Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλ. Μηχανικών & Μηχανικών Υπολογιστών
Εργαστήριο Λειτουργικών Συστημάτων
Χειμερινό Εξάμηνο 2022-2023

Ομάδα oslab30
Ιωάννης Δρέσσος - 03119608

Αναφορά 2ης Εργαστηριακής Άσκησης (lunix:TNG)

Μας δίνεται ο σκελετός ενός οδηγού συσκευής για Linux, συγκεκριμένα για ασύρματο δίκτυο αισθητήρων του οποίου ο δέκτης είναι συνδεδεμένος με TTY (Serial) over USB στο σύστημα μας. Καλούμαστε να συμπληρώσουμε κώδικα ώστε να εξασφαλίσουμε την σωστή και ασφαλή λειτουργία του οδηγού.

Ο κώδικας που συμπληρώνουμε αφορά το μέρος συσκευής χαρακτήρων του οδηγού και βρίσκεται στα αρχεία lunix-chrdev.h και lunix-chrdev.c. Στην αναφορά περιλαμβάνονται μόνο οι αλλαγές που έγιναν και όχι ολόκληρος ο κώδικας.

Τα παρακάτω αποσπάσματα εισήχθησαν στο αρχείο **lunix-chrdev.c**:

```
static int lunix_chrdev_state_needs_refresh(struct
lunix_chrdev_state_struct *state)
{
      ...

      if(state->buf_timestamp <
(sensor->msr_data[state->type]->last_update)) {
            debug("update needed\n");
            return 1;
      }
      else {
            debug("no update needed\n");
            return 0;
      }
}
```

Αλλαγές στην συνάρτηση **lunix_chrdev_state_needs_refresh()**:

- Προστέθηκε κώδικας σύγκρισης της χρονοσφραγίδας της δομής κατάστασης της συσκευής χαρακτήρων (character device state struct) σε σχέση με την τελευταία ενημέρωση των μετρήσεων του σχετικού αισθητήρα, με σκοπό να προσδιορίσουμε εάν χρειάζεται ενημέρωση η δομή κατάστασης.

```
static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct
*state)
{
	struct lunix_sensor_struct *sensor;
	uint32_t new_value;
	long parsed_value;
	int int_part, dec_part;

	debug("leaving\n");

	...

	if(!lunix_chrdev_state_needs_refresh(state)) return -EAGAIN;

	...

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

	spin_lock_irq(&sensor->lock);
	new_value = sensor->msr_data[state->type]->values[0];
	spin_unlock_irq(&sensor->lock);

	...

	switch(state->type) {
		case BATT:
			parsed_value = lookup_voltage[new_value];
			break;
		case TEMP:
			parsed_value = lookup_temperature[new_value];
```

```
                break;
        case LIGHT:
                parsed_value = lookup_light[new_value];
                break;
        default:
                debug("unknown measurement type");
    }

    debug("parsed_value = %ld\n", parsed_value);

    int_part = parsed_value / 1000;
    dec_part = parsed_value % 1000;

    debug("actual_value = %d.%d\n", int_part, dec_part);

    state->buf_lim = snprintf(state->buf_data, LUNIX_CHRDEV_BUFSZ,
"%d.%d ", int_part, dec_part);

    if(state->buf_lim >= LUNIX_CHRDEV_BUFSZ) debug("snprintf
truncated string\n");

    state->buf_timestamp =
sensor->msr_data[state->type]->last_update;

    ...
}
```

Αλλαγές στην συνάρτηση **lunix_chrdev_state_update()**:

- Σε πρώτη φάση έγιναν δηλώσεις μεταβλητών που θα χρησιμοποιηθούν παρακάτω στον κώδικα.
- Επειτα, ελέγχουμε εάν χρειάζεται ανανέωση η δομή κατάστασης της συσκευής χαρακτήρων.
- Στο κύριο σημείο της συνάρτησης, αντλούμε τα δεδομένα απευθείας απο τη δομή αισθητήρα ενώ βρισκόμαστε εντός περιστρεφόμενου κλειδώματος.
- Αφού έχουμε τη νέα τιμή της μέτρησης, την μετατρέπουμε μορφή αναγνώσιμη από άνθρωπο χρησιμοποιώντας τους πίνακες αναζήτησης που μας δίνονται, αφού προσδιορίσουμε το είδος της μέτρησης.

- Τέλος, αποθηκεύουμε την τελική τιμή στον buffer και ανανεώνουμε την χρονοσφραγίδα της δομής κατάστασης βάσει τον χρόνο τελευταίας ενημέρωσης των μετρήσεων από τη δομή του αισθητήρα.

