ΑΝΑΦΟΡΑ 2ΗΣ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΗΛΙΑΣ ΝΤΟΝΤΟΡΟΣ

el19206

Ο κώδικας ο οποίος έπρεπε να συμπληρώσουμε ήταν το σώμα 4 βασικών συναρτήσεων (init,open,read,release) και 2 βοηθητικών (lunix_chrdev_state_update, lunix_chrdev_state_needs_refresh).

INIT:

```
 1. int lunix_chrdev_init(void)
 2. {
 3.     /*
 4.      * Register the character device with the kernel, asking for
 5.      * a range of minor numbers (number of sensors * 8 measurements / sensor)
 6.      * beginning with LINUX_CHRDEV_MAJOR:0
 7.      */
 8.     int ret;
 9.     dev_t dev_no;
10.     unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;
11.
12.     debug("initializing character device\n");
13.     cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
14.     lunix_chrdev_cdev.owner = THIS_MODULE;
15.
16.     dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
17.
18.     ret = register_chrdev_region(dev_no, lunix_minor_cnt, "Lunix");
19.
20.     if (ret < 0)
21.     {
22.         debug("failed to register region, ret = %d\n", ret);
23.         goto out;
24.     }
25.
26.     ret = cdev_add(&lunix_chrdev_cdev, dev_no, 123);
27.     if (ret < 0)
28.     {
29.         debug("failed to add character device\n");
30.         goto out_with_chrdev_region;
31.     }
32.     debug("completed successfully\n");
33.     return 0;
34.
35. out_with_chrdev_region:
36.     unregister_chrdev_region(dev_no, lunix_minor_cnt);
37. out:
38.     return ret;
39. }
40.
```

Η init θα κληθεί όταν κάνουμε attach το module μας οπότε αρχικά στην γραμμή 13 κάνουμε initialize τον δικό μας character device driver. Έπειτα στις γραμμές 18 και 26 κάνουμε register τον driver με το major number που έχουμε επιλέξει και για πόσους Minor numbers μπορεί να κληθεί ο lunix driver.

OPEN:

```
 1. static int lunix_chrdev_open(struct inode *inode, struct file *filp)
 2. {
 3.     /* Declarations */
 4.     int ret;
 5.     struct lunix_chrdev_state_struct *state;
 6.     dev_t nodeminor;
 7.     int type;
 8.
 9.     debug("entering\n");
10.     ret = -ENODEV;
11.     if ((ret = nonseekable_open(inode, filp)) < 0)
12.         goto out;
13.     /*
14.      * Associate this open file with the relevant sensor based on
15.      * the minor number of the device node [/dev/sensor<NO>-<TYPE>]
16.      */
17.
18.     nodeminor = iminor(inode);
19.     type = nodeminor % 8;
20.     if (type >= 4)
21.     {
22.         ret = -ENODEV;
23.         goto out;
24.     }
25.
26.     /* Allocate a new Lunix character device private state structure */
27.     state = kmalloc(sizeof(struct lunix_chrdev_state_struct), GFP_KERNEL);
28.     if (!state)
29.     {
30.         ret = -EFAULT;
31.         goto out;
32.     }
33.
34.     if (type == 0)
35.         state->type = BATT;
36.     if (type == 1)
37.         state->type = TEMP;
38.     if (type == 2)
39.         state->type = LIGHT;
40.
41.     state->buf_lim = 0;
42.     state->buf_timestamp = 0;
43.     state->sensor = &lunix_sensors[(nodeminor >> 3)];
44.
45.     sema_init(&state->lock, 1);
46.     filp->private_data = state;
47.
48. out:
49.     debug("leaving, with ret = %d\n", ret);
50.     return ret;
51. }
```

RELEASE:

```
1. static int lunix_chrdev_release(struct inode *inode, struct file *filp)
2. {
3.     kfree(filp->private_data);
4.
5.     return 0;
6. }
7.
```

Κάθε φορά που κάποια διεργασία θα θελήσει να ανοίξει κάποιο από τα αρχεία που έχουμε δημιουργήσει με τα δεδομένα των αισθητήρων θα κληθεί η lunix_chrdev_open. Ώστε να μπορέσει ο driver μας να καταλάβει ποιο ακριβώς αρχείο είναι ώστε να στείλει τα κατάλληλα δεδομένα καλεί στην γραμμή 18 την iminor που επιστρέφει τον Minor number του αρχείο για το οποίο καλέστηκε η open. Από τον minor number βρίσκουμε τον τύπο των δεδομένων που ζητούνται. Στην γραμμή 27 ζητείται η απαραίτητη μνήμη από τον πυρήνα για το chrdev_state_struct και αμέσως μετά γίνονται initialize τα πεδία του state. Τέλος όλο το state αποθηκεύεται στο private_data του filp ώστε να μπορούν να ανατρέξουν οι υπόλοιπες συναρτήσεις που θα κληθούν στην συνέχεια. Η μνήμη η οποία έχει δεσμευτεί στην γραμμή 27 της open απελευθερώνεται όταν γίνει close το αρχείο και καλεστεί η release.

