Εργαστήριο Λειτουργικών Συστημάτων

Δεύτερη Άσκηση

Οδηγός Ασύρματου Δικτύου Αισθητήρων στο λειτουργικό σύστημα Linux

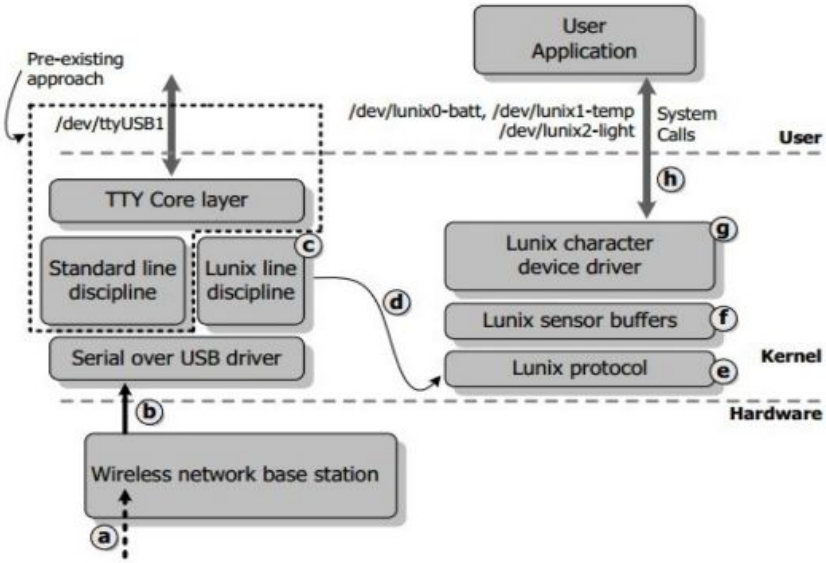
Ομάδα 31
Αρβανίτης Χρήστος 03114622
Μπαγάκης Εμμανουήλ 03114157

Εισαγωγή

Στη συγκεκριμένη εργαστηριακή άσκηση ζητείται η υλοποίηση ενός οδηγού συσκευής (driver) για ένα ασύρματο δίκτυο αισθητήρων υπό το λειτουργικό σύστημα Linux. Το δίκτυο αυτό αποτελείται από επιμέρους αισθητήρες σε συνδεσμολογία mesh ώστε όλοι οι αισθητήρες να αποδίδουν δεδομένα στον κεντρικό σταθμό βάσης. Κάθε ένας από αυτούς λαμβάνει μετρήσεις για τρία μεγέθη, τη φωτεινότητα, την τάση της μπαταρίας και τη θερμοκρασία περιβάλλοντος. Κατά την διάρκεια της ανάπτυξης του οδηγού, χρησιμοποιήσαμε εικονικό μηχάνημα QEMU-KVM το οποίο αρχικοποιείται με το βοηθητικό utopia.sh bash script που μας δίνεται. Ο σταθμός βάσης λαμβάνει τα πακέτα από τους αισθητήρες και τα αποστέλλει μέσω USB στο εργαστήριο. Η διασύνδεση αυτή υλοποιείται με κύκλωμα serial over USB.

Επειδή δεν είναι δυνατό να διαθέτει κάθε ένας από εμάς ένα τέτοιο δίκτυο αισθητήρων, στο εργαστήριο της σχολής υλοποιείται μια τέτοια διάταξη και τα αποτελέσματα όλων των μετρήσεων εμφανίζονται μαζικά στην εικονική σειριακή συσκευή /dev/ttyUSB1 του υπολογιστή του εργαστηρίου. Στον ίδιο υπολογιστή εγκαθίσταται TCP/IP server ο οποίος μεταδίδει τις μετρήσεις και στη συνέχεια, μέσω του utopia.sh, όλες αυτές ανακατευθύνονται στη S0 σειριακή θύρα του QEMU μηχανήματος (dev/ttyS0).
Τελικό ζητούμενο αποτελεί η υλοποίηση συστήματος εξαγωγής των δεδομένων από την σειριακή θύρα σε ένα σύνολο από επιμέρους συσκευές χαρακτήρων, για τις οποίες έχουμε ορίσει σύμβαση για την ονομασία τους, οι οποίες παράγονται από το βοηθητικό script lunix_dev_nodes.sh σύμφωνα με την ίδια σύμβαση.

