Introduction to Operating Systems

Project 3: Dynamically-Loadable Kernel Modules (DLKMs)

0616087 賴沛冠

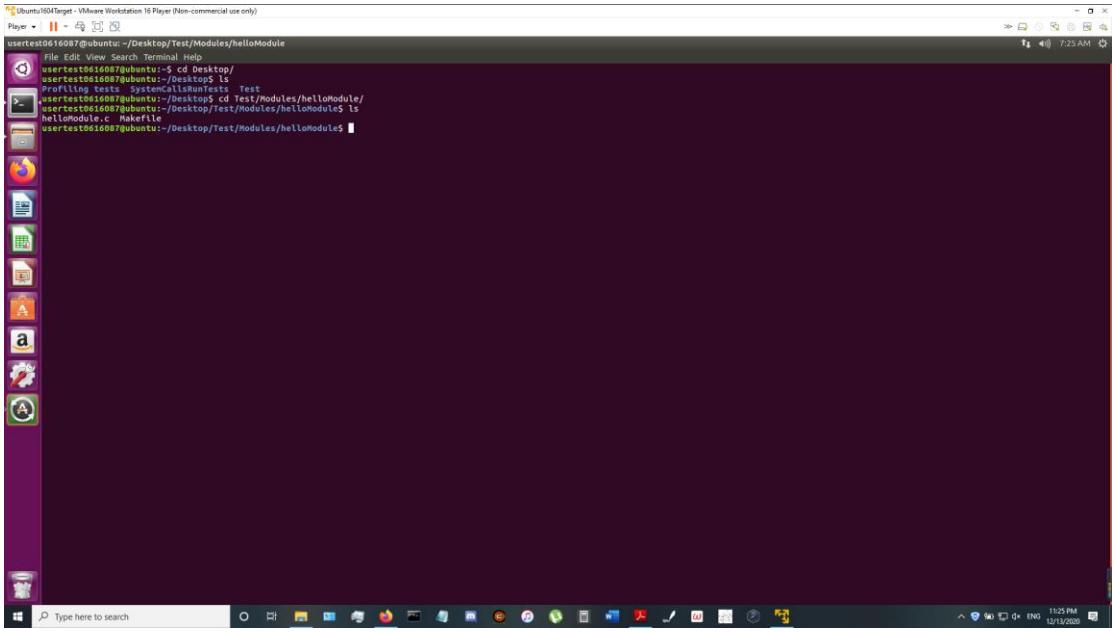
1. Demo video link

https://youtu.be/1pgu1upz2Aw

2. Report

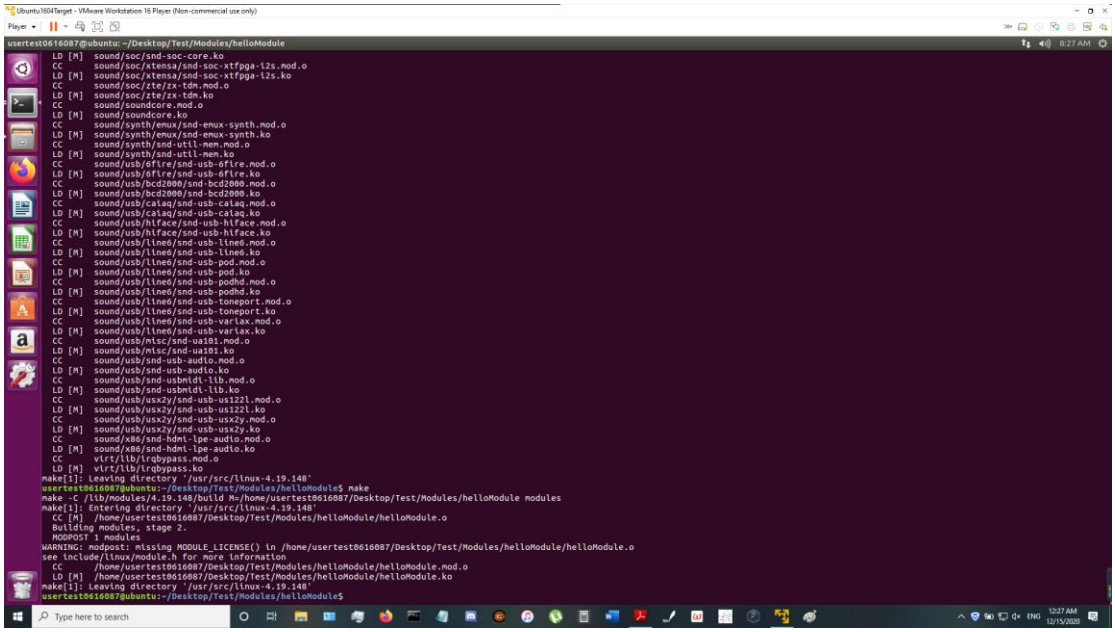
[Screenshot #1: Create a screenshot showing these two files and the folder where you save them.]

As you can see, I created a folder /helloModule under /Desktop/Test/Module, and created two files: helloModule.c and Makefile.



[Screenshot #2: Create a screenshot showing the result of the make process]

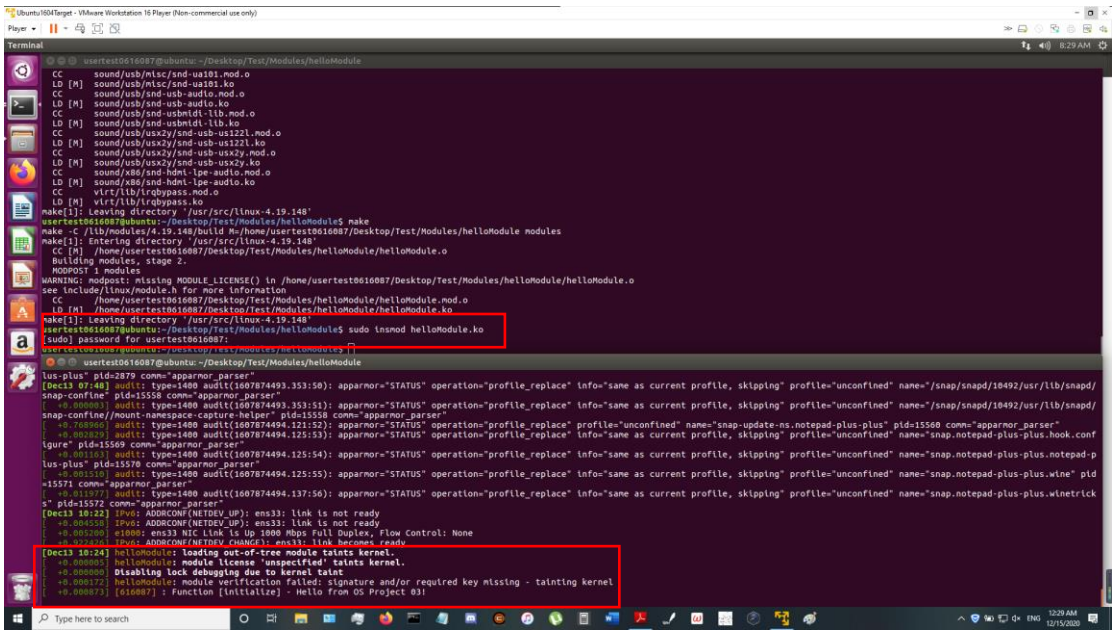
The screenshot shows the result of $make.



[Screenshot #3: Create a screenshot showing both Terminal 1 and 2 when you load the module]

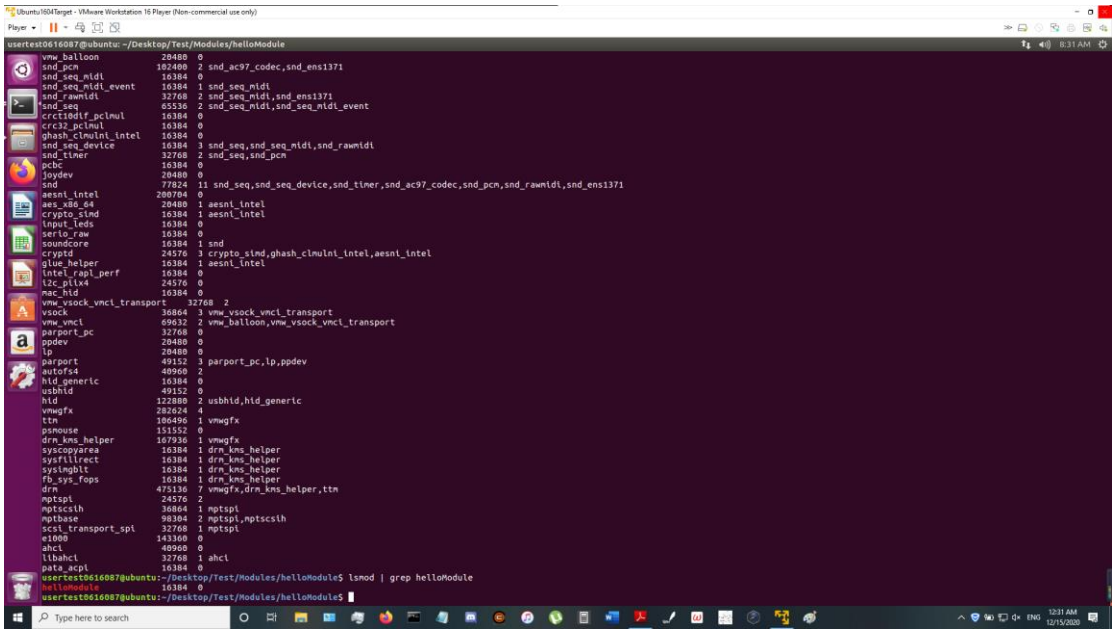
The upside is Terminal 1, and it has installed a module “helloModule.ko” created by making the makefile, with the command $sudo insmod helloModule.ko.