READ:

```
 1. static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf, size_t cnt, loff_t
*f_pos)
 2. {
 3.     ssize_t ret;
 4.
 5.     struct lunix_sensor_struct *sensor;
 6.     struct lunix_chrdev_state_struct *state;
 7.
 8.     state = filp->private_data;
 9.     WARN_ON(!state);
10.
11.     sensor = state->sensor;
12.     WARN_ON(!sensor);
13.
14.     if (down_interruptible(&state->lock))
15.         ret = -ERESTARTSYS;
16.     /*
17.      * If the cached character device state needs to be
18.      * updated by actual sensor data (i.e. we need to report
19.      * on a "fresh" measurement, do so
20.      */
21.     if (*f_pos >= (state->buf_lim))
22.         *f_pos = 0;
23.
24.     if (*f_pos == 0)
25.     {
26.         while (lunix_chrdev_state_update(state) == -EAGAIN)
27.         {
28.
29.             /* The process needs to sleep */
30.             up(&state->lock);
31.
32.             if (filp->f_flags & O_NONBLOCK)
33.             {
34.                 ret = -EAGAIN;
35.                 goto out;
36.             }
37.
38.             if (wait_event_interruptible(sensor->wq,
lunix_chrdev_state_needs_refresh(state)))
39.             {
40.                 ret = -ERESTARTSYS;
41.                 goto out;
42.             }
43.
44.             if (down_interruptible(&state->lock))
45.             {
```

```
46.                 ret = -ERESTARTSYS;
47.                 goto out;
48.             }
49.         }
50.     }
51.
52.     /* End of file */
53.
54.     /* Determine the number of cached bytes to copy to userspace */
55.
56.     if (state->buf_lim < *f_pos + cnt)
57.         cnt = state->buf_lim - *f_pos;
58.
59.     if (copy_to_user(usrbuf, state->buf_data + *f_pos, cnt))
60.     {
61.         ret = -EFAULT;
62.         goto out;
63.     }
64.
65.     *f_pos += cnt;
66.     ret = cnt;
67.     /* Auto-rewind on EOF mode? */
68.
69. out:
70.     up(&state->lock);
71.     return ret;
72. }
73.
```

Όταν καλείται η read από κάποια διεργασία, αρχικά επαναφέρουμε το state που είχαμε αποθηκεύσει με την εκτέλεση της open. Έπειτα ελέγχουμε από ποιο σημείο θέλει να ξεκινήσει να διαβάσει δεδομένα η διεργασία (f_pos). στην γραμμή 26 ελέγχουμε αν υπάρχουν νέα δεδομένα που μπορούμε να δώσουμε στο χρήστη. Εάν δεν υπάρχουν νέα δεδομένα ελέγχουμε αν υπάρχει το O_NONBLOCK στα flags με τα οποία καλέστηκε η read καθώς αμέσως μετά η διεργασία που κάλεσε την read θα πρέπει να μπει σε sleeping state μέχρι να υπάρχει κάποια αλλαγή και να έρθουν νέα δεδομένα (γραμμή 38). Όταν τελειώσει το while loop σημαίνει ότι έχουν έρθει νέα δεδομένα από τον συγκεκριμένο αισθητήρα και τα δίνουμε στον χρήστη (copy_to_user γραμμή 59) αφού πρώτα ενημερώσουμε το καινούριο f_pos και τέλος επιστρέφουμε το cnt δηλαδή τον αριθμό των χαρακτήρων που δώσαμε στο χρήστη.

STATE_UPDATE:

```
 1. static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
 2. {
 3.     struct lunix_sensor_struct *sensor;
 4.     unsigned long flags;
 5.     uint32_t data;
 6.     unsigned int dec, fract;
 7.     unsigned char sign;
 8.     long values;
 9.     long *lookup[N_LUNIX_MSR] = {lookup_voltage, lookup_temperature, lookup_light};
10.
11.     /*
12.      * Grab the raw data quickly, hold the
13.      * spinlock for as little as possible.
14.      */
15.     sensor = state->sensor;
16.     spin_lock_irqsave(&sensor->lock, flags);
```

```
17.
18.     /*
19.      * Any new data available?
20.      */
21.
22.     if (!lunix_chrdev_state_needs_refresh(state))
23.     {
24.         spin_unlock_irqrestore(&sensor->lock, flags);
25.         return -EAGAIN;
26.     }
27.     data = sensor->msr_data[state->type]->values[0];
28.     state->buf_timestamp = sensor->msr_data[state->type]->last_update;
29.
30.     spin_unlock_irqrestore(&sensor->lock, flags);
31.     /*
32.      * Now we can take our time to format them,
33.      * holding only the private state semaphore
34.      */
35.     values = lookup[state->type][data];
36.     if (values >= 0)
37.         sign = ' ';
38.     else
39.         sign = '-';
40.
41.     dec = values / 1000;
42.     fract = values % 1000;
43.     sprintf(state->buf_data, " %c%d.%d  ", sign, dec, fract);
44.
45.     state->buf_lim = strnlen(state->buf_data, 20);
46.
47.     debug("leaving\n");
48.     return 0;
49. }
```

STATE_NEEDS_REFRESH:

```
 1. static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *state)
 2. {
 3.     struct lunix_sensor_struct *sensor;
 4.
 5.     WARN_ON(!(sensor = state->sensor));
 6.     if (sensor->msr_data[state->type]->last_update > state->buf_timestamp)
 7.         return 1;
 8.     else
 9.         return 0;
10. }
```

Η συνάρτηση lunix_chrdev_state_update καλείται από την read και παίρνει τα καινούρια δεδομένα που πιθανώς να έχουν έρθει από τον αισθητήρα .Αρχικά παίρνει το spinlock και καλεί την luni_chrdev_needs_refresh η οποία ελέγχει αν τα τελευταία δεδομένα που έχουν έρθει είναι πιο καινούρια από τα τελευταία που έχουν αποθηκευτεί στο lunix_chrdev_state_struct. Εάν είναι τότε γίνεται αντιγραφή τους στην μεταβλητή data και απελευθερώνεται το spinlock Έπειτα φτιάχνουμε τα δεκαδικά ψηφιά ώστε να εμφανίζεται σωστά η τιμή και αποθηκεύουμε τα σωστά δεδομένα μέσα στο state ώστε να διαβαστούν από την read και να περαστούν στον χρήστη.