CS5250

Assignment 3

Zhang Shuhao

A0120258N

## Question 1:

1. When will **module\_init** and **module\_exit** be loading/called?

``**lsmod** ($module).ko” will load the module named ($module), and subsequently, **module\_init** will be called.

Correspondingly, ``**rmmod**” will unload it, and **module\_exit** will be called.

1. What is the command of building the module, installing the module and removing the module?

I write a makefile to support those functions. Specifically, they are implemented as follows.

1. To build the module:

TARGET\_MODULE:=hello\_world\_mod

obj-m += $(TARGET\_MODULE).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

1. To install the module:

load:

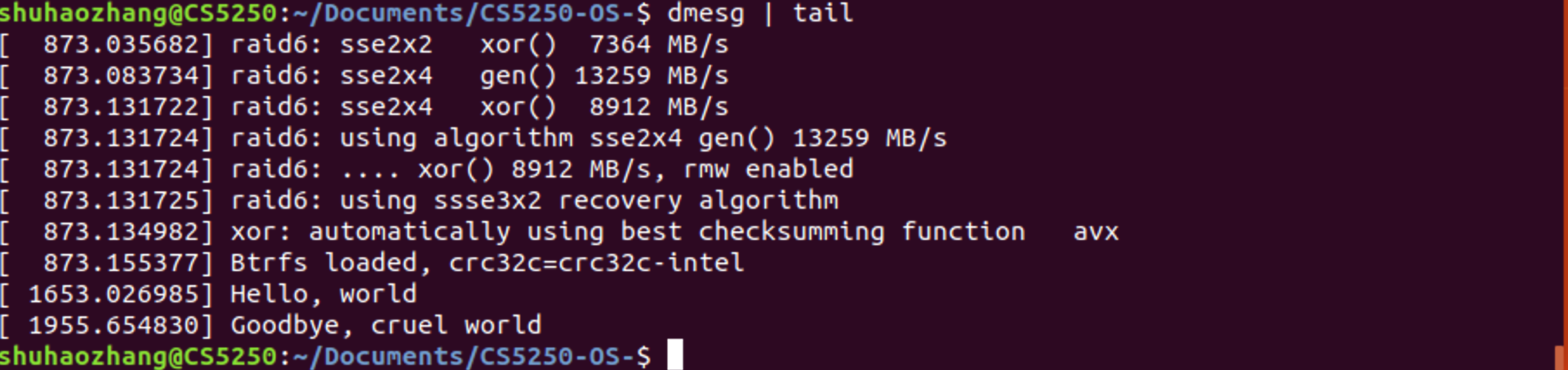
insmod ./$(TARGET\_MODULE).ko

1. To remove the module

unload:

rmmod ./$(TARGET\_MODULE).ko

1. Give the screenshot of the previous three commands and their results if any in the shell.

The following is the screenshot of ``**dmesg | tail**” after calling ``**lsmod**” and ``**rmmod**” of the hello world module. 

1. Add a <who> parameter to your module so that your module will show hello <who> during init stage.
2. Parameters are declared with the module\_param macro, which is defined in moduleparam.h. I highlight the changes in red as follows.

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/module.h>

#include <linux/moduleparam.h> //for module\_param

MODULE\_LICENSE("GPL");

static char \*who = "world";

module\_param(who, charp, S\_IRUSR|S\_IWUSR);

MODULE\_PARM\_DESC(who, "Tell me your name.");

static int hello\_init(void) {

printk(KERN\_ALERT "Hello, %s!\n", who);

return 0;

}

static void hello\_exit(void) {

printk(KERN\_ALERT "Goodbye, %s\n",who);

