Εργαστήριο Λειτουργικών Συστήματων
2η Άσκηση

Αλέξιος Ζαμάνης
03115010
Μιχαήλ Μεγγίσογλου
03115014

4 Δεκεμβρίου 2018

1 Σχολιασμός των συναρτήσεων

1.1 lunix_chrdev_init

Ζητάμε από τον πυρήνα το major number 60 για το character device μας με όνομα Lunix:TNG, μαζί με μία σειρά από minor numbers για κάθε πιθανό ζεύγος αισθητήρα-μέτρησης. Αν μας δοθεί, καταχωρούμε το device μας στο σύστημα και αυτό γίνεται άμεσα διαθέσιμο.

1.2 lunix_chrdev_open

Παίρνουμε από το inode, ήτοι το ζεύγος αισθητήρα-μέτρησης που μας ενδιαφέρει, το minor number του και δεσμεύουμε-αρχικοποιούμε βάσει αυτού το πεδίο private_data που αντιστοιχεί στο εκάστοτε άνοιγμα αρχείου του νήματος εκτέλεσής μας. Στην ίδια δομή αρχικοποιούμε ένα mutex, που θα χρησιμοποιήσουμε παρακάτω.

1.3 lunix_chrdev_release

Στο κλείσιμο του instance του ανοιχτού αρχείου που βλέπει το νήμα εκτέλεσης μας απελευθερώνομε τους πόρους που είχε δεσμεύσει το πεδίο private data.

1.4 lunix_chrdev_read

Εδώ βρίσκεται η καρδιά του οδηγού συσκευών μας. Το ενδιαφερόμενο για ανάγνωση νήμα εκτέλεσης κλειδώνει ένα mutex, μεταφέρει στο χώρο χρήστη όσα δεδομένα ζήτησε να διαβάσει (ή, αν είναι λιγότερα, όσα είναι διαθέσιμα), τοποθετεί το file offset στη σωστή θέση (ή το μηδενίζει, αν φτάσαμε στο τέλος των διαθέσιμων δεδομένων), ξεκλειδώνει το mutex και επιστρέφει το πλήθος των δεδομένων που διάβασε. Με το mutex αποφεύγουμε την περίπτωση ταυτόχρονης πρόσβασης στα δεδομένα, που θα οδηγούσε σε εσφαλμένη τελική τιμή στο offset.

Στην περίπτωση που δεν υπάρχουν δεδομένα, το νήμα εκτέλεσης ζητά ατομικά την ανανέωση του buffer, καλώντας την lunix_chrdev_state_update, την οποία θα συζητήσουμε παρακάτω. Αν αυτή επιστρέψει κατά σύμβαση -EAGAIN, ήτοι δεν υπάρχουν νέα δεδομένα, ξεκλειδώνουμε το mutex, και μπαίνουμε στην ουρά αναμονής που αντιστοιχεί στο ζεύγος αισθητήρα-μέτρησης που μας ενδιαφέρει, έως ότου ικανοποιηθεί η συνθήκη που περιγράφει η συνάρτηση lunix_chrdev_state_needs_refresh (βλέπε παρκάτω). Όσο περιμένουμε, κοιμόμαστε, ώστε να μη σπαταλάμε το χρόνο του επεξεργαστή. Όταν πλέον ξυπνήσουμε, ξαναπαίρνουμε το κλείδωμα και επαναλαμβάνουμε το αίτημα ανανέωσης του buffer.

Αξίζει να σημειωθεί ότι χρησιμοποιούμε τις interruptible εκδοχές των κλειδωμάτων και της ουράς αναμονής, ώστε πιθανά interrupts, π.χ. ένα σήμα SIGKILL

από το χρήστη, να μπορούν να ξυπνήσουν το νήμα εκτέλεσης που έχει κοιμηθεί. Σε διαφορετική περίπτωση θα παρουσιάζονταν έντονο latency προς το μέρος του χρήστη.

