Student ID: 1806235731

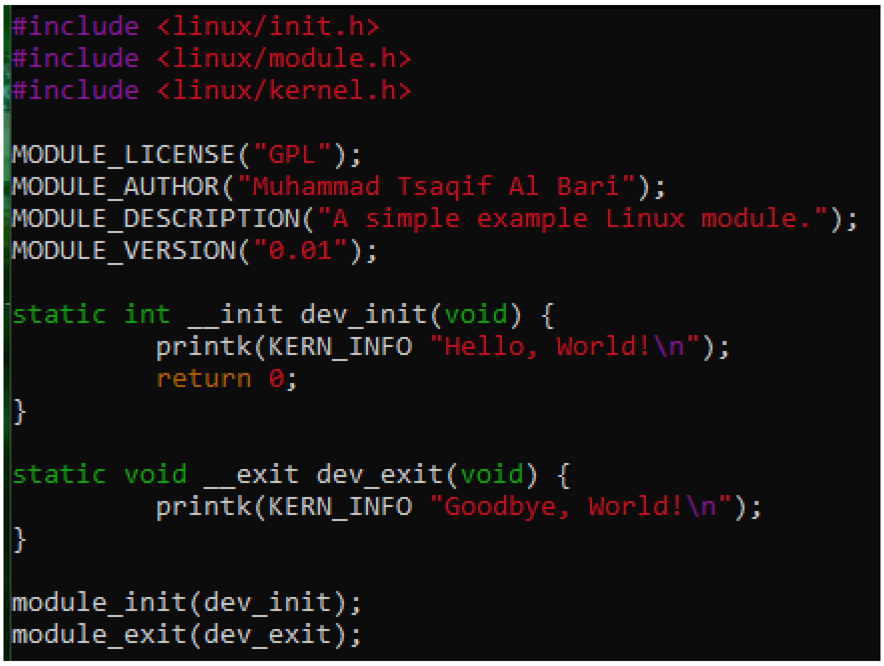
Name: Muhammad Tsaqif Al Bari

Class: B

Week 13 Logbook

Creating a custom driver

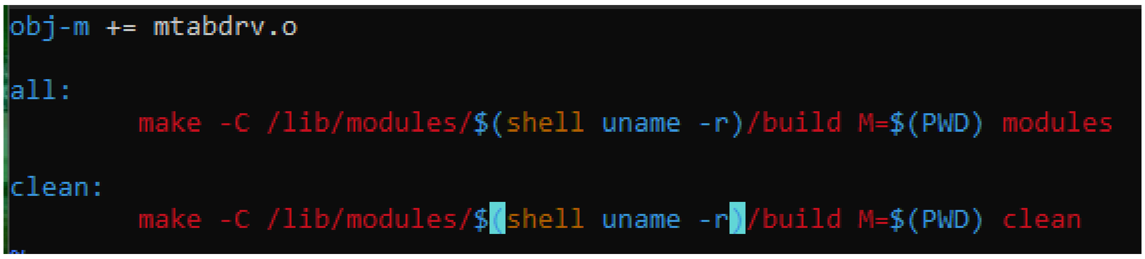
1. Create a basic custom module on C.



```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Muhammad Tsaqif Al Bari");
MODULE_DESCRIPTION("A simple example Linux module.");
MODULE_VERSION("0.01");

static int __init dev_init(void) {
        printk(KERN_INFO "Hello, World!\n");
        return 0;
}

static void __exit dev_exit(void) {
        printk(KERN_INFO "Goodbye, World!\n");
}

module_init(dev_init);
module_exit(dev_exit);
```

   First we would need to create a basic module for our work base. There are 2 methods define here which are dev_init and dev_exit. Both methods have __init and __exit respectively. These will tell our kernel that theses two methods are used when we install or uninstall our module.