}

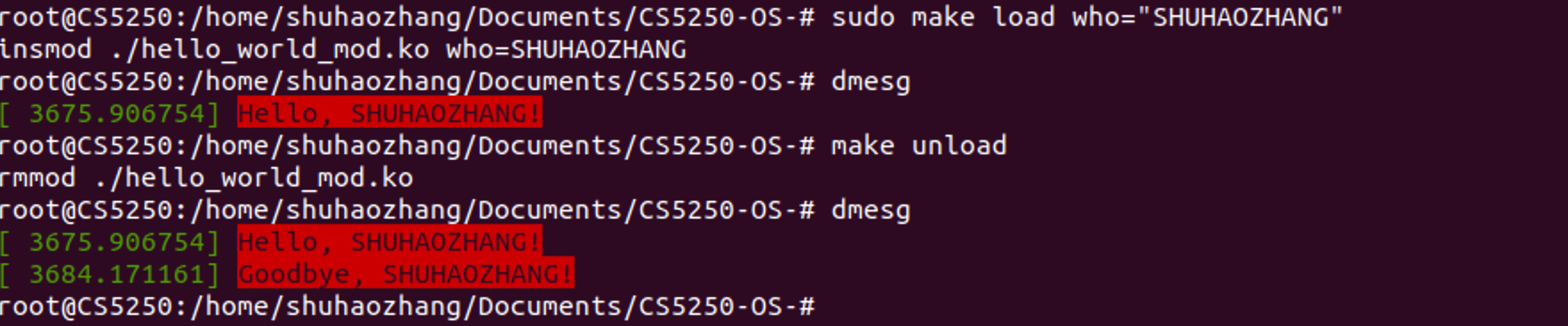
module\_init(hello\_init);

module\_exit(hello\_exit);

1. lsmod call needs to be updated as well.

insmod ./$(TARGET\_MODULE).ko who=$(who)

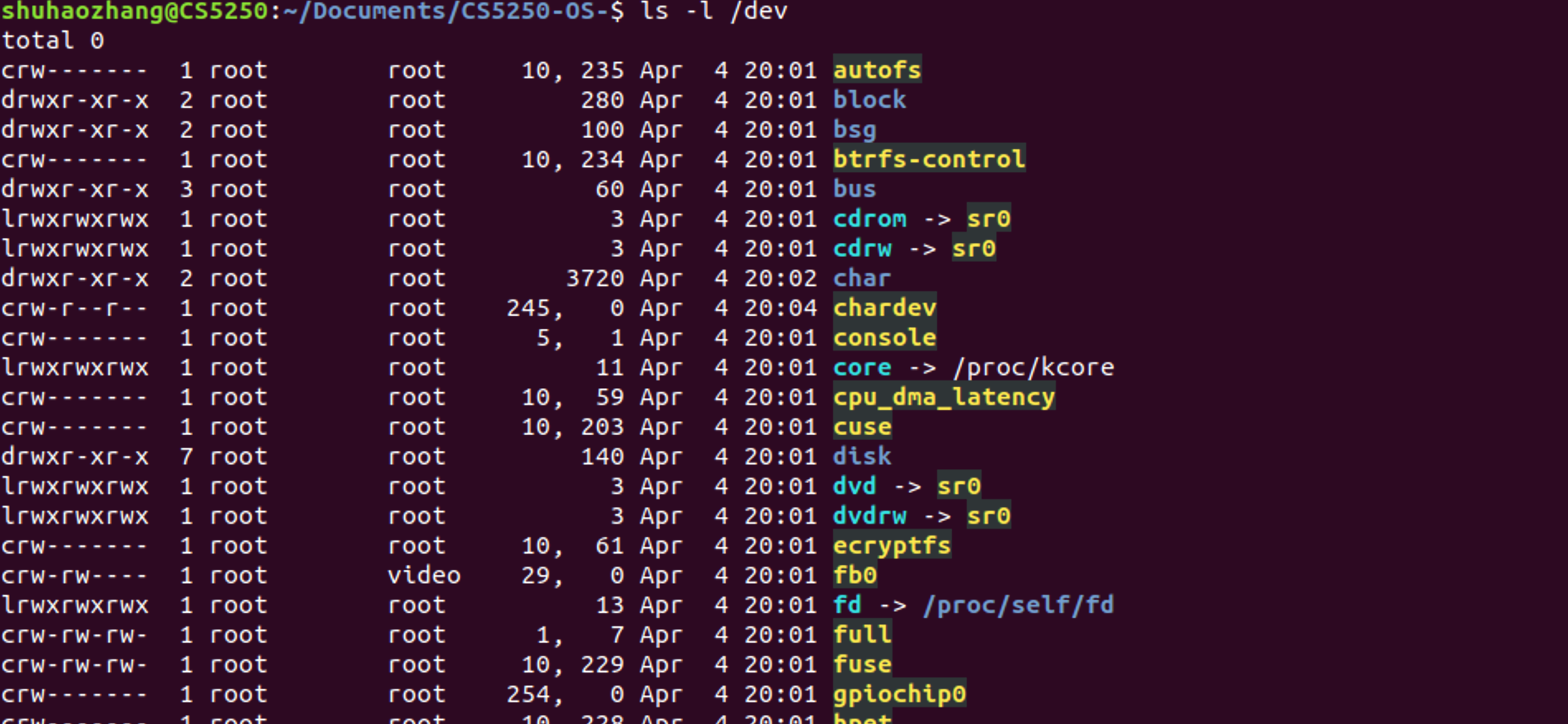
I test it with “sudo make load who=``SHUHAOZHANG”, “sudo make unload”. I clean dmesg first by **dmesg –c** The following screens shot shows the results of **dmesg**.



