Σιγλίδης Γιάννης
Κωνσταντινίδης Ορέστης
Ομάδα Α22

Εργαστήριο Λειτουργικών Συστημάτων :

Οδηγός Ασύρματου Δικτύου Αισθητήρων στο λειτουργικό σύστημα Linux

2η Άσκηση για το μάθημα Εργαστήριο Λειτουργικών Συστημάτων.
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Διδάσκοντες : Ν. Κοζύρης, Π. Τσανάκας

1 Ζητούμενα της Άσκησης

Στην συγκεκριμένη άσκηση, δεν δίνονται ερωτήσεις προς απάντηση. Μετά από υπόδειξη ενός εκ των βοηθών του εργαστηρίου, θα προσπαθήσουμε, σε αυτή την αναφορά να γίνει περιγραφή της κατασκευής του κώδικα που ζητήθηκε ώστε να παρουσιαστεί με τον καλύτερο τρόπο η προσπάθειά μας και το επίπεδο κατανόησης του τι τελικά κατασκευάσαμε.

Αρχικά, θα προσπαθήσουμε να περιγράψουμε τις συναρτήσεις και τον ρόλο τους στην υλοποίηση του οδηγού, που βρίσκονται στο κομμάτι του κώδικα που ζητήθηκε να αναπτύξουμε. Ο καλύτερος τρόπος για να γίνει αυτό θεωρήσαμε ότι είναι να παρουσιάσουμε τις συναρτήσεις σε "χρονική" σειρά, με την λογική του ποιά θα χρησιμοποιηθεί πρώτη, όταν τελικά "τρέξει" ο οδηγός. Οπότε, έχουμε:

1.1 Init

Αρχικά με την εκτέλεση της εντολής insmod, θα τρέξει η συνάρτηση lunix_chrdev_init η οποία θα κάνει την αρχικοποίηση της συσκευής χαρακτήρων. Στην συγκεκριμένη άσκηση είχαμε μια συσκευή χαρακτήρων. Ορίζεται η μεταβλητή lunix_minor_cnt η οποία "μετράει" το πόσες διαφορετικές τιμές θέλουμε να αποθηκεύονται. Αφήνουμε χώρο για περαιτέρω ανάπτυξη του οδηγού δίνοντας χώρο για 8 είδη μετρήσεων. Η μεταβλητή dev_no περιέχει τον major και τον minor αριθμό της συσκευής.

Οι σημαντικότερες συναρτήσεις που χρησιμοποιούνται εδώ είναι οι register_chrdev_region και cdev_add. Η πρώτη ενημερώνει το λειτουργικό σύστημα για τον χώρο που θα χρειαστούμε. Το μέγεθος των δεδομένων που θα χρησιμοποιηθούν,, το όνομα της συσκευής καθώς και τον major και τον minor αριθμό της συσκευής, ώστε να "προετοιμαστεί" το έδαφος κατάλληλα για την χρησιμοποίησή της. Η δεύτερη, πραγματοποιεί την δέσμευση αυτού του χώρου. Ο λόγος που χρησιμοποιούνται δύο συναρτήσεις για αυτή την λειτουργία γίνεται περισσότερο κατανοητός αν σκεφτούμε ότι ίσως σε κάποιον οδηγό να θέλουμε να χρησιμοποιήσουμε παραπάνω από μία συσκευή και στην συνέχεια να δεσμεύσουμε για την κάθε συσκευή τον χώρο που της αναλογεί. Μπορούμε να παρατηρήσουμε ότι κατά την δέσμευση του χώρου δίνεται δείκτης προς τον "χώρο" της συσκευής.

Το υπόλοιπο "κομμάτι" της lunix_chrdev_init απλά ελέγχει ότι όλες οι συναρτήσεις εκτελέσαν την λειτουργία τους χωρίς λάθη.

