

## Introduction to Operating Systems

Instructor: Prof. Ying-Dar Lin

TA: Ricardo Pontaza

### Project 3: Dynamically-Loadable Kernel Modules (DLKMs)

#### Deadline:

2020-12-18 (Fri) 23:59:59

#### Q&A:

If you have any questions, please post it on the E3 discussion board, and it will be answered in two days.

#### Deliverables:

1. **Demo video** (5-7 minutes – upload to YouTube – add link in the first page of the report).
2. **Report** (pdf file with file name of the form **OS\_Project03\_StudentID.pdf**)
  - Screenshots + one explanation paragraph per screenshot.
  - Answers to questions below.
3. **C code:**
  - a. **calculator.c (Section 1.5)**
  - b. **calculatorModule.c (Section 1.5)**
4. In the demo video and report, for each screenshot, explain:
  - a. What has been done, and
  - b. The reasoning behind the steps.
5. In the c code, please keep correct indentation, clean code and clear comments.

**Objective:** The objective of this project is to help the student to get familiar with Linux Kernel Modules. The student will learn the definition, usage and how to implement custom Linux Kernel Modules. The student will also learn how to mount and unmount them without re-compiling the kernel.

**Scope:** Understand the concept of kernel modules, implement custom modules and learn how to mount and unmount them.

#### Questions to be answered in the report:

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)?
2. Why does adding a system call require kernel re-compilation, while adding a kernel module does not?
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**).
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`
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`
6. There is a 0644 in the line

```
module_param(studentId, int, 0644);
```

inside **paramsModule.c** (Section 1.3). What does 0644 mean?

- 7. What happens if the initialization function of the module returns -1? What type of error do you get?
- 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?
- 9. What is the `/sys/module` folder for?
- 10. In Section 1.3 (`paramsModule.c`), the variable **charparameter** is of type **charp**. What is charp?

### Table of contents:

- 1. Example 1 – Hello world:
  - a. Module creation.
  - b. (Manually) module load and unload.
  - c. Display message from inside module.
- 2. Example 2 - Sending parameters to a module and reading and modifying module variables:
  - a. Module creation.
  - b. (Manually) module load and unload.
  - c. Send parameters in loading time.
  - d. Display message from inside module.
  - e. Execution of code inside module.
  - f. Read parameters' values.
  - g. Read module information.
  - h. Warning when modifying module parameters' values.
- 3. Example 3 - Dynamically loading and unloading a module by user-space application:
  - a. Module creation.
  - b. (Dynamically) module load and unload by c program.
  - c. Send parameters to module from c program.
- 4. Final exercise – Simple calculator

**SECTION 1:**  
**DYNAMICALLY-LOADABLE KERNEL MODULES**  
**(DLKMs) – BASIC EXAMPLE**

## Section 1: Dynamically-Loadable Kernel Modules

### Section 1.1: System requirements

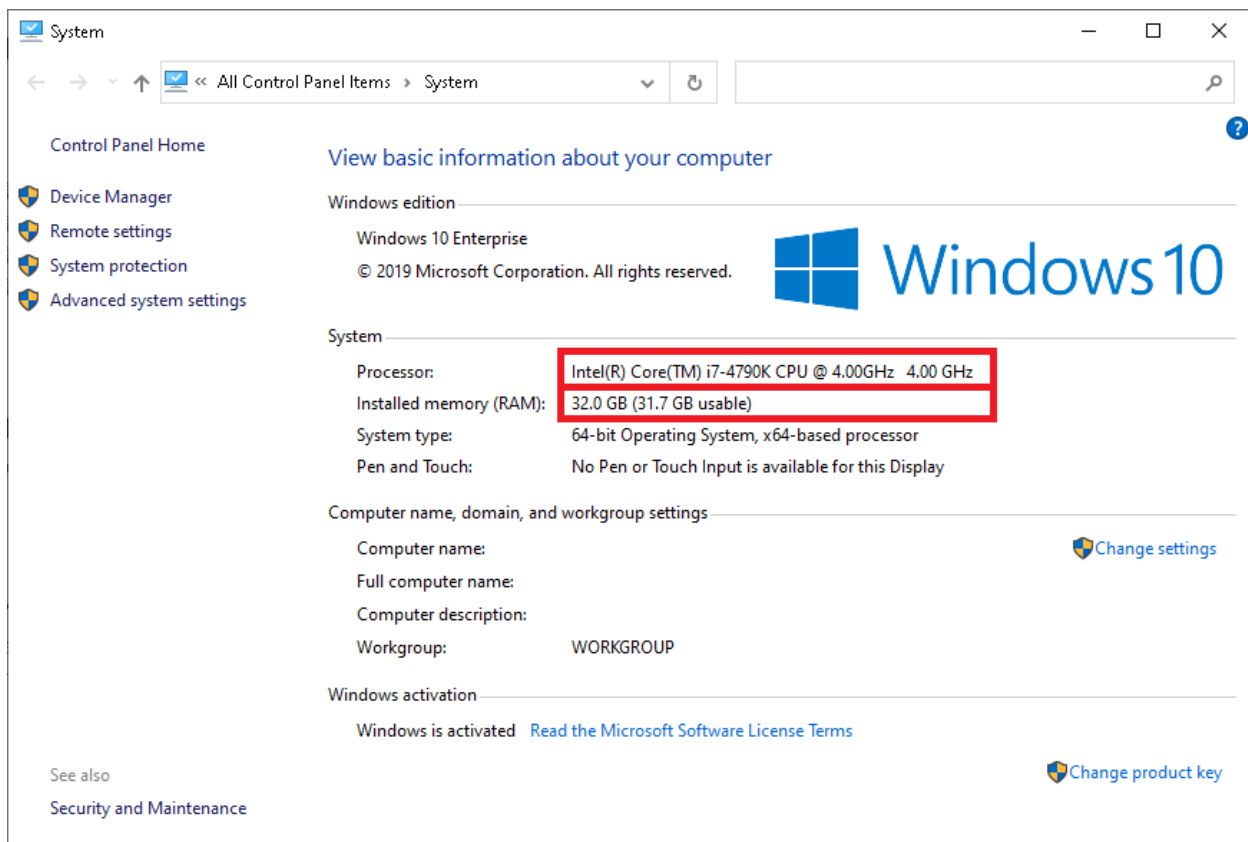
Below are the system requirements your system must match to continue:

1. Ubuntu 16.04 - 64 bits – desktop edition.
2. Kernel 4.19.148.

<https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.148.tar.xz>

Also check your RAM-CPU in both your physical computer and virtual machines:

1. Check your CPU model (intel i3-xxxx, intel i5-xxxx, intel i7-xxxx or similar), and check how many cores and how many threads it supports.



(If it is an intel CPU, check on [ark.intel.com](http://ark.intel.com)).

2. If your computer has only 2 cores with at most 2 threads:
  - a. **Case 1:** 2 cores – 1 thread or
  - b. **Case 2:** 2 cores – 2 threads,

Then only allocate 1 core per virtual machine and leave 2 cores for the physical machine.

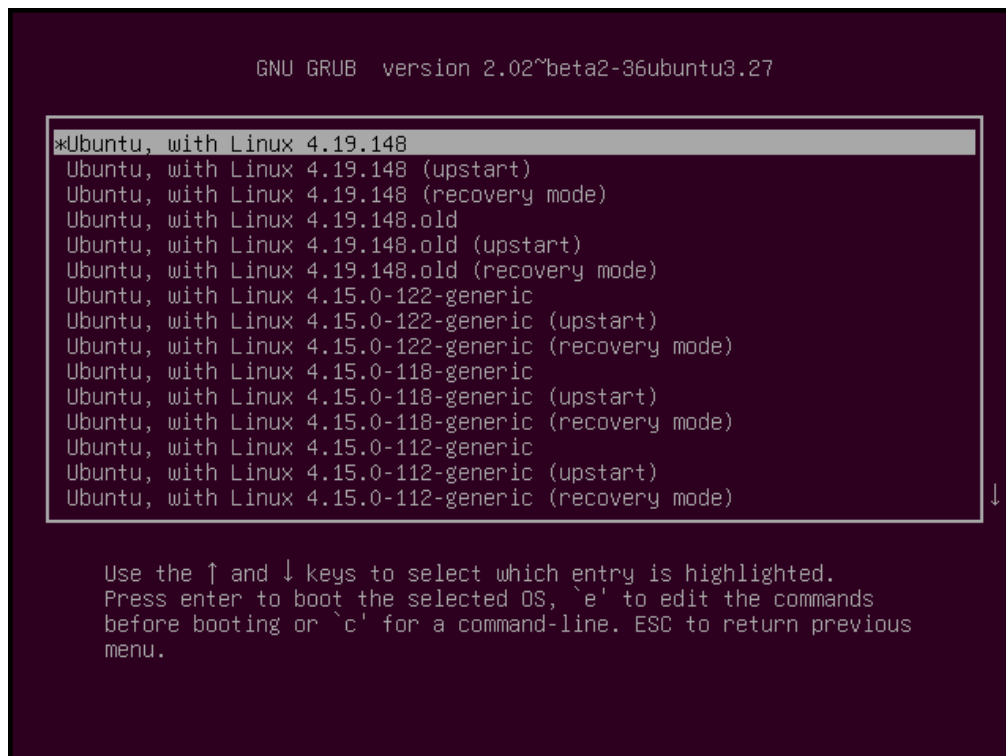
If your physical computer has more cores (i.e.,  $4 < \text{CPUs} = \text{\#of cores} * \text{\#of threads}$ ), then you can allocate more CPUs to the virtual machines.

3. If your computer only has between 4 and 6 GB of RAM, then only allocate 1-2 GB of RAM for each virtual machine and leave 2-4 GB of RAM to the physical machine.

If your computer has more than 6 GB of RAM, then you can proceed to allocate more RAM to the virtual machines.

#### **IMPORTANT NOTES:**

1. If your computer does not cover the minimum requirements, **please solve them before posting any questions in the forum.**
2. Forum support will be restricted to only **kernel 4.19.148**. (Other kernels and other systems are not supported in the forum – you can still use other kernels but no forum support will be given).
3. Forum support will be restricted to **kernel module-related questions only**. (If you have **vmware-related**, **linux-related** or **c-programming** related questions, please search online – **no support will be given in the forum for these topics**).
4. If you have problems booting up the virtual machines, and/or downloading or building the kernel, please refer to project 1A.
5. Please always select the correct kernel when booting up your virtual machine.



And always check by using **\$ uname -r**.

## Section 1.2: Dynamically-Loadable Kernel Modules – Example 1: Hello world

In this section we will create a basic kernel module that displays a message, and learn how to (manually) mount and unmount it. All these steps will be performed in the Target machine. The host machine will be off during this project.