The downside is Terminal 2, it uses $dmesg -wH to print the kernel ring buffer, as you can see there is a sentence “Hello from OS Project 03” means we successfully make the module.



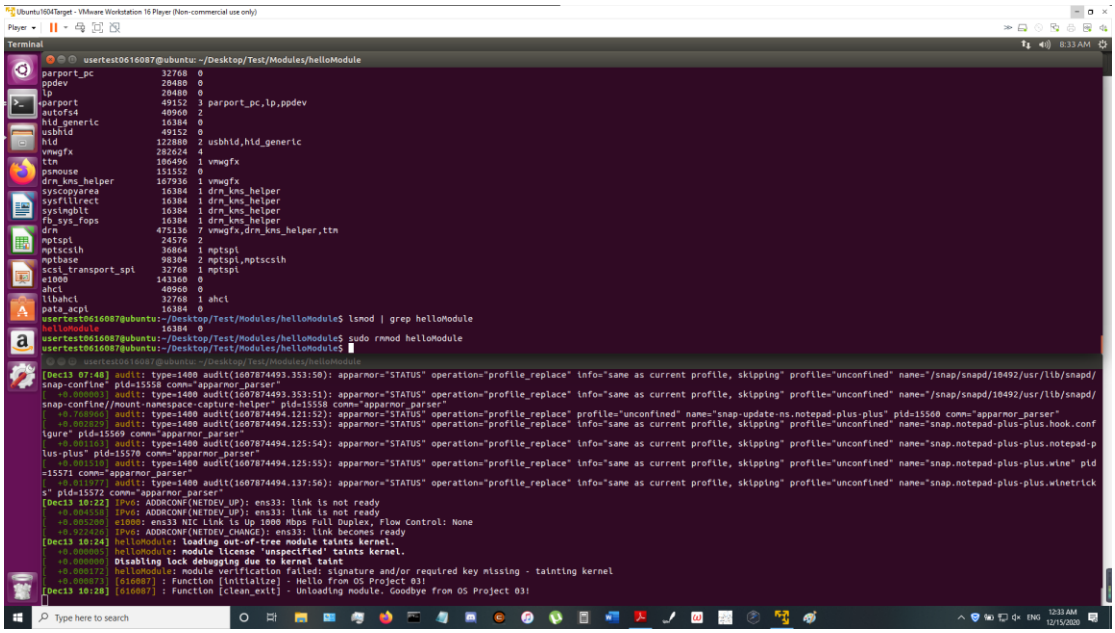
[Screenshot #4: Create a screenshot showing the list of loaded modules from step 8 and 9]

By running $ lsmod or $ lsmod | grep helloModule in Terminal 1, we can see that there is a module called helloModule now in the modules list.



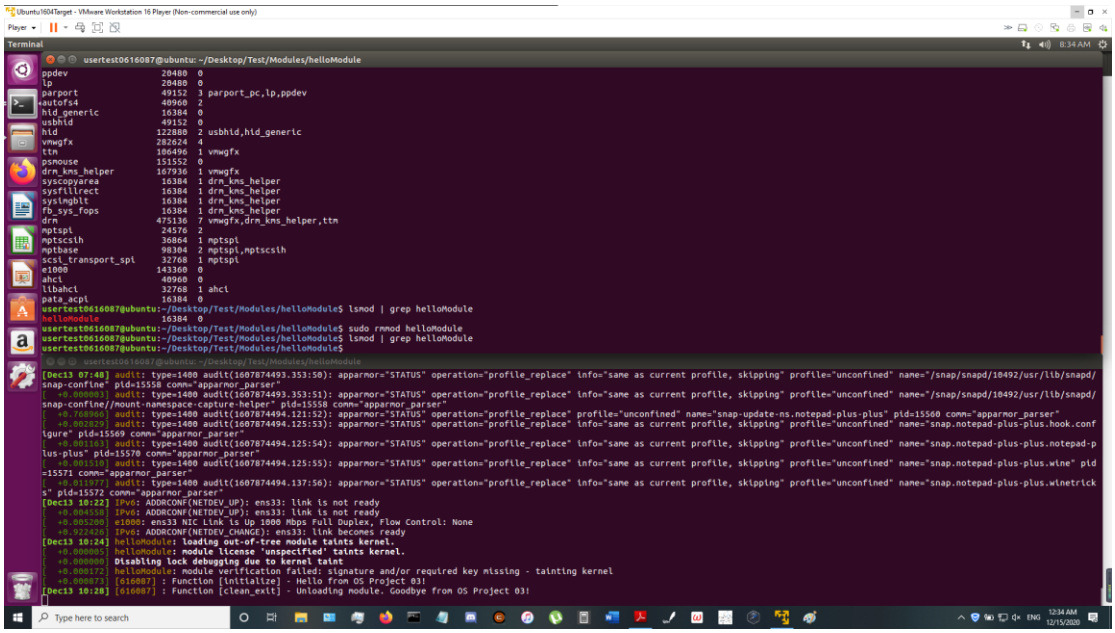
[Screenshot #5: Create a screenshot showing both terminals when unloading the module]

By running $ sudo rmmod helloModule, you can see Terminal 1 worked, and Terminal 2 prints message inside the clean_exit function of the helloModule.c file.

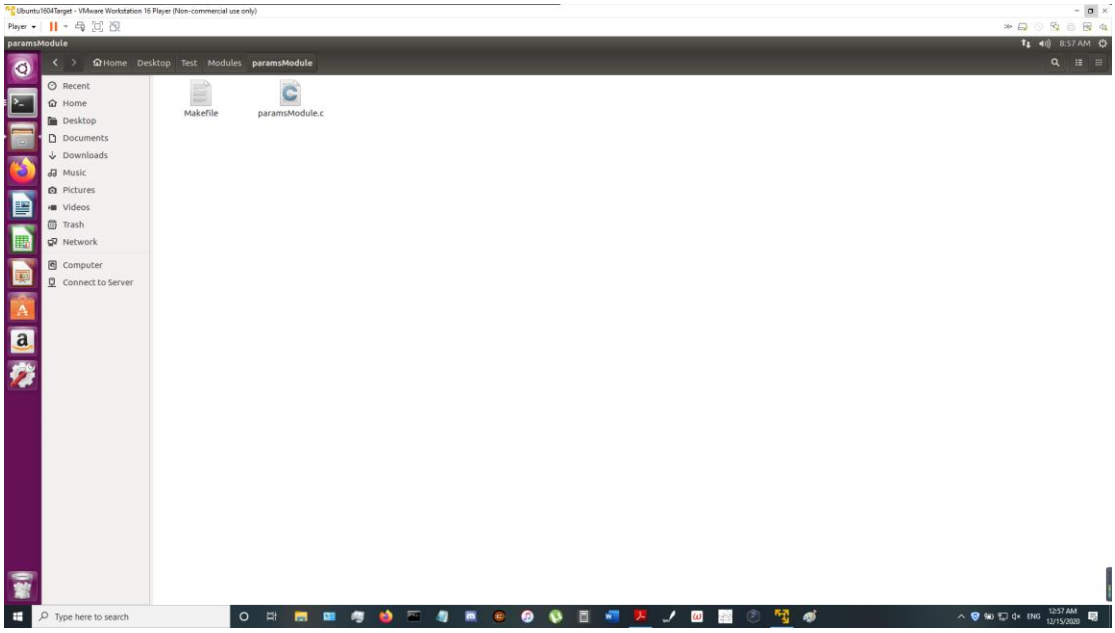


[Screenshot #6: Create a screenshot showing the result of when searching for the module after it was unloaded].

When searching for the module after it was unloaded, we can't see anything because here is no helloModule mounted.

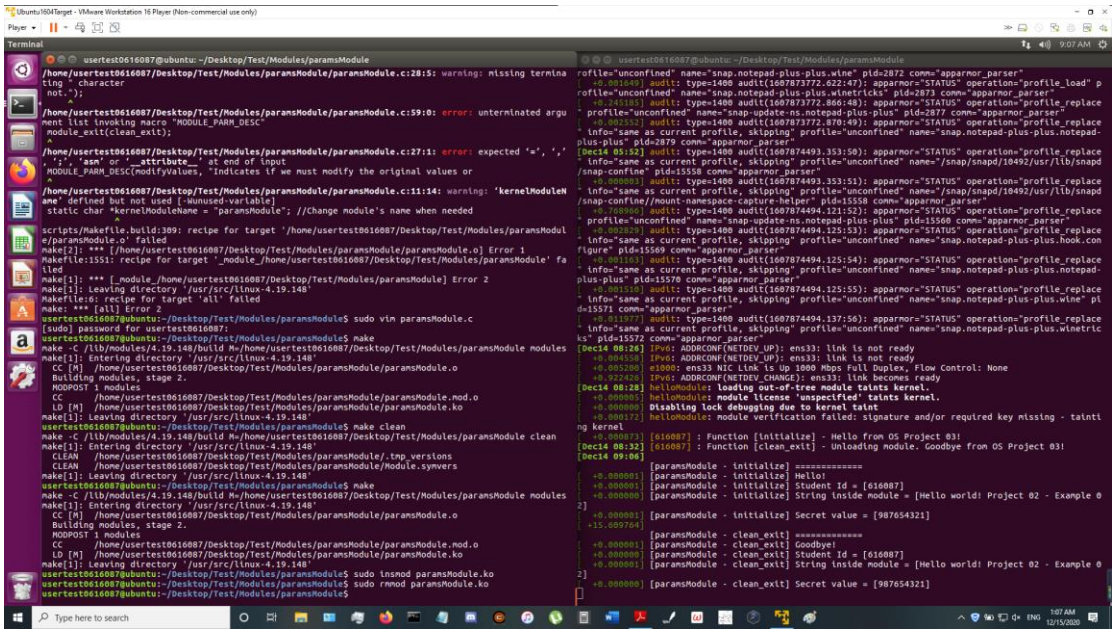


[Screenshot #7: Create a screenshot showing these two files.]