## Question 2:

1. Give the mknod command you use.

``sudo mknod /dev/chardev c **245** 0”. Note that, I use dynamic assigning major version instead of fixing at 61, which avoids potential conflict with other device drivers.

1. Give the screenshot of your device with “ls -l /dev” command and highlight your device. 
2. Give the codes of read and write functions that you implemented and the screenshots of the four testing cases.

**READ:**

ssize\_t onebyte\_read(struct file \*filep, char \*buf, size\_t count, loff\_t \*f\_pos)

{

/\*please complete the function on your own\*/

int bytes\_read = 0;

/\* Check if the buffer has been written \*/

if(\*buf != 0){

return 0;

}

copy\_to\_user(buf, onebyte\_data, sizeof(char));

bytes\_read ++;

return bytes\_read;

}

**WRITE:**

ssize\_t onebyte\_write(struct file \*filep, const char \*buf, size\_t count, loff\_t \*f\_pos)

{

/\*please complete the function on your own\*/

/\*\*

\* copy\_from\_user: Returns number of bytes that could not be copied. On success, this will be zero.

\*

\* to

\* Destination address, in kernel space.

\*

\* from

\* Source address, in user space.

\*

\* Number of bytes to copy.

\*/

int bytes\_write=0;

copy\_from\_user(onebyte\_data, buf, sizeof(char));

/\* Check the length of the bytes that cannot be written\*/

if(count> sizeof(char))

{

printk(KERN\_ALERT "No space left on device\n");