1.2 Open

Στην συνέχεια, παρουσιάζουμε την συνάρτηση lunix_chrdev_open η οποία παίρνει δύο εισόδους. Η πρώτη είναι δείκτης σε μια δομή που περιέχει τα στοιχεία της συσκευής (major number, minor number, etc), το γνωστό και ως inode και η δεύτερη είναι μία δομή του πυρήνα που υπάρχει μόνο κατά την εκτέλεση του module και βοηθάει στην καταγφή των δεδομένων της συσκευής. Χρησιμοποιούμε την συνάρτηση πυρήνα kmalloc για την δέσμευση του απαραίτητου χώρου για την δομή κατάστασης. Η δομή αυτή είναι η lunix_chrdev_state_struct που ορίζεται μέσα στο αρχείο επικεφαλίδων lunix-chrdev.h, και αναπαριστά ένα στιγμιότυπο ενός open system-call επάνω σε μία συσκευή που έχει major_number που χειρίζεται το παρόν module.

Το σημαντικότερο έργο αυτής της συνάρτησης είναι η σύνδεση του οδηγού με την συσκευή. Αυτό γίνεται με την ανάθεση στους δείκτες sensor της δομής κατάστασης lunix_chrdev_state_struct με κατάλληλο minor_number (προσδιορίζεται με βάση την πράξη: $minor\_number\ div\ 8$ και την ανάθεση της ως δείκτη στον πίνακα lunix_sensors που έχει τις διευθύνσεις όλων των υπάρχοντων αισθητήρων της δομής lunix_sensor_struct όπως αυτές καταγράφονται από άλλο τμήμα του module, που δεν σχεδιάσαμε εμείς) το οποίο είναι καταγεγραμμένο μέσα στην δομή inode (δίνεται ως είσοδος σε αυτήν την συνάρτηση) και τον προσδιορισμό του τύπου της συσκευή με typecast, με βάση το enumeration lunix_msr_enum που βρίσκεται στο αρχείο επικεφαλίδων lunix.h. Ακόμα αρχικοποιούμε έναν σημαφόρο και έναν spinlock σε αυτή την συνάρτηση οι οποίοι θα μας φανούν χρήσιμοι στην συνέχεια. για τον συγχρονισμό σε περίπτωση hardware interrupt, αφού εκεί δεν υπάρχει κάποια διεργασία η οποία περιμένουμε να "ξυπνήσει" και η χρησιμοποίηση σημαφόρου δεν θα δούλευε. Τέλος, συνδέεται το πεδίο της δομή του πυρήνα file, private_data με

όλη την κατάσταση lunix_chrdev_state_struct προκειμένου να μπορούμε να αποθηκεύσουμε το "άνοιγμα" του αρχείου σε περίπτωση που το ζητήσει η read ή κάποια άλλη συνάρτηση.

1.3 Read

Η συνάρτηση lunix_chrdev_read υλοποιεί το "διάβασμα" μίας διεργασίας από την συσκευή χαρακτήρων. Αφού πάρουμε τα δεδομένα από το πεδίο private_data της δομής file που δίνεται ως είσοδος, χρησιμοποιούμε την συνάρτηση down_interruptible για να κλειδώσουμε τον semaphore προκειμένου να διαβάσουμε δεδομένα από τη συσκευή, συνάρτηση που επιλέγουμε γιατί θα θέλαμε χειρισμό σημάτων όπως του SIGINIT. Ο σημαφόρος επιλέγεται για την επικοινωνία του οδηγού με μία διεργασία χώρου χρήστη που κάνει, open στην αντίστοιχη συσκευή και θα χρησιμοποιηθεί στην περίπτωση που ένα ήδη forked παιδί της διεργασίας, πάει να διαβάσει στο κληρονομημένο ανοιχτό αρχείο (και με το κληρονομημένο lunix_chrdev_state_struct από τον πατέρα της). Έτσι επιλέγουμε να βάλουμε τη διεργασία για "ύπνο", ξέρωντας ότι "αξίζει τον κόπο" μιας και δεν θα ξυπνήσει τόσο άμεσα ώστε να την κρατάγαμε ξύπνια σε busy-wait.