Περιγραφή υλοποίησης



Σχήμα 1: Αρχιτεκτονική λογισμικού του υπό εξέταση συστήματος

Κλήση insmod

Έστω ότι έχουμε ήδη υλοποιήσει τον οδηγό, πρέπει να τον φορτώσουμε στον πυρήνα με την εντολή `insmod()`. Αυτή πυροδοτεί την εκτέλεση της συνάρτησης `lunix_module_init()` που υλοποιείται στο αρχείο `lunix-module.c` και εκτελείται λόγω της εκτέλεσης της εντολής `module_init(lunix_module_init);` η οποία ορίζει ποια συνάρτηση θα κληθεί κατά τη φόρτωση. Η `lunix_module_init` είναι υπεύθυνη για ορισμένες λειτουργίες αρχικοποίησης του οδηγού χαρακτήρων μας και των απαραίτητων δομών του. Συγκεκριμένα:

```
lunix_sensors = kzalloc(sizeof(*lunix_sensors) * lunix_sensor_cnt,
GFP_KERNEL);
```

Δέσμευση μνήμης για τις δομές sensor που αποθηκεύουν τα δεδομένα από το line discipline (στάδιο f σχήμα 1). Ο πίνακας `lunix_sensors[]` είναι global για κάθε αρχείο της υλοποίησής μας και δηλώνεται στο αρχείο επικεφαλίδας lunix.h .

```
lunix_protocol_init(&lunix_protocol_state);
```

Υλοποιείται στο αρχείο lunix_protocol.c και ευθύνεται για την αρχικοποίηση της μηχανής καταστάσεων του πρωτοκόλλου του οδηγού μας.

```
for (si_done = -1; si_done < lunix_sensor_cnt - 1; si_done++) {
        debug("initializing sensor %d\n", si_done + 1);
        ret = lunix_sensor_init(&lunix_sensors[si_done + 1]);
```

```
            debug("initialized sensor %d, ret = %d\n", si_done + 1, ret);
            if (ret < 0) {
                  goto out_with_sensors;
            }
      }

      /*
       * Initialize the Lunix line discipline
       */
      if ((ret = lunix_ldisc_init()) < 0)
            goto out_with_sensors;

      /*
       * Initialize the Lunix character device
       */
      if ((ret = lunix_chrdev_init()) < 0)
            goto out_with_ldisc;

      return 0;
```

Αρχικοποίηση τιμών των αισθητήρων (lunix_sensor_struct), της διάταξης γραμμής καθώς και κλήση της lunix_chrdev_init() η οποία υλοποιείται από εμάς στο αρχείο lunix-chrdev.c.

Στη συνέχεια, καλείται η `lunix_chrdev_init()`. Ο κώδικάς της ακολουθεί:

```
int lunix_chrdev_init(void)   // running when insmod
{
      /*
       * Register the character device with the kernel, asking for
       * a range of minor numbers (number of sensors * 8 measurements /
sensor)
       * beginning with LINUX_CHRDEV_MAJOR:0
       */
      int ret;
      dev_t dev_no;
      unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;

      debug("initializing character device\n");
      cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
      lunix_chrdev_cdev.owner = THIS_MODULE;

      dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);
```

```
    ret = register_chrdev_region(dev_no,lunix_minor_cnt,"lunix");
    if (ret < 0) {
       debug("failed to register region, ret = %d\n", ret);
       goto out;
    }

    ret = cdev_add(&lunix_chrdev_cdev,dev_no,lunix_minor_cnt);
    if (ret < 0) {
       debug("failed to add character device\n");
       goto out_with_chrdev_region;
    }
    debug("completed successfully\n");
    return 0;

out_with_chrdev_region:
    unregister_chrdev_region(dev_no, lunix_minor_cnt);
out:
    return ret;
}
```

Όσον αφορά τις συσκευές χαρακτήρων και την ονομασία τους, όπως αναφέρθηκε και παραπάνω το script lunix_dev_nodes.sh είναι υπεύθυνο για την δημιουργία των αρχείων που θα χρησιμοποιήσουμε για την ανάγνωση των μετρήσεων. Πιο συγκεκριμένα για κάθε sensor (16 στο σύνολο), δημιουργούνται τρία αρχεία τα οποία αναφέρονται στη μέτρηση μπαταρίας,θερμοκρασίας και φωτεινότητας αντίστοιχα. Το identifier των αρχείων αυτών είναι το major και το minor number τους. Το major number χρησιμοποιείται από τον kernel για να προσδιορίσει τον οδηγό συσκευής που αφορά ο κόμβος αυτός. Επιλέγουμε τον major number 60 ο οποίος είναι δεσμευμένος για πειραματική χρήση. Για τον minor number επιλέγουμε την τιμή : minor = αισθητήρας* 8 + μέτρηση, όπου μέτρηση = τάση μπαταρίας(0), θερμοκρασία(1) και φωτεινότητα(2).