[Screenshot #8: Create a screenshot showing the values you obtain when mounting and unmounting the module, and what commands you ran to do so.]

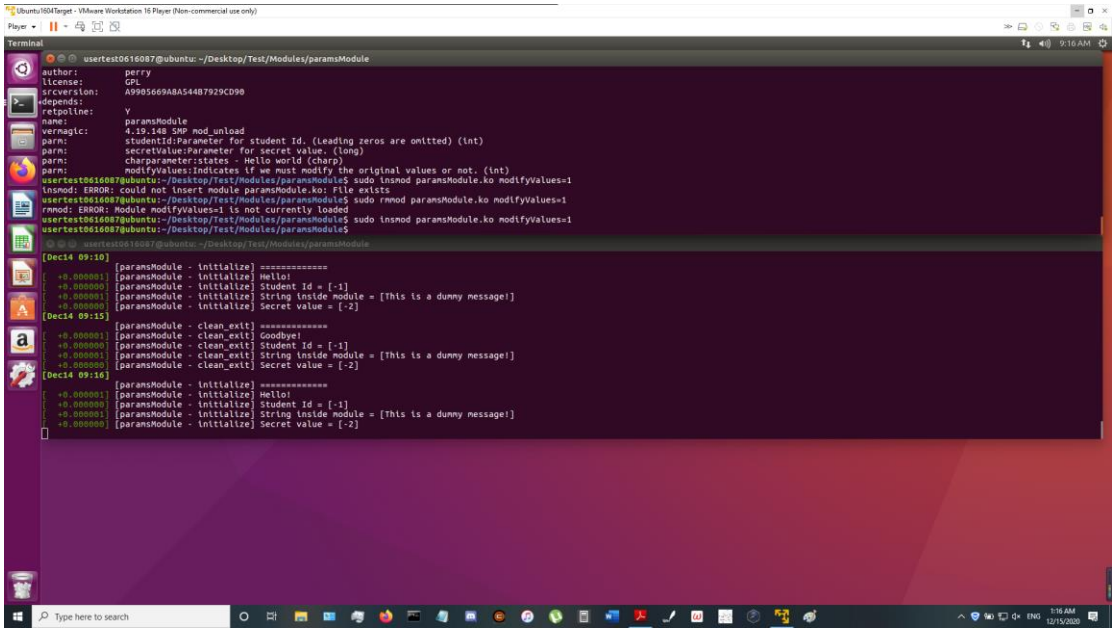
I use $make clean and $make to re-build my module, and $ sudo insmod paramsModule.ko to mount the three values(studentId (616087), secretValue (987654321), and charparameter ("Hello world! Project 02 – Example 02")). Finally, use $ sudo rmmod paramsModule.ko to unmounts the module.



[Screenshot #9-10: Create two screenshots explaining steps 7 and 8.]

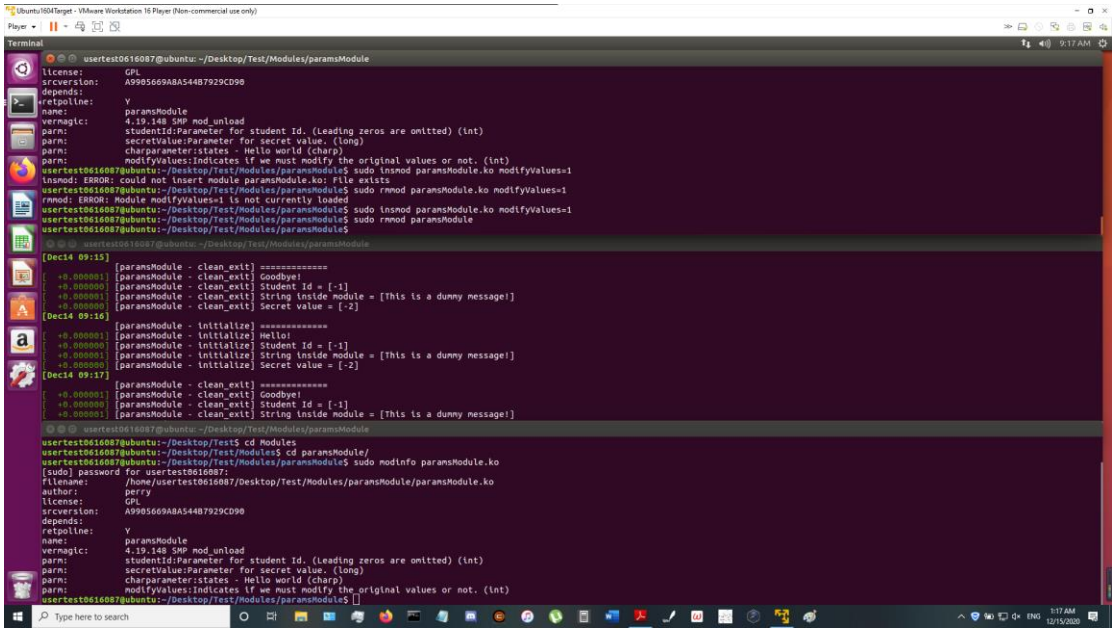
#9

In step 7 we modified the value of modifyValues to 1 by using **$ sudo insmod paramsModule.ko modifyValues=1**, and check weather it changes on Terminal 2.



#10

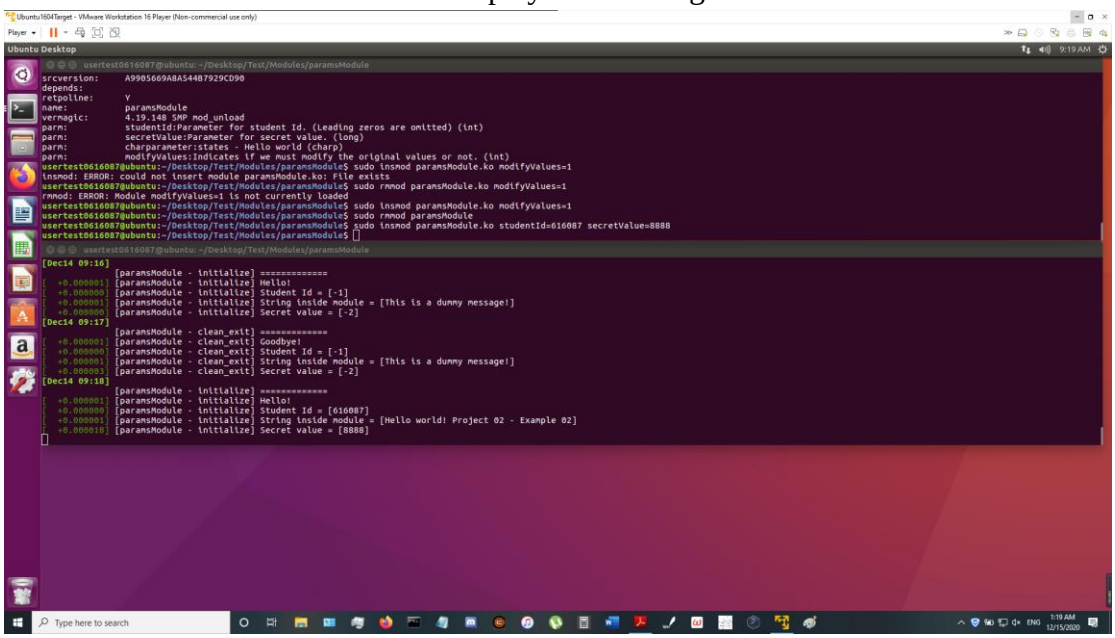
In step 8 we use $ sudo modinfo paramsModule.ko to check the information inside the module.It also shows the datatype of the variables (int, long, charp).
Then unmount the module in Terminal 1, and watch that the variables when the clean_exit function is executed in Terminal 2.



[Screenshot #11-12-13: Create three screenshots explaining steps 9, 10 and 11.]

#11

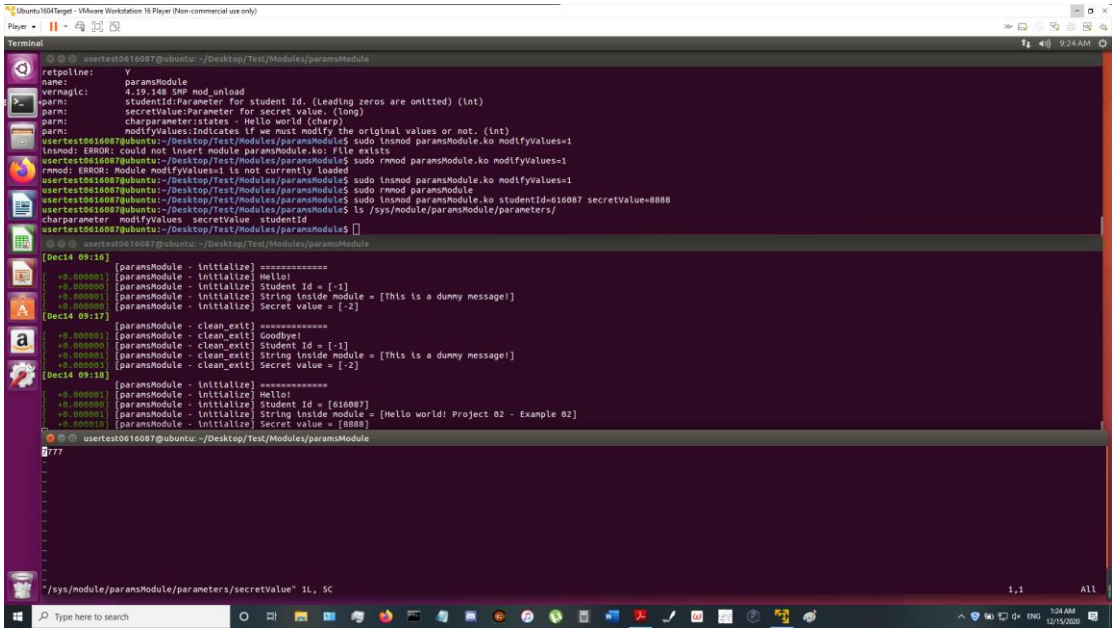
In step 9 we mount again the module in Terminal 1 with values studentId=[616087] and secretValue=8888. Terminal 2 displays the messages with those values.



