CS5250 ASSIGNMENT 4

YANG MO – A0091836X

Task 1

1. Screenshot of the github commit & Code

Please see **appendix A** for github screenshot and **appendix B** for device driver code

Task 2

1. New `kmalloc` and `kfree` code in `init` and `exit` function

- `kmalloc`:

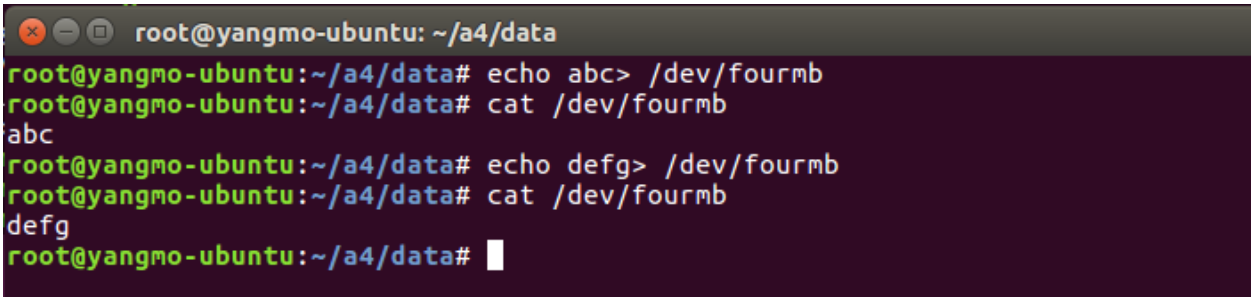
```
// allocate 4 MB of memory for storage
// kmalloc is just like malloc, the second parameter is
// the type of memory to be allocated.
// To release the memory allocated by kmalloc, use kfree.
fourMB_data = kmalloc(1024*1024*4*sizeof(char), GFP_KERNEL);
```

- `kfree`:

```
// if the pointer is pointing to something
if (fourMB_data) {
        // free the memory and assign the pointer to NULL
        kfree(fourMB_data);
        fourMB_data = NULL;
}
```

2. Output of the test cases:

As can be seen below, new device can take more than 1 byte:



3. Copy a 5MB file to your device and display total bytes written and show unwritten part

I am using a `printk` function in write function to print the total bytes written so far:

```
printk("Total Bytes written so far: %d ", bytes_written_total);
```

This `bytes_written_total` will record the latest total written bytes so far since the open function is called. It will be reset to zero each time open function is called and will accumulate if write function is called multiple times to write an large file (such as in our case 4MB or 5MB file).

- Preparation to create exactly 5MB file and 4MB file:
  Create exactly 4MB file:

```
root@yangmo-ubuntu: ~/a4/data
root@yangmo-ubuntu:~/a4/data# perl -e 'print "1" x 4194303 ; print"X" x 1' > fourmb.txt
root@yangmo-ubuntu:~/a4/data#
```

  Create exactly 5MB file:

```
root@yangmo-ubuntu:~/a4/data# perl -e 'print "1" x 4194303 ; print"X" x 1; print "2" x 1048576' >
 fivemb.txt
root@yangmo-ubuntu:~/a4/data#
```

  Created files:

```
root@yangmo-ubuntu: ~/a4/data
root@yangmo-ubuntu:~/a4/data# ls -l
total 9216
-rw-r--r-- 1 root root 5242880 Apr  2 21:56 fivemb.txt
-rw-r--r-- 1 root root 4194304 Apr  2 21:55 fourmb.txt
root@yangmo-ubuntu:~/a4/data#
```

```
root@yangmo-ubuntu: ~/a4/data
root@yangmo-ubuntu:~/a4/data# ls -lh
total 9.0M
-rw-r--r-- 1 root root 5.0M Apr  2 21:56 fivemb.txt
-rw-r--r-- 1 root root 4.0M Apr  2 21:55 fourmb.txt
```

  To make it clear, `fourmb.txt` file contains 4194303 bytes of '1' with last byte to be 'X', which makes it totally 4194304 bytes (4MB).

```
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111Xroot@yangmo-ubuntu:~/a4/data#
```

  While `fivemb.txt` have its first 4MB to be exactly the same content as fourmb.txt while the last 1MB to be all '2':

```
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222root@yangmo-ubuntu:~/a4/data#
```

  So the expected results from my device will be only 4MB data will be written.

