Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων & Μηχανικων Υπολογιστων τομεας τεχνολογιας πληροφορικης και υπολογιστων εργαστηριο υπολογιστικων συστηματων
www.cslab.ece.ntua.gr

Εργαστήριο Λειτουργικών Συστημάτων
Εργαστηριακή Άσκηση 2 Lunix-TNG
Σταύρος Λαζόπουλος 03120843, Ομάδα 66

Για την υλοποίηση της άσκησης προσθέσαμε κώδικα στα αρχεία lunix-chrdev.h και lunix-chrdev.c.

Στο αρχείο lunix-chrdev.h περιέχεται το structure με τα private data του ανοιχτού αρχείου που αντιστοιχεί στην συσκευή. Σε αυτό προσθέσαμε το int nonblocking το οποίο χρησιμοποιούμε για να ξεχωρίζουμε τις λειτουργίες blocking και non-blocking I/O.

```
30 /*
31  * Private state for an open character device node
32  */
33 struct lunix_chrdev_state_struct {
34   enum lunix_msr_enum type;
35   struct lunix_sensor_struct *sensor;
36
37   /* A buffer used to hold cached textual info */
38   int buf_lim;
39   unsigned char buf_data[LUNIX_CHRDEV_BUFSZ];
40   uint32_t buf_timestamp;
41
42   struct semaphore lock;
43
44   int nonblocking;
45   /*
46    * Fixme: Any mode settings? e.g. blocking vs. non-blocking
47    */
48 };
```

Στο αρχείο lunix-chrdev.c αρχικά υλοποιούμε την lunix_chrdev_init μέσω της οποίας θα γίνεται registered η συσκευή χαρακτήρων με τον kernel. Συγκεκριμένα θέλουμε να κάνουμε initialize το structure cdev lunix_chrdev_cdev.

Εμάς αρχικά μας ζητείται να συμπληρώσουμε την κλήση της register_chrdev_region η οποία αναθέτει device numbers στην συσκευή. Την καλούμε με ορίσματα ένα ζευγάρι major και minor number από τα οποία θα ξεκινήσει η ανάθεση, τον συνολικό αριθμό των συσκευών και το όνομά τους.

Στην συνέχεια μας ζητείται να συμπληρώσουμε την κλήση της cdev_add η οποία γνωστοποιεί στον πυρήνα τις συσκευές. Την καλούμε με ορίσματα ένα pointer στο cdev structure, τους major/minor numbers, και τον συνολικό αριθμό των συσκευών.

```
266 int lunix_chrdev_init(void)
267 {
268   /*
269    * Register the character device with the kernel, asking for
270    * a range of minor numbers (number of sensors * 8 measurements / sensor)
271    * beginning with LINUX_CHRDEV_MAJOR:0
272    */
273   int ret;
274   dev_t dev_no;
275   unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;
276
277   debug("initializing character device\n");
278   cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
279   lunix_chrdev_cdev.owner = THIS_MODULE;
280
281   dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
282   /* int register_chrdev_region(dev_t first, unsigned int count, char *name);
283   ret = register_chrdev_region(dev_no, lunix_minor_cnt, "lunix");
284
285   if (ret < 0) {
286     debug("failed to register region, ret = %d\n", ret);
287     goto out;
288   }
289   /* int cdev_add(struct cdev *dev, dev_t num, unsigned int count); */
290   ret = cdev_add(&lunix_chrdev_cdev, dev_no, lunix_minor_cnt);
291
292   if (ret < 0) {
293     debug("failed to add character device\n");
294     goto out_with_chrdev_region;
295   }
296   debug("completed successfully\n");
297   return 0;
298
299 out_with_chrdev_region:
300   unregister_chrdev_region(dev_no, lunix_minor_cnt);
301 out:
302   return ret;
303 }
```

Στην συνέχεια ξεκινάμε να υλοποιούμε τα file_operations των συσκευών μας.

```
static struct file_operations lunix_chrdev_fops =
{
  .owner          = THIS_MODULE,
  .open           = lunix_chrdev_open,
  .release        = lunix_chrdev_release,
  .read           = lunix_chrdev_read,
  .unlocked_ioctl = lunix_chrdev_ioctl,
  .mmap           = lunix_chrdev_mmap
};
```