1.5 lunix_chrdev_state_update

Η συνάρτηση αυτή καλείται από ένα νήμα εκτέλεσης που έχει λάβει το κλείδωμα. Σε πρώτο στάδιο κλειδώνει ένα spinlock, καλεί τη lunix_chrdev_state_needs_refresh (βλέπε παρακάτω), και ανάλογα με την απάντησή της, είτε αποθηκεύει τοπικά τα νέα δεδομένα, ανανεώνοντας παράλληλα τη χρονοσφραγίδα και αφήνοντας εν συνεχεία το spinlock, είτε ξεκλειδώνει το spinlock και επιστρέφει κατα σύμβαση -EAGAIN στον καλούντα.

Η χρήση spinlock προστατεύει τον αισθητήρα από ταυτόχρονες εγγραφές σε interrupt context. Μάλιστα, η εκδοχή των spinlocks που χρησιμοποιούμε απενεργοποιεί τα interrupts στον τοπικό επεξαργαστή, αποθηκεύοντας παράλληλα την προηγούμενη κατάσταση (ήτοι αν ήταν ήδη απενεργοποιημένα), αποτρέποντας πιθανά αδιέξοδα.

Στο δεύτερο στάδιο δίνουμε στα raw δεδομένα τις τιμές που αντιστοιχούν στο μέγεθος που περιγράφουν, βάσει λεξικού, και τα μετατρέπουμε σε πλασματική δεκαδική αναπαράσταση, αποθηκεύοντας παράλληλα τη χρονοσφραγίδα και τη μορφοποιημένη τιμή στο πεδίο private_data του ανοιχτού αρχείου. Η επιλογή χρήσης ακεραίων οφείλεται σε περιορισμούς που επιβάλλει το λειτουργικό συστημα, αναφορικά με τους καταχωρητές των πράξεων κινητής υποδιαστολής.

1.6 lunix_chrdev_state_needs_refresh

Εδώ γρήγορα ελέγχουμε αν η χρονοσφραγίδα που διαθέτουμε προηγείται χρονικά της τελευταίας ανανέωσης του ζεύγους αισθητήρα-μέτρησης που μας αφορά και επιστρέφουμε την αντίστοιχη αληθοτιμή.