  Now let us copy the `fivemb.txt` file to the device:

```
root@yangmo-ubuntu: ~/a4/data
root@yangmo-ubuntu:~/a4/data# cat fivemb.txt > /dev/fourmb
cat: write error: No space left on device
root@yangmo-ubuntu:~/a4/data#
```

  As expected, no space left on device error is thrown. Next to check how many bytes are written to the device by checking the `dmesg`:

```
root@yangmo-ubuntu:~/a4/data# cat fivemb.txt > /dev/fourmb
cat: write error: No space left on device
root@yangmo-ubuntu:~/a4/data# dmesg
[21474.698130] FourMB device module is unloaded
[21477.185424] This is a fourMB device module
[21552.405699] Offset: 0 Total Bytes written so far: 131072 Total Bytes written so far: 262144 T
tal Bytes written so far: 393216 Total Bytes written so far: 524288 Total Bytes written so far:
55360 Total Bytes written so far: 786432 Total Bytes written so far: 917504 Total Bytes written
o far: 1048576 Total Bytes written so far: 1179648 Total Bytes written so far: 1310720 Total Byt
s written so far: 1441792 Total Bytes written so far: 1572864 Total Bytes written so far: 170393
 Total Bytes written so far: 1835008 Total Bytes written so far: 1966080 Total Bytes written so
ar: 2097152 Total Bytes written so far: 2228224 Total Bytes written so far: 2359296 Total Bytes
ritten so far: 2490368 Total Bytes written so far: 2621440 Total Bytes written so far: 2752512 T
tal Bytes written so far: 2883584
[21552.462718] Total Bytes written so far: 3014656 Total Bytes written so far: 3145728 Total Byt
s written so far: 3276800 Total Bytes written so far: 3407872 Total Bytes written so far: 353894
 Total Bytes written so far: 3670016 Total Bytes written so far: 3801088 Total Bytes written so
ar: 3932160 Total Bytes written so far: 4063232 Total Bytes written so far: 4194304 Total Bytes
ritten so far: 4194304
[21552.484744] No space left on device
root@yangmo-ubuntu:~/a4/data#
```

As can be seen above, the write functions were called multiple times and the total bytes written are exactly 4194304 bytes which is 4MB.

To prove the above `dmesg` is correct, we need to check the actual data in `/dev/fourmb:`

- Checking head section (first 200 bytes):

```
root@yangmo-ubuntu: ~/a4/data
root@yangmo-ubuntu:~/a4/data# cat /dev/fourmb | head -c 200
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111root@yangmo-ubuntu:~/a4/data#
```

- Checking tail section(last 200 bytes):

```
root@yangmo-ubuntu:~/a4/data# cat /dev/fourmb | tail -c 200
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111X root@yangmo-ubuntu:~/a4/data#
```

We can see from the tail section of the device that the write stopped right at the 'X' of the `fivemb.txt` file and no '2' in the following section was written to the device.

In addition, I also double checked the written data by ouputing the content from the device to an output.txt file and checked its size to be exactly 4MB:

```
root@yangmo-ubuntu:~/a4/data# cat /dev/fourmb > output.txt
root@yangmo-ubuntu:~/a4/data# ls -l
total 13312
-rw-r--r-- 1 root root 5242880 Apr  2 21:56 fivemb.txt
-rw-r--r-- 1 root root 4194304 Apr  2 21:55 fourmb.txt
-rw-r--r-- 1 root root 4194304 Apr  2 22:23 output.txt
root@yangmo-ubuntu:~/a4/data# ls -lh
total 13M
-rw-r--r-- 1 root root 5.0M Apr  2 21:56 fivemb.txt
-rw-r--r-- 1 root root 4.0M Apr  2 21:55 fourmb.txt
-rw-r--r-- 1 root root 4.0M Apr  2 22:23 output.txt
root@yangmo-ubuntu:~/a4/data#
```

Task 3

1. List arguments of llseek function and explain their meanings

List of the arguments is:

```
loff_t fourMB_lseek(struct file *file, loff_t offset, int whence);
```

Meanings:

`file`: The file whose current file offset we want to change.

`offset`: The amount the byte offset is to be changed. The sign indicates whether the offset is to be moved forward (positive) or backward (negative).