#12

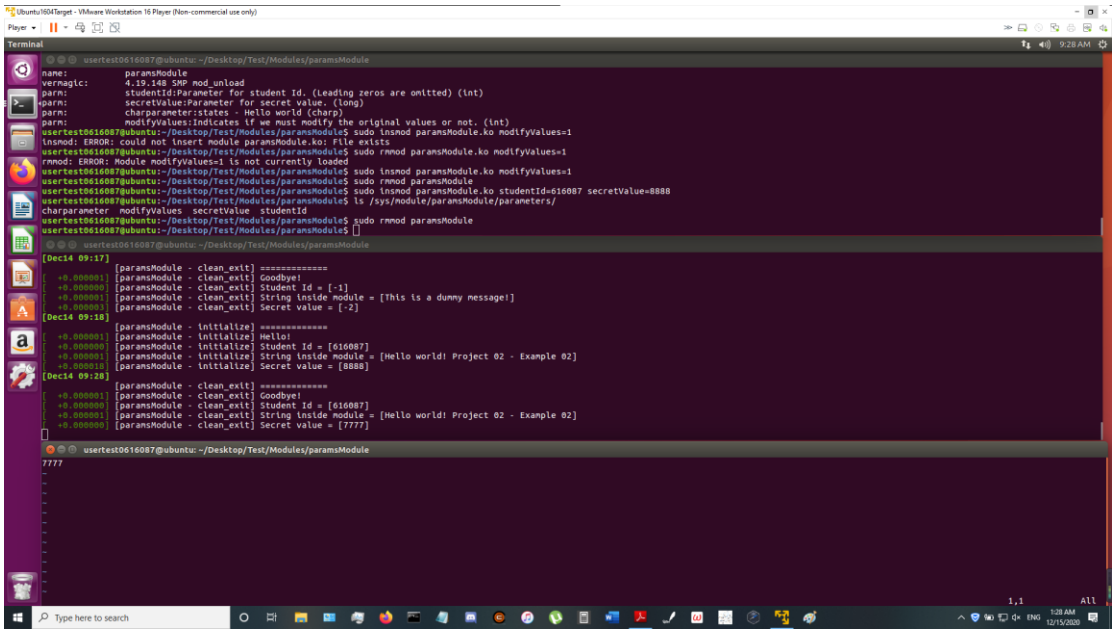
In step 10 we can see Linux manages module variables as files. When the module is mount, we can find them in /sys/module/paramsModule/parameters

Ans in Terminal 3, we can modify the value of those variables with $ sudo vim /sys/module/paramsModule/parameters/secretValue and make secretValue=7777.



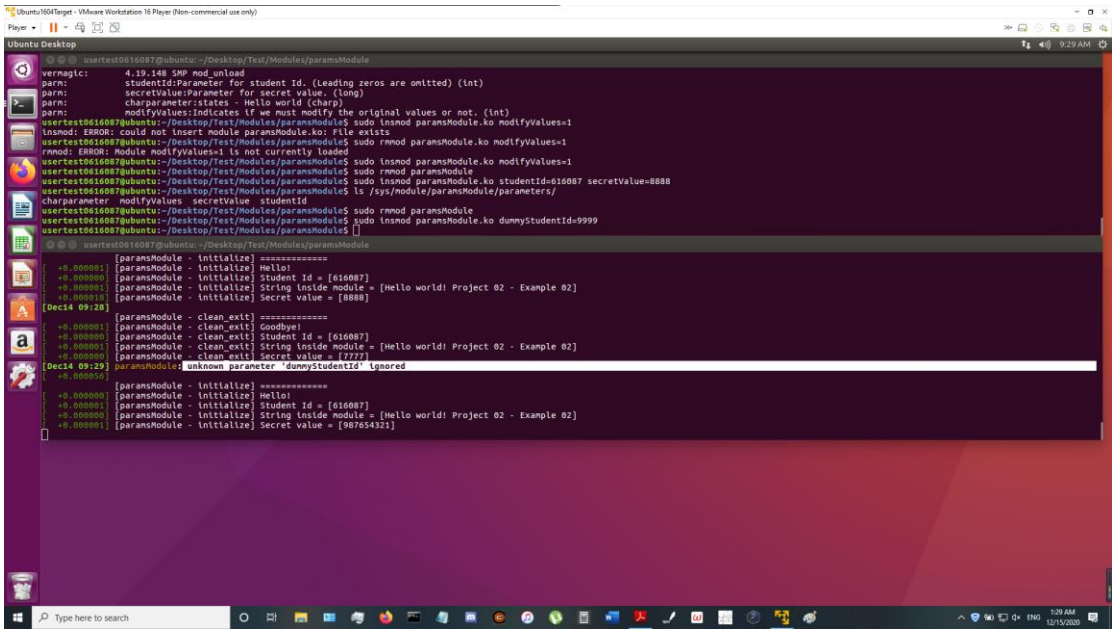
#13

In step 11, that the secretValue file has been modified, when we unload the module in Terminal 1, we will see the new value in Terminal 2



[Screenshot #14: Create a screenshot explaining step 12.]

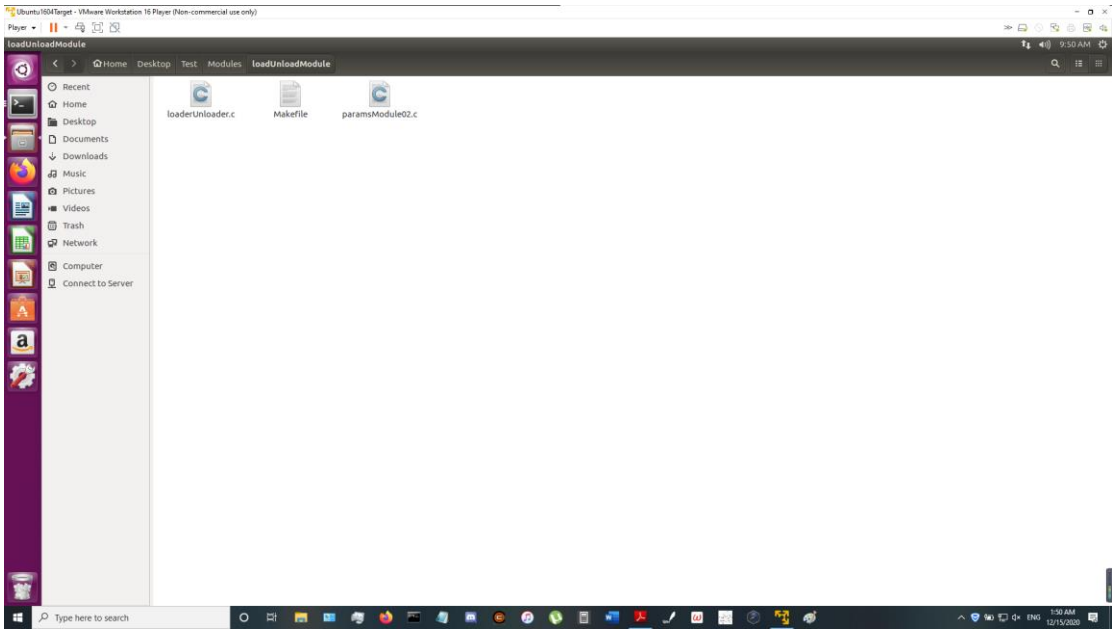
In step 12, the reason **paramsModule: unknown parameter 'dummyStudentId' ignored** appears is because there is no parameter called dummyStudentId, so the kernel will ignore it and print the original values.