Αν το όρισμα f_pos που δείχνει την θέση ανάγνωση στο ανοιχτό αρχείο είναι στο μηδέν, δηλαδή δεν έχουμε διαβάσει από το καινούργιο αρχείο τότε αν υπάρχει το αποκτούμε μέσω της update, ενω ο έλεγχος ύπαρξης γίνεται μέσω της refresh. Αν δεν υπάρχει κάτι καινούργιο να διαβάσουμε τοτε βάζουμε την διεργασία για ύπνο μέσω της wait_event_interruptible (ενώ έχουμε κάνει ήδη up τον semaphore) με βάση την συνθήκη refresh. (Τοποθετείται σε if statement για τον χειρισμό σημάτων όπως του SIGINIT)

Παρακάτω ελέγχεται ο αριθμός των δεδομένων που ζητείται από το πρόγραμμα χρήστη και βλέπουμε αν συμβαδίζει με τον αριθμό των δεδομένων που είναι αποθηκευμένα στους καταχωρητές που υλοποιούνται σε προηγούμενο βήμα της διαδικασίας. Σε περίπτωση που ζητούνται παραπάνω από αυτά που υπάρχουν επιστρέφεται ο συνολικός όγκος των δεδομένων που υπάρχουν. Η μεταφορά των δεδομένων στον χώρο χρήστη γίνεται μέσω της copy_to_user ενώ αν φτάσουμε στο τέλος buf_lim του πινακα buf_data της δομής lunix_chrdev_state_struct που περιέχει τα δεδομένα όπως δόθηκαν από την update γυρνάμε το f_pos στην αρχή (στο 0), ενώ στις άλλες περιπτώσεις το μετακινούμε στον επόμενο χαρακτήρα από αυτόν που διαβάσαμε.

1.4 Update

Η lunix_chrdev_state_update πέρνει δεδομενα από την δομή lunix_sensor_struct όπως αυτή ανατίθεται στο στάδιο της open. Η δομή χρησιμοποιείται και από άλλα τμήματα του κώδικα του module τα οποία της αναθέτουν τιμές στο πεδίο msr_data. Επειδή αυτό γίνεται μέσα από interrupt του hardware και όχι από διεργασία, δεν μπορούμε να χρησιμοποιήσουμε semaphore (καθώς δεν υπάρχει κάτι το οποίο θα κοιμηθεί) και γι'αυτό χρησιμοποιούμε spinlock. Αφου κλειδώσουμε φροντίζουμε την κατάλληλη μεταφορά των δεδομένων και την "μετάφρασή" τους για την αποστολή στον χρήστη με την βοήθεια των look_up tables, με βάση το type lunix_chrdev_state_struct αποθηκεύοντας τα ως string στο buf_data. Κάτι τέτοιο γίνεται με βάση το format των δεδομένων (εκφώνηση) (xxyyy -> xx.yyy). Επίσης ανανεώνουμε το buf_timestamp του lunix_chrdev_state_struct που χρειάζεται στη refresh. Η επεξεργασία των δεδομένων γίνεται εκτός του spinlock το οποίο κρατάμε όσο λιγότερο γίνεται.

1.5 Refresh

Η συνάρτηση που "βοηθά" την update είναι η συνάρτηση lunix_chrdev_state_needs_refresh η ελέγχει αν το timestamp της τελευταίας τιμής του sensor όπως ανανεώνεται από άλλο μέρος του module, μέσα από interrupts, είναι μεγαλύτερο από αυτό που έχει το αντίστοιχο lunix_chrdev_state_struct στο πεδίο buf_timestamp και συνεπώς αν υπάρχουν νέες τιμές προκειμένου να γίνει η κλήση της update.