**NOTE:** You must finish project 1A, 1B and (preferably) project 2 in order to do this one.

1. In the Desktop, create a folder named **Tests** and inside create a folder named **Modules**. Inside the Modules folder create another one called **helloModule**.
2. Inside the helloModule folder, create a file named **helloModule.c** with the following content:

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

static int studentId = 12345;

static int initialize(void)
{
    printk(KERN_INFO "[%d] : Function [%s] - Hello from OS Project 03!\n",
studentId, __func__);
    return 0;
}

static void clean_exit(void){
    printk(KERN_INFO "[%d] : Function [%s] - Unloading module. Goodbye from OS
Project 03!\n", studentId, __func__);
}

module_init(initialize);
module_exit(clean_exit);
```

3. Inside the helloModule folder, also create a file named **Makefile** with the following content:

```
obj-m = helloModule.o

KVERSION = $(shell uname -r)

all:
    make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

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

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

**Important Note:** Notice that, to compile, load and unload a kernel module, it is not necessary to re-compile the whole kernel.

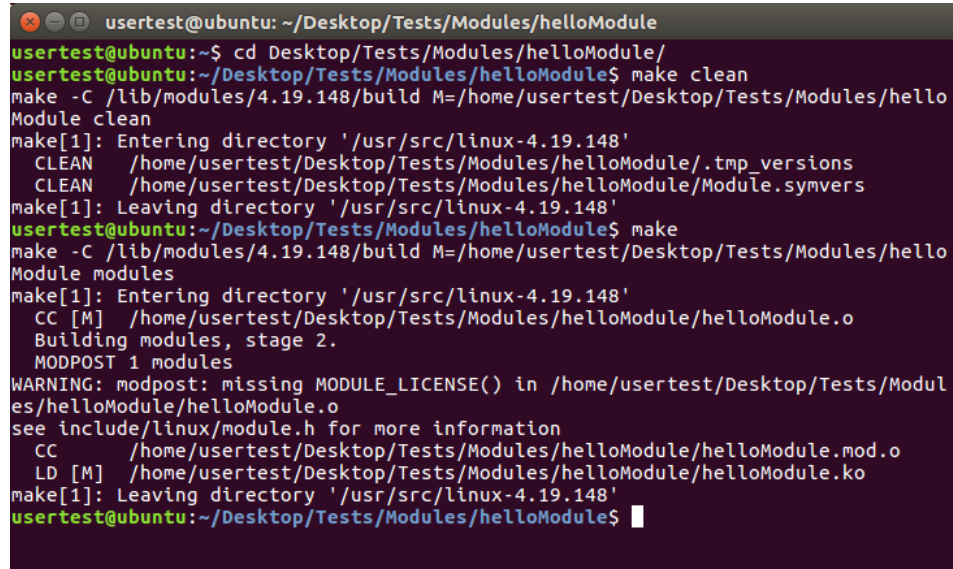
4. From this point on (until mentioned again), we will open two terminals and place them side by side. In this document they will be referred as **terminal 1** and **terminal 2**.

Open **terminal 1** and **terminal 2**, and in both of them go to the helloModule folder we just created.

5. In **Terminal 1** run the following commands, which build our module

**\$make clean**

**\$make**



```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~$ cd Desktop/Tests/Modules/helloModule/
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ make clean
make -C /lib/modules/4.19.148/build M=/home/usertest/Desktop/Tests/Modules/helloModule clean
make[1]: Entering directory '/usr/src/linux-4.19.148'
  CLEAN   /home/usertest/Desktop/Tests/Modules/helloModule/.tmp_versions
  CLEAN   /home/usertest/Desktop/Tests/Modules/helloModule/Module.symvers
make[1]: Leaving directory '/usr/src/linux-4.19.148'
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ make
make -C /lib/modules/4.19.148/build M=/home/usertest/Desktop/Tests/Modules/helloModule modules
make[1]: Entering directory '/usr/src/linux-4.19.148'
  CC [M]   /home/usertest/Desktop/Tests/Modules/helloModule/helloModule.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/usertest/Desktop/Tests/Modules/helloModule/helloModule.o
see include/linux/module.h for more information
  CC       /home/usertest/Desktop/Tests/Modules/helloModule/helloModule.mod.o
  LD [M]   /home/usertest/Desktop/Tests/Modules/helloModule/helloModule.ko
make[1]: Leaving directory '/usr/src/linux-4.19.148'
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$
```

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

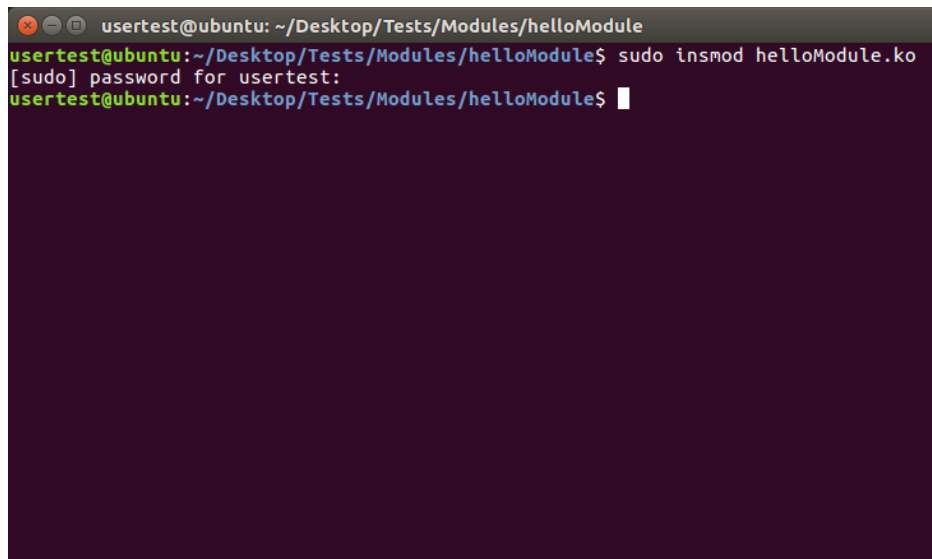
6. In **Terminal 2** run the command

**\$ dmesg -wH**

And check what is the last line it appears.

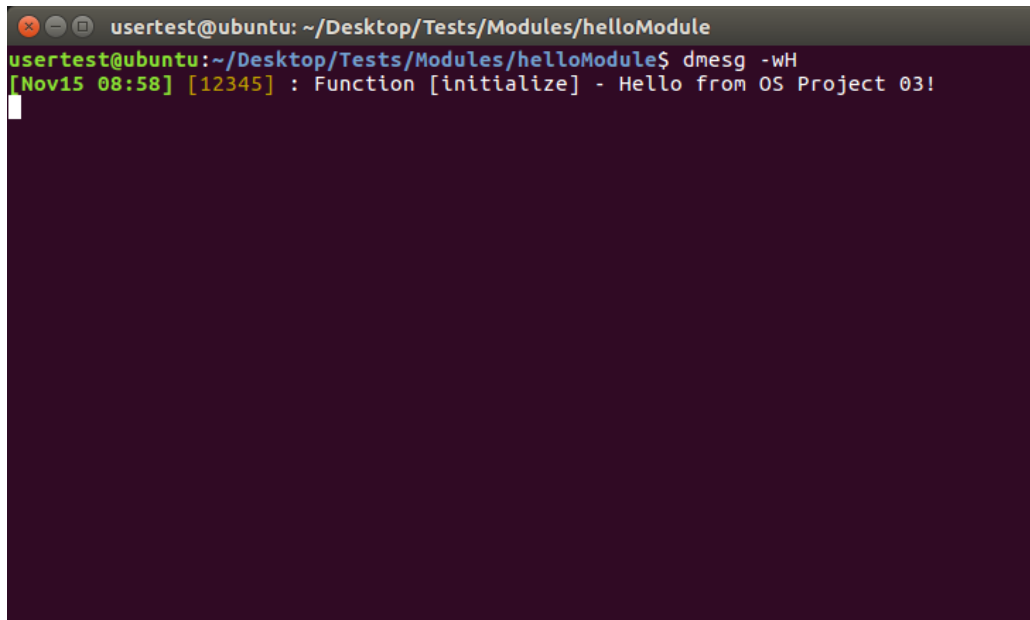
7. In **Terminal 1** run the following command to load the module into the kernel

**\$ sudo insmod helloModule.ko**



```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo insmod helloModule.ko
[sudo] password for usertest:
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$
```

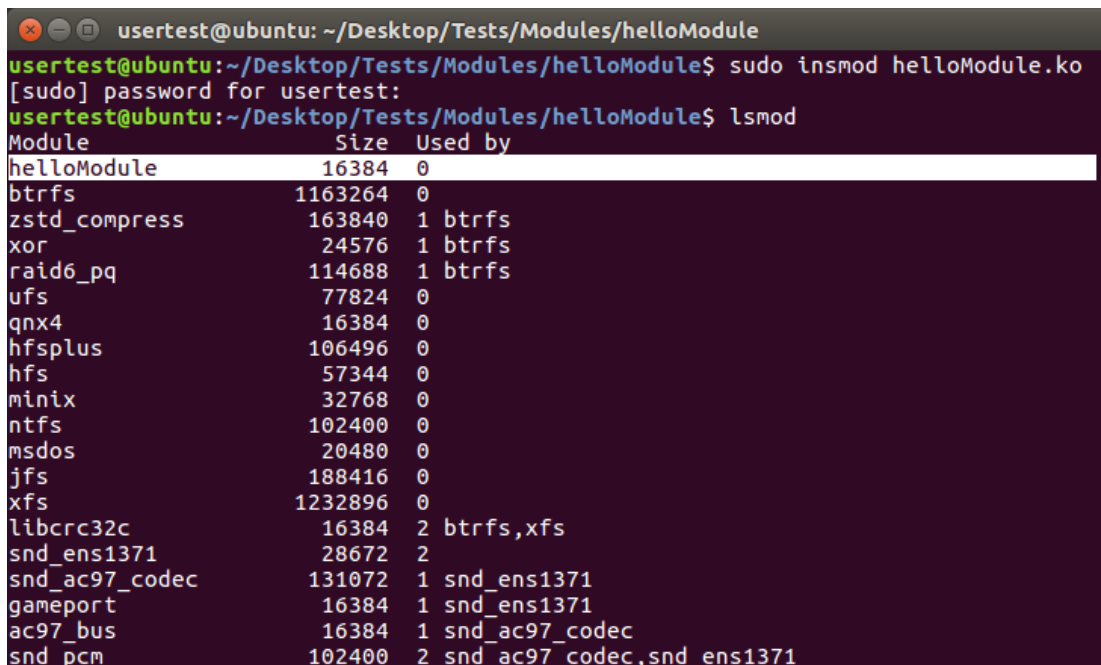
In Terminal 2 the message inside the **initialize** function appears.



```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ dmesg -wH
[Nov15 08:58] [12345] : Function [initialize] - Hello from OS Project 03!
```

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

8. Now the module has been created and loaded to the kernel. By running **\$ lsmod** we can see that there is a module called **helloModule** now in the modules list.