`Whence`: controls the behavior and it could have three possible values:

1) ***SEEK_SET :*** the file offset should be set to offset bytes.
2) ***SEEK_CUR :*** the file offset should be set to its current location plus offset
3) ***SEEK_END:*** the file offset should be set to the size of the file (in our case is current number of bytes in the device driver) plus offset.

2. Run the test program and give screenshot of the results

There is a small change I made to the `test.c`:

```
printf("unable to open lcd");
exit(EXIT_FAILURE);
```

I changed `printk` to `printf` in order to compile it to `test.o`:

```
root@yangmo-ubuntu:~/a4/task3# gcc test.c -o test.o
```

Run the test and get the results:

```
root@yangmo-ubuntu:~/a4/task3# ./test.o
lseek = 0
test begin!
lseek = 4
written = 3
lseek = 10
lseek = 6
lseek = -1
root@yangmo-ubuntu:~/a4/task3# cat /dev/lcd
1111222111root@yangmo-ubuntu:~/a4/task3#
```

The reason why `lseek` returns 10 for "`k = lseek(lcd, 0, SEEK_END);`" after writing 3 bytes is that before writing another `lseek` function set the pointer to offset 4, which then cause the 3 bytes writing function to overwrite the old value from offset 4. Hence the total bytes are still 10 and this is why `SEEK_END` with offset 0 will return value 10.

Task 4

1. Basic Hello function and show hello message

There is one extra line needed to be added to define `SCULL_IOC_MAXNR` in addition to the sample hello code:

```
#define SCULL_IOC_MAGIC 'k'
#define SCULL_HELLO _IO(SCULL_IOC_MAGIC, 1)
#define SCULL_IOC_MAXNR 1
```

Run the test program as following and check the `dmesg`:

```
root@yangmo-ubuntu:~/a4/task4# ./test.o
test begin!
written = 3
result = 0
root@yangmo-ubuntu:~/a4/task4# dmesg
[34553.537088] Total Bytes written so far: 3
[34553.537095] hello
root@yangmo-ubuntu:~/a4/task4#
```

Hello was printed as expected.

2. Implement and test _IOW and _IOR

Added cases in `ioctl` method to handle `WRITE` and `READ`

```
case SCULL_WRITE_MESSAGE:
        max_length = MAX_DEV_MSG_LENGTH;
        msg = (char *)arg;
        tmp_dev_msg = dev_msg;
        while(*msg && max_length > 1){
                copy_from_user(tmp_dev_msg++, msg++, sizeof(char));
                max_length --;
        }
        *tmp_dev_msg = '\0';
        printk(KERN_WARNING "message written: %s\n", dev_msg);
        break;
case SCULL_READ_MESSAGE:
        msg = (char *)arg;
        tmp_dev_msg = dev_msg;
        while(*tmp_dev_msg){
                copy_to_user(msg++, tmp_dev_msg++, sizeof(char));
        }

        //terminate the input message
        put_user('\0', msg);
        break;
```

It is important that in read and write section, we need to put '\0' in the end to terminate the char array properly so when it is being printed, it could be printed properly as expected.

I also made an assumption here that the max length of the `dev_msg` is 1000:

```
//set the dev msg max length to be 1000
#define MAX_DEV_MSG_LENGTH 1000
```

I added following test cases in the `test.c`:

```
int k, i, sum;
char s[3];
char *msg = "Yang Mo A0091836X";
char msg_read[30];

memset(s, '2', sizeof(s));
printf("test begin!\n");

k = write(lcd, s, sizeof(s));
printf("written = %d\n", k);

k = ioctl(lcd, SCULL_HELLO);
printf("result = %d\n", k);

k = ioctl(lcd, SCULL_WRITE_MESSAGE, msg);
printf("result = %d\n", k);

k = ioctl(lcd, SCULL_READ_MESSAGE, msg_read);
printf("result = %d\n", k);
printf("result = %s\n", msg_read);
```

Run the test and get following result and check `dmesg` for IOW method:

```
root@yangmo-ubuntu:~/a4/task4# ./test.o
test begin!
written = 3
result = 0
result = 0
result = 0
result = Yang Mo A0091836X
root@yangmo-ubuntu:~/a4/task4# dmesg
[39404.348954] Total Bytes written so far: 3
[39404.348962] hello
[39404.348965] message written: Yang Mo A0091836X
```

3. Implement and test _IOWR

We need to add an `original_dev_msg` to store the old value of `dev_msg`

```
char *original_dev_msg = NULL;
```