2 Παράρτημα πηγαίου κώδικα

2.1 lunix-chrdev.c

```
1 /*
2  * lunix-chrdev.c
3  *
4  * Implementation of character devices
5  * for Lunix:TNG
6  *
7  * Alexios Zamanis
8  * Michalis Megisoglou
```

```
 9   *
10   */
11
12  #include <linux/mm.h>
13  #include <linux/fs.h>
14  #include <linux/init.h>
15  #include <linux/list.h>
16  #include <linux/cdev.h>
17  #include <linux/poll.h>
18  #include <linux/slab.h>
19  #include <linux/sched.h>
20  #include <linux/ioctl.h>
21  #include <linux/types.h>
22  #include <linux/module.h>
23  #include <linux/kernel.h>
24  #include <linux/mmzone.h>
25  #include <linux/vmalloc.h>
26  #include <linux/spinlock.h>
27
28  #include "lunix.h"
29  #include "lunix-chrdev.h"
30  #include "lunix-lookup.h"
31
32  /*
33   * Global data
34   */
35  struct cdev lunix_chrdev_cdev;
36
37  /*
38   * Just a quick [unlocked] check to see if the cached
39   * chrdev state needs to be updated from sensor
        measurements.
40   */
41  static int lunix_chrdev_state_needs_refresh(struct
      lunix_chrdev_state_struct *state)
42  {
43      struct lunix_sensor_struct *sensor;
44
45      WARN_ON(!(sensor = state->sensor));
```

```
46      if (state->buf_timestamp < sensor->msr_data[state->
           type]->last_update)
47          return 1;
48      return 0;
49  }
50
51  /*
52   * Updates the cached state of a character device
53   * based on sensor data. Must be called with the
54   * character device state lock held.
55   */
56  static int lunix_chrdev_state_update(struct
       lunix_chrdev_state_struct *state)
57  {
58      struct lunix_sensor_struct *sensor;
59      unsigned long flags;
60      uint32_t timestamp;
61      uint16_t tmp;
62      long long_format;
63      unsigned char sign;
64      int integer, fractional;
65
66      debug("leaving\n");
67
68      /*
69       * Grab the raw data quickly, hold the
70       * spinlock for as little as possible.
71       */
72      sensor = state->sensor;
73      spin_lock_irqsave(&sensor->lock, flags);
74
75      /*
76       * Any new data available?
77       */
78      if (lunix_chrdev_state_needs_refresh(state))
79      {
80          timestamp = sensor->msr_data[state->type]->
               last_update;
81          tmp = sensor->msr_data[state->type]->values[0];
82      }
```

```
83      else
84      {
85          spin_unlock_irqrestore(&sensor->lock, flags);
86          return -EAGAIN;
87      }
88      spin_unlock_irqrestore(&sensor->lock, flags);
89
90      /*
91       * Now we can take our time to format them,
92       * holding only the private state semaphore
93       */
94
95      switch (state->type)
96      {
97      case BATT:
98          long_format = lookup_voltage[tmp];
99          break;
100     case LIGHT:
101         long_format = lookup_light[tmp];
102         break;
103     case TEMP:
104         long_format = lookup_temperature[tmp];
105     }
106
107     sign = long_format >= 0 ? ' ' : '-';
108     integer = long_format / 1000;
109     fractional = long_format % 1000;
110
111     state->buf_lim = snprintf(state->buf_data,
            LUNIX_CHRDEV_BUFSZ, "%c%d.%d\n", sign, integer,
            fractional);
112     state->buf_timestamp = timestamp;
113
114     debug("leaving\n");
115     return 0;
116 }
117
118 /*********************************
119  * Implementation of file operations
120  * for the Lunix character device
```

```
121  ************************************/
122
123 static int lunix_chrdev_open(struct inode *inode, struct
        file *filp)
124 {
125     /* Declarations */
126     unsigned int minor;
127     struct lunix_chrdev_state_struct *private_data;
128     int ret;
129     extern struct lunix_sensor_struct *lunix_sensors;
130
131     debug("entering\n");
132     ret = -ENODEV;
133     if ((ret = nonseekable_open(inode, filp)) < 0)
134         goto out;
135
136     /*
137      * Associate this open file with the relevant sensor
            based on
138      * the minor number of the device node [/dev/sensor<
           NO>-<TYPE>]
139      */
140
141     minor = iminor(inode);
142
143     /* Allocate a new Lunix character device private
           state structure */
144     private_data = kmalloc(sizeof(struct
           lunix_chrdev_state_struct), GFP_KERNEL);
145     private_data->type = minor & 7;
146     private_data->sensor = &lunix_sensors[minor >> 3];
147     private_data->buf_timestamp = private_data->sensor->
           msr_data[private_data->type]->last_update;
148     sema_init(&private_data->lock, 1);
149
150     filp->private_data = private_data;
151 out:
152     debug("leaving, with ret = %d\n", ret);
153     return ret;
154 }
```

```
155
156 static int lunix_chrdev_release(struct inode *inode,
       struct file *filp)
157 {
158     kfree(filp->private_data);
159     return 0;
160 }
161
162 static long lunix_chrdev_ioctl(struct file *filp,
       unsigned int cmd, unsigned long arg)
163 {
164     return -EINVAL;
165 }
166
167 static ssize_t lunix_chrdev_read(struct file *filp, char
      __user *usrbuf, size_t cnt, loff_t *f_pos)
168 {
169     ssize_t ret;
170
171     struct lunix_sensor_struct *sensor;
172     struct lunix_chrdev_state_struct *state;
173
174     state = filp->private_data;
175     WARN_ON(!state);
176
177     sensor = state->sensor;
178     WARN_ON(!sensor);
179
180     if (down_interruptible(&state->lock))
181         return -ERESTARTSYS;
182     /*
183      * If the cached character device state needs to be
184      * updated by actual sensor data (i.e. we need to
          report
185      * on a "fresh" measurement, do so
186      */
187     if (*f_pos == 0)
188     {
189         while (lunix_chrdev_state_update(state) == -
            EAGAIN)
```

```
190             {
191                 up(&state->lock);
192                 if (filp->f_flags & O_NONBLOCK)
193                     return -EAGAIN;
194                 if (wait_event_interruptible(sensor->wq,
                        lunix_chrdev_state_needs_refresh(state)))
195                     return -ERESTARTSYS;
196                 /* The process needs to sleep */
197                 if (down_interruptible(&state->lock))
198                     return -ERESTARTSYS;
199             }
200         }
201 
202         /* End of file */
203         if (state->buf_lim == 0)
204         {
205             ret = 0;
206             goto out;
207         }
208 
209         /* Determine the number of cached bytes to copy to
               userspace */
210         if (cnt >= state->buf_lim - *f_pos)
211             cnt = state->buf_lim - *f_pos;
212         if (copy_to_user(usrbuf, state->buf_data + *f_pos,
               cnt))
213         {
214             ret = -EFAULT;
215             goto out;
216         }
217         *f_pos += cnt;
218         ret = cnt;
219 
220         /* Auto-rewind on EOF mode? */
221         if (*f_pos == state->buf_lim)
222             *f_pos = 0;
223 out:
224         up(&state->lock);
225         return ret;
226 }
```

```
227
228 static int lunix_chrdev_mmap(struct file *filp, struct
       vm_area_struct *vma)
229 {
230     return -EINVAL;
231 }
232
233 static struct file_operations lunix_chrdev_fops = {
234     .owner = THIS_MODULE,
235     .open = lunix_chrdev_open,
236     .release = lunix_chrdev_release,
237     .read = lunix_chrdev_read,
238     .unlocked_ioctl = lunix_chrdev_ioctl,
239     .mmap = lunix_chrdev_mmap};
240
241 int lunix_chrdev_init(void)
242 {
243     /*
244      * Register the character device with the kernel,
           asking for
245      * a range of minor numbers (number of sensors * 8
           measurements / sensor)
246      * beginning with LINUX_CHRDEV_MAJOR:0
247      */
248     int ret;
249     dev_t dev_no;
250     unsigned int lunix_minor_cnt = lunix_sensor_cnt <<
          3;
251
252     debug("initializing character device\n");
253     cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
254     lunix_chrdev_cdev.owner = THIS_MODULE;
255
256     dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
257     ret = register_chrdev_region(dev_no, lunix_minor_cnt
          , "Lunix:TNG");
258     if (ret < 0)
259     {
260         debug("failed to register region, ret = %d\n",
              ret);
```

```
261             goto out;
262         }
263         ret = cdev_add(&lunix_chrdev_cdev, dev_no,
                lunix_minor_cnt);
264         if (ret < 0)
265         {
266             debug("failed to add character device\n");
267             goto out_with_chrdev_region;
268         }
269         debug("completed successfully\n");
270         return 0;
271 
272     out_with_chrdev_region:
273         unregister_chrdev_region(dev_no, lunix_minor_cnt);
274     out:
275         return ret;
276     }
277 
278     void lunix_chrdev_destroy(void)
279     {
280         dev_t dev_no;
281         unsigned int lunix_minor_cnt = lunix_sensor_cnt <<
                3;
282 
283         debug("entering\n");
284         dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
285         cdev_del(&lunix_chrdev_cdev);
286         unregister_chrdev_region(dev_no, lunix_minor_cnt);
287         debug("leaving\n");
288     }
```