1.6 Release και Destroy

Η συνάρτηση release απελευθερώνει τον "χώρο" των δεδομένων, αναιρώντας ότι έχει κάνει η open και η destroy καταστρέφει τις δεσμεύσεις που έχουμε κάνει, αναιρώντας ότι έχει κάνει η init. Η πρότη καλείται όταν κλείνουμε το αρχείο, ενώ η δεύτερη όταν κάνουμε rmmod.

Παρακάτω παρατίθεται ο κώδικας που αναπτύξαμε όπου μπορούν να φανούν τα στοιχεία των συναρτήσεων που αναφέρθηκαν.

lunix–chrdev.c

```c
/*
 * lunix-chrdev.c
 *
 * Implementation of character devices
 * for Lunix:TNG
 *
 * < Your name here >
 *
 */

#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/list.h>
#include <linux/cdev.h>
#include <linux/poll.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/ioctl.h>
#include <linux/types.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/mmzone.h>
#include <linux/vmalloc.h>
#include <linux/spinlock.h>
#include <linux/string.h>

#include "lunix.h"
#include "lunix-chrdev.h"
#include "lunix-lookup.h"

/*
 * Global data
 */
struct cdev lunix_chrdev_cdev;

/*
 * Just a quick [unlocked] check to see if the cached
 * chrdev state needs to be updated from sensor measurements.
 */
static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *state)
{
        struct lunix_sensor_struct *sensor;
        sensor = state->sensor;
        WARN_ON ( !(sensor = state->sensor));
        printk(KERN_DEBUG "Entered refresh\n");
        /* ? */
        if(state->sensor->msr_data[state->type]==NULL){
                return 0;
        }
        return (state->buf_timestamp < sensor->msr_data[state->type]->last_update); /* ? */
}

/*
 * Updates the cached state of a character device
 * based on sensor data. Must be called with the
 * character device state lock held.
 */
static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
{
        struct lunix_sensor_struct *sensor;
        uint32_t read;
        unsigned char A[20];
        long temp;
        unsigned long flag;

        debug("leaving\n");
        printk(KERN_DEBUG "update entered\n");
        sensor = state->sensor;

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

```
74          */
75          spin_lock_irqsave(&sensor->lock,flag);
76          printk(KERN_DEBUG "Enter update and locked \n");
77          /* ? */
78          if(sensor->msr_data[state->type]->values == NULL || !lunix_chrdev_state_needs_refresh(state
    ))){
79                  spin_unlock(&sensor->lock);
80                  return -EAGAIN;
81          }
82          else{
83                  read = sensor->msr_data[state->type]->values[0];
84                  state->buf_timestamp = sensor->msr_data[state->type]->last_update;
85          }
86          /* Why use spinlocks? See LDD3, p. 119 */
87
88          /*
89           * Any new data available?
90           */
91          spin_unlock_irqrestore(&sensor->lock,flag);
92          switch(state->type){
93                  case BATT :
94                          temp = lookup_voltage[(int)read];
95                          break;
96                  case TEMP :
97                          temp = lookup_temperature[(int)read];
98                          break;
99                  case LIGHT:
100                         temp = lookup_light[(int)read];
101                         break;
102                 default :
103                         return -EFAULT;
104         }
105
106         /* ? */
107         if(temp<0){
108                 sprintf(A,"%ld,%ld\n",temp/1000,(-temp)%1000);
109         }
110         else{
111                 sprintf(A,"%ld,%ld\n",temp/1000,(temp)%1000);
112         }
113         strcpy(state->buf_data,A);
114         state->buf_lim = (unsigned int)strlen(A);
115         /*
116          * Now we can take our time to format them,
117          * holding only the private state semaphore
118          */
119
120         /* ? */
121
122 //out:
123         debug("leaving\n");
124         return 0;
125 }
126
127 /*************************************
128  * Implementation of file operations
129  * for the Lunix character device
130  *************************************/
131
132 static int lunix_chrdev_open(struct inode *inode, struct file *filp)
133 {
134         /* Declarations */
135         /* ? */
136         int ret;
137         struct lunix_chrdev_state_struct *s = (struct lunix_chrdev_state_struct *)kmalloc(sizeof(
    struct lunix_chrdev_state_struct), GFP_KERNEL);
138
139         debug("entering\n");
140         printk(KERN_DEBUG "Open started\n");
141         ret = -ENODEV;
142         if ((ret = nonseekable_open(inode, filp)) < 0)
143                 goto out;
144
145         //my code
146         s->type=(enum lunix_msr_enum)(iminor(inode)%8);
147         printk(KERN_DEBUG "open : s->type = %d\n",s->type);
```