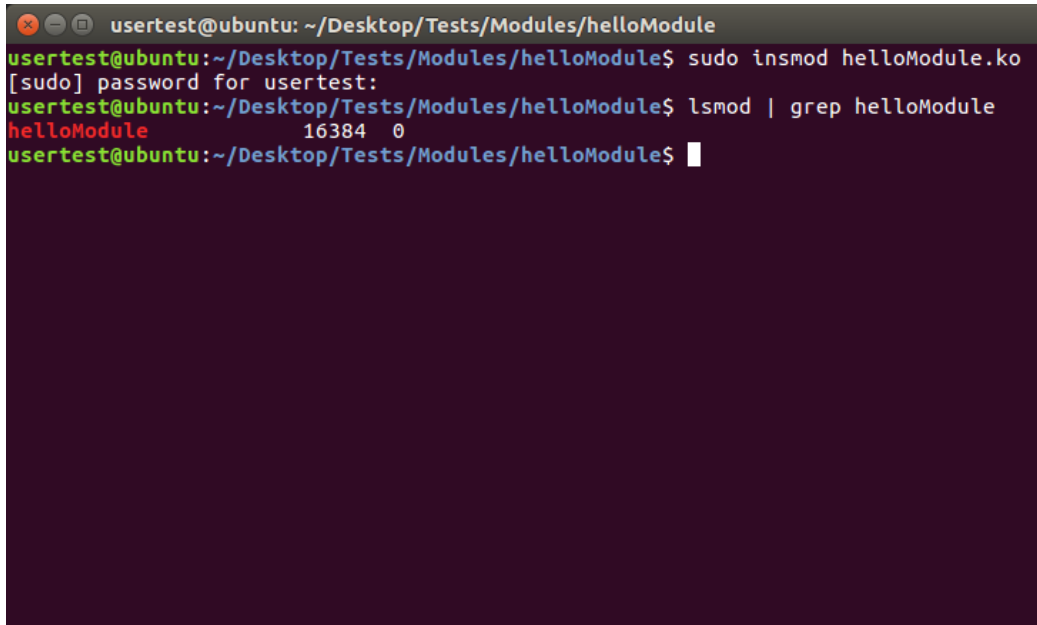


```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo insmod helloModule.ko
[sudo] password for usertest:
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod
```

| Module         | Size    | Used by                      |
|----------------|---------|------------------------------|
| helloModule    | 16384   | 0                            |
| btrfs          | 1163264 | 0                            |
| zstd_compress  | 163840  | 1 btrfs                      |
| xor            | 24576   | 1 btrfs                      |
| raid6_pq       | 114688  | 1 btrfs                      |
| ufs            | 77824   | 0                            |
| qnx4           | 16384   | 0                            |
| hfsplus        | 106496  | 0                            |
| hfs            | 57344   | 0                            |
| minix          | 32768   | 0                            |
| ntfs           | 102400  | 0                            |
| msdos          | 20480   | 0                            |
| jfs            | 188416  | 0                            |
| xfs            | 1232896 | 0                            |
| libcrc32c      | 16384   | 2 btrfs,xfs                  |
| snd_ens1371    | 28672   | 2                            |
| snd_ac97_codec | 131072  | 1 snd_ens1371                |
| gameport       | 16384   | 1 snd_ens1371                |
| ac97_bus       | 16384   | 1 snd_ac97_codec             |
| snd_pcm        | 102400  | 2 snd_ac97_codec,snd_ens1371 |



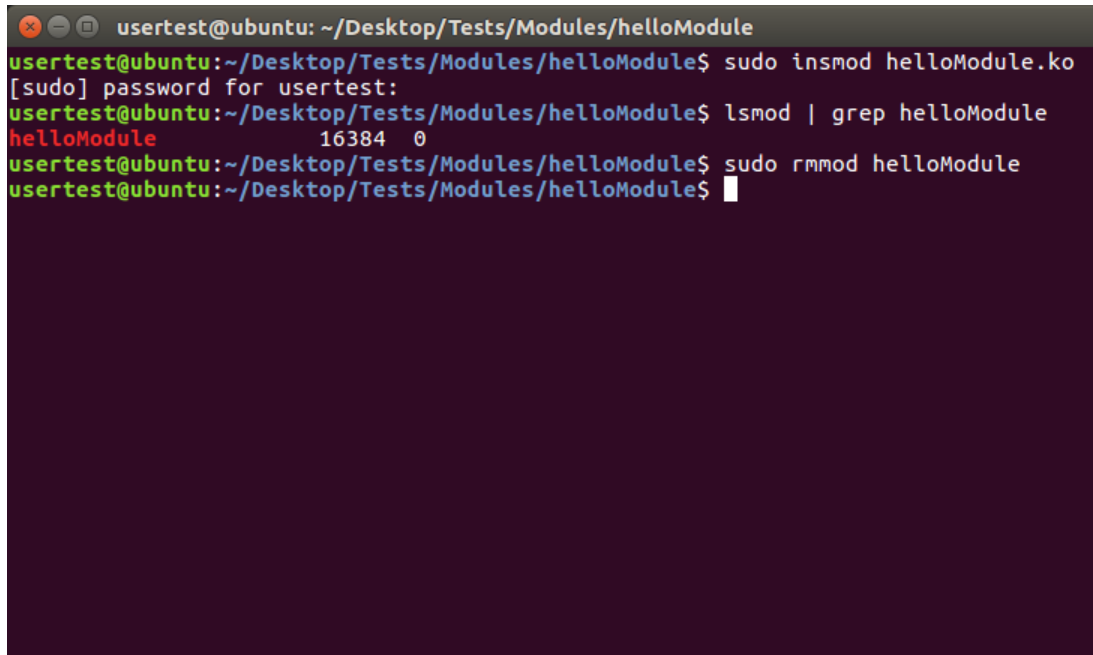
9. In **Terminal 1**, you can also check if the module was loaded or not by running the command  
**\$ lsmod | grep helloModule**

A terminal window with a dark purple background. The title bar shows 'usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule'. The terminal text shows the user running 'sudo insmod helloModule.ko', followed by a password prompt '[sudo] password for usertest:'. Then, the user runs 'lsmod | grep helloModule', which outputs 'helloModule 16384 0'. The prompt returns to the user's shell.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo insmod helloModule.ko
[sudo] password for usertest:
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod | grep helloModule
helloModule          16384  0
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$
```

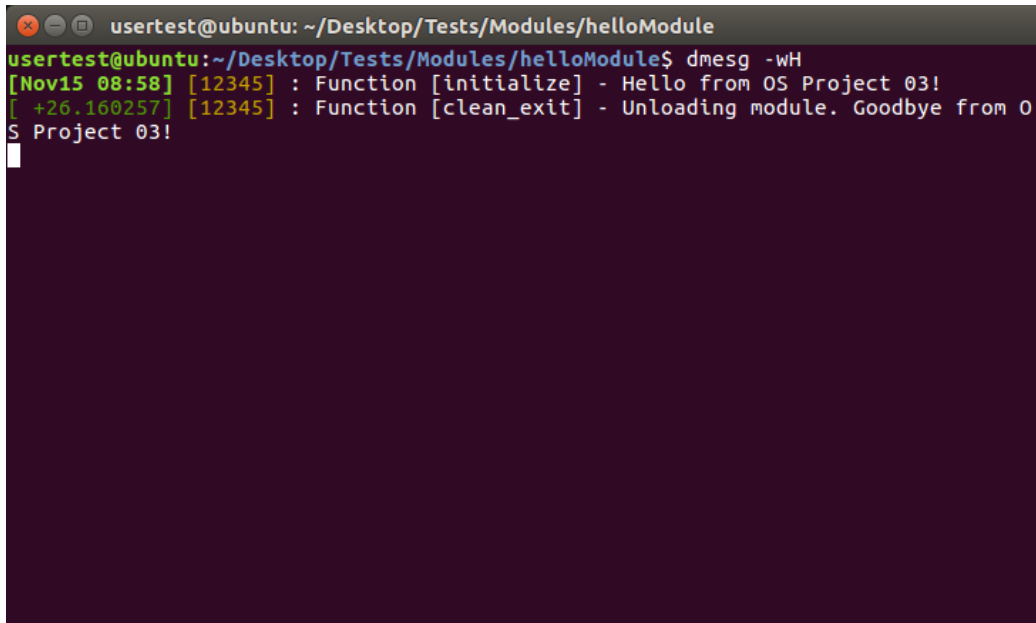
**[Screenshot #4: Create a screenshot showing the list of loaded modules from step 8 and 9]**

10. To unload the module, in **Terminal 1** type  
**\$ sudo rmmod helloModule**

A terminal window with a dark purple background. The title bar shows 'usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule'. The terminal text shows the user running 'sudo insmod helloModule.ko', followed by a password prompt '[sudo] password for usertest:'. Then, the user runs 'lsmod | grep helloModule', which outputs 'helloModule 16384 0'. Finally, the user runs 'sudo rmmod helloModule', and the prompt returns to the user's shell.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo insmod helloModule.ko
[sudo] password for usertest:
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod | grep helloModule
helloModule          16384  0
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo rmmod helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$
```

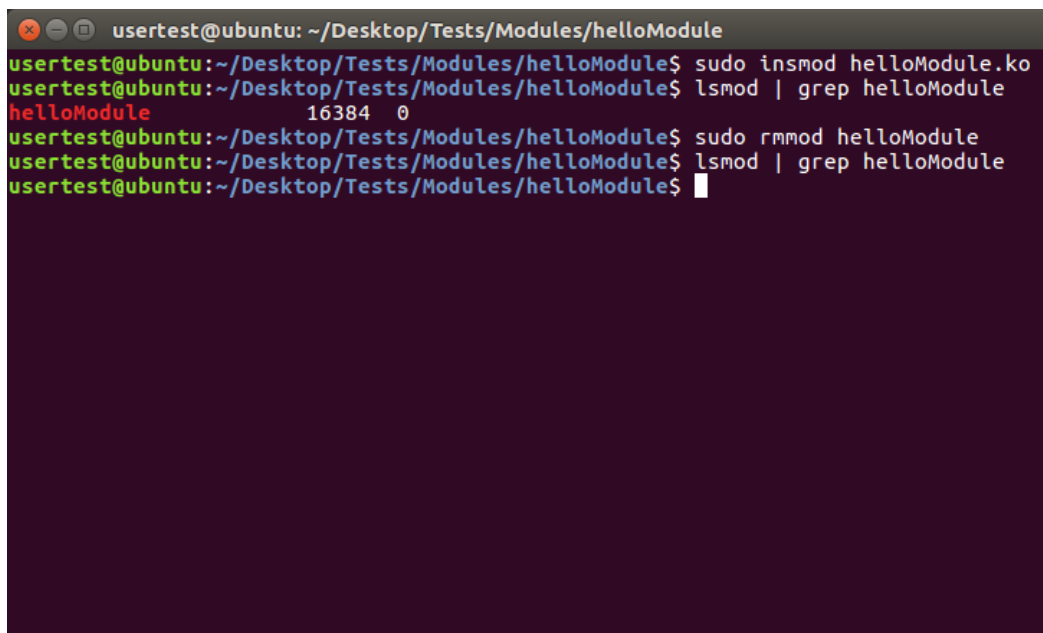
In **Terminal 2**, the message inside the `clean_exit` function of the `helloModule.c` file appears.



```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ dmesg -wH
[Nov15 08:58] [12345] : Function [initialize] - Hello from OS Project 03!
[ +26.160257] [12345] : Function [clean_exit] - Unloading module. Goodbye from OS Project 03!
```

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

11. In **Terminal 1**, if we look again for the module using `lsmod`, it should not return any result.



```
usertest@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo insmod helloModule.ko
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod | grep helloModule
helloModule          16384  0
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo rmmod helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod | grep helloModule
usertest@ubuntu:~/Desktop/Tests/Modules/helloModule$
```

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

### **Section 1.3: Dynamically-Loadable Kernel Modules – Example 2:**

#### **Sending parameters to a module and reading and modifying module variables**

In this section we will create a kernel module that is capable of receiving parameters, and we will also learn how to read variables inside the module. Here we will use three terminals **(Terminal 1, 2 and 3)**.