[Screenshot #15-16: Create two screenshots showing the files above in their folder and EXPLAIN the loaderUnloader.c file]

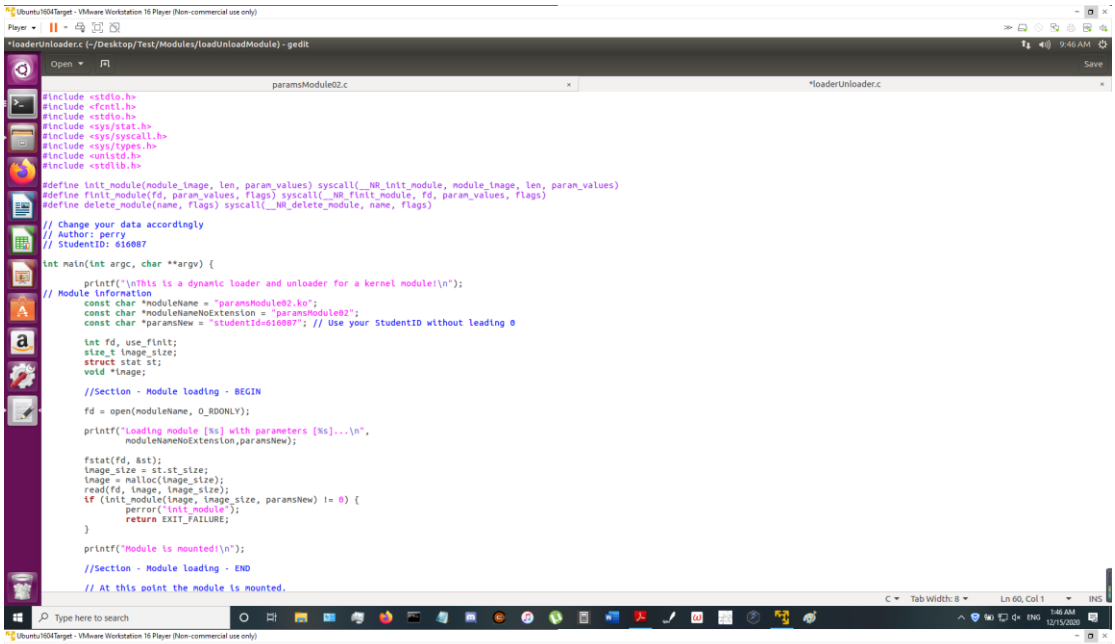
#15

Files above in their folder

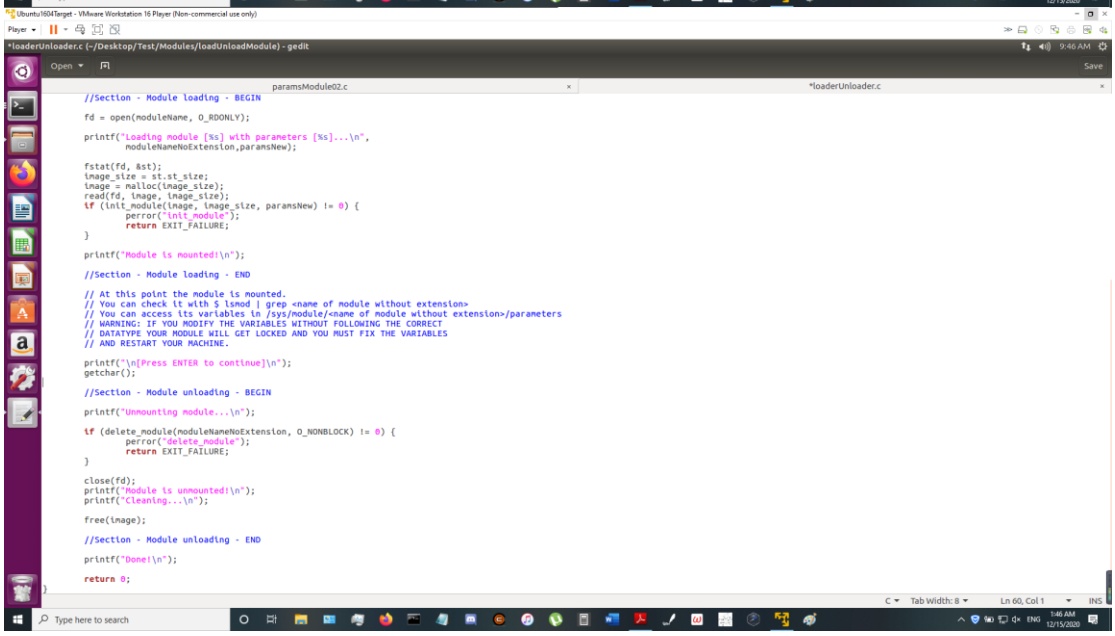


#16

The loaderUnloader.c is like putting parameters **paramsNew** into init_module, moduleName into fd, use init_module to deliver paramsNew into module and mount the module. After that we close(fd) to unmounts the module.



```c
#include <stdio.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

#define init_module(module_image, len, param_values) syscall(__NR_init_module, module_image, len, param_values)
#define finit_module(fd, param_values, flags) syscall(__NR_finit_module, fd, param_values, flags)
#define delete_module(name, flags) syscall(__NR_delete_module, name, flags)

// Change your data accordingly
// Author: perry
// StudentID: 616087

int main(int argc, char **argv) {

        printf("\nThis is a dynamic loader and unloader for a kernel module!\n");
// Module information
        const char *moduleName = "paramsModule02.ko";
        const char *moduleNameNoExtension = "paramsModule02";
        const char *paramsNew = "studentId=616087"; // Use your StudentID without leading 0

        int fd, use_finit;
        size_t image_size;
        struct stat st;
        void *image;

        //Section - Module loading - BEGIN

        fd = open(moduleName, O_RDONLY);

        printf("Loading module [%s] with parameters [%s]...\n",
                moduleNameNoExtension,paramsNew);

        fstat(fd, &st);
        image_size = st.st_size;
        image = malloc(image_size);
        read(fd, image, image_size);
        if (init_module(image, image_size, paramsNew) != 0) {
                perror("init_module");
                return EXIT_FAILURE;
        }

        printf("Module is mounted!\n");

        //Section - Module loading - END

        // At this point the module is mounted.
```

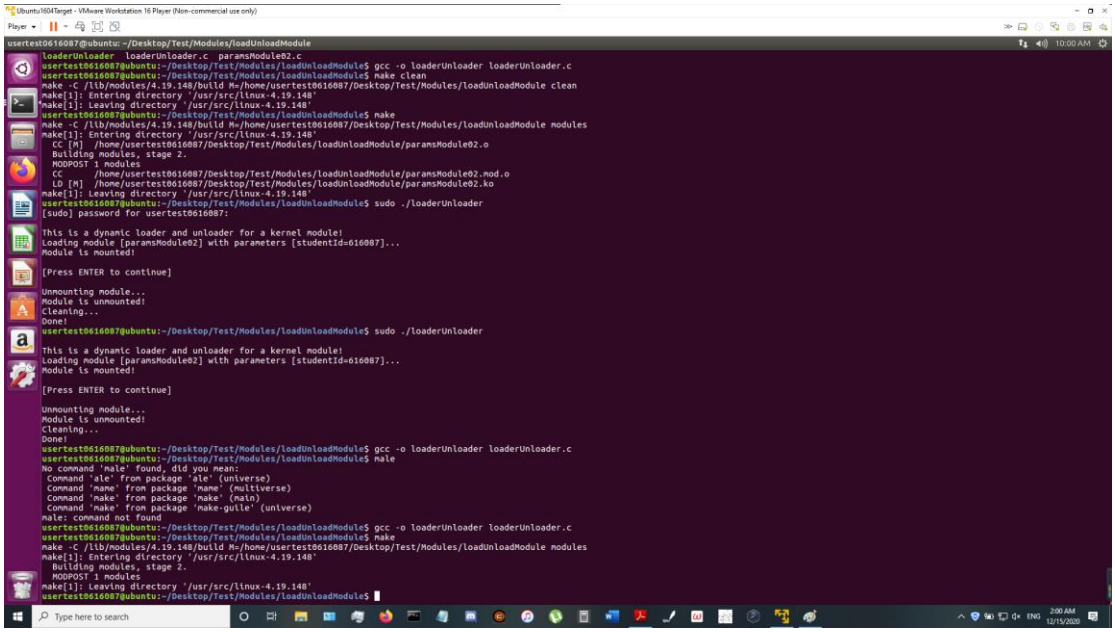


```c
        //Section - Module loading - BEGIN

        fd = open(moduleName, O_RDONLY);

        printf("Loading module [%s] with parameters [%s]...\n",
                moduleNameNoExtension,paramsNew);

        fstat(fd, &st);
        image_size = st.st_size;
        image = malloc(image_size);
        read(fd, image, image_size);
        if (init_module(image, image_size, paramsNew) != 0) {
                perror("init_module");
                return EXIT_FAILURE;
        }

        printf("Module is mounted!\n");

        //Section - Module loading - END

        // At this point the module is mounted.
        // You can check it with $ lsmod | grep <name of module without extension>
        // You can access its variables in /sys/module/<name of module without extension>/parameters
        // WARNING: IF YOU MODIFY THE VARIABLES WITHOUT FOLLOWING THE CORRECT
        // DATATYPE YOUR MODULE WILL GET LOCKED AND YOU MUST FIX THE VARIABLES
        // AND RESTART YOUR MACHINE.

        printf("\n[Press ENTER to continue]\n");
        getchar();

        //Section - Module unloading - BEGIN

        printf("Unmounting module...\n");

        if (delete_module(moduleNameNoExtension, O_NONBLOCK) != 0) {
                perror("delete_module");
                return EXIT_FAILURE;
        }

        close(fd);
        printf("Module is unmounted!\n");
        printf("Cleaning...\n");

        free(image);

        //Section - Module unloading - END

        printf("Done!\n");

        return 0;
}
```

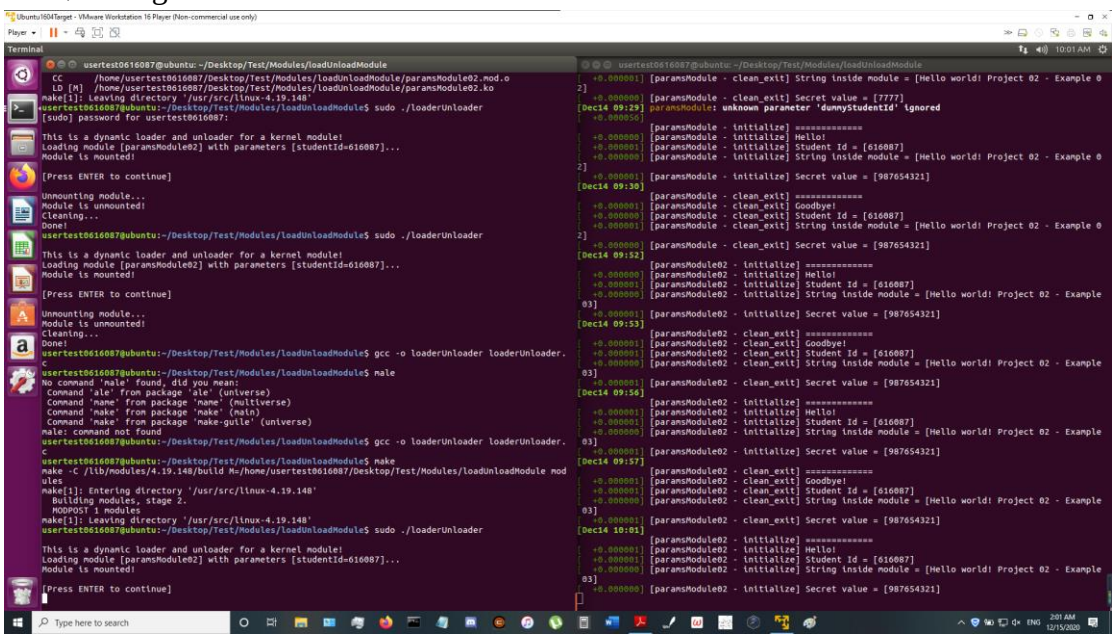
[Screenshot #17-18-19-20: Create four screenshots explaining steps 4,5,6,7,8.]

