    /* Add the char device to the system  */
    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;
}
```

Το παραπάνω system call αρχικοποιεί τα ειδικά αρχεία του driver. Αρχικά δημιουργούμε τον πρώτο αριθμό συσκευής dev_no με την εντολή MKDEV(LUNIX_CHRDEV_MAJOR, 0), όπου το major number LUNIX_CHRDEV_MAJOR είναι 60. Στην συνέχεια χρησιμοποιώντας την register_chrdev_region καταχωρούμε έναν εύρος (0-16*3) αριθμών συσκευών (lunix_minor_cnt) με το ίδιο major number και διαδοχικά minor numbers. Τέλος χρησιμοποιώντας την cdev_add προσθέτουμε τον οδηγό συσκευής στο σύστημα.