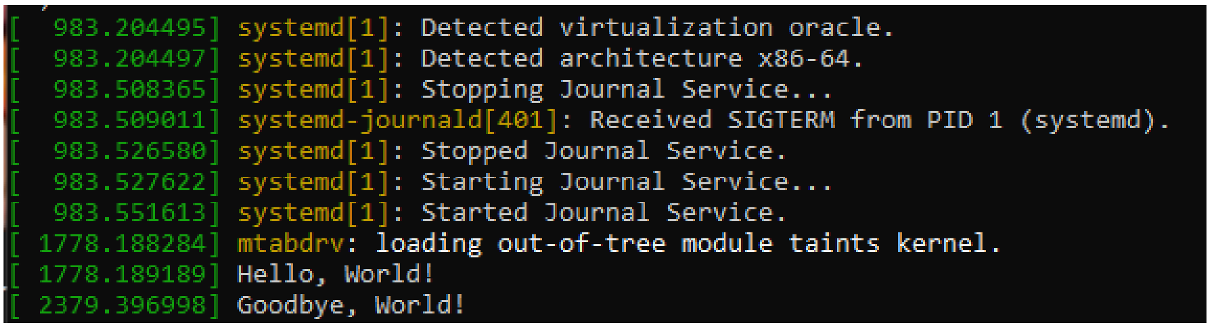
2. Create a makefile



```
obj-m += mtabdrv.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

   After creating our module template on C, we need to compile them into .ko, which stands for Kernel Object. After compiling it .ko, we can install them.

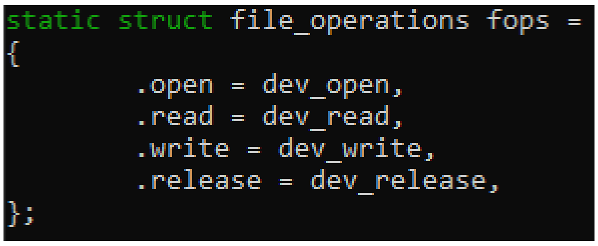
3. Installing the module



```
[  983.204495] systemd[1]: Detected virtualization oracle.
[  983.204497] systemd[1]: Detected architecture x86-64.
[  983.508365] systemd[1]: Stopping Journal Service...
[  983.509011] systemd-journald[401]: Received SIGTERM from PID 1 (systemd).
[  983.526580] systemd[1]: Stopped Journal Service.
[  983.527622] systemd[1]: Starting Journal Service...
[  983.551613] systemd[1]: Started Journal Service.
[ 1778.188284] mtabdrv: loading out-of-tree module taints kernel.
[ 1778.189189] Hello, World!
[ 2379.396998] Goodbye, World!
```

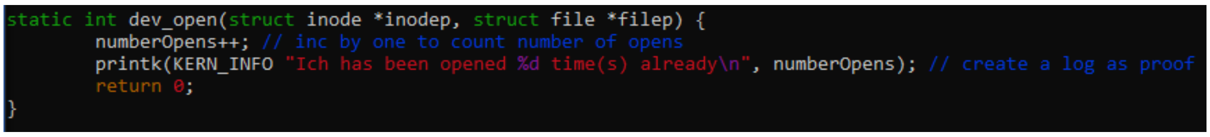
After compiling them to .ko we can install them with insmod moduleName.ko, in this case it’s insmod mtabdrv.ko. We can check if it’s installed by looking on dmesg. Since we need to install a driver for this task, I would go and uninstall it with rmmod mtabdrv.ko.

4. Adding open,close,write,read capabilities

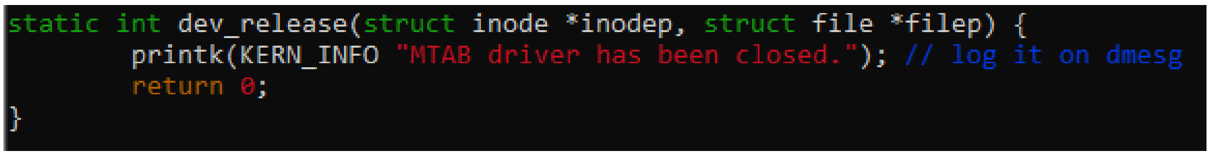