Πρώτα, μέσω της κλήσης cdev_init() αρχικοποιούμε τη συσκευή χαρακτήρων με τη αντίστοιχη δομή των file operations. Στη συνέχεια δεσμεύουμε στον πυρήνα ένα range απο minor numbers, συγκεκριμένα απαιτούνται 16 αισθητήρες με οχτώ μετρήσεις ο καθένας οπότε 128 minor numbers. Τέλος ενεργοποιούμε αυτές τις συσκευές μέσω της cdev_add() κλήσης.

Επιλέγουμε να παρουσιάσουμε την υλοποίηση με τη σειρά που καλούνται οι επιμέρους συναρτήσεις κατά τη κλήση της εντολής cat σε ένα αρχείο χαρακτήρων.

Strace κλήσης cat σε /dev/lunix0-batt:

```
execve("/bin/cat", ["cat", "/dev/lunix0-temp"], [/* 24 vars */]) = 0
brk(0)                                  = 0x2328000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fe6143ab000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=29532, ...}) = 0
mmap(NULL, 29532, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe6143a3000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\34\2\0\0\0\0\0"..., 832) =
832
fstat(3, {st_mode=S_IFREG|0755, st_size=1738176, ...}) = 0
mmap(NULL, 3844640, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fe613de2000
mprotect(0x7fe613f83000, 2097152, PROT_NONE) = 0
mmap(0x7fe614183000,              24576,              PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a1000) = 0x7fe614183000
mmap(0x7fe614189000,              14880,              PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe614189000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fe6143a2000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fe6143a1000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fe6143a0000
arch_prctl(ARCH_SET_FS, 0x7fe6143a1700) = 0
mprotect(0x7fe614183000, 16384, PROT_READ) = 0
mprotect(0x60b000, 4096, PROT_READ)     = 0
mprotect(0x7fe6143ad000, 4096, PROT_READ) = 0
munmap(0x7fe6143a3000, 29532)           = 0
brk(0)                                  = 0x2328000
brk(0x2349000)                          = 0x2349000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=1859120, ...}) = 0
```

```
mmap(NULL, 1859120, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe6141da000
close(3)                                = 0
fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(136, 1), ...}) = 0
open("/dev/lunix0-temp", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFCHR|0644, st_rdev=makedev(60, 1), ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7fe6141b8000
read(3, "24.198\n", 131072)             = 7
write(1, "24.198\n", 724.198
)                 = 7
read(3, "24.198\n", 131072)             = 7
write(1, "24.198\n", 724.198
)                 = 7
read(3, ^CProcess 2925 detached
 <detached ...>
```

Κλήση open σε ειδικό αρχείο dev/lunix{x}-{batt,temp,light}

Κατά το άνοιγμα ενός αρχείου dev/lunix{x}-{batt,temp,light} καλείται η αντίστοιχή, από το struct lunix_chrdev_fops, συνάρτηση, δηλαδή η lunix_chrdev_open(). Ο κώδικάς της δίνεται παρακάτω:

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
    /* Declarations */
    /* ? */
    int ret,minor;
    //    struct lunix_sensor_struct *sensor; Have to associate inode with
the required sensor given by the filp parameter
    struct lunix_sensor_struct *sensor;
    struct lunix_chrdev_state_struct *state;

    //Also point state from private_data for future reference without all
those container_of() calls
    //maybe in future versions check flags for read only access or any
other thing like raw or cooked data

    minor = iminor(inode);
    sensor = &(lunix_sensors[minor/8]);
```

```
    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>]
     * from inode go to major minor from minor get type and put it in new
state struct
     */

    debug("Got minor = %d for BATT measurement\n",minor);