As shown in next page, before overwriting the `dev_msg`, we need to copy its old value to `original_dev_msg`, the data of which will be copied to the user message param later before the method returns.

```
case SCULL_WRITE_READ_MESSAGE:
        strncpy(original_dev_msg, dev_msg, sizeof(char)*MAX_DEV_MSG_LENGTH);
        max_length = MAX_DEV_MSG_LENGTH;
        msg = (char *)arg;
        tmp_dev_msg = dev_msg;
        while(*msg && max_length){
                copy_from_user(tmp_dev_msg++, msg++, sizeof(char));
                max_length --;
        }
        *tmp_dev_msg = '\0';
        printk(KERN_WARNING "new dev_msg written: %s\n", dev_msg);

        //now read original value back
        msg = (char *)arg;
        tmp_dev_msg = original_dev_msg;
        while(*tmp_dev_msg){
                copy_to_user(msg++, tmp_dev_msg++, sizeof(char));
        }

        //terminate the input message
        put_user('\0', msg);

        break;
```

Then we again need to update the `test.c` to test the new IOWR method:

```
int k, i, sum;
char s[3];
char *msg = "Yang Mo A0091836X";
char msg_read[30];
char newMsg[30] = "This is a new msg";

memset(s, '2', sizeof(s));
printf("test begin!\n");

k = write(lcd, s, sizeof(s));
printf("written = %d\n", k);

k = ioctl(lcd, SCULL_HELLO);
printf("result = %d\n", k);

k = ioctl(lcd, SCULL_WRITE_MESSAGE, msg);
printf("result = %d\n", k);

k = ioctl(lcd, SCULL_READ_MESSAGE, msg_read);
printf("result = %d\n", k);
printf("result = %s\n", msg_read);

k = ioctl(lcd, SCULL_WRITE_READ_MESSAGE, newMsg);
printf("result = %d\n", k);
printf("Original Message = %s\n", newMsg);
```

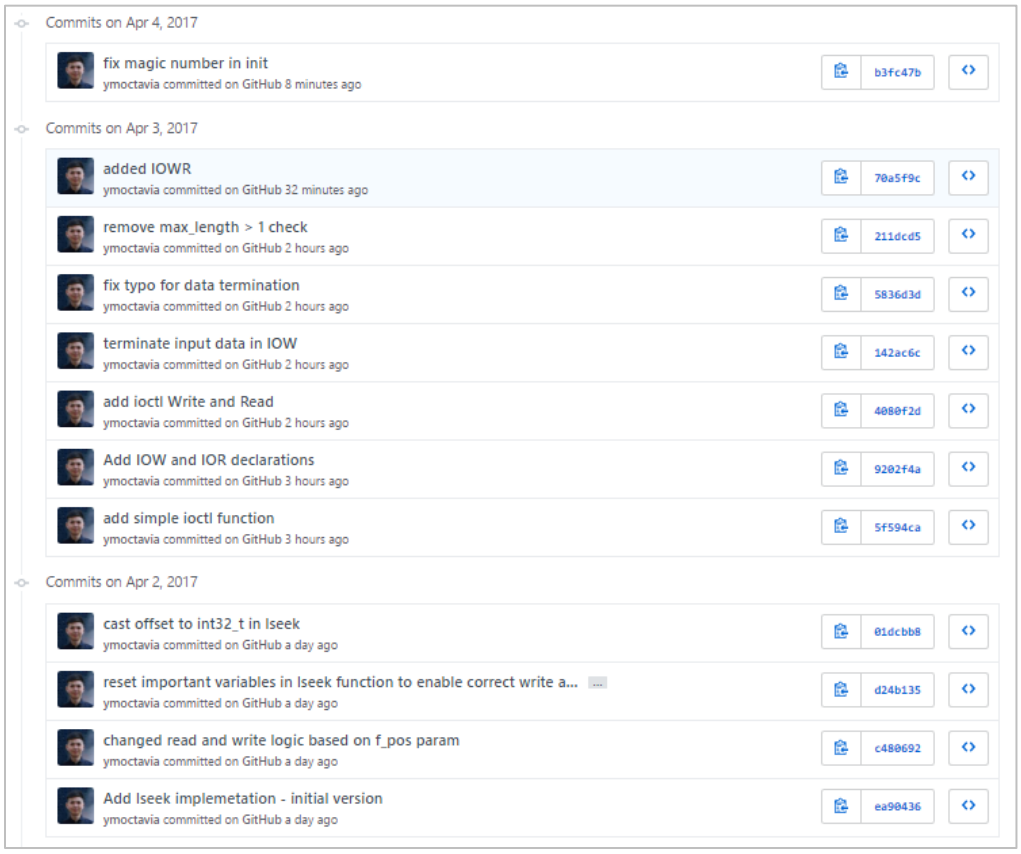
Now let's run the test to display old value and check the `dmesg` for new value:

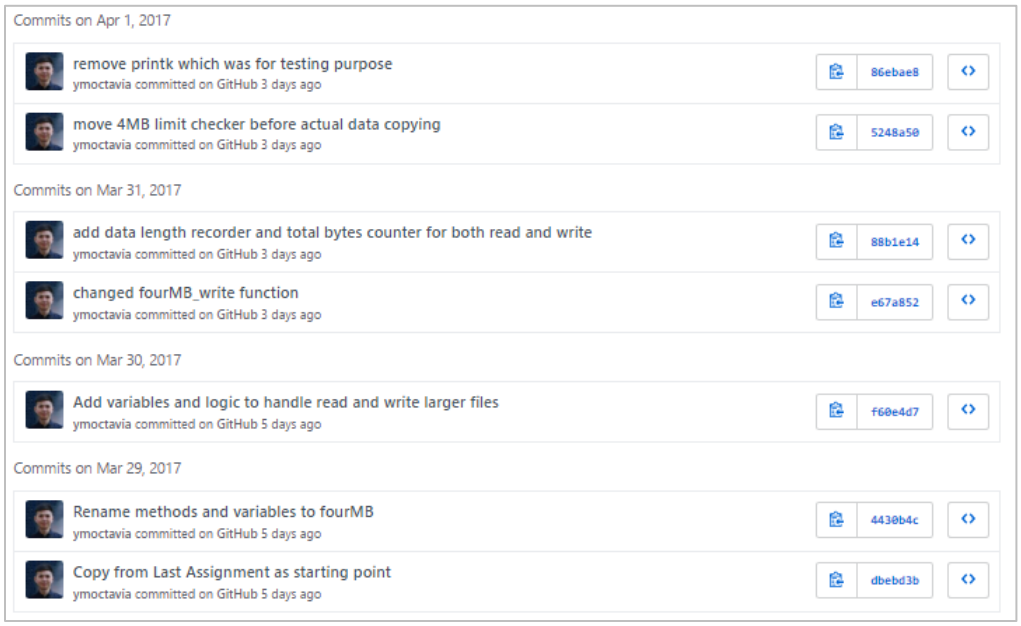
```
root@yangmo-ubuntu:~/a4/task4# ./test.o
test begin!
written = 3
result = 0
result = 0
result = 0
result = Yang Mo A0091836X
result = 0
Original Message = Yang Mo A0091836X
root@yangmo-ubuntu:~/a4/task4# dmesg
[45634.155537] Total Bytes written so far: 3
[45634.155547] hello
[45634.155550] message written: Yang Mo A0091836X
[45634.155556] new dev_msg written: This is a new msg
```

As can be seen, the original value was successfully returned back for display while the new value has been successfully written to `dev_msg` and logged by `printk`.

Appendix A

Github link: https://github.com/ymoctavia/CS5250/commits/master





Appendix B

Code Link: https://github.com/ymoctavia/CS5250/blob/master/Assignment%204/FourMBDevice.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/ioctl.h> /* needed for the _IOW etc stuff used later*/

#define SCULL_IOC_MAGIC 'k'
#define SCULL_HELLO _IO(SCULL_IOC_MAGIC, 1)
#define SCULL_WRITE_MESSAGE _IOW(SCULL_IOC_MAGIC, 2, char *)
#define SCULL_READ_MESSAGE _IOR(SCULL_IOC_MAGIC, 3, char *)
#define SCULL_WRITE_READ_MESSAGE _IOWR(SCULL_IOC_MAGIC, 4, char *)