```
static struct file_operations fops =
{
        .open = dev_open,
        .read = dev_read,
        .write = dev_write,
        .release = dev_release,
};
```

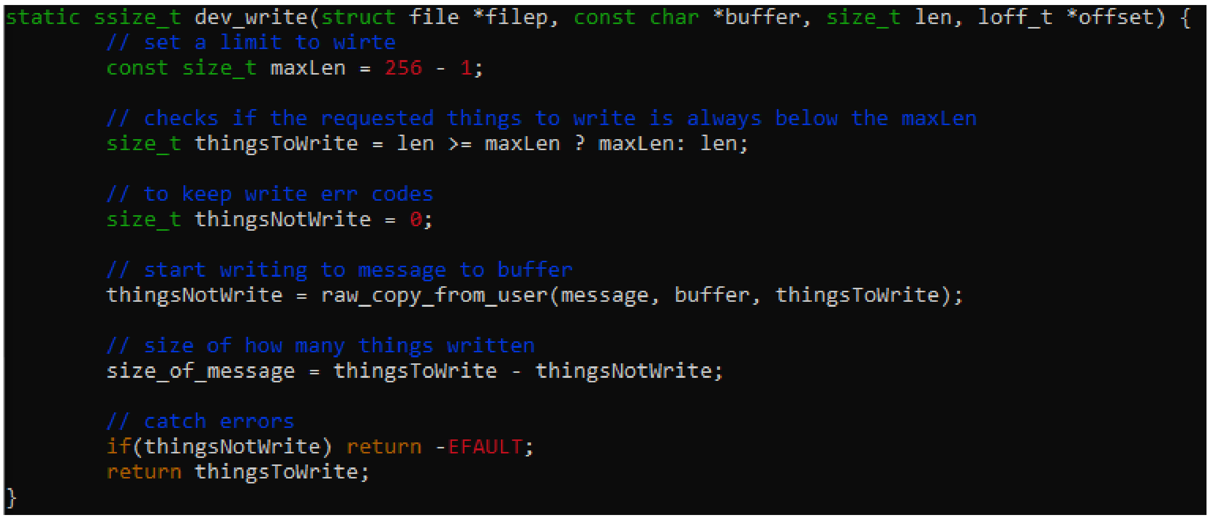
To turn our module into a device driver, we need to implement open(),close(),read(),and write() capabilities to our module.



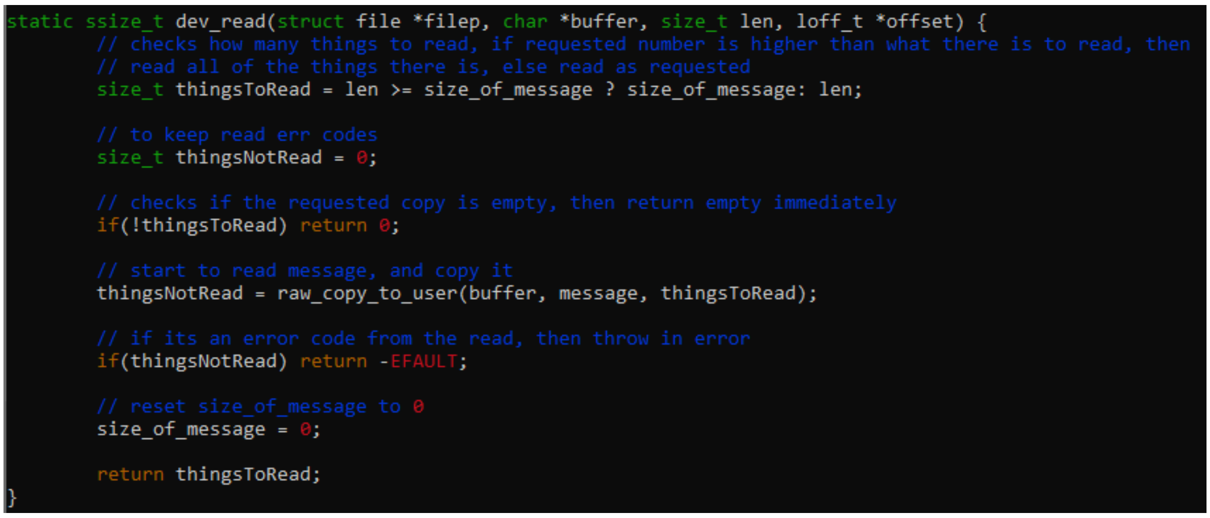
```
static int dev_open(struct inode *inodep, struct file *filep) {
        numberOpens++; // inc by one to count number of opens
        printk(KERN_INFO "Ich has been opened %d time(s) already\n", numberOpens); // create a log as proof
        return 0;
}
```