Open:

```c
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
  /* Declarations */
  struct lunix_chrdev_state_struct *chrdev_state;
  unsigned int sensor_no, sensor_type;
  /* ? */

  int ret;

  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>]
   */
  sensor_no = iminor(inode) / 8;
  sensor_type = iminor(inode) % 8;

  /* Allocate a new Lunix character device private state structure */
  if(!(chrdev_state = kmalloc(sizeof(struct lunix_chrdev_state_struct), GFP_KERNEL))){
    debug("failed to allocate memory for device private state\n");
    ret = -ENOMEM;
    goto out;
  }
  chrdev_state->type = sensor_type;
  chrdev_state->sensor = &lunix_sensors[sensor_no];
  chrdev_state->buf_lim = 0;
  chrdev_state->buf_timestamp = 0;
  sema_init(&chrdev_state->lock, 1);
  chrdev_state->nonblocking = 0; //(filp->f_flags & O_NONBLOCK) ? 1 : 0;

  filp->private_data =chrdev_state;

  debug("chrdev_state initialized successfully");

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

Στην open αρχικά συσχετίζουμε το συγκεκριμένο αρχείο με ένα σένσορα σύμφωνα με τον minor number του inode. Στην συνέχεια κάνουμε allocate αρκετό χώρο για το lunix_chrdev_state_struct στο οποίο έχουμε ένα pointer chrdev_state. Μέσω αυτού του pointer συμπληρώνουμε τα μέλη του structure με τα κατάλληλα δεδομένα. Τέλος αντιγράφουμε αυτό το pointer στο private data της δομής file.

Release:

```c
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
  /* ? */
  kfree(filp->private_data);
  return 0;
}
```

Στην release αποδεσμεύουμε την μνήμη του lunix_chrdev_state_struct στο οποίο έδειχνε το private data του αρχείου.

Read:

```
182 static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf, size_t cnt, loff_t *f_pos)
183 {
184   ssize_t ret, cached_bytes;
185   struct lunix_sensor_struct *sensor;
186   struct lunix_chrdev_state_struct *state;
187
188   state = filp->private_data;
189   WARN_ON(!state);
190
191   sensor = state->sensor;
192   WARN_ON(!sensor);
193
194   /* Lock? */
195   if (down_interruptible(&state->lock))
196     return -ERESTARTSYS;
197   /*
198    * If the cached character device state needs to be
199    * updated by actual sensor data (i.e. we need to report
200    * on a "fresh" measurement, do so
201    */
202   if (*f_pos == 0) {
203     while (lunix_chrdev_state_update(state) == -EAGAIN) {
204       /* ? */
205       /* The process needs to sleep */
206       up(&state->lock);
207       if (state->nonblocking)
208         return -EAGAIN;
209       /*wait_event_interruptible(queue, condition) wait queue until */
210       if (wait_event_interruptible(sensor->wq, lunix_chrdev_state_needs_refresh(state)))
211         return -ERESTARTSYS; /* signal: tell the fs layer to handle it */
212
213       if (down_interruptible(&state->lock))
214         return -ERESTARTSYS;
215       /* See LDD3, page 153 for a hint */
216     }
217   }
218
219   /* End of file */
220   if (!state->buf_lim){
221     ret = 0 ;
222     goto out;
223   }
224   /* ? */
225
226   /* Determine the number of cached bytes to copy to userspace */
227   cached_bytes = state->buf_lim - *f_pos;
228   if (cached_bytes < cnt)
229     cnt = cached_bytes;
230
231   if (copy_to_user(usrbuf, state->buf_data + *f_pos, cnt)){
232     ret = -EFAULT;
233     goto out;
234   }
235   ret = cnt;
236   *f_pos += cnt;
237   /* ? */
238
239   /* Auto-rewind on EOF mode? */
240   if (*f_pos >= state->buf_lim){
241     *f_pos = 0;
242   }
243   /* ? */
244 out:
245   /* Unlock? */
246   up(&state->lock);
247   return ret;
248 }
```

Στην read ελέγχουμε αρχικά χρησιμοποιούμε το semaphore του state structure Έτσι ώστε συσκευές με το ίδιο ανοιχτό αρχείο (από fork ή threads, έτσι ώστε να έχουν ίδιο open file entry) να μην προσπαθουν να διαβάσουν ή/και ανανεώσουν τα δεδομένα ταυτόχρονα . Στην συνέχεια μπαίνουμε σε ένα loop μέχρι να έρθουν νέα δεδομένα (δηλαδή μέχρι η unix_chrdev_state_update να τελειώσει επιτυχώς). Για να μήν είναι busy loop αρχικά κάνουμε release το semaphore και στην συνέχεια μπαίνουμε σε ουρά αναμονή και “κοιμόμαστε” μέχρι να χρειαστεί να έρθουν νέα δεδομένα (condition η lunix_chrdev_state_needs_refresh). Αν είχαμε ορίσει nonblocking =1 τότε κάνουμε return -EAGAIN. Όταν βγούμε από την ουρά ξαναπαίρνουμε το lock και προσπαθούμε πάλι να ανανεώσουμε τα δεδομένα.

Εφόσον βγούμε από το loop ελέγχουμε αρχικά ότι υπάρχουν όντως δεδομένα στο buffer. Στην συνέχεια βρίσκουμε το μέγεθος των δεδομένων και τα αντιγράφουμε στο userspace μέσω της copy_to_user. Γράφουμε μέγιστο cnt δεδομένα διαβάζουμε από την θέση του pointer, f_pos, αυξάνουμε τον pointer κατά cnt αφού αντιγράψουμε τα δεδομένα και αν ξεπεράσουμε το μέγεθος του buffer (τεχνικά πάντα θα φτάνουμε στο τέλος του buffer χωρίς να το ξεπεράσουμε) επιστρέφουμε τον pointer στην αρχή. Τέλος αφήνουμε το lock και επιστρέφουμε το πόσο γράψαμε.

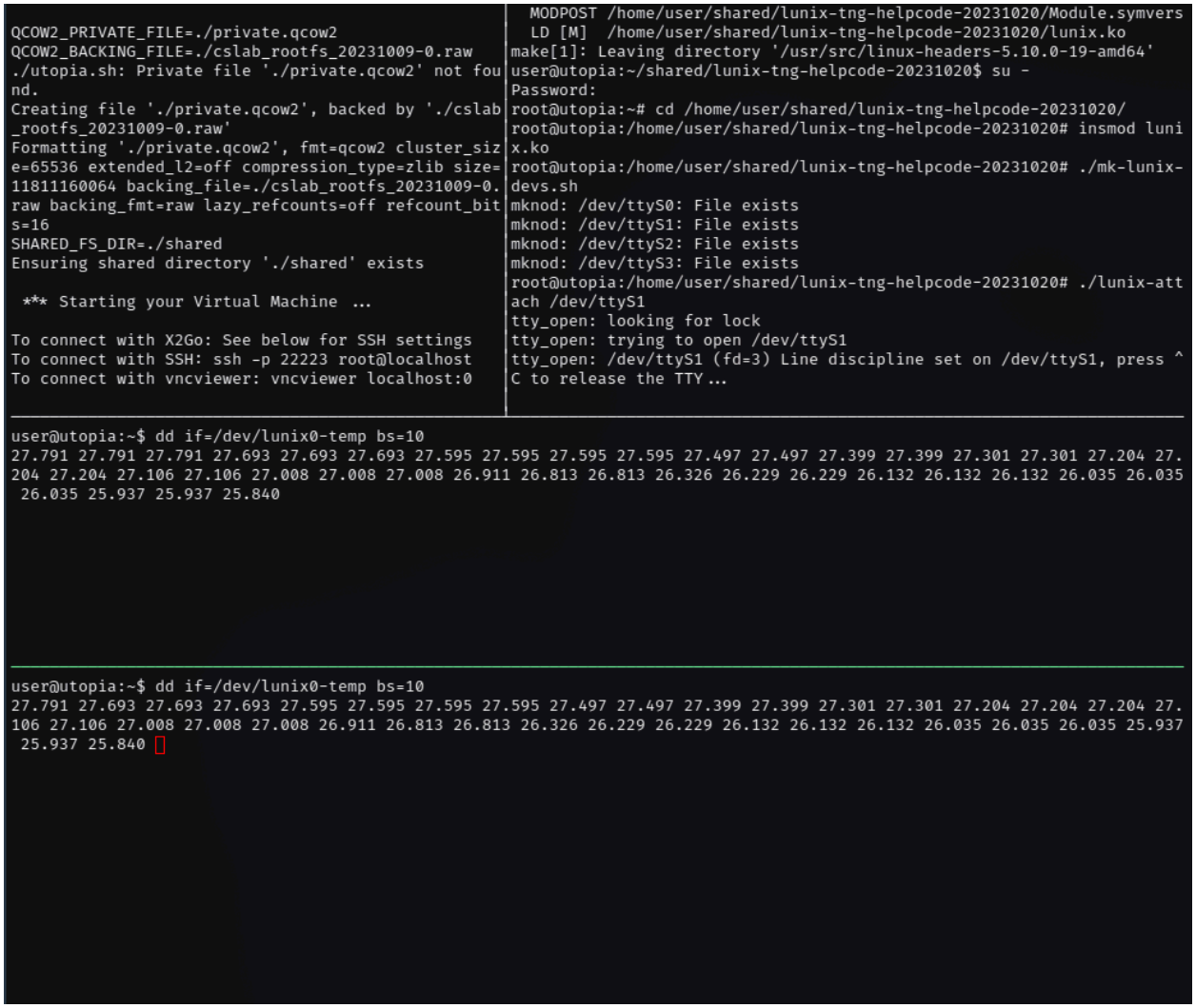
Η lunix_chrdev_state_update διαβάζει γρήγορα τα raw δεδομένα από τον sensor χρησιμοποιώντας spinlock σώζοντας το interrupt state διότι η συνάρτηση ανανέωσης των sensor τρέχει σε interrupt context. Αφού διαβάσουμε τα δεδομένα ελέγχουμε αν όντως υπάρχουν νέα δεδομένα. Στην συνέχεια ανάλογα με τον τύπο των δεδομένων με το lookup table παράγουμε τα cooked data. Τέλος ανανεώνουμε το state.