    //allocate character device private state structure:
lunix_chrdev_state_struct. CONNECT THE SENSOR FIELD

    state = kzalloc(sizeof(struct lunix_chrdev_state_struct), GFP_KERNEL);
//allocate kernel ram
    if (!state) {
       ret = -22;
       goto out;
    }
    state->type = minor%8;
    state->sensor = sensor; //Allocate chrdev lunix sensor with sensor
state struct
    ret = sema_init(&(state->lock),1); //initialize state semaphore
    if (ret) {
       goto out;
    }
    //Other fields of state are initialized by other functions

    filp->private_data = state; /* for other methods without calling
container_of() or iminor()*/
out:
    debug("leaving, with ret = %d\n", ret);
    return ret;
}
```

Όπως φαίνεται παραπάνω, η συγκεκριμένη συνάρτηση ευθύνεται για τα εξής:

- Δέσμευση χώρου μνήμης για το lunix_chrdev_state_struct που αποτελεί buffer με την τελευταία μέτρηση και αντιστοιχεί σε κάθε νέο άνοιγμα αρχείου προκειμένου να καθίσταται δυνατή η προσπέλαση του ίδιου αισθητήρα από διαφορετικές διεργασίες.

```
kzalloc(sizeof(struct lunix_chrdev_state_struct), GFP_KERNEL);
```

- Αρχικοποίηση τιμών του state σχετικών με το είδος του αισθητήρα, του σημαφόρου για παράλληλη πρόσβαση στο struct και αντιστοίχηση του με τον κατάλληλο sensor buffer.

```
state->type = minor%8;
state->sensor = sensor; //Allocate chrdev lunix sensor with sensor state struct
sema_init(&(state->lock),1); //initialize state semaphore
```

- Αντιστοίχηση του ανοιχτού αρχείου που δίνεται από τον δείκτη παράμετρο *filp με το κατάλληλο state (και κατ επέκταση sensor) μέσω του πεδίου private_data για μελλοντική χρήση χωρίς να αναζητάμε τους major και minor numbers.

```
filp->private_data = state; /* for other methods without calling container_of() or iminor()*/
```

Σε αυτή τη στιγμή μπορούμε να αναφέρουμε και τη release συνάρτηση η οποία απελευθερώνει απλά το state αφού για τα υπόλοιπα αναλαμβάνουν άλλες συναρτήσεις του lunix-module.c.

```
xstatic int lunix_chrdev_release(struct inode *inode, struct file *filp)
{    struct lunix_chrdev_state_struct *state;
    state = filp->private_data;
    debug("releasing chrdev %d of type %d\n",iminor(inode) , state->type);
    kfree(filp->private_data);
    return 0;
}
```

Κλήση read()

Στη συνέχεια καλείται η read() επί του αρχείου, η οποία μέσω της δομής lunix_chrdev_fops αντιστοιχεί στην συνάρτηση lunix_chrdev_read(). Η συγκεκριμένη συνάρτηση αποτελεί τον κύριο κορμό της υλοποίησής μας. Ο κώδικάς της δίνεται παρακάτω.

```
static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf,
size_t cnt, loff_t *f_pos)
{
    ssize_t ret;

    struct lunix_sensor_struct *sensor;
    struct lunix_chrdev_state_struct *state;
```

```
    state = filp->private_data;
    WARN_ON(!state);
    int cur_bytes;

    sensor = state->sensor;
    WARN_ON(!sensor);

    enum    lunix_msr_enum sensor_type = state->type;
    debug("Sensor type %d\n",sensor_type);
    debug("RAW Sensor data= %lld\n",
sensor->msr_data[sensor_type]->values[0]);

    if(down_interruptible(&(state->lock))){
       ret = -1;
       goto out;
    };

       if (*f_pos == 0) {
                  while (lunix_chrdev_state_update(state) == -EAGAIN) {
                        up(&(state->lock));
                        //IF wait == 0 or wait == -RESTARTSYS (C like
0=false everything else true)
                        if (wait_event_interruptible(sensor->wq,
(lunix_chrdev_state_needs_refresh(state)))){
                              debug("CAUGHT SIGNAL\n");
                              return -ERESTARTSYS;
                        }