```
static int dev_release(struct inode *inodep, struct file *filep) {
        printk(KERN_INFO "MTAB driver has been closed."); // log it on dmesg
        return 0;
}
```

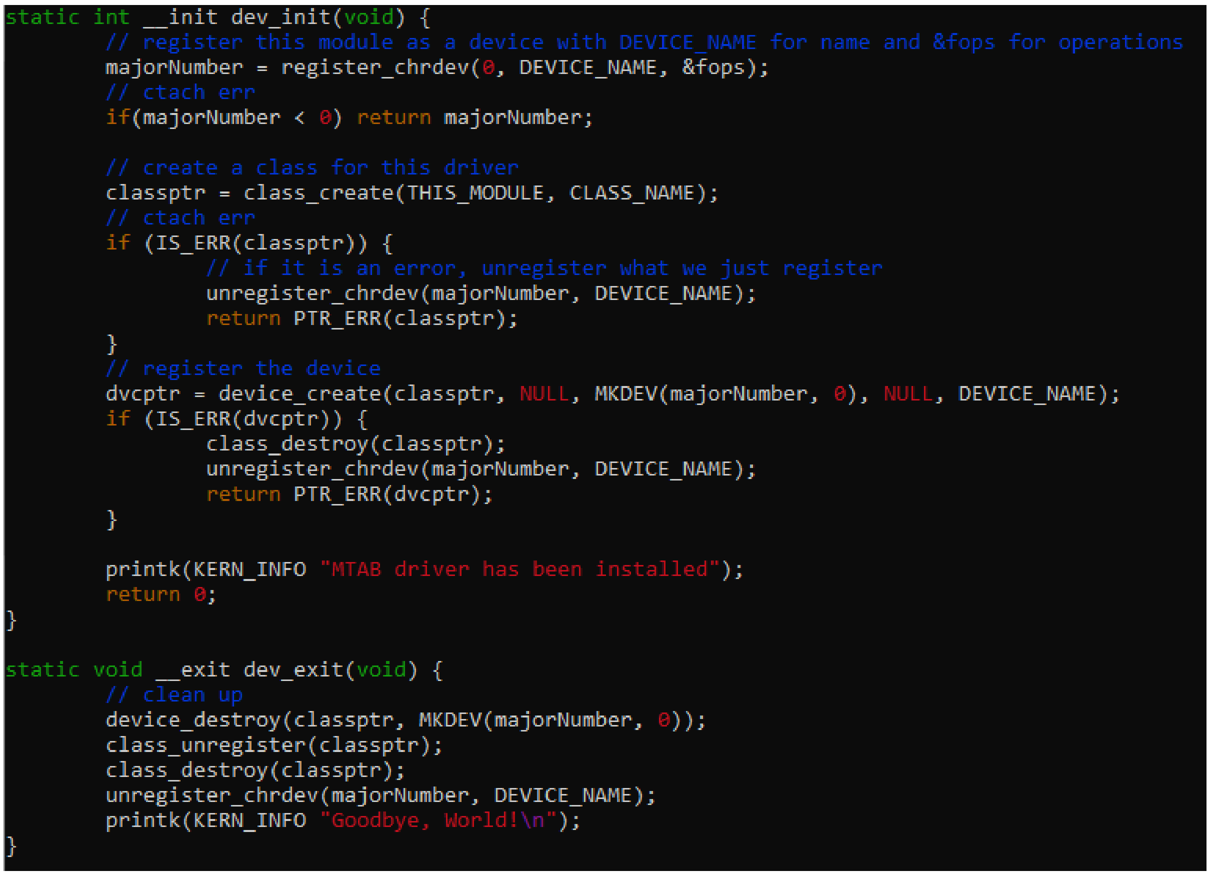


```
static ssize_t dev_write(struct file *filep, const char *buffer, size_t len, loff_t *offset) {
        // set a limit to wirte
        const size_t maxLen = 256 - 1;

        // checks if the requested things to write is always below the maxLen
        size_t thingsToWrite = len >= maxLen ? maxLen: len;

        // to keep write err codes
        size_t thingsNotWrite = 0;

        // start writing to message to buffer
        thingsNotWrite = raw_copy_from_user(message, buffer, thingsToWrite);

        // size of how many things written
        size_of_message = thingsToWrite - thingsNotWrite;

        // catch errors
        if(thingsNotWrite) return -EFAULT;
        return thingsToWrite;
}
```