#17 ompile loaderUnloader.c with **$ gcc -o loaderUnloader loaderUnloader.c.**

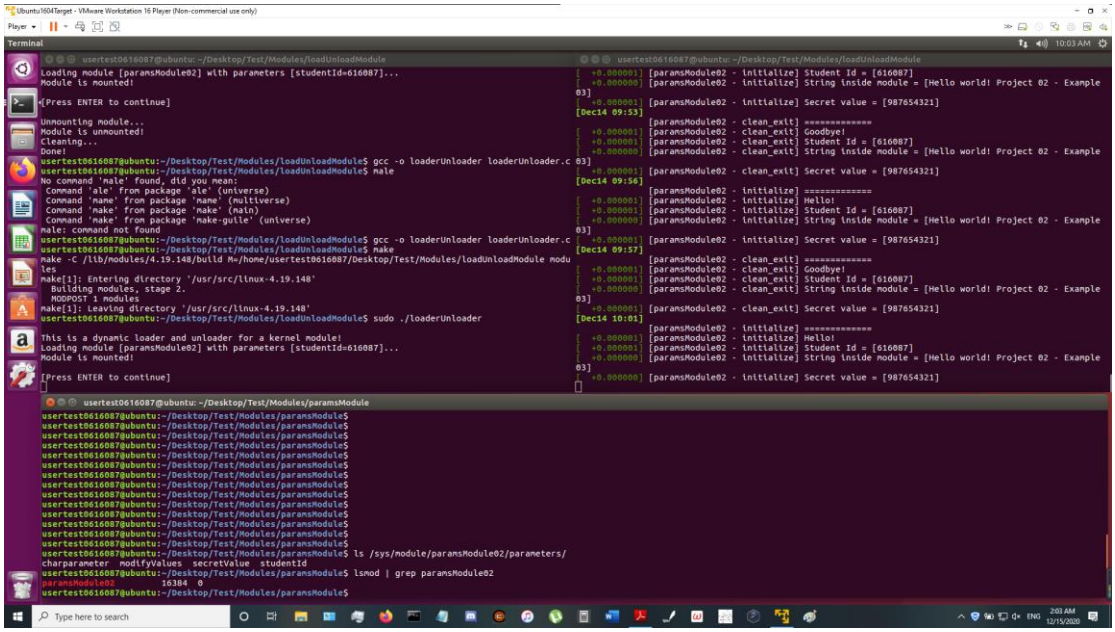
And then make the module.



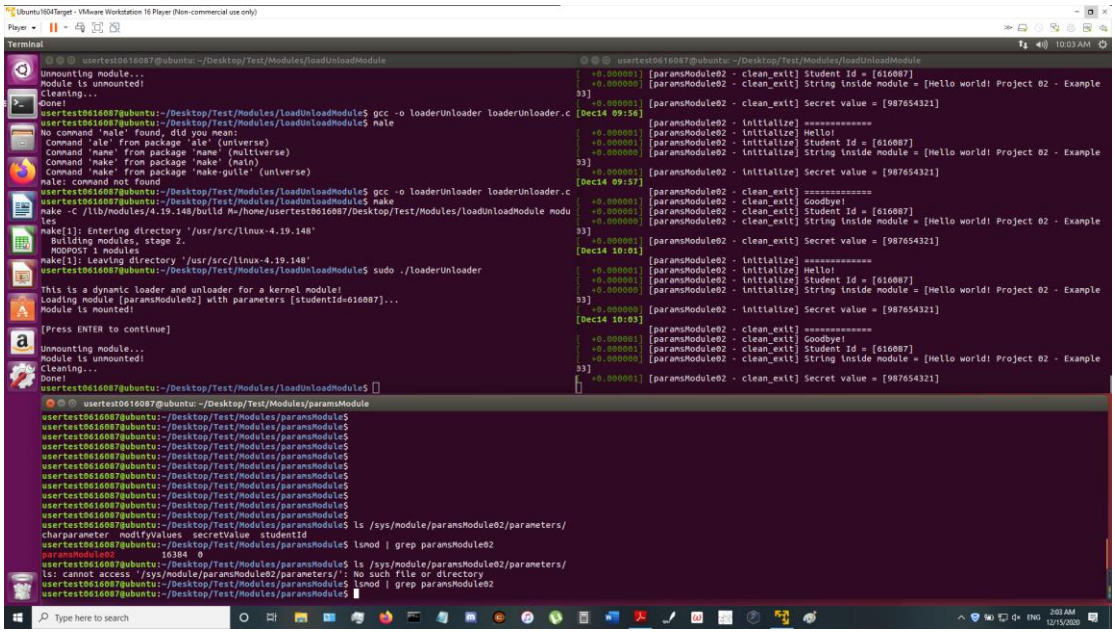
#18 $ dmesg –wH in Terminal 2



#19 $ sudo ./loaderUnloader. We can see the parameters shows on the screen. At this point the module has been loaded with studentId=9999. Because the module is loaded, we can read the variables inside it in Terminal 3