```
148         s->sensor = &lunix_sensors[iminor(inode)/8];
149         s->buf_timestamp=0;
150         sema_init(&s->lock,1);
151         spin_lock_init(&s->sensor->lock);
152         filp->private_data = s;
153         //end
154
155         /*
156          * Associate this open file with the relevant sensor based on
157          * the minor number of the device node [/dev/sensor<NO>-<TYPE>]
158          */
159
160
161         /* Allocate a new Lunix character device private state structure */
162         /* ? */
163 out:
164         debug("leaving, with ret = %d\n", ret);
165         return ret;
166 }
167
168 static int lunix_chrdev_release(struct inode *inode, struct file *filp)
169 {
170         kfree(filp->private_data);
171         return 0;
172 }
173
174 static long lunix_chrdev_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
175 {
176         /* Why? */
177         return -EINVAL;
178 }
179
180 static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf, size_t cnt, loff_t *f_pos)
181 {
182         ssize_t ret;
183
184         struct lunix_sensor_struct *sensor;
185         struct lunix_chrdev_state_struct *state;
186
187         state = filp->private_data;
188         WARN_ON(!state);
189
190         sensor = state->sensor;
191         WARN_ON(!sensor);
192
193         /* Lock? */
194         if (down_interruptible(&state->lock))
195                 return -ERESTARTSYS;
196         printk(KERN_DEBUG "locked!!! read\n");
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                printk(KERN_DEBUG "read from scratch\n");
204                 while (lunix_chrdev_state_update(state) == -EAGAIN) {
205                         /* ? */
206                         up(&state->lock);
207                         printk(KERN_DEBUG "no update\n");
208                         if(wait_event_interruptible(sensor->wq,lunix_chrdev_state_needs_refresh(
    state)))
209                                 return -ERESTARTSYS;
210                         if (down_interruptible(&state->lock))
211                                 return -ERESTARTSYS;
212                         /* The process needs to sleep */
213                         /* See LDD3, page 153 for a hint */
214                 }
215                 printk(KERN_DEBUG "READ SUCCESS .... \n");
216
217                 printk(KERN_DEBUG "Lets see: \n");
218                 printk(KERN_DEBUG "BUFF_DATA: %s \n", state->buf_data);
219         }
220         debug("Continue Reading\n");
221         if(((int)(*f_pos) + (int)cnt) <= state->buf_lim){
222                 int ll;
```