1. In your Desktop, inside the Tests/Modules/ folder, create a new folder called **paramsModule**.
2. Inside **paramsModule**, create a file named **paramsModule.c** and copy the code below. **(Please read this code to understand the rest of this section)**

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

#include <linux/string.h>

#define AUTHOR "Ricardo Pontaza OS TA 2020 NCTU"    //Change your name here
MODULE_LICENSE("GPL");
MODULE_AUTHOR(AUTHOR);

static char *kernelModuleName = "paramsModule"; //Change module's name when needed

static int studentId = 12345; // real studentId = 012345, removed 0 for display purposes
module_param(studentId, int, 0644);
MODULE_PARM_DESC(studentId, "Parameter for student Id. (Leading zeros are omitted)");

static long secretValue = 987654321;
module_param(secretValue, long, 0644);
MODULE_PARM_DESC(secretValue, "Parameter for secret value.");

static char *charparameter = "Hello world! Project 02 - Example 02";
module_param(charparameter, charp, 0644);
MODULE_PARM_DESC(charparameter, "states - Hello world");

static int modifyValues = 0;
module_param(modifyValues, int, 0644);
MODULE_PARM_DESC(modifyValues, "Indicates if we must modify the original values or not.");

static int dummyStudentId = -1;
static long dummySecretValue = -2;

static int initialize(void){

    if(modifyValues==1)
    {
        studentId = dummyStudentId;
        secretValue = dummySecretValue;
        charparameter = "This is a dummy message!";
    }
}
```

---

```

        printk(KERN_INFO "\n[%s - %s] =====\n", kernelModuleName, __func__);
        printk(KERN_INFO "[%s - %s] Hello!\n", kernelModuleName, __func__);
        printk(KERN_INFO "[%s - %s] Student Id = [%d]\n", kernelModuleName, __func__,
studentId);
        printk(KERN_INFO "[%s - %s] String inside module = [%s]\n", kernelModuleName,
__func__, charparameter);
        printk(KERN_INFO "[%s - %s] Secret value = [%ld]\n", kernelModuleName,
__func__, secretValue);

        return 0;
}

static void clean_exit(void){
    printk(KERN_INFO "\n[%s - %s] =====\n", kernelModuleName, __func__);
    printk(KERN_INFO "[%s - %s] Goodbye!\n", kernelModuleName, __func__);
    printk(KERN_INFO "[%s - %s] Student Id = [%d]\n", kernelModuleName, __func__,
studentId);
    printk(KERN_INFO "[%s - %s] String inside module = [%s]\n", kernelModuleName,
__func__, charparameter);
    printk(KERN_INFO "[%s - %s] Secret value = [%ld]\n", kernelModuleName,
__func__, secretValue);
}

module_init(initialize);
module_exit(clean_exit);

```

3. Also create a Makefile in the same folder and add the following code

```

obj-m = paramsModule.o

KVERSION = $(shell uname -r)

all:
    make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean

```

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

4. In Terminal 1 build your module (**\$make clean** and **\$make**).
5. In Terminal 2 open dmesg with **\$ dmesg -wH**
6. By mounting the module in **Terminal 1** with

**\$ sudo insmod paramsModule.ko**

We can see in **Terminal 2** that **\$ dmesg -wH** shows the default values of

- **studentId** (12345),
- **secretValue** (987654321), and
- **charparameter** ("Hello world! Project 02 – Example 02")

Also when we unmount the module in **Terminal 1** with

**\$ sudo rmmod paramsModule.ko**

We see that the values remain the same in **Terminal 2**.

(Please read the code in **paramsModule.c** to see where and how these values are defined).

#### Terminal 1:

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~$ cd Desktop/Tests/Modules/paramsModule/
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.ko
[sudo] password for usertest:
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

#### Terminal 2:

```
usertest@ubuntu: ~
usertest@ubuntu:~$ sudo dmesg -wH
[Nov17 06:13]
[paramsModule - initialize] =====
[ +0.000002] [paramsModule - initialize] Hello!
[ +0.000000] [paramsModule - initialize] Student Id = [12345]
[ +0.000001] [paramsModule - initialize] String inside module = [Hello world! P
roject 02 - Example 02]
[ +0.000000] [paramsModule - initialize] Secret value = [987654321]
[ +33.094458]
[paramsModule - clean_exit] =====
[ +0.000002] [paramsModule - clean_exit] Goodbye!
[ +0.000001] [paramsModule - clean_exit] Student Id = [12345]
[ +0.000000] [paramsModule - clean_exit] String inside module = [Hello world! P
roject 02 - Example 02]
[ +0.000001] [paramsModule - clean_exit] Secret value = [987654321]
```

UP TO THIS POINT, THE VALUES OF THE VARIABLES INSIDE THE MODULE HAVE THE SAME VALUES WHEN IT IS LOADED AND WHEN IT IS UNLOADED.

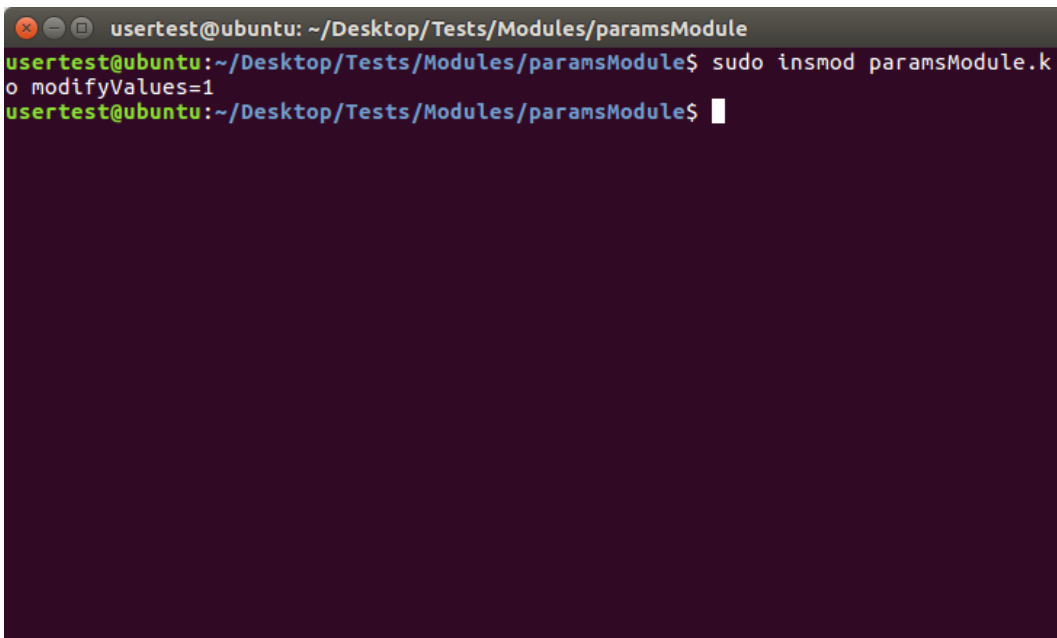
[Screenshot #8: Create a screenshot showing the values you obtain when mounting and unmounting the module, and what commands you ran to do so.]

7. We can execute code when the module is loaded (and also when it is unloaded). Inside the function **initialize** we have the following piece of code

```
static int initialize(void){  
  
    if(modifyValues==1)  
    {  
        studentId = dummyStudentId;  
        dummySecretVaue = secretValue;  
        charparameter = "This is a dummy message!\n";  
    }  
  
    [...]  
  
    return 0;  
}
```

Which means that if the **modifyValues** variable has the value of 1 when the module is loaded, then the **studentId**, **secretValue** and **charparameter** values change.

To execute this code, mount the module in **Terminal 1** sending the value of the parameter  
**\$ sudo insmod paramsModule.ko modifyValues=1**

A screenshot of a terminal window. The title bar shows 'usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule'. The terminal text shows the user 'usertest' at 'ubuntu' in the directory '~/Desktop/Tests/Modules/paramsModule'. They have entered the command 'sudo insmod paramsModule.ko modifyValues=1'. The prompt is now 'usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule\$' with a cursor. The terminal background is dark purple.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule  
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.k  
o modifyValues=1  
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

And in Terminal 2 you will see the modified values.

```
usertest@ubuntu: ~  
usertest@ubuntu:~$ sudo dmesg -wH  
[Nov17 06:41]  
[ +0.000002] [paramsModule - initialize] =====  
[ +0.000000] [paramsModule - initialize] Hello!  
[ +0.000000] [paramsModule - initialize] Student Id = [-1]  
[ +0.000001] [paramsModule - initialize] String inside module = [This is a dummy message!]  
[ +0.000000] [paramsModule - initialize] Secret value = [-2]
```

8. Point **Terminal 3** to the folder where you have the module, and run the command  
**\$ sudo modinfo paramsModule.ko**

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule  
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo modinfo paramsModule.ko  
filename:      /home/usertest/Desktop/Tests/Modules/paramsModule/paramsModule.ko  
author:        Ricardo Pontaza OS TA 2020 NCTU  
license:        GPL  
srcversion:     024D6C3AC25891EB07D10BD  
depends:  
retpoline:      Y  
name:           paramsModule  
vermagic:       4.19.148 SMP mod unload  
parm:          studentId:Parameter for student Id. (Leading zeros are omitted) (int)  
parm:          secretValue:Parameter for secret value. (long)  
parm:          charparameter:states - Hello world (charp)  
parm:          modifyValues:Indicates if we must modify the original values or not. (int)  
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

This command shows you the information of the module (author, license, parameters and more). IT ALSO SHOWS THE DATATYPE OF THE VARIABLES (int, long, charp). We will use them in a future step (Note that *charparameter* is of type charp).

**IMPORTANT NOTE:** Note that **paramsModule.c** has two variables (**dummyStudentId** and **dummySecretValue** that are not displayed by **modinfo**).

Unmount the module in **Terminal 1**, and watch that the variables when the **clean\_exit** function is executed in **Terminal 2**.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo modinfo paramsModule.ko
filename:          /home/usertest/Desktop/Tests/Modules/paramsModule/paramsModule.ko
author:            Ricardo Pontaza OS TA 2020 NCTU
license:           GPL
srcversion:        024D6C3AC25891EB07D10BD
depends:
retpoline:         Y
name:              paramsModule
vermagic:          4.19.148 SMP mod_unload
parm:              studentId:Parameter for student Id. (Leading zeros are omitted) (int)
parm:              secretValue:Parameter for secret value. (long)
parm:              charparameter:states - Hello world (charp)
parm:              modifyValues:Indicates if we must modify the original values or not. (int)
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