/\*Return Linux System Error<28>: No space left on device \*/

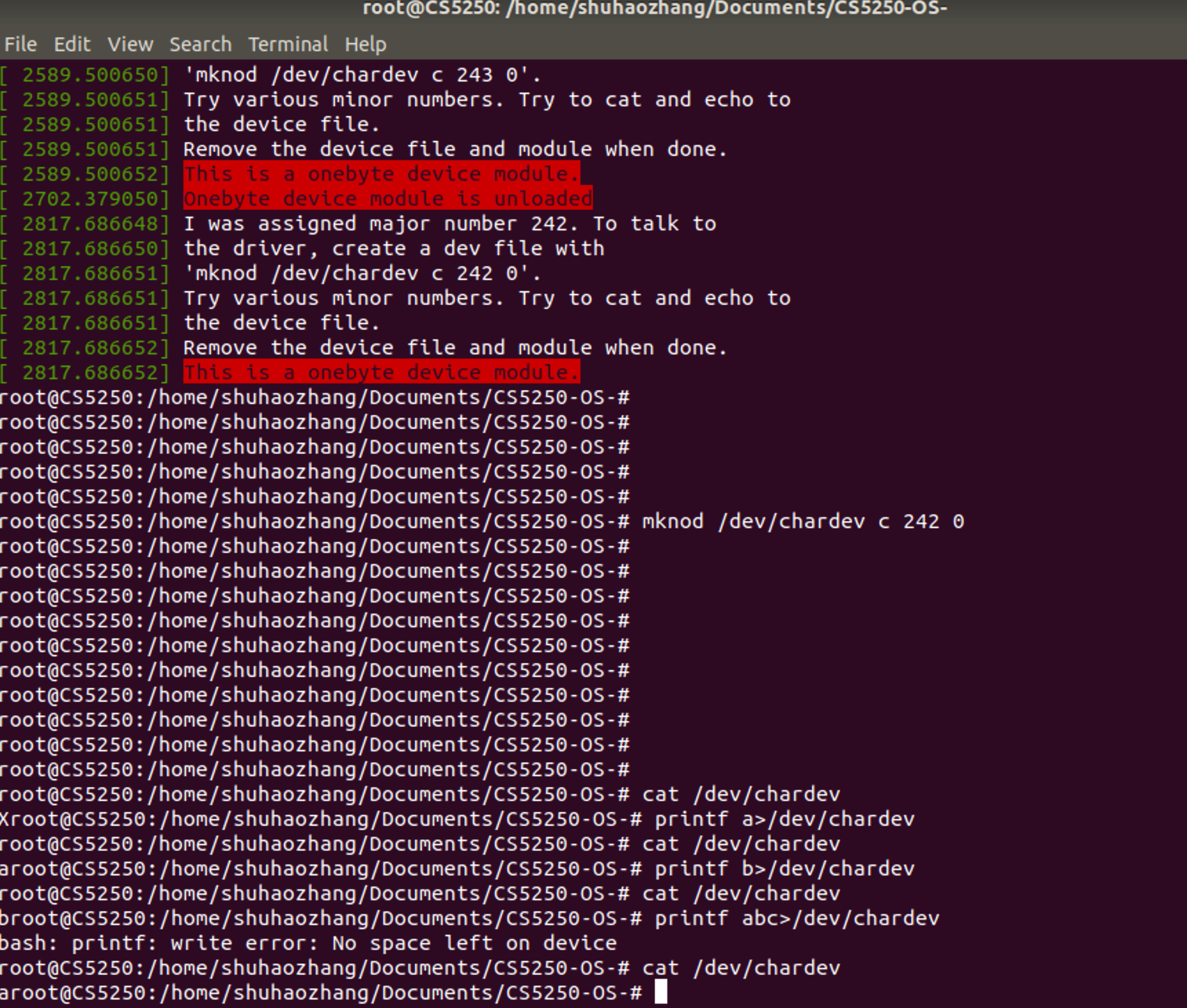
return -ENOSPC;

