Report

Exercise 2 Accessing EEPROM via I2C bus

by

Ankit Rathi (1207543476)

Accessing calls from multiple user threads

To accept calls from multiple threads, the driver can be modified with below mentioned steps, the dirty data read and write needs to be protected from multiple user threads accessing the shared resources.

- Declare two global semaphores in the driver,
  - *struct semaphore queue_sem;*
  - *struct semaphore data_sem;*
- Initialize the two semaphores in *static int __init i2c_eeprom_init(void)* method using the below statements.

  sema_init(&queue_sem,1);
  sema_init(&data_sem,1);
- In the "*static ssize_t i2c_eeprom_read_from_queue(struct file *file, char __user *buf, size_t count, loff_t *offset)*" and "*static ssize_t i2c_eeprom_write_into_queue(struct file *file, const char __user *buf, size_t count, loff_t *offset)*" function call,
  1. The part of the code to send and receive messages over i2c must be protected by semaphore.
     ```
     down_interruptible(&queue_sem);
     i2c_master_send();
     i2c_master_recv();
     up(&queue_sem);
     ```
  2. The part of the code to put data into workers thread must also be protected from dirty read and write.
     ```
     down_interruptible(&data_sem);
     send_work_queue->work_id = ++WORK_ID_COUNTER;
     send_work_queue->read_or_write     = 'W';
     send_work_queue->status_Flag   = 'Q';
     up(&data_sem);
     ```

Work on a different EEPROM chip (Different Number of Pages)

For the driver to be compatible with EEPROM Chip of different Page Size, a simple change of providing a new value of the page size should be done in the driver.

```
#define EEPROM_PAGE_SIZE      64
```

Here, the new page size can be used instead of 64. The driver program accesses the *EEPROM_PAGE_SIZE* all across, to calculate wrapping at end of page, memory allocation for temp buffers. So a change in this size will make driver compatible with EEPROM Chip of different page size.

Work on a different EEPROM chip (Different Slave Address)

For driver to be use a EEPROM Chip at different Slave Address, a change in address defined in the driver needs to be changed.

```
#define SLAVE_ADDRESS      0x54
```

Here, the slave address of the EEPROM Chip address is put into *SLAVE_ADDRESS*. The code all across uses this SLAVE_ADDRESS as address for the EEPROM Chip. Hence, a change in this value defined in the driver will make driver do all the transactions with the EEPROM Chip at different address.

<u>Work on a different EEPROM chip (Adding a new Chip)</u>
To add a new EEPROM Chip into the already existing setup, a new entry is required into i2c_board_info, which defines the device name and its slave address.

```
static struct i2c_board_info i2c_eeprom_board_info[] = {
        {
                I2C_BOARD_INFO(DEVICE_NAME, SLAVE_ADDRESS),
        }
};
```

Also the semaphores to protect data needs to be moved into the device data structure and instead of using a *struct i2c_EEPROM_dev *i2c_EEPROM_device_list;* , we will have to use a array of device data structure or a pointer so that it can be allocated dynamically to add new devices and respective changes needs to be made to add a new adapter as well.