```
usertest@ubuntu: ~
usertest@ubuntu:~$ sudo dmesg -wH
[Nov17 06:41]
[paramsModule - initialize] =====
[ +0.000002] [paramsModule - initialize] Hello!
[ +0.000000] [paramsModule - initialize] Student Id = [-1]
[ +0.000001] [paramsModule - initialize] String inside module = [This is a dummy message!]
[ +0.000000] [paramsModule - initialize] Secret value = [-2]
[Nov17 06:43]
[paramsModule - clean_exit] =====
[ +0.000002] [paramsModule - clean_exit] Goodbye!
[ +0.000000] [paramsModule - clean_exit] Student Id = [-1]
[ +0.000001] [paramsModule - clean_exit] String inside module = [This is a dummy message!]
[ +0.000000] [paramsModule - clean_exit] Secret value = [-2]
```

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

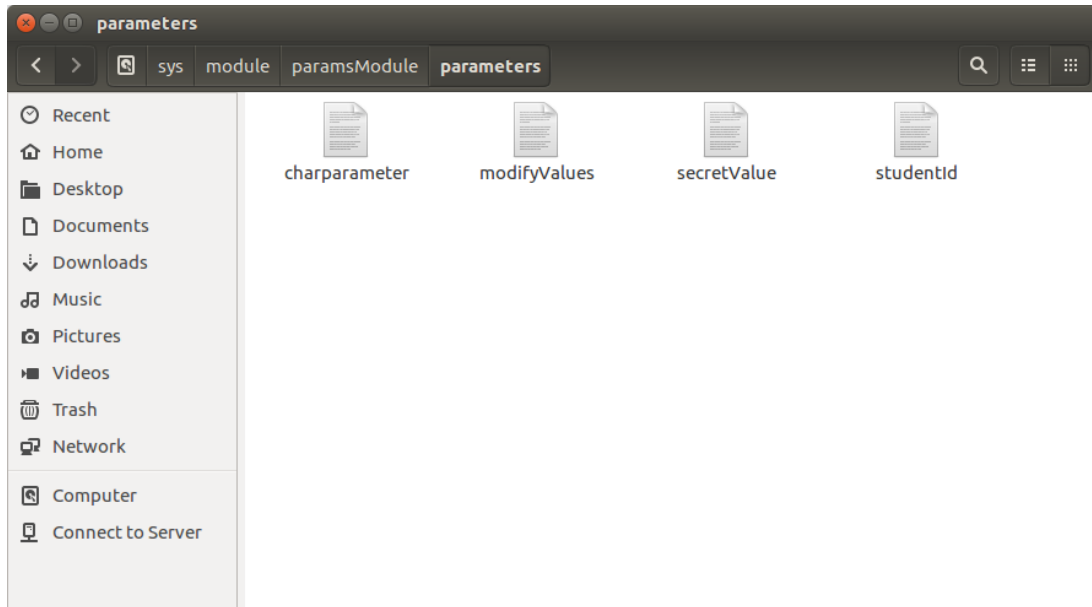


9. Mount again the module in Terminal 1 with values **studentId=[your student ID WITHOUT LEADING 0]** and **secretValue=8888**. Terminal 2 will display the messages with those values.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.k
o studentId=9999 secretValue=8888
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

```
usertest@ubuntu: ~
usertest@ubuntu:~$ sudo dmesg -wH
[Nov17 06:44]
[paramsModule - initialize] =====
[ +0.000002] [paramsModule - initialize] Hello!
[ +0.000000] [paramsModule - initialize] Student Id = [9999]
[ +0.000002] [paramsModule - initialize] String inside module = [Hello world! P
roject 02 - Example 02]
[ +0.000001] [paramsModule - initialize] Secret value = [8888]
```

10. Linux manages module variables as files. **WHEN THE MODULE IS MOUNTED**, you can find them in the following location: **/sys/module/<name of module>/parameters**  
For our example, the variables are located in /sys/module/paramsModule/parameters

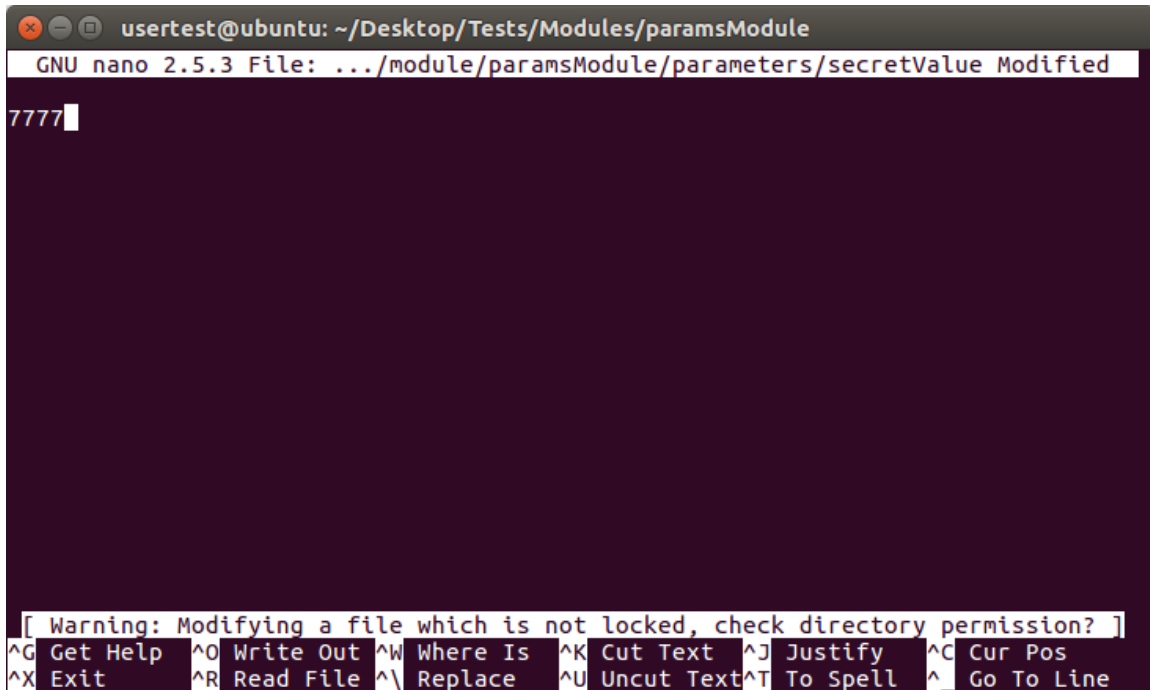


In Terminal 3, we can modify the value of those variables with nano:

**\$ sudo nano /sys/module/paramsModule/parameters/secretValue**

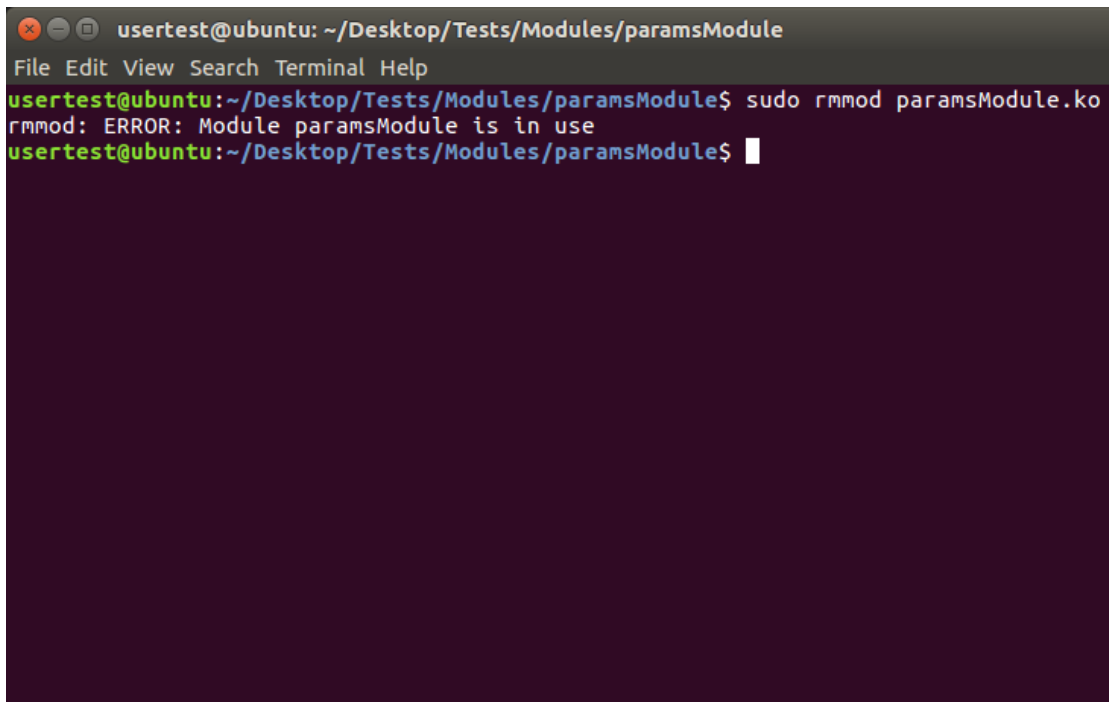
```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ ls /sys/module/paramsModule/parameters/
charparameter  modifyValues  secretValue  studentId
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo nano /sys/module/paramsModule/parameters/secretValue
```

For this example, we will make `secretValue=7777`



```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
GNU nano 2.5.3 File: ../module/paramsModule/parameters/secretValue Modified
7777
[ Warning: Modifying a file which is not locked, check directory permission? ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

**WARNING:** When you run `modinfo`, it shows to you the variable datatype (int, long, char, etc). If you save a value that is not the correct datatype, your module will get locked and you won't be able to unload it, so you will have to fix the error, and restart your machine.



```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
File Edit View Search Terminal Help
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule.ko
rmmod: ERROR: Module paramsModule is in use
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

11. Now that the secretValue file has been modified, when you unload the module in Terminal 1, you will see the new value in Terminal 2.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ ls /sys/module/paramsModule/parameters/
charparameter modifyValues secretValue studentId
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo nano /sys/module/paramsModule/parameters/secretValue
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