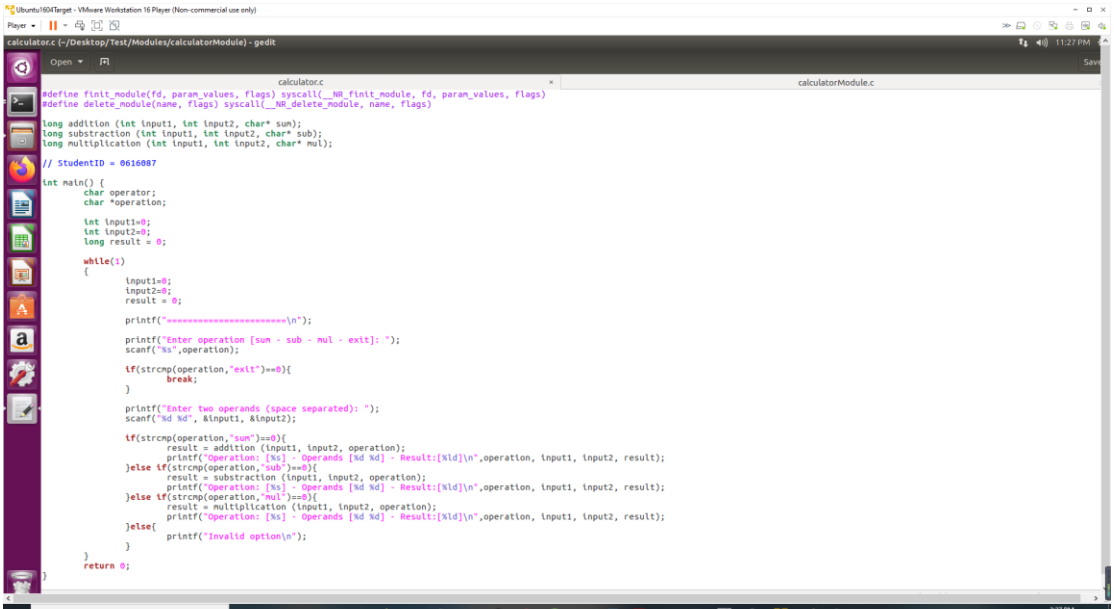


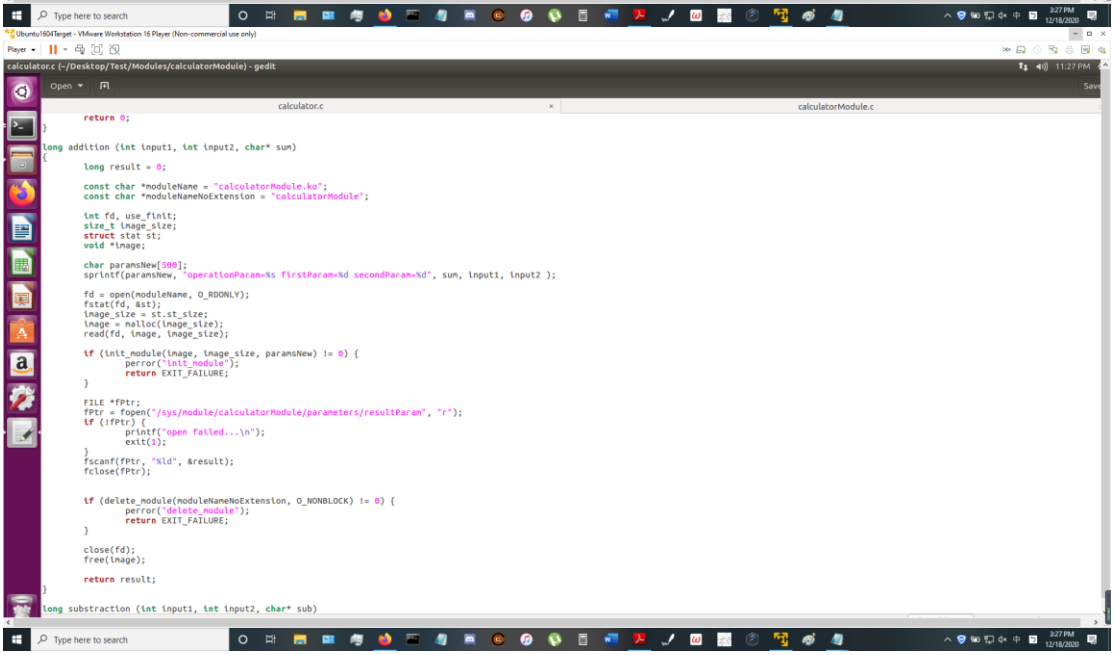
#20 In Terminal 1 press ENTER. This will continue the execution of ./loaderUnloader, and it will proceed to unload the module. So In Terminal 2 we will see that the module was unloaded. And in Terminal 3 you will see that you cannot access the module's folder in /sys/ anymore and the module has been removed from the modules list.



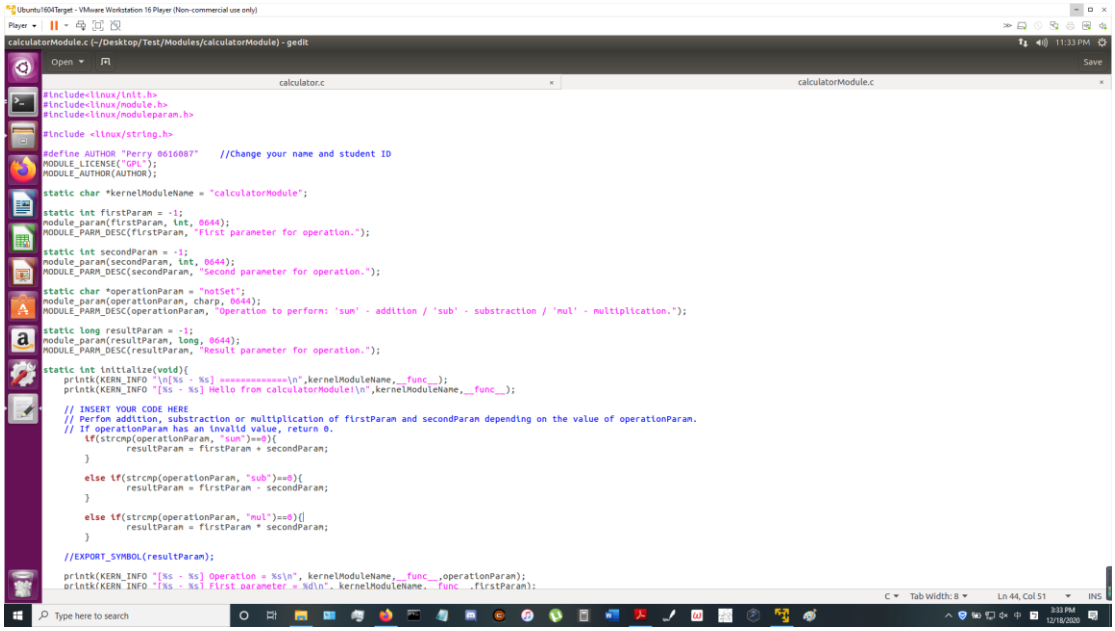
[Screenshot #21-22-23-24: Create four screenshots explaining how you solve this problem.]

In screenshot#21-22 is my calculator.c. I add a veriable in every function call, and use sprintf to put string in paramsNew, then put the parameters into module by using init_module. When one input “operator” “input1” “input2”, it will mount and deliver to calculatorModule.c and put into /sys/module/calculatorModule/parameters. The module would calculate and return results into returnParam, then calculator.c would read the files and print value inside resultParam. After that the module will dynamically unmount.





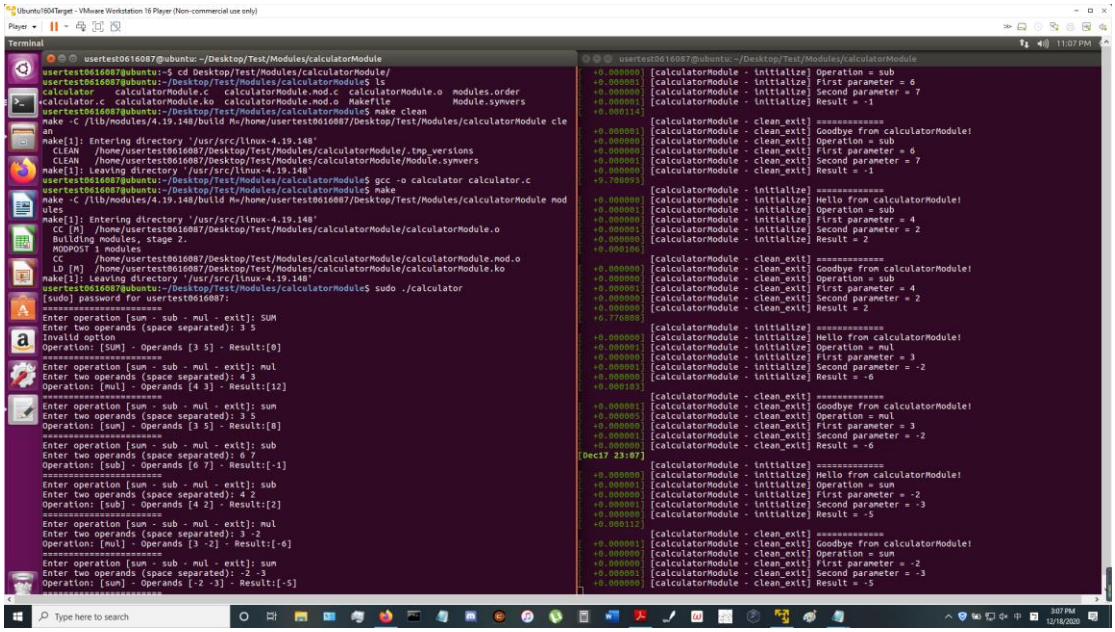
#23 This is my calculatorModule.c. I use strcmp to identify the operation and calculate the result. Because the values (firstParam, secondParam, operationParam) has been modified by init_module, the result=firstParam(operation)secondParam, and finish the calculation.



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

#include <linux/string.h>

#define AUTHOR "Perry 0616087"    //Change your name and student ID
MODULE_LICENSE("GPL");
MODULE_AUTHOR(AUTHOR);

static char *kernelModuleName = "calculatorModule";

static int firstParam = -1;
module_param(firstParam, int, 0644);
MODULE_PARM_DESC(firstParam, "first parameter for operation.");

static int secondParam = -1;
module_param(secondParam, int, 0644);
MODULE_PARM_DESC(secondParam, "Second parameter for operation.");

static char *operationParam = "notSet";
module_param(operationParam, charp, 0644);
MODULE_PARM_DESC(operationParam, "Operation to perform: 'sum' - addition / 'sub' - substraction / 'mul' - multiplication.");

static long resultParam = -1;
module_param(resultParam, long, 0644);
MODULE_PARM_DESC(resultParam, "Result parameter for operation.");

static int initialize(void){
    printk(KERN_INFO "\n[%s - %s] ============\n",kernelModuleName,__func__);
    printk(KERN_INFO "[%s - %s] Hello from calculatorModule!\n",kernelModuleName,__func__);

    // INSERT YOUR CODE HERE
    // Perfon addition, substraction or multiplication of firstParam and secondParan depending on the value of operationParam.
    // If operationParam has an invalid value, return 0.
        if(strcmp(operationParan, "sum")==0){
                resultParam = firstParan + secondParan;
        }

        else if(strcmp(operationParan, "sub")==0){
                resultParam = firstParan - secondParan;
        }

        else if(strcmp(operationParan, "mul")==0){
                resultParam = firstParan * secondParan;
        }

    //EXPORT_SYMBOL(resultParam);

    printk(KERN_INFO "[%s - %s] Operation = %s\n", kernelModuleName,__func__,operationParan);
    printk(KERN_INFO "[%s - %s] First parameter = %d\n", kernelModuleName, __func__,firstParam);
```

#24

This screenshot is my demo result. As you can see every time I enter operation and inputs, the module will mount automatically, and after the calculate is over, the result will output, and the module will unmounts, and finish this project.