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
	...

	struct lunix_chrdev_state_struct *new_state;
	int msr_type;
	int sensor_id;

	...

	sensor_id = iminor(inode) >> 3;
	debug("sensor_id  = %d\n", sensor_id);
	msr_type = iminor(inode) % 8;
	debug("msr_type = %d\n", msr_type);
	if(msr_type >= N_LUNIX_MSR) { // Invalid measurement type
		ret = -ENODEV;
		goto out;
	}

	...

	new_state = kmalloc(sizeof(struct lunix_chrdev_state_struct),
GFP_KERNEL);

	if(!new_state) {
		printk(KERN_ERR "lunix_chrdev_open: Could not allocate
private state structure\n");
		ret = -EFAULT;
		goto out;
	}

	new_state->type = msr_type;
	new_state->sensor = &lunix_sensors[sensor_id];
	new_state->buf_lim = 0;
```

```
	new_state->buf_timestamp = 0;

	sema_init(&new_state->lock, 1);

	filp->private_data = new_state;

	...
}
```

Αλλαγές στην συνάρτηση **lunix_chrdev_open()**:

- Ως γνωστόν στην αρχή έγιναν οι δηλώσεις μεταβλητών που θα χρησιμοποιηθούν.
- Ξεκινάμε με τον προσδιορισμό του αισθητήρα και τύπου μέτρησης χρησιμοποιώντας το minor number του inode από τα ορίσματα της συνάρτησης.
- Δημιουργούμε χώρο για τη νέα δομή κατάστασης σε χώρο πυρήνα. Περιλαμβάνεται έλεγχος αποτυχίας για την συγκεκριμένη εντολή.
- Τέλος, αρχικοποιούμε τα πεδία και τον σημαφόρο της δομής και την κάνουμε προσβάσιμη από το πεδίο private_data της δομής αρχείου από τα ορίσματα της συνάρτησης.

```
static int lunix_chrdev_release(struct inode *inode, struct file
*filp)
{
	kfree(filp->private_data);

	return 0;
}
```

Αλλαγές στην συνάρτηση **lunix_chrdev_release()**:

- Προστέθηκε κώδικας για να ελευθερώσει την μνήμη που είναι δεσμευμένη για την δομή κατάστασης που αντιστοιχεί στο αρχείο που δίνεται σαν όρισμα στην συνάρτηση.

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

      int temp;
      ...

      if(down_interruptible(&state->lock)) return -ERESTARTSYS;

      ...

      if (*f_pos == 0) {
            while (lunix_chrdev_state_update(state) == -EAGAIN) {
                  up(&state->lock);

                  debug("no new data, sleeping..\n");
                  if(wait_event_interruptible(sensor->wq,
lunix_chrdev_state_needs_refresh(state))) {
                        debug("interrupted\n");
                        return -ERESTARTSYS;
                  }
                  debug("state updated, woken up\n");

                  if(down_interruptible(&state->lock)) return
-ERESTARTSYS;
            }
      }

      if(state->buf_lim == 0) {
ret = 0;
                  goto out;
      }

      ...

      temp = state->buf_lim - *f_pos;

      if(temp < cnt) cnt = temp;
```

```
      debug("copying data to user space..\n");
      if(copy_to_user(usrbuf, state->buf_data + *f_pos, cnt)) {
                ret = -EFAULT;
                goto out;
      }
      debug("copy_to_user succeeded\n");

      ...

      *f_pos += cnt;

      if(*f_pos >= state->buf_lim) {
            *f_pos = 0;
            ret = cnt;
            goto out;
      }

      ret = cnt;

out:
      up(&state->lock);

      return ret;
}
```

Αλλαγές στην συνάρτηση **lunix_chrdev_read()**:

- Πάνω πάνω ορίζουμε την μεταβλητή temp που θα χρησιμοποιηθεί παρακάτω.
- Κλειδώνουμε τον σημαφόρο της δομής κατάστασης.
- Αναμένουμε νέα δεδομένα από τον αισθητήρα εάν δεν υπάρχουν ήδη στον buffer. Προσέχουμε να επιτρέπουμε διακοπές από σήματα στον σημαφόρο μας και κατα την αναμονή ανανέωσης της δομής κατάστασης.
- Πραγματοποιούνται επιπλέον έλεγχοι για EOF, προσδιορίζουμε το μήκος δεδομένων που θα αντιγραφεί σε χώρο χρήστη από χώρο πυρήνα.
- Αντιγράφουμε τα δεδομένα σε χώρο χρήστη, ελέγχοντας για σφάλματα.
- Ανανεώνουμε την τιμή του δείκτη θέσης αρχείου και την επαναφέρουμε εάν χρειάζεται.
- Ξεκλειδώνουμε τον σημαφόρο και επιστρέφουμε.

```
int lunix_chrdev_init(void)
{
      ...

      ret = register_chrdev_region(dev_no, lunix_minor_cnt,
"lunix:TNG");

      ...

      ret = cdev_add(&lunix_chrdev_cdev, dev_no, lunix_minor_cnt);

      ...
}
```

Αλλαγές στην συνάρτηση **lunix_chrdev_init()**:

- Κάνουμε εγγραφή της περιοχής τιμών της συσκευής χαρακτήρων μας.
- Προσθέτουμε την συσκευή χαρακτήρα στο σύστημα.