```
usertest@ubuntu: ~
usertest@ubuntu:~$ sudo dmesg -wH
[Nov17 06:44]
[paramsModule - initialize] =====
[ +0.000002] [paramsModule - initialize] Hello!
[ +0.000000] [paramsModule - initialize] Student Id = [9999]
[ +0.000002] [paramsModule - initialize] String inside module = [Hello world! Project 02 - Example 02]
[ +0.000001] [paramsModule - initialize] Secret value = [8888]
[Nov17 06:47]
[paramsModule - clean_exit] =====
[ +0.000002] [paramsModule - clean_exit] Goodbye!
[ +0.000001] [paramsModule - clean_exit] Student Id = [9999]
[ +0.000000] [paramsModule - clean_exit] String inside module = [Hello world! Project 02 - Example 02]
[ +0.000001] [paramsModule - clean_exit] Secret value = [7777]
```

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

12. If you try to send a parameter that does not appear in **modinfo**, the module is loaded but the parameter ignored.

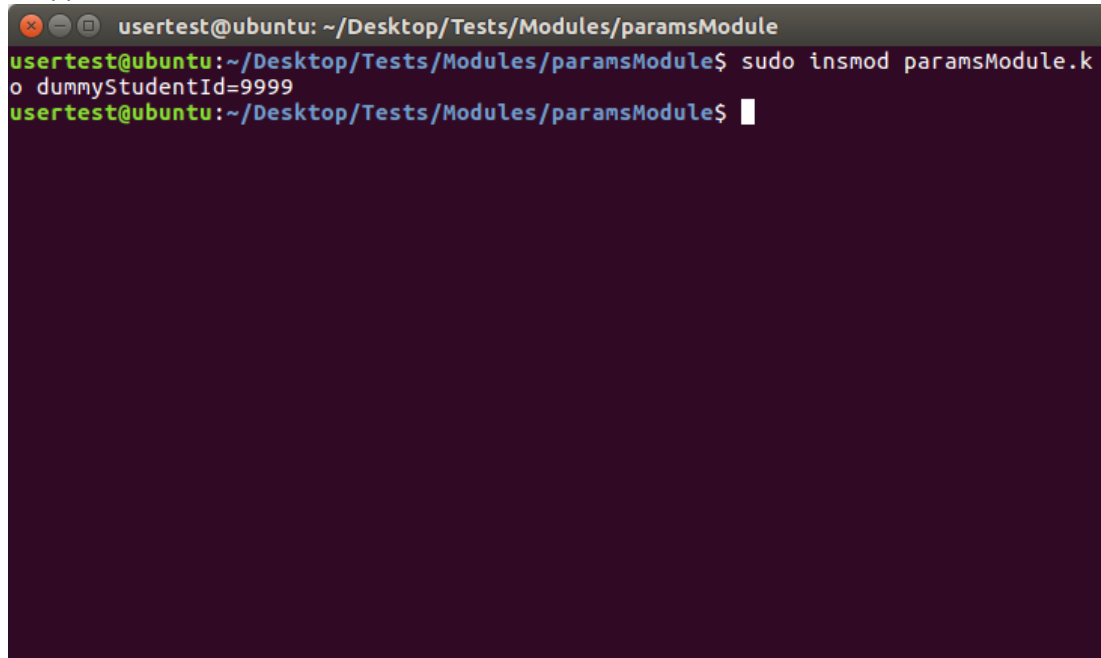
In Terminal 1 mount the module with

**\$ sudo insmod paramsModule.ko dummyStudentId=9999**

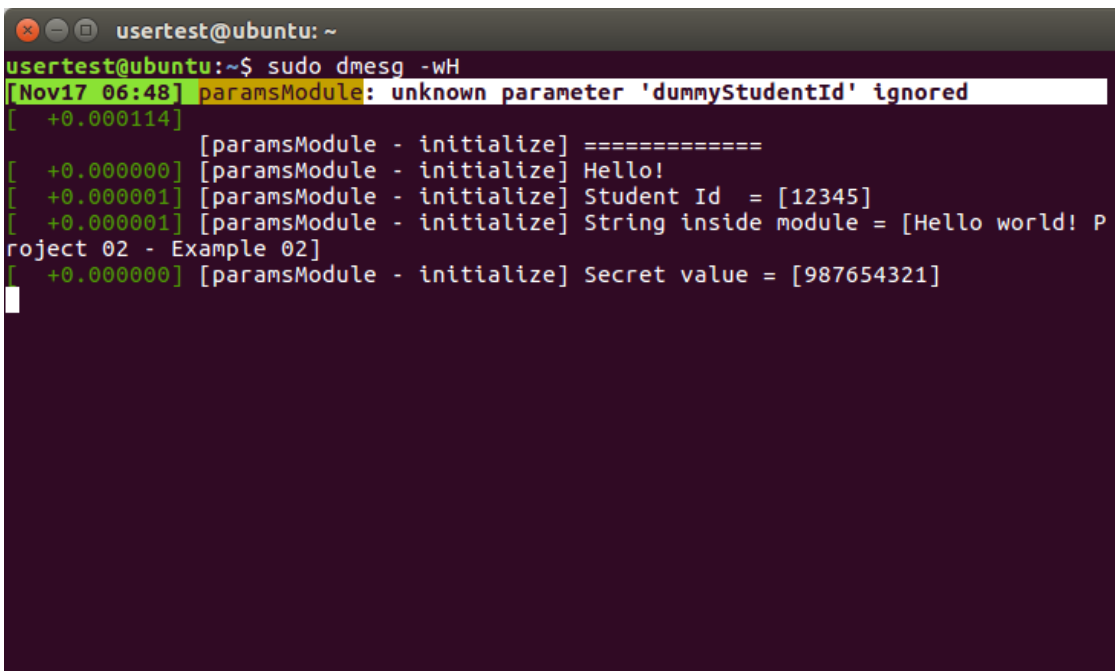
and see that in Terminal 2 the module is loaded, but the message

***paramsModule: unknown parameter 'dummyStudentId' ignored***

appears.



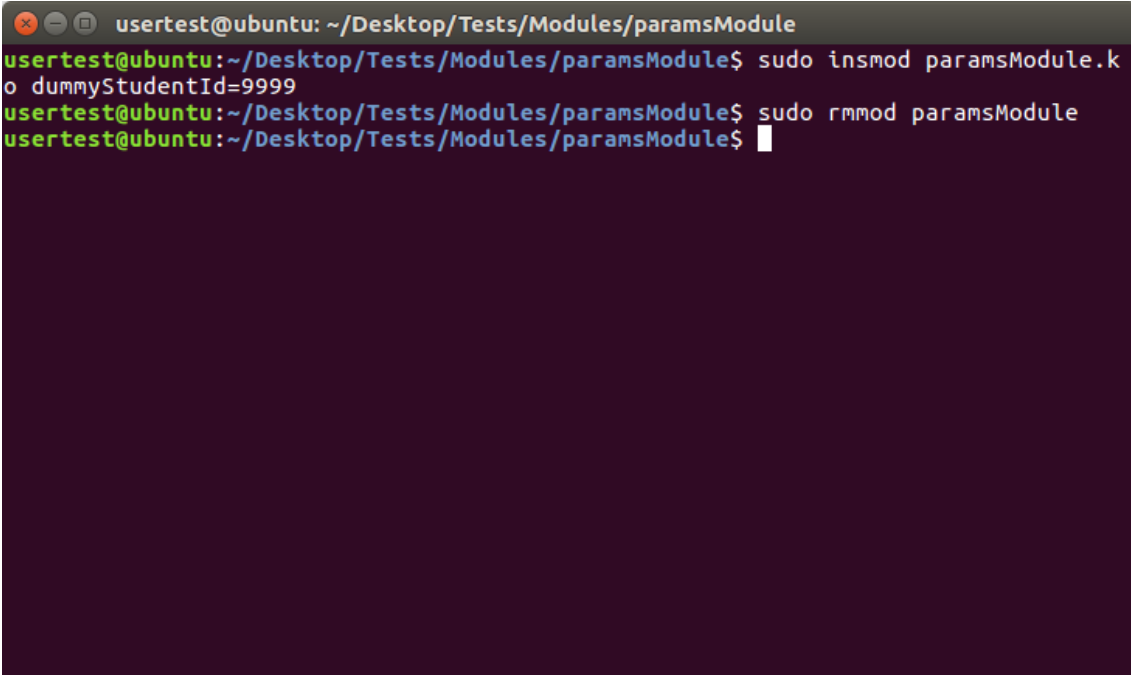
```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.ko dummyStudentId=9999
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```



```
usertest@ubuntu: ~
usertest@ubuntu:~$ sudo dmesg -wH
[Nov17 06:48] paramsModule: unknown parameter 'dummyStudentId' ignored
[ +0.000114]
[ +0.000000] [paramsModule - initialize] =====
[ +0.000000] [paramsModule - initialize] Hello!
[ +0.000001] [paramsModule - initialize] Student Id = [12345]
[ +0.000001] [paramsModule - initialize] String inside module = [Hello world! Project 02 - Example 02]
[ +0.000000] [paramsModule - initialize] Secret value = [987654321]
```

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

13. Finally, remove the module in Terminal 1.

A terminal window with a dark purple background and a grey title bar. The title bar contains window control icons and the text 'usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule'. The terminal shows three lines of text: the first line is a command 'sudo insmod paramsModule.k o dummyStudentId=9999' with the second line being its output; the second line is a command 'sudo rmmod paramsModule'; and the third line is the prompt 'usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule\$' followed by a white cursor.

```
usertest@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.k
o dummyStudentId=9999
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule
usertest@ubuntu:~/Desktop/Tests/Modules/paramsModule$
```

### Section 1.4: Dynamically-Loadable Kernel Modules – Example 3: Dynamically loading and unloading a module by user-space application

In this section we will create a c program that is capable of dynamically load, send parameters, and unload a kernel. Here we will again use three terminals (**Terminal 1, 2 and 3**).

1. In your Desktop, inside the Tests/Modules/ folder, create a new folder called **loadUnloadModule**, and inside it create the file **loaderUnloader.c** with the following code.

```
#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: Ricardo Pontaza
// StudentID: 12345

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=9999"; // 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.
// 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;
}

```

**Note:** This code has 3 sections: Module loading (which shows how to load a module), the middle section (where `getchar()` is located, here you can implement your own code to read the variables from `/sys/module/<module name>/parameters` if needed), and Module unloading (which shows how to unload a module).



