**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.