#define SCULL_IOC_MAXNR 4

//set the dev msg max length to be 1000
#define MAX_DEV_MSG_LENGTH 1000

#define MAJOR_NUMBER 61/* forward declaration */


int fourMB_open(struct inode *inode, struct file *filep);
int fourMB_release(struct inode *inode, struct file *filep);
ssize_t fourMB_read(struct file *filep, char *buf, size_t count, loff_t *f_pos);
ssize_t fourMB_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos);
loff_t fourMB_lseek(struct file *file, loff_t offset, int whence);
long fourMB_ioctl(struct file *filp, unsigned int cmd, unsigned long arg);
static void fourMB_exit(void);


/* definition of file_operation structure */
struct file_operations fourMB_fops = {
        read: fourMB_read,
        write: fourMB_write,
        open: fourMB_open,
        release: fourMB_release,
        llseek: fourMB_lseek,
        unlocked_ioctl : fourMB_ioctl,
};

char *fourMB_data = NULL;

char *data_pointer = NULL;
size_t data_length_written = 0;
size_t data_length_to_read = 0;
int bytes_written_total = 0;
int bytes_read_total = 0;
```

```
char *dev_msg = NULL;
char *original_dev_msg = NULL;

loff_t fourMB_lseek(struct file *file, loff_t offset, int whence) {

        loff_t new_position = 0;

        switch(whence) {
                case SEEK_SET :
                    new_position = (int32_t)offset;
                    break;
                case SEEK_CUR :
                    new_position = file->f_pos + (int32_t)offset;
                    break;
                case SEEK_END :
                    new_position = data_length_written + (int32_t)offset;
                    break;
        }

        //check boundary
        if(new_position > data_length_written){
                new_position = data_length_written;
        }

        if(new_position < 0){
                new_position = 0;
        }

        file->f_pos = new_position;

        //reset data pointer
        data_pointer = fourMB_data;

        //reset important variables
        data_length_to_read = data_length_written;
        bytes_written_total = 0;
        bytes_read_total = 0;

        return new_position;
}

long fourMB_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
        int err = 0;
        int retval = 0;
        char *msg;
        char *tmp_dev_msg;
        int max_length;

        /*
        * extract the type and number bitfields, and don't decode
```

```
        * wrong cmds: return ENOTTY (inappropriate ioctl) before access_ok()
        */

        if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC) return -ENOTTY;
        if (_IOC_NR(cmd) > SCULL_IOC_MAXNR) return -ENOTTY;

        /*
        * the direction is a bitmask, and VERIFY_WRITE catches R/W
        * transfers. 'Type' is user-oriented, while
        * access_ok is kernel-oriented, so the concept of "read" and
        * "write" is reversed
        */

        if (_IOC_DIR(cmd) & _IOC_READ)
                err = !access_ok(VERIFY_WRITE, (void __user *)arg, _IOC_SIZE(cmd));
        else if (_IOC_DIR(cmd) & _IOC_WRITE)
                err = !access_ok(VERIFY_READ, (void __user *)arg, _IOC_SIZE(cmd));

        if (err) return -EFAULT;

        switch(cmd) {
                case SCULL_HELLO:
                        printk(KERN_WARNING "hello\n");
                        break;
                case SCULL_WRITE_MESSAGE:
                        max_length = MAX_DEV_MSG_LENGTH;
                        msg = (char *)arg;
                        tmp_dev_msg = dev_msg;
                        while(*msg && max_length > 1){
                                copy_from_user(tmp_dev_msg++, msg++, sizeof(char));
                                max_length --;
                        }
                        *tmp_dev_msg = '\0';
                        printk(KERN_WARNING "message written: %s\n", dev_msg);
                        break;
                case SCULL_READ_MESSAGE:
                        msg = (char *)arg;
                        tmp_dev_msg = dev_msg;
                        while(*tmp_dev_msg){
                                copy_to_user(msg++, tmp_dev_msg++, sizeof(char));
                        }

                        //terminate the input message
                        put_user('\0', msg);
                        break;
                case SCULL_WRITE_READ_MESSAGE:
                        strncpy(original_dev_msg, dev_msg, sizeof(char)*MAX_DEV_MSG_LENGTH);
                        max_length = MAX_DEV_MSG_LENGTH;
                        msg = (char *)arg;
                        tmp_dev_msg = dev_msg;
                        while(*msg && max_length){
```

```
                                copy_from_user(tmp_dev_msg++, msg++, sizeof(char));
                                max_length --;
                        }
                        *tmp_dev_msg = '\0';
                        printk(KERN_WARNING "new dev_msg written: %s\n", dev_msg);