2. Also inside Tests/Modules/loadUnloadModule create the **paramsModule02.c** file

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

#include <linux/string.h>

#define AUTHOR "Ricardo Pontaza OS TA 2020 NCTU"    //Change your name here
MODULE_LICENSE("GPL");
MODULE_AUTHOR(AUTHOR);

static char *kernelModuleName = "paramsModule02"; //Change module's name when needed

static int studentId = 12345; // real studentId = 012345, removed 0 for display purposes
module_param(studentId, int, 0644);
MODULE_PARM_DESC(studentId, "Parameter for student Id. (Leading zeros are omitted)");

static long secretValue = 987654321;
module_param(secretValue, long, 0644);
MODULE_PARM_DESC(secretValue, "Parameter for secret value.");

static char *charparameter = "Hello world! Project 02 - Example 03";
module_param(charparameter, charp, 0644);
MODULE_PARM_DESC(charparameter, "states - Hello world");

static int modifyValues = 0;
module_param(modifyValues, int, 0644);
MODULE_PARM_DESC(modifyValues, "Indicates if we must modify the original values or not.");

static int dummyStudentId = -1;
static long dummySecretValue = -2;

static int initialize(void){

    if(modifyValues==1)
    {
        studentId = dummyStudentId;
        secretValue = dummySecretValue;
        charparameter = "This is a dummy message!";
    }

    printk(KERN_INFO "\n[%s - %s] =====\n",kernelModuleName,__func__);
    printk(KERN_INFO "[%s - %s] Hello!\n",kernelModuleName,__func__);
    printk(KERN_INFO "[%s - %s] Student Id = [%d]\n",kernelModuleName, __func__,
studentId);
    printk(KERN_INFO "[%s - %s] String inside module = [%s]\n", kernelModuleName,
__func__, charparameter);
    printk(KERN_INFO "[%s - %s] Secret value = [%ld]\n", kernelModuleName, __func__,
secretValue);

    return 0;
}
```

```
static void clean_exit(void){
    printk(KERN_INFO "\n[%s - %s] =====\n",kernelModuleName,__func__);
    printk(KERN_INFO "[%s - %s] Goodbye!\n",kernelModuleName,__func__);
    printk(KERN_INFO "[%s - %s] Student Id = [%d]\n",kernelModuleName, __func__,
studentId);
    printk(KERN_INFO "[%s - %s] String inside module = [%s]\n", kernelModuleName,
__func__, charparameter);
    printk(KERN_INFO "[%s - %s] Secret value = [%ld]\n", kernelModuleName,
__func__, secretValue);
}

module_init(initialize);
module_exit(clean_exit);
```

3. Also inside the Tests/Modules/loadUnloadModule create a Makefile

```
obj-m = paramsModule02.o

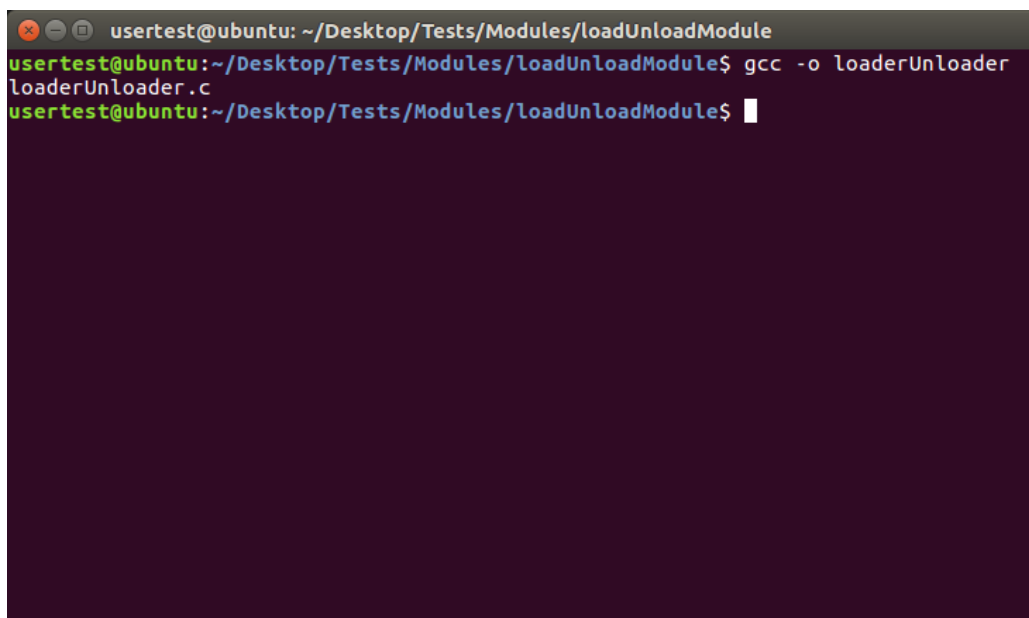
KVERSION = $(shell uname -r)

all:
    make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

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

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

4. Point Terminal 1 to the loadUnloadModule folder, and build the c application using gcc  
**\$ gcc -o loaderUnloader loaderUnloader.c**  
 Also build your module with **\$ make**



```
usertest@ubuntu: ~/Desktop/Tests/Modules/loadUnloadModule
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$ gcc -o loaderUnloader
loaderUnloader.c
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$
```

5. In Terminal 2 open **\$ dmesg -wH**
6. In Terminal 1 run the **./loaderUnloader** application using **sudo**  
**\$ sudo ./loaderUnloader**

```
usertest@ubuntu: ~/Desktop/Tests/Modules/loadUnloadModule
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$ gcc -o loaderUnloader loaderUnloader.c
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$ sudo ./loaderUnloader

This is a dynamic loader and unloader for a kernel module!
Loading module [paramsModule02] with parameters [studentId=9999]...
Module is mounted!

[Press ENTER to continue]
```

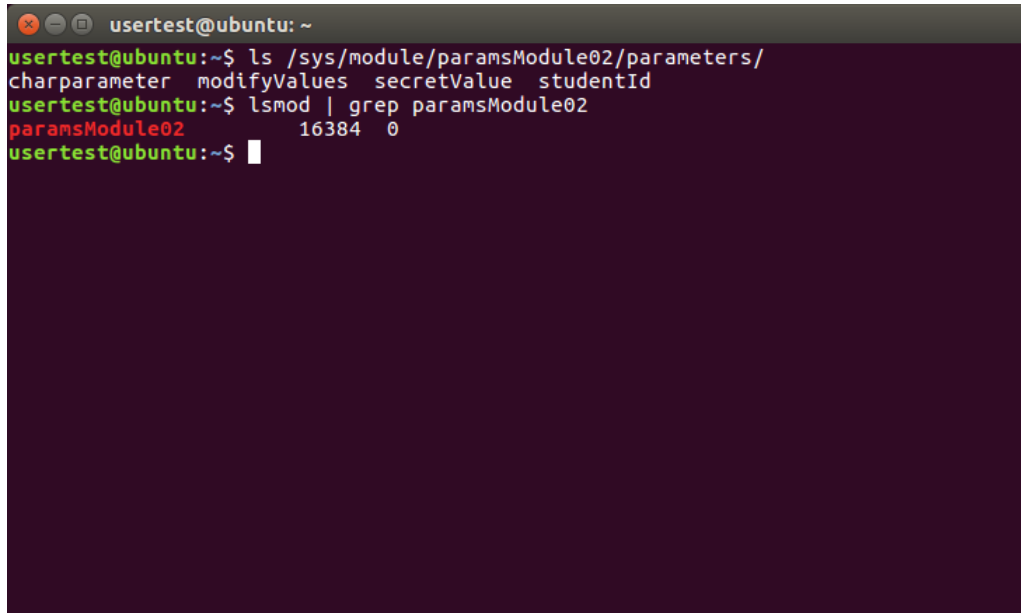
You will see that **./loaderUnloader** is executed until it reaches the **getchar()** command. You will also see the execution of the initialization function in Terminal 2. AT THIS POINT THE MODULE HAS BEEN LOADED WITH **studentId=9999** [Use your real student ID here without leading 0].

```
usertest@ubuntu: ~
usertest@ubuntu:~$ dmesg -wH
[Nov17 10:24]
[paramsModule02 - initialize] =====
[ +0.000001] [paramsModule02 - initialize] Hello!
[ +0.000001] [paramsModule02 - initialize] Student Id = [9999]
[ +0.000000] [paramsModule02 - initialize] String inside module = [Hello world!
Project 02 - Example 03]
[ +0.000001] [paramsModule02 - initialize] Secret value = [987654321]
```

7. Because the module is loaded, you can read the variables inside it in Terminal 3. You can also check that it has been loaded in the modules list by the commands:

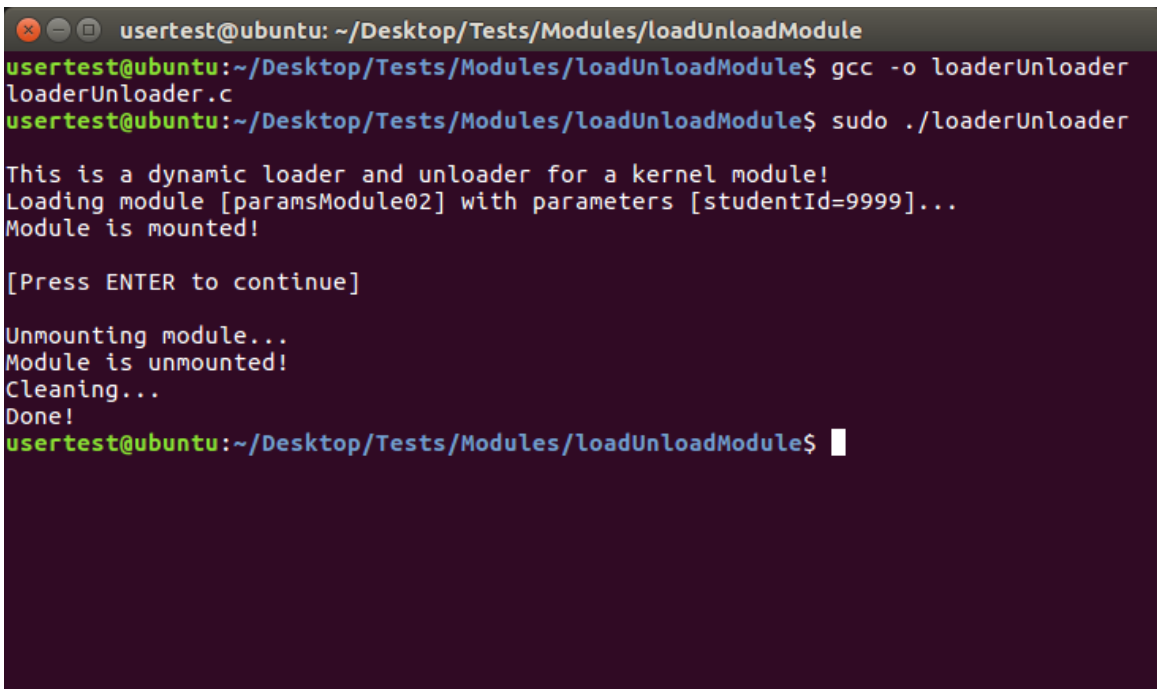
```
$ ls /sys/module/paramsModule02/parameters/
```

```
$ lsmod | grep paramsModule02
```



```
usertest@ubuntu: ~  
usertest@ubuntu:~$ ls /sys/module/paramsModule02/parameters/  
charparameter  modifyValues  secretValue  studentId  
usertest@ubuntu:~$ lsmod | grep paramsModule02  
paramsModule02 16384 0  
usertest@ubuntu:~$
```

