National Technical University of Athens
School of Electrical and Computer Engineering
Operating Systems Lab
Winter Semester 2022-2023

Team oslab30
Ioannis Dressos - 03119608

Report of 2nd Lab Exercise (lunix:TNG)

We are given the codebase of a Linux device driver for a wireless sensor network, the station/receiver of which is connected to our computer through TTY (Serial) over USB. We must add code to the kernel program to achieve safe and proper driver functionality.

The code we must add concerns the character device part of the driver and is located in the files lunix-chrdev.h and lunix-chrdev.c. Only changes made to the files are included in this report.

The following snippets were added to the **lunix-chrdev.c** file:

```
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;
	}
}
```

Changes made to the **lunix_chrdev_state_needs_refresh()** function:

- Added code to compare the timestamp of the character device state struct to the last update time of the measurements of the respective sensor, in order to determine if the state struct must be updated.

```
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;

      ...
}
```

Changes made to the **lunix_chrdev_state_update()** function:

- First, variable declarations were made to use in the code below.
- Then we check if the character device state struct needs updating.
- In the main part of the function, we fetch the measurement data value from the sensor struct while inside a spinlock.
- Now that we have the new measurement value we parse it, converting it to human-readable form by using the lookup tables provided, after we determine the type of measurement.
- Finally we save the canonical measurement value to the buffer and we update the timestamp of the state struct to be the last update time of the measurements from the sensor struct.

```
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;
```

```
      ...
}
```

Changes made to the **lunix_chrdev_open()** function:

- Again, first come variable declarations.
- Using the inode minor number from the function parameters, we determine the sensor and the measurement type.
- We allocate memory for the new state struct in kernel space. Checks for allocation failure are included.
- Finally, we initialize the state fields and semaphore, and we point the private_data field from the file struct in the function parameters to the new state struct.

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

      return 0;
}
```

Changes made to the **lunix_chrdev_release()** function:

- Added code to actually free the memory allocated for the character device state struct.

```
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;
}
```

Changes made to the **lunix_chrdev_read()** function:

- Declare the variable *temp*, which will be used below.
- Lock the state struct semaphore.
- Wait for new data from the sensors if not buffered. We make sure to allow for interruptions to our semaphore and while waiting for the state struct to be updated.
- Perform additional checks for EOF, determine the length of the data to be copied to user space.
- Copy the data to user space, checking for errors in the process.
- Update the file position value and reset it if necessary.
- Unlock the semaphore and return.

```
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);

```

```
        ...
}
```

Changes made to the **lunix_chrdev_init()** function:

- Register the character device value range (region).
- Add the character device to the system.