                        //now read original value back
                        msg = (char *)arg;
                        tmp_dev_msg = original_dev_msg;
                        while(*tmp_dev_msg){
                                copy_to_user(msg++, tmp_dev_msg++, sizeof(char));
                        }

                        //terminate the input message
                        put_user('\0', msg);

                        break;
                default: /* redundant, as cmd was checked against MAXNR */
                        return -ENOTTY;
        }

        return retval;
}

int fourMB_open(struct inode *inode, struct file *filep)
{
        //re-align data pointer to the start of data section
        data_pointer = fourMB_data;

        //need to set how many bytes to read for read operation
        data_length_to_read = data_length_written;

        bytes_written_total = 0;
        bytes_read_total = 0;

        return 0; // always successful
}


int fourMB_release(struct inode *inode, struct file *filep)
{
        return 0; // always successful
}


ssize_t fourMB_read(struct file *filep, char *buf, size_t count, loff_t *f_pos)
{
        int bytes_read = 0;

        /* Check if it the end of the data section */
        if (data_length_to_read == 0){
```

```
                printk("Total Bytes read: %d ", bytes_read_total);
                return 0;
        }

        data_pointer = *f_pos + data_pointer;
        data_length_to_read -= *f_pos;

        while (count && *data_pointer && data_length_to_read) {

                copy_to_user(buf++, data_pointer++, sizeof(char));

                count--;
                bytes_read++;
                bytes_read_total ++;
                data_length_to_read --;
        }

        return bytes_read;
}


ssize_t fourMB_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos)
{
        int bytes_written = 0;

        //if this function is called first time during current write operation
        //simple reset the written data length
        if(data_pointer == fourMB_data){
                //printk("Offset: %d ", *f_pos);
                data_pointer += *f_pos;
        }

        while (count && *buf) {
                //detect if have writen 4MB data
                if(data_length_written >= 1024*1024*4*sizeof(char)){
                        break;
                }

                copy_from_user(data_pointer++, buf++, sizeof(char));

                count--;
                bytes_written++;
                bytes_written_total++;

                if((bytes_written_total + *f_pos) > data_length_written){
                        data_length_written ++;
                }
        }

        printk("Total Bytes written so far: %d ", bytes_written_total);
```

```
        //check if data more than 4MB left to be written
        if(count > 0)
        {
                printk(KERN_ALERT "No space left on device\n");

                /*Return Linux System Error<28>: No space left on device */
                return -ENOSPC;
        }

        return bytes_written;

}


static int fourMB_init(void)
{
        int result;
        // register the device
        result = register_chrdev(MAJOR_NUMBER, "fourMB", &fourMB_fops);
        if (result < 0) {
                return result;
        }
        // allocate 4 MB of memory for storage
        // kmalloc is just like malloc, the second parameter is
        // the type of memory to be allocated.
        // To release the memory allocated by kmalloc, use kfree.
        fourMB_data = kmalloc(1024*1024*4*sizeof(char), GFP_KERNEL);

        dev_msg = kmalloc(MAX_DEV_MSG_LENGTH*sizeof(char), GFP_KERNEL);
        original_dev_msg = kmalloc(MAX_DEV_MSG_LENGTH*sizeof(char), GFP_KERNEL);


        if (!fourMB_data) {
                fourMB_exit();
                // cannot allocate memory
                // return no memory error, negative signify a failure
                return -ENOMEM;
        }

        // initialize the value to be X
        *fourMB_data = 'X';
        data_length_written ++;


        printk(KERN_ALERT "This is a fourMB device module\n");
        return 0;
}


static void fourMB_exit(void)
{
```

```
{
        // if the pointer is pointing to something
        if (fourMB_data) {
                // free the memory and assign the pointer to NULL
                kfree(fourMB_data);
                fourMB_data = NULL;
        }

        if (dev_msg) {
                kfree(dev_msg);
                dev_msg = NULL;
        }
        if (original_dev_msg) {
                kfree(original_dev_msg);
                original_dev_msg = NULL;
        }

        // unregister the device
        unregister_chrdev(MAJOR_NUMBER, "fourMB");
        printk(KERN_ALERT "FourMB device module is unloaded\n");
}

MODULE_LICENSE("GPL");
module_init(fourMB_init);
module_exit(fourMB_exit);
```