                        /* ? */
                        /* The process needs to sleep */
                        /* See LDD3, page 153 for a hint */

down_interruptible(&(state->lock)); //
                  }
            }

            cur_bytes = state->buf_lim - *f_pos; //bytes that are
available for reading in the same measurement

            if (cnt <= cur_bytes){
                  ret = cnt;
```

```
		}
		else{
			ret = cur_bytes;
		}

		if (copy_to_user(usrbuf, state->buf_data + *f_pos, ret)) {
				up(&(state->lock));
				return -EFAULT;
			}

		if (ret == cur_bytes)
		{
			*f_pos = 0;
		}
		else{
			*f_pos += cnt; // update fpos only if copy to user was
succesful
		}


out:
	 up(&(state->lock));
	/* Unlock? */
	return ret;
}
```

Αρχικά, λαμβάνουμε τον state buffer από το πεδίο private_data του filp και από αυτόν τον αντίστοιχό sensor struct και τον τύπο του αισθητήρα. Στη συνέχεια, φροντίζουμε να λάβουμε το lock του state buffer ώστε να τον τροποποιήσουμε κατάλληλα με τη νεότερη τιμή αν αυτή υπάρχει με χρήση της down_interruptible() ώστε να μπορεί να ξυπνήσει η διεργασία και με σήματα. Στη συνέχεια ελέγχουμε τη θέση του δείκτη f_pos ώστε να γνωρίζουμε σε ποιο σημείο σταμάτησε η προηγούμενη μέτρηση. Αν αυτή δεν είναι μηδενική, τότε έχουμε ήδη δεδομένα στο πεδίο state->buf_data οπότε και επιστρέφουμε αυτά φροντίζοντας να γνωρίζουμε το κατάλληλο offset του string από το οποίο θα διαβάσουμε. Αν πάλι ξεκινάμε καινούρια μέτρηση φροντίζουμε να ελέγξουμε για το αν ο state buffer έχει τα πιο πρόσφατα δεδομένα. Συγκεκριμένα:
Χρησιμοποιούμε τη lunix_chrdev_state_update() η οποία αν δεν υπάρχουν νέα δεδομένα, οπότε και δεν έγινε κάποια αλλαγή στον state buffer επιστρέφει -EAGAIN. Περιμένοντας δεδομένα, θα πρέπει να κοιμίζουμε τη διεργασία που ζητά δεδομένα μη διαθέσιμα, οπότε και χρησιμοποιούμε την εξής κλήση:

```
if (wait_event_interruptible(sensor->wq,
```

```
(lunix_chrdev_state_needs_refresh(state))){
                                debug("CAUGHT SIGNAL\n");
                                return -ERESTARTSYS;
                        }
```

Αναλυτικά, τοποθετούμε στην ουρά αναμονής κάθε αισθητήρα τη τρέχουσα διεργασία μέχρι να ικανοποιηθεί η λογική έκφραση της δεύτερης παραμέτρου. Το lunix_chrdev_state_needs_refresh(state) αποτελεί ένα shortcut που ελέγχει, όπως θα περιγράψουμε και στη συνέχεια, αν χρειάζεται να γίνει update, δηλαδή αν ο sensor buffer έχει πιο πρόσφατα δεδομένα από τον state buffer. Όταν τελικά βρεθούν νέα δεδομένα, τότε επιστρέφουμε -ERESTARTSYS ώστε να επανεκινήσουμε την read() και να λάβουμε τα νέα δεδομένα. Χρησιμοποιούμε την interruptible έκδοση της εντολής ώστε να έχουμε τη δυνατότητα αποστολής σημάτων κατά τη διάρκεια του waiting της διεργασίας (π.χ. Ctrl + C για τερματισμό της διεργασίας).

Αφού τελικά λάβουμε τα νεότερα δεδομένα, πλέον υπολογίζουμε το offset από το οποίο θα ξεκινήσουμε την επιστροφή χαρακτήρων του state->buf_data και στη συνέχεια εκτελούμε τη copy_to_user() ώστε να περάσουμε σε userspace χώρο μνήμης τους επιθυμητούς χαρακτήρες. Φροντίζουμε να δώσουμε τη σωστή τιμή στο *f_pos ώστε επόμενες αναγνώσεις παιδιών που μοιράζονται το ίδιο file struct να ξεκινήσουν από το σωστό σημείο ενώ επιστρέφουμε τον αριθμό των bytes που τελικά διαβάστηκαν (ret).

Κλήση lunix_chrdev_state_update()

Η lunix_chrdev_read καλεί την lunix_chrdev_state_update η οποία είναι υπεύθυνη για την ενημέρωση των τιμών των μετρήσεων από τους αισθητήρες καθώς και την μορφοποίηση των δεδομένων έτσι ώστε να μπορούν να διαβάζονται σε μορφή χαρακτήρων. Επειδή ακριβώς δύναται να αλλάζει την κατάσταση των buffers οφείλουμε να φροντίζουμε τον αποκλεισμό οποιουδήποτε race condition. Για αυτόν τον λόγο χρησιμοποιούμε εντός της συνάρτησης το κλείδωμα για το sensor struct ενώ κατά τη καλούσα read() το κλείδωμα για το state buffer(). Σχετικά με το κλείδωμα, υλοποιούμε το πρόγραμμα χρησιμοποιώντας κλήσεις spin_*. Αυτό θα μπορούσε να δημιουργήσει ζητήματα σε περίπτωση που έρθουν δύο συνεχόμενες διακοπές από το υλικό χωρίς να έχει τερματίσει η πρώτη. Το γεγονός αυτό αντιμετωπίζεται με τη χρήση κλήσεων spin_lock_irqsave και spin_unlock_irqsave ώστε να μην κολλήσει ποτέ το πρόγραμμα με interrupt που δεν μπορεί να εξυπηρετηθεί λόγω των spinlocks.
Αν η συνάρτηση κάνει πράγματι update την τιμή της μέτρησης, επιστρέφει 1 αλλιώς αν δεν χρειάζεται update, επιστρέφει -EAGAIN.
Ακολουθεί ο κώδικας της lunix_chrdev_update():