}

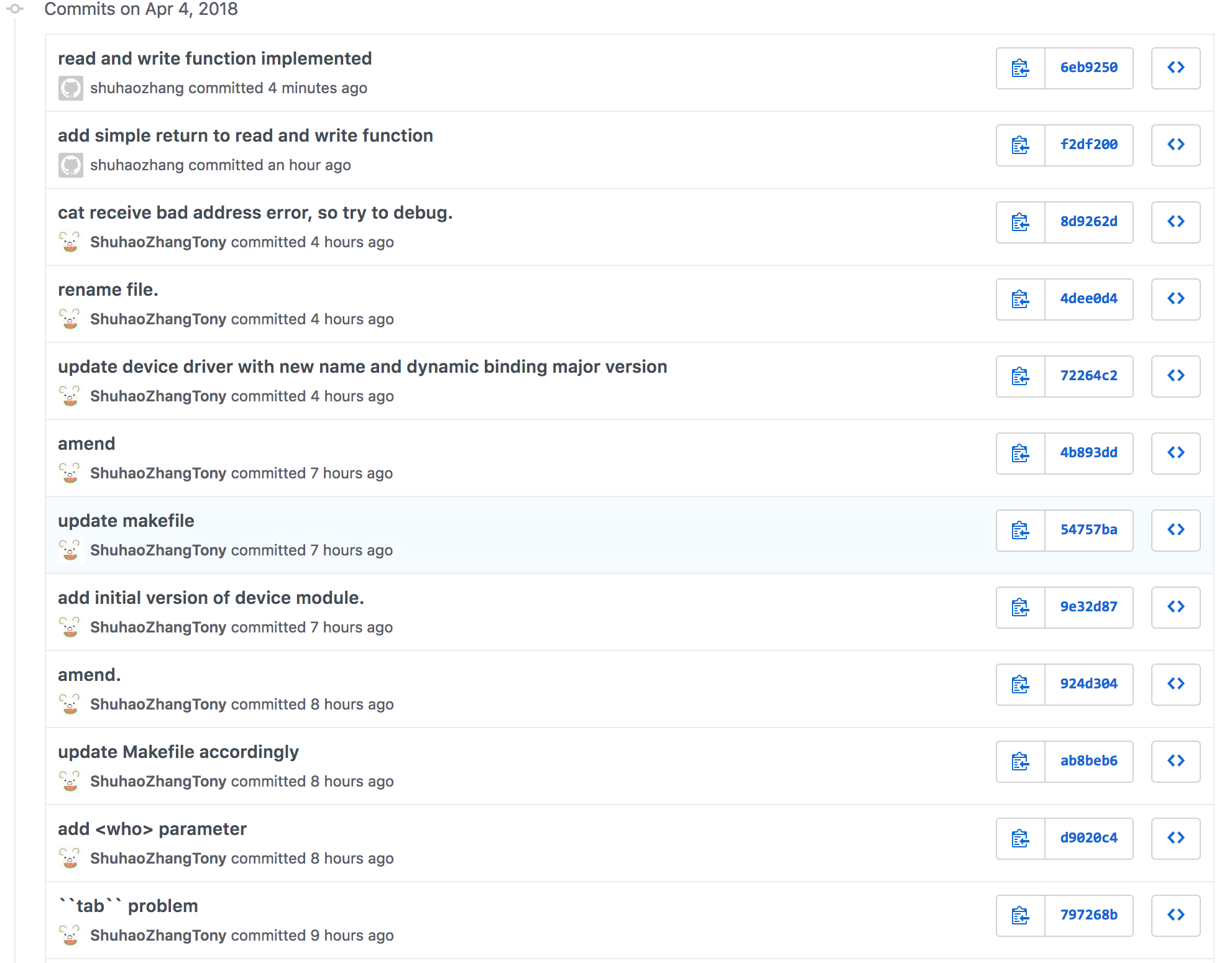
bytes\_write++;

return bytes\_write;

}



The following is the screenshot of github commit. The last two commit is made inside the virtual machine.



APPENDIX

The following are the source code I used in this project, I have highlighted the changes or important code lines in red.

**Chardev.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/uaccess.h>

#define MAJOR\_NUMBER 61/\* forward declaration \*/

static int Major; /\* Major number assigned to our device driver \*/

#define DEVICE\_NAME "chardev" /\* Dev name as it appears in /proc/devices \*/

int onebyte\_open(struct inode \*inode, struct file \*filep);

int onebyte\_release(struct inode \*inode, struct file \*filep);

ssize\_t onebyte\_read(struct file \*filep,

char \*buf, size\_t count,

loff\_t \*f\_pos);

ssize\_t onebyte\_write(struct file \*filep,

const char \*buf, size\_t count,

loff\_t \*f\_pos);

static void onebyte\_exit(void);

/\* definition of file\_operation structure \*/

struct file\_operations onebyte\_fops = {

read: onebyte\_read,

write: onebyte\_write,

open: onebyte\_open,

release: onebyte\_release

};

char \*onebyte\_data = NULL;

/\*

\* Called when a process tries to open the device file, like

\* "cat /dev/mycharfile"

\*/

int onebyte\_open(struct inode \*inode, struct file \*filep)