8. In Terminal 1 press ENTER. This will continue the execution of `./loaderUnloader`, and it will proceed to unload the module.



```
usertest@ubuntu: ~/Desktop/Tests/Modules/loadUnloadModule  
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$ gcc -o loaderUnloader  
loaderUnloader.c  
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$ sudo ./loaderUnloader  
  
This is a dynamic loader and unloader for a kernel module!  
Loading module [paramsModule02] with parameters [studentId=9999]...  
Module is mounted!  
  
[Press ENTER to continue]  
  
Unmounting module...  
Module is unmounted!  
Cleaning...  
Done!  
usertest@ubuntu:~/Desktop/Tests/Modules/loadUnloadModule$
```

In Terminal 2 you will see that the module was unloaded

```
usertest@ubuntu: ~  
usertest@ubuntu:~$ dmesg -wH  
[Nov17 10:24]  
[paramsModule02 - initialize] =====  
[ +0.000001] [paramsModule02 - initialize] Hello!  
[ +0.000001] [paramsModule02 - initialize] Student Id = [9999]  
[ +0.000000] [paramsModule02 - initialize] String inside module = [Hello world!  
Project 02 - Example 03]  
[ +0.000001] [paramsModule02 - initialize] Secret value = [987654321]  
[Nov17 10:25]  
[paramsModule02 - clean_exit] =====  
[ +0.000002] [paramsModule02 - clean_exit] Goodbye!  
[ +0.000001] [paramsModule02 - clean_exit] Student Id = [9999]  
[ +0.000000] [paramsModule02 - clean_exit] String inside module = [Hello world!  
Project 02 - Example 03]  
[ +0.000001] [paramsModule02 - clean_exit] Secret value = [987654321]  
█
```

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.

```
usertest@ubuntu: ~  
usertest@ubuntu:~$ ls /sys/module/paramsModule02/parameters/  
charparameter modifyValues secretValue studentId  
usertest@ubuntu:~$ lsmod | grep paramsModule02  
paramsModule02      16384  0  
usertest@ubuntu:~$ ls /sys/module/paramsModule02/parameters/  
ls: cannot access '/sys/module/paramsModule02/parameters/': No such file or dire  
ctory  
usertest@ubuntu:~$ lsmod | grep paramsModule02  
usertest@ubuntu:~$ █
```

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

## Section 1.5: Dynamically-Loadable Kernel Modules – Final Exercise: Simple calculator

The previous examples covered the following:

- **Example 1:** How to display a message in a module, and how to (manually) load and unload it.
- **Example 2:** How to send parameters to a module when it is loaded, how to read its variables, how to execute code inside the module and how to (manually) load and unload it.
- **Example 3:** How to (automatically) load a module from inside a c program, how to send parameters to it, how to make the c program to wait and how to (automatically) unload the module from inside the c program.

With all the knowledge acquired from these three examples, you must finish the following task (you can create a folder named Tests/Modules/calculatorModule and work inside it):

1. You are given a **calculator.c** file with the following code

```
#include <stdio.h>
#include <string.h>

long addition (int input1, int input2);
long substraction (int input1, int input2);
long multiplication (int input1, int input2);

// StudentID = 12345 // replace with your student ID

int main() {
    char operator;
    char *operation;

    int input1=0;
    int input2=0;
    long result = 0;

    while(1)
    {
        input1=0;
        input2=0;
        result = 0;

        printf("=====\n");
        printf("Enter operation [sum - sub - mul - exit]: ");
        scanf("%s",operation);

        if(strcmp(operation,"exit")==0){
            break;
        }

        printf("Enter two operands (space separated): ");
        scanf("%d %d", &input1, &input2);
```

```

        if(strcmp(operation,"sum")==0){
            result = addition (input1, input2);
            printf("Operation: [%s] - Operands [%d %d] - Result:[%ld]\n",operation,
                input1, input2, result);
        }else if(strcmp(operation,"sub")==0){
            result = subtraction (input1, input2);
            printf("Operation: [%s] - Operands [%d %d] - Result:[%ld]\n",operation,
                input1, input2, result);
        }else if(strcmp(operation,"mul")==0){
            result = multiplication (input1, input2);
            printf("Operation: [%s] - Operands [%d %d] - Result:[%ld]\n",operation,
                input1, input2, result);
        }else{
            printf("Invalid option\n");
        }
    }

    return 0;
}

long addition (int input1, int input2)
{
    long result = 0;

    //INSERT YOUR CODE HERE TO LOAD/UNLOAD AND USE MODULE

    return result;
}

long subtraction (int input1, int input2)
{
    long result = 0;

    //INSERT YOUR CODE HERE TO LOAD/UNLOAD AND USE MODULE

    return result;
}

long multiplication (int input1, int input2)
{
    long result = 0;

    //INSERT YOUR CODE HERE TO LOAD/UNLOAD AND USE MODULE

    return result;
}

```

2. You are also given a **calculatorModule.c** module

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

#include <linux/string.h>

#define AUTHOR "Ricardo Pontaza OS TA 2020 NCTU"    //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' - subtraction / '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\n",
           calculatorModule!\n",kernelModuleName,__func__);

    // INSERT YOUR CODE HERE
    // Perform addition, subtraction or multiplication of firstParam and secondParam
    // depending on the value of operationParam.
    // If operationParam has an invalid value, return 0.

    printk(KERN_INFO "[%s - %s] Operation = %s\n",
           kernelModuleName,__func__,operationParam);
    printk(KERN_INFO "[%s - %s] First parameter = %d\n",
           kernelModuleName,__func__,firstParam);
    printk(KERN_INFO "[%s - %s] Second parameter = %d\n",
           kernelModuleName,__func__,secondParam);
    printk(KERN_INFO "[%s - %s] Result = %ld\n", kernelModuleName,__func__,resultParam);

    return 0;
}
```



```

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

    printk(KERN_INFO "[%s - %s] Operation = %s\n",
        kernelModuleName,__func__,operationParam);
    printk(KERN_INFO "[%s - %s] First parameter = %d\n",
        kernelModuleName,__func__,firstParam);
    printk(KERN_INFO "[%s - %s] Second parameter = %d\n",
        kernelModuleName,__func__,secondParam);
    printk(KERN_INFO "[%s - %s] Result = %ld\n",
        kernelModuleName,__func__,resultParam);
}

module_init(initialize);
module_exit(clean_exit);

```

### 3. And a Makefile

```

obj-m = calculatorModule.o

KVERSION = $(shell uname -r)

all:
    make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean

```

### 4. Your objective is simple. Finish both the **calculator.c** and **calculatorModule.c** files, so the following happens:

- a. The calculatorModule.ko is able to calculate additions, subtractions and multiplications of **firstParam** and **secondParam**, depending on the value of **operationParam**.
- b. The valid values of **operationParam** are:
  - i. sum = addition
  - ii. sub = subtraction
  - iii. mul = multiplication
  - iv. (other) = return 0
- c. The generated application **./calculator** must take parameters as follows:
  - i. Ask for operation (sum – sub – mul – exit)
  - ii. Ask for two integer operands separated by space
  - iii. Return the selected operation, operands and result (please refer to screenshots)
- d. The application **./calculator** must:
  - i. **Dynamically** load the **calculatorModule.ko**,
  - ii. Calculate the result inside the module,
  - iii. Read the result from **/sys/module/calculatorModule/parameters**,
  - iv. Display to user, and
  - v. Unload the module.

```

userstest@ubuntu: ~/Desktop/Tests/Modules/calculatorModule
userstest@ubuntu:~/Desktop/Tests/Modules/calculatorModule$ gcc -o calculator calculator.c
userstest@ubuntu:~/Desktop/Tests/Modules/calculatorModule$ sudo ./calculator
[sudo] password for userstest:
=====
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]: 

```

```

userstest@ubuntu: ~
userstest@ubuntu:~$ dmesg -wH
[Nov17 13:29]
[calculatorModule - initialize] =====
[ +0.000001] [calculatorModule - initialize] Hello from calculatorModule!
[ +0.000001] [calculatorModule - initialize] Operation = sum
[ +0.000000] [calculatorModule - initialize] First parameter = 3
[ +0.000001] [calculatorModule - initialize] Second parameter = 5
[ +0.000000] [calculatorModule - initialize] Result = 8
[Nov17 13:30]
[calculatorModule - clean_exit] =====
[ +0.000001] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[ +0.000002] [calculatorModule - clean_exit] Operation = sum
[ +0.000001] [calculatorModule - clean_exit] First parameter = 3
[ +0.000000] [calculatorModule - clean_exit] Second parameter = 5
[ +0.000001] [calculatorModule - clean_exit] Result = 8

```

```
usertest@ubuntu: ~/Desktop/Tests/Modules/calculatorModule
usertest@ubuntu:~/Desktop/Tests/Modules/calculatorModule$ gcc -o calculator calculator.c
usertest@ubuntu:~/Desktop/Tests/Modules/calculatorModule$ sudo ./calculator
[sudo] password for usertest:
=====
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]: mul
Enter two operands (space separated): 4 3
Operation: [mul] - Operands [4 3] - Result:[12]
=====
Enter operation [sum - sub - mul - exit]:
```

```
usertest@ubuntu: ~
[ +0.000001] [calculatorModule - initialize] Second parameter = 5
[ +0.000000] [calculatorModule - initialize] Result = 8
[Nov17 13:30]
[calculatorModule - clean_exit] =====
[ +0.000001] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[ +0.000002] [calculatorModule - clean_exit] Operation = sum
[ +0.000001] [calculatorModule - clean_exit] First parameter = 3
[ +0.000000] [calculatorModule - clean_exit] Second parameter = 5
[ +0.000001] [calculatorModule - clean_exit] Result = 8
[ +43.386742]
[calculatorModule - initialize] =====
[ +0.000001] [calculatorModule - initialize] Hello from calculatorModule!
[ +0.000001] [calculatorModule - initialize] Operation = mul
[ +0.000000] [calculatorModule - initialize] First parameter = 4
[ +0.000001] [calculatorModule - initialize] Second parameter = 3
[ +0.000000] [calculatorModule - initialize] Result = 12
[Nov17 13:31]
[calculatorModule - clean_exit] =====
[ +0.000002] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[ +0.000001] [calculatorModule - clean_exit] Operation = mul
[ +0.000000] [calculatorModule - clean_exit] First parameter = 4
[ +0.000001] [calculatorModule - clean_exit] Second parameter = 3
[ +0.000001] [calculatorModule - clean_exit] Result = 12
```

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

**NOTES:**

1. **YOU MUST UPLOAD THE FINISHED calculator.c AND calculatorModule.c TO E3.**
2. We will build your code (using gcc) and your module (using \$ make) and test your files with random numbers and operations.

**HINTS:**

1. First finish the module, manually test it and check that all the cases work.
2. Search online how to read text files from inside a c program (Check fopen – fclose).