```
62 static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
63 {
64   struct lunix_sensor_struct *sensor;
65   /**/
66   uint32_t raw_data;
67   uint32_t timestamp;
68   long cooked_data;
69   unsigned long flags;
70
71   /*
72    * Grab the raw data quickly, hold the
73    * spinlock for as little as possible.
74    */
75   WARN_ON(!(sensor = state->sensor));
76   spin_lock_irqsave(&sensor->lock, flags);
77   raw_data = sensor->msr_data[state->type]->values[0];
78   timestamp = sensor->msr_data[state->type]->last_update;
79   spin_unlock_irqrestore(&sensor->lock, flags);
80   /* ? */
81
82   /* Why use spinlocks? See LDD3, p. 119 */
83
84   /*
85    * Any new data available?
86    */
87   if(!(state->buf_timestamp < timestamp))
88     return -EAGAIN;
89   /* ? */
90
91   /*
92    * Now we can take our time to format them,
93    * holding only the private state semaphore
94    */
95   if (state->type == BATT)
96     cooked_data = lookup_voltage[raw_data];
97   else if (state->type == TEMP)
98     cooked_data = lookup_temperature[raw_data];
99   else if (state->type == LIGHT)
100    cooked_data = lookup_light[raw_data];
101  else
102    goto out;
103
104
105
106  state->buf_lim = snprintf(state->buf_data, LUNIX_CHRDEV_BUFSZ,
107      "%ld.%03ld ", cooked_data/1000, cooked_data%1000);
108  state->buf_timestamp = timestamp;
109
110  /* ? */
111
112 out:
113   debug("leaving\n");
114   return 0;
115 }
```

Η lunix_chrdev_state_needs_refresh συγκρίνει το timestamp του state με το last update του σενσορα όπως και η lunix_chrdev_state_update για να δει αν χρειαζόμαστε ανανέωση.

```c
static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *state)
{
  struct lunix_sensor_struct *sensor;
  debug("enetering\n");
  /* ? */
  sensor = state->sensor;
  if (state->buf_timestamp < sensor->msr_data[state->type]->last_update)
    return 1;
  /* The following return is bogus, just for the stub to compile */
  return 0; /* ? */
}
```

Ελέγχουμε την λειτουργία του οδηγού με πολλαπλά processes να κάνουν accesss το ίδιο αρχείο.



Αν κοιτάξουμε την εντολή με strace βλέπουμε ότι ανοίγει το μαγικό αρχείο με fd 3. Αντιγράφει το fd 3 στο 0. Προσπαθεί να κάνει seek αλλά ο οδηγός μας δεν το υποστηρίζει. Τέλος κάνει read τα δεδομένα και τα γράφει στο terminal.