```
static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct
```

```
*state)
{
    struct lunix_sensor_struct *sensor;
    uint32_t sens_value,last_update;
    int aker = 0,dekad = 0;
    long human_value;

    sensor = state->sensor;
    enum    lunix_msr_enum sensor_type = state->type;

    if (lunix_chrdev_state_needs_refresh(state)) {

       spin_lock(&sensor->lock);

       sens_value = sensor->msr_data[sensor_type]->values[0];
       last_update = sensor->msr_data[sensor_type]->last_update;

       //format data and pass to state
       //first the timestamps
       state->buf_timestamp = sensor->msr_data[sensor_type]->last_update;
       spin_unlock(&sensor->lock);

       //Then buff limit and buffer value
       switch (sensor_type) {
             case 0:
                   human_value = lookup_voltage[sens_value];
             break;
             case 1:
                   human_value = lookup_temperature[sens_value];
             break;
             case 2:
                   human_value = lookup_light[sens_value];
             break;
       }
       debug("SENSOR VAL = %ld\n",human_value);
       //get integer value
       aker = human_value/1000;
       // get decimal value
       dekad = human_value%1000;

       state->buf_lim = digit_num(aker) + digit_num(dekad) + 2; // +2 for
newline and point
```

```
        debug("STATE UPDATE limit = %d\n",state->buf_lim);
        //then long to string

 snprintf(state->buf_data,sizeof(state->buf_data),"%d.%d\n",aker,dekad);
        debug("STATE UPDATE string = %s",state->buf_data);

    }
    else{
        debug("NO UPDATE STATE\n");
        return -EAGAIN;
    }

    return 1;
 }
```

Κρατώντας το state κλείδωμα, αρχικά ελέγχουμε μέσω της συνάρτησης lunix_chrdev_state_needs_refresh() αν πράγματι υπάρχει κάποια καινούργια μέτρηση στον αισθητήρα. Ο τρόπος με τον οποίο γίνεται αυτός ο έλεγχος αναλύεται παρακάτω. Δεδομένου ότι έχουμε μία νέα μέτρηση, λαμβάνουμε το sensor κλείδωμα και μπαίνουμε σε κομμάτι κρίσιμου κώδικα (spin_lock(&sensor->lock)). Στη συνέχεια αποθηκεύουμε τις τρέχουσες τιμές του αισθητήρα για τον οποίο έχουμε νέα μέτρηση, καθώς και το πότε έγινε η τελευταία μέτρηση :

```
 sens_value = sensor->msr_data[sensor_type]->values[0];
 last_update = sensor->msr_data[sensor_type]->last_update;
```

Έπειτα η διαδικασία μορφοποίησης των δεδομένων και αποθήκευσης τους στο state ξεκινά με το timestamps της τελευταίας ενημέρωσης :