```
223                 printk(KERN_DEBUG "SECTOR 1 with f_pos = %d and buf_lim = %d\n",(int)*f_pos,state->
    buf_lim);
224                 if ((ll=copy_to_user(usrbuf, &state->buf_data[(int)*f_pos], cnt))!=0) {
225                         debug("What the fuck happened: EFAULT(1) : copy_to_user = %d",ll);
226                         printk(KERN_DEBUG "Was about to read < %d > data beggining from character <
     %c > of string < %s >, while *fpos = %d, and state->buf_lim=%d \n",(int)cnt,state->buf_data[(
    int)*f_pos],state->buf_data,(int)*f_pos,state->buf_lim);
227                         ret = -EFAULT;
228                         goto out;
229                 }
230                 *f_pos = *f_pos + cnt;
231                 if(*f_pos == state->buf_lim){
232                         *f_pos = 0;
233                 }
234                 ret = cnt;
235                 printk(KERN_DEBUG "exiting ... SECTOR 1 with ret = %d,cnt = %d\n",ret,cnt);
236         }
237         else{
238                 printk(KERN_DEBUG "SECTOR 2 with f_pos = %d and buf_lim = %d\n",*f_pos,state->
    buf_lim);
239                 int lll;
240                 if ((lll=copy_to_user(usrbuf, &state->buf_data[(int)*f_pos], (state->buf_lim - (int
    )*f_pos)))!=0){
241                         debug("What the fuck happened: EFAULT(2) : copy_to_user = %d",lll);
242                         printk(KERN_DEBUG "Was about to read < %d > data beggining from character <
     %c > of string < %s >, while *fpos = %d, and state->buf_lim=%d \n",(state->buf_lim - (int)*
    f_pos),state->buf_data[(int)*f_pos],state->buf_data,(int)*f_pos,state->buf_lim);
243                         ret = -EFAULT;
244                         goto out;
245                 }
246                 ret =state->buf_lim - (int)*f_pos;
247                 printk(KERN_DEBUG "exiting ... SECTOR 2 with ret = %d,buf_lim = %d, *f_pos = %d \n"
    ,ret,state->buf_lim,(int)*f_pos);
248                 *f_pos = 0;
249         }
250         /* End of file */
251         /* ? */
252
253         /* Determine the number of cached bytes to copy to userspace */
254         /* ? */
255
256         /* Auto-rewind on EOF mode? */
257         /* ? */
258 out:
259         /* Unlock? */
260         up(&state->lock);
261         return ret;
262 }
263
264 static int lunix_chrdev_mmap(struct file *filp, struct vm_area_struct *vma)
265 {
266         return -EINVAL;
267 }
268
269 static struct file_operations lunix_chrdev_fops =
270 {
271         .owner          = THIS_MODULE,
272         .open           = lunix_chrdev_open,
273         .release        = lunix_chrdev_release,
274         .read           = lunix_chrdev_read,
275         .unlocked_ioctl = lunix_chrdev_ioctl,
276         .mmap           = lunix_chrdev_mmap
277 };
278
279 int lunix_chrdev_init(void)
280 {
281         /*
282          * Register the character device with the kernel, asking for
283          * a range of minor numbers (number of sensors * 8 measurements / sensor)
284          * beginning with LINUX_CHRDEV_MAJOR:0
285          */
286         int ret;
287         dev_t dev_no;
288         unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;
289
290         debug("initializing character device\n");
```

```
291         cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
292         lunix_chrdev_cdev.owner = THIS_MODULE;
293 
294         dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
295         /* ? */
296         ret = register_chrdev_region(dev_no,lunix_minor_cnt,"lunix-tng");
297         /* register_chrdev_region(LOOK OK) */
298         if (ret < 0) {
299                 debug("failed to register region, ret = %d\n", ret);
300                 goto out;
301         }
302         /* ? */
303         ret = cdev_add(&lunix_chrdev_cdev,dev_no,lunix_minor_cnt);
304         /* cdev_add? */
305         if (ret < 0) {
306                 debug("failed to add character device\n");
307                 goto out_with_chrdev_region;
308         }
309         debug("completed successfully\n");
310         return 0;
311 
312 out_with_chrdev_region:
313         unregister_chrdev_region(dev_no, lunix_minor_cnt);
314 out:
315         return ret;
316 }
317 
318 void lunix_chrdev_destroy(void)
319 {
320         dev_t dev_no;
321         unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;
322 
323         debug("entering the destruction protzes\n");
324         dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
325         cdev_del(&lunix_chrdev_cdev);
326         unregister_chrdev_region(dev_no, lunix_minor_cnt);
327         debug("leaving\n");
328 }
```