```c
static ssize_t dev_read(struct file *filep, char *buffer, size_t len, loff_t *offset) {
        // checks how many things to read, if requested number is higher than what there is to read, then
        // read all of the things there is, else read as requested
        size_t thingsToRead = len >= size_of_message ? size_of_message: len;

        // to keep read err codes
        size_t thingsNotRead = 0;

        // checks if the requested copy is empty, then return empty immediately
        if(!thingsToRead) return 0;

        // start to read message, and copy it
        thingsNotRead = raw_copy_to_user(buffer, message, thingsToRead);

        // if its an error code from the read, then throw in error
        if(thingsNotRead) return -EFAULT;

        // reset size_of_message to 0
        size_of_message = 0;

        return thingsToRead;
}
```

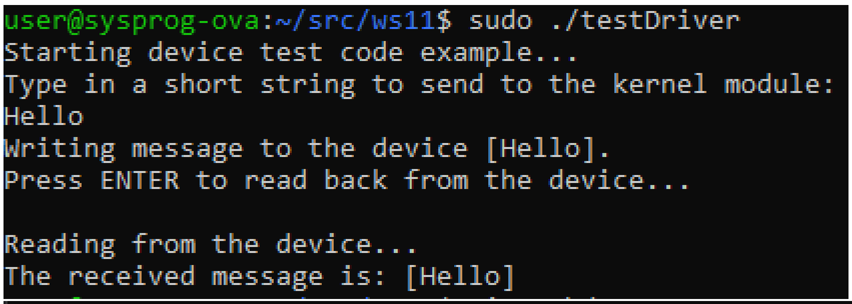
5. Register our module as a driver



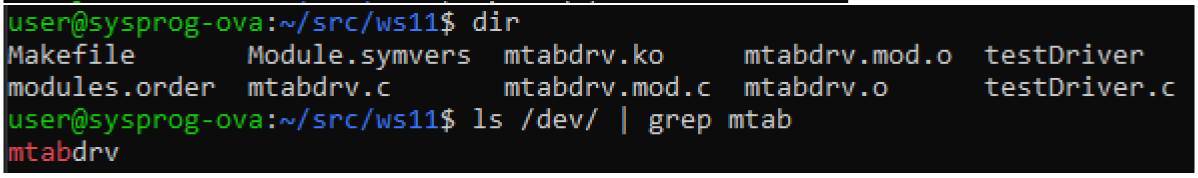
```c
static int __init dev_init(void) {
        // register this module as a device with DEVICE_NAME for name and &fops for operations
        majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
        // ctach err
        if(majorNumber < 0) return majorNumber;

        // create a class for this driver
        classptr = class_create(THIS_MODULE, CLASS_NAME);
        // ctach err
        if (IS_ERR(classptr)) {
                // if it is an error, unregister what we just register
                unregister_chrdev(majorNumber, DEVICE_NAME);
                return PTR_ERR(classptr);
        }
        // register the device
        dvcptr = device_create(classptr, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
        if (IS_ERR(dvcptr)) {
                class_destroy(classptr);
                unregister_chrdev(majorNumber, DEVICE_NAME);
                return PTR_ERR(dvcptr);
        }

        printk(KERN_INFO "MTAB driver has been installed");
        return 0;
}

static void __exit dev_exit(void) {
        // clean up
        device_destroy(classptr, MKDEV(majorNumber, 0));
        class_unregister(classptr);
        class_destroy(classptr);
        unregister_chrdev(majorNumber, DEVICE_NAME);
        printk(KERN_INFO "Goodbye, World!\n");
}
```

To register our module as a driver, we need to register it a major number to the kernel. To do that, we need to modify our init and exit methods to register a major number on install, and remove it on uninstalling.

6. Test run



```
user@sysprog-ova:~/src/ws11$ sudo ./testDriver
Starting device test code example...
Type in a short string to send to the kernel module:
Hello
Writing message to the device [Hello].
Press ENTER to read back from the device...

Reading from the device...
The received message is: [Hello]
```



```
user@sysprog-ova:~/src/ws11$ dir
Makefile       Module.symvers  mtabdrv.ko     mtabdrv.mod.o  testDriver
modules.order  mtabdrv.c       mtabdrv.mod.c  mtabdrv.o      testDriver.c
user@sysprog-ova:~/src/ws11$ ls /dev/ | grep mtab
mtabdrv
```