```
 state->buf_timestamp = sensor->msr_data[sensor_type]->last_update;
```

Στην συνέχεια φεύγουμε από το κρίσιμο κομμάτι κώδικα οπότε καλούμε τη spin_unlock(&sensor->lock); και ανάλογα με το είδος της μέτρησης που πήραμε, ανατρέχουμε στο αντίστοιχο lookup_table. Τα lookup_tables που ορίζονται στο lunix-lookup.h χρησιμοποιούνται προκειμένου να μετατρέψουμε τις τιμές των μετρήσεων σε human readable μορφή. Αφού λάβουμε την τιμή την διαχωρίζουμε το ακέραιο και το δεκαδικό μέρος προσθέτοντας την υποδιαστολή. Στη συνέχεια ενημερώνουμε την τιμή του buf_lim ως το πλήθος των ψηφίων της μέτρησης + 2 (υποδιαστολή και newline) και τελικά αποθηκεύουμε τη συμβολοσειρά στο state->buf_data (π.χ. 23.451). Το πλήθος των ψηφίων της μέτρησης υπολογίζεται από την παρακάνω συνάρτηση :

```
 static int digit_num(int inp){
```

```
    int n = inp;
    int count = 0;
      do
      {
       // n = n/10
      n /= 10;
      ++count;
      } while (n !=0);
    return count;
}
```

Κλήση lunix_chrdev_state_needs_refresh()

Όπως αναφέραμε και παραπάνω η συνάρτηση αυτή καλείται από την state_update και σκοπό έχει να αποφανθεί αν υπάρχει καινούργια μέτρηση για τον αισθητήρα. Για να το πετύχει αυτό συγκρίνουμε τα δύο πεδία timestamps sensor->msr_data[sensor_type]->last_update και state->buf_timestamp. Αν το last_update είναι μεγαλύτερο, σημαίνει ότι έχω νέα μέτρηση και επομένως η state_update εκτελείται κανονικά. Σε αντίθετη περίπτωση επιστρέφει με τιμή -EAGAIN. Τα πεδία αυτά περιέχουν τη πληροφορία για την ώρα μέτρησης της κάθε μέτρησης σε μία πολύ συνηθισμένη αριθμητική μορφή για το λειτουργικό σύστημα Linux, το Linux epoch, δηλαδή την διαφορά σε seconds από την 1η Ιανουαρίου του 1970. Η υλοποίηση της ακολουθεί:

```
/*
 * 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));


    enum    lunix_msr_enum sensor_type;
    sensor_type = state->type;

    //check timestamps
    //in state struct = uint32_t buf_timestamp;
    //in sensor struct = uint32_t msr_data[type of sensor
```

```
enum]->last_update
    debug("COMPARISON %ld -
%ld\n",sensor->msr_data[sensor_type]->last_update,state->buf_timestamp);
    return (sensor->msr_data[sensor_type]->last_update >
state->buf_timestamp);
}
```

Κλήση rmmod

Κατά τη κλήση του rmmod εκτελείται μία σειρά από συναρτήσεις. Αναλυτικότερα, καλείται η lunix_module_cleanup() του αρχείου lunix-module.c η οποία καλεί την υλοποιημένη στο lunix-chrdev.c lunix_chrdev_destroy(); η οποία καθαριζει όποιο υπόλειμμα στη μνήμη απο το kernel module μας.

Δοκιμή της υλοποίησής μας

Προκειμένου να ελέγξουμε την σωστή λειτουργία του driver μας, υλοποιήσαμε ένα πρόγραμμα του οποίου ο κώδικας φαίνεται παρακάτω:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>

int  main(int argc , char** argv){

     int fd = open("/dev/lunix0-temp", O_RDONLY);
     char buf[5];
     char buf2[5];
    if (fork()==0){
    //child
    sleep(2);
    int numread = read(fd,buf2,3);
    write(1,buf2,numread);
    printf("\n");
    }
    else {
    int numread =  read(fd,buf,2);
```

```
    write(1,buf,numread);
    printf("\n");
    wait(NULL);
    }
    close(fd);

    return 0;
}
```

Το πρόγραμμα αρχικά κάνει open σε κάποιο είδος μέτρησης (στη συγκεκριμένη στη θερμοκρασία του πρώτου αισθητήρα). Στην συνέχεια με την βοήθεια της fork(); δημιουργείται μία ακόμα διεργασία. Το παιδί αρχικά περιμένει 2” και στη συνέχεια διαβάζει από το αρχείο 3 χαρακτήρες , ενώ ο γονιός διαβάζει δύο χαρακτήρες και περιμένει να τερματίσει το παιδί.

Example output :

```
root@utopia:/home/user/lunixcode#  ./forkstress
26
.71
root@utopia:/home/user/lunixcode# ./forkstress
26
.81
root@utopia:/home/user/lunixcode# ./forkstress
26
 .91
```