```
usertest0616087@ubuntu:~$ cd Desktop/Test/Modules/calculatorModule/
usertest0616087@ubuntu:~/Desktop/Test/Modules/calculatorModule$ ls
calculator   calculatorModule.c   calculatorModule.mod.c  calculatorModule.o  modules.order
calculator.c  calculatorModule.ko  calculatorModule.mod.o  Makefile        Module.symvers
usertest0616087@ubuntu:~/Desktop/Test/Modules/calculatorModule$ make clean
make -C /lib/modules/4.19.148/build M=/home/usertest0616087/Desktop/Test/Modules/calculatorModule cle
an
make[1]: Entering directory '/usr/src/linux-4.19.148'
  CLEAN   /home/usertest0616087/Desktop/Test/Modules/calculatorModule/.tmp_versions
  CLEAN   /home/usertest0616087/Desktop/Test/Modules/calculatorModule/Module.symvers
make[1]: Leaving directory '/usr/src/linux-4.19.148'
usertest0616087@ubuntu:~/Desktop/Test/Modules/calculatorModule$ gcc -o calculator calculator.c
usertest0616087@ubuntu:~/Desktop/Test/Modules/calculatorModule$ make
make -C /lib/modules/4.19.148/build M=/home/usertest0616087/Desktop/Test/Modules/calculatorModule mod
ules
make[1]: Entering directory '/usr/src/linux-4.19.148'
  CC [M]  /home/usertest0616087/Desktop/Test/Modules/calculatorModule/calculatorModule.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/usertest0616087/Desktop/Test/Modules/calculatorModule/calculatorModule.mod.o
  LD [M]  /home/usertest0616087/Desktop/Test/Modules/calculatorModule/calculatorModule.ko
make[1]: Leaving directory '/usr/src/linux-4.19.148'
usertest0616087@ubuntu:~/Desktop/Test/Modules/calculatorModule$ sudo ./calculator
[sudo] password for usertest0616087:
======================
Enter operation [sum - sub - mul - exit]: SUM
Enter two operands (space separated): 3 5
Invalid option
Operation: [SUM] - Operands [3 5] - Result:[0]
======================
Enter operation [sum - sub - mul - exit]: mul
Enter two operands (space separated): 4 3
Operation: [mul] - Operands [4 3] - Result:[12]
======================
Enter operation [sum - sub - mul - exit]: sum
Enter two operands (space separated): 3 5
Operation: [sum] - Operands [3 5] - Result:[8]
======================
Enter operation [sum - sub - mul - exit]: sub
Enter two operands (space separated): 6 7
Operation: [sub] - Operands [6 7] - Result:[-1]
======================
Enter operation [sum - sub - mul - exit]: sub
Enter two operands (space separated): 4 2
Operation: [sub] - Operands [4 2] - Result:[2]
======================
Enter operation [sum - sub - mul - exit]: mul
Enter two operands (space separated): 3 -2
Operation: [mul] - Operands [3 -2] - Result:[-6]
======================
Enter operation [sum - sub - mul - exit]: sum
Enter two operands (space separated): -2 -3
Operation: [sum] - Operands [-2 -3] - Result:[-5]
======================
```

```
+0.000000] [calculatorModule - initialize] Operation = sub
+0.000001] [calculatorModule - initialize] First parameter = 6
+0.000000] [calculatorModule - initialize] Second parameter = 7
+0.000001] [calculatorModule - initialize] Result = -1
+0.000114]

[calculatorModule - clean_exit] ============
+0.000001] [calculatorModule - clean_exit] Goodbye from calculatorModule!
+0.000000] [calculatorModule - clean_exit] Operation = sub
+0.000000] [calculatorModule - clean_exit] First parameter = 6
+0.000001] [calculatorModule - clean_exit] Second parameter = 7
+0.000000] [calculatorModule - clean_exit] Result = -1
+9.788093]

[calculatorModule - initialize] ============
+0.000000] [calculatorModule - initialize] Hello from calculatorModule!
+0.000001] [calculatorModule - initialize] Operation = sub
+0.000000] [calculatorModule - initialize] First parameter = 4
+0.000001] [calculatorModule - initialize] Second parameter = 2
+0.000000] [calculatorModule - initialize] Result = 2
+0.000106]

[calculatorModule - clean_exit] ============
+0.000000] [calculatorModule - clean_exit] Goodbye from calculatorModule!
+0.000000] [calculatorModule - clean_exit] Operation = sub
+0.000001] [calculatorModule - clean_exit] First parameter = 4
+0.000000] [calculatorModule - clean_exit] Second parameter = 2
+0.000000] [calculatorModule - clean_exit] Result = 2
+6.776808]

[calculatorModule - initialize] ============
+0.000000] [calculatorModule - initialize] Hello from calculatorModule!
+0.000001] [calculatorModule - initialize] Operation = mul
+0.000000] [calculatorModule - initialize] First parameter = 3
+0.000001] [calculatorModule - initialize] Second parameter = -2
+0.000000] [calculatorModule - initialize] Result = -6
+0.000103]

[calculatorModule - clean_exit] ============
+0.000001] [calculatorModule - clean_exit] Goodbye from calculatorModule!
+0.000005] [calculatorModule - clean_exit] Operation = mul
+0.000000] [calculatorModule - clean_exit] First parameter = 3
+0.000001] [calculatorModule - clean_exit] Second parameter = -2
+0.000000] [calculatorModule - clean_exit] Result = -6
[Dec17 23:07]
[calculatorModule - initialize] ============
+0.000000] [calculatorModule - initialize] Hello from calculatorModule!
+0.000001] [calculatorModule - initialize] Operation = sun
+0.000000] [calculatorModule - initialize] First parameter = -2
+0.000001] [calculatorModule - initialize] Second parameter = -3
+0.000000] [calculatorModule - initialize] Result = -5
+0.000112]

[calculatorModule - clean_exit] ============
+0.000001] [calculatorModule - clean_exit] Goodbye from calculatorModule!
+0.000000] [calculatorModule - clean_exit] Operation = sun
+0.000000] [calculatorModule - clean_exit] First parameter = -2
+0.000001] [calculatorModule - clean_exit] Second parameter = -3
+0.000000] [calculatorModule - clean_exit] Result = -5
```

Part2-Q&A

1. **What is a static kernel module? What is a dynamic kernel module? What is the other name of a dynamic kernel module? What are the differences between system calls and dynamic kernel modules (mention at least 3)?**

Ans:

- Static kernel modules are those which are compiled as part of the base kernel and it is available at any time.
- Dynamic kernel modules are compiled as modules separately and loaded based on user demand.
- Dynamic kernel module is also called as Loadable Kernel Module(LKM)
- System call is a function in the kernel, and is visible for users. Also a user need to execute a syscall to get services from kernel. While dynamic kernel module is an object file that can be insert into kernel. It's used for expanding the kerne's functionality, also it don't need to re-compile the kernel to add new dynamic kernel module.

2. **Why does adding a system call require kernel re-compilation, while adding a kernel module does not?**

Ans:

Because it's not possible to hot-patch a new syscall into a running kernel. The only way of properly doing so is by getting the kernel source, modifying it, configuring it, and then re-compiling it as a brand new kernel.

However, kernel modules cannot add new system calls, so build and load kernel modules don't need to recompiling the kernel.

3. **What are the commands insmod, rmmod and modinfo for? How do you use them? (Write how would you use them with a module named dummyModule.ko).**

Ans:

insmod:

Load module into kernel. **$sudo insmod dummyModule.ko**

rmmod:

Remove loaded module from kernel. **$sudo rmmod dummyModule.ko**

modinfo:

Check base information of module. **$sudo modinfo dummyModule.ko**

4. **Write the usage (parameters, what data type they are and what do they do) of the following commands: a. module_init b. module_exit c. MODULE_LICENSE d. module_param e. MODULE_PARM_DESC**

Ans:

a. module_init is used to mark a function to be used as the entry-point of a Linux device-driver. It is called. during do_initcalls() (for a builtin driver) or at module insertion time (for a *.ko module)

b. module_exit will wrap the driver clean-up code with cleanup_module when used with rmmod when the driver is a module. If the driver is statically compiled into the kernel, module_exit has no effect.

c. MODULE_LICENSE() is a macro which allows loadable kernel modules to declare their license to the world. Its purpose is to let the kernel developers know when a non-free module has been inserted into a given kernel.

d. The module_param() macro takes 3 arguments: the name of the variable, its type and permissions for the corresponding file in sysfs, and bring them to module.

e. MODULE_PARM_DESC() is used to document arguments that the module can take. It takes two parameters: a variable name and a free form string describing that variable.

5. **What do the following terminal commands mean (explain what they do and what does the -x mean in each case): a. cat b. ls -l c. dmesg -wH d. lsmod e. lsmod | grep**

Ans:

a. The 'cat' [short for “concatenate“] command is one of the most frequently used commands in Linux and other operating systems. The cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

b. The 'ls' means list all contents under the path. The option '-l' tells the command to use a long list format.

c. 'dmesg' (diagnostic message) is a command that prints the message buffer of the kernel. The output includes messages produced by the device drivers. '-wH' enables user-friendly features like colors or relative time.

d. The 'lsmod' command is used to display the status information of the modules that have been loaded into the kernel. After executing the lsmod command, all modules loaded into the system will be listed.

e. Since the output result is very long, we can use lsmod | grep to display a specific module string

6. **There is a 0644 in the line module_param(studentId, int, 0644); inside paramsModule.c (Section 1.3). What does 0644 mean?**

Ans:

0644 means which users have promission could modify the parameter.

7. **What happens if the initialization function of the module returns -1? What type of error do you get?**

Ans:

```
if (init_module(image, image_size, buffer) != 0) {
    perror("init_module");
    return EXIT_FAILURE;
}
```

If init_module() !=0, it will print descriptive error messages stderr, and return the parameter value of EXIT_FAILURE, which is 1.

8. **In Section 1.3 – step 6, modinfo shows the information of some variables inside the module but two of them are not displayed. Why is it?**

Ans:

the reason, modinfo shows the information of some variables inside the module but two of them are not displayed,is because there is no parameter called dummyStudentId, so the kernel will ignore it and print the original values.

9. **What is the /sys/module folder for?**

Ans:

For storing those parameters delivered by init_module and could be used by mounted module.

10. **In Section 1.3 (paramsModule.c), the variable charparameter is of type charp. What is charp?**

Ans:

If the module parameter is a string, the ‘charp’ type is usually used to define the module parameter. The core copies the string provided by the user to the memory, and the corresponding variable points to this string.