{

return 0; // always successful

}

int onebyte\_release(struct inode \*inode, struct file \*filep)

{

return 0; // always successful

}

ssize\_t onebyte\_read(struct file \*filep, char \*buf, size\_t count, loff\_t \*f\_pos)

{

/\*please complete the function on your own\*/

int bytes\_read = 0;

/\* Check if the buffer has been written \*/

if(\*buf != 0){

return 0;

}

copy\_to\_user(buf, onebyte\_data, sizeof(char));

bytes\_read ++;

return bytes\_read;

}

ssize\_t onebyte\_write(struct file \*filep, const char \*buf, size\_t count, loff\_t \*f\_pos)

{

/\*please complete the function on your own\*/

/\*\*

\* copy\_from\_user: Returns number of bytes that could not be copied. On success, this will be zero.

\*

\* to

\* Destination address, in kernel space.

\*

\* from

\* Source address, in user space.

\*

\* n

\* Number of bytes to copy.

\*/

int bytes\_write=0;

copy\_from\_user(onebyte\_data, buf, sizeof(char));

/\* Check the length of the bytes that cannot be written\*/

if(count> sizeof(char))

{

printk(KERN\_ALERT "No space left on device\n");

/\*Return Linux System Error<28>: No space left on device \*/

return -ENOSPC;

}

bytes\_write++;

return bytes\_write;

}

static int onebyte\_init(void)

{

// int result;

// register the device

Major = register\_chrdev(0, DEVICE\_NAME, &onebyte\_fops);

if (Major < 0) {

printk(KERN\_ALERT "Registering char device failed with %d\n", Major);

return Major;

}

printk(KERN\_INFO "I was assigned major number %d. To talk to\n", Major);

printk(KERN\_INFO "the driver, create a dev file with\n");

printk(KERN\_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE\_NAME, Major);

printk(KERN\_INFO "Try various minor numbers. Try to cat and echo to\n");

printk(KERN\_INFO "the device file.\n");

printk(KERN\_INFO "Remove the device file and module when done.\n");

// allocate one byte 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.

onebyte\_data = kmalloc(sizeof(char), GFP\_KERNEL);

if (!onebyte\_data) {

onebyte\_exit();

// cannot allocate memory

// return no memory error, negative signify a failure

return -ENOMEM;

}

// // initialize the value to be X

\*onebyte\_data = 'X';

printk(KERN\_ALERT "This is a onebyte device module.\n");

return 0;

}

static void onebyte\_exit(void)

{

// if the pointer is pointing to something

if (onebyte\_data) {

// free the memory and assign the pointer to NULL

kfree(onebyte\_data);

onebyte\_data = NULL;

}

// unregister the device

unregister\_chrdev(MAJOR\_NUMBER, DEVICE\_NAME);

printk(KERN\_ALERT "Onebyte device module is unloaded\n");

}

MODULE\_LICENSE("GPL");

module\_init(onebyte\_init);

module\_exit(onebyte\_exit);

**Makefile**

TARGET\_MODULE:=chardev

obj-m += $(TARGET\_MODULE).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

load:

insmod ./$(TARGET\_MODULE).ko

unload:

rmmod ./$(TARGET\_MODULE).ko

**Hello\_world\_mod**

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/module.h>

#include <linux/moduleparam.h> //for module\_param

MODULE\_LICENSE("GPL");

static char \*who = "world";

module\_param(who, charp, S\_IRUSR|S\_IWUSR);

MODULE\_PARM\_DESC(who, "Tell me your name.");

static int hello\_init(void)

{

printk(KERN\_ALERT "Hello, %s!\n", who);

return 0;

}

static void hello\_exit(void)

{

printk(KERN\_ALERT "Goodbye, %s!\n", who);

}

module\_init(hello\_init);

module\_exit(hello\_exit);

**Makefile**

TARGET\_MODULE:=hello\_world\_mod

obj-m += $(TARGET\_MODULE).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

load:

insmod ./$(TARGET\_MODULE).ko who=($who)

unload:

rmmod ./$(TARGET\_MODULE).ko