Introduction to Operating Systems

Project 3: Dynamically-Loadable Kernel Modules (DLKMs)

(To be lectured on 2021-11-26)

Instructor: Prof. Ying-Dar Lin (林盈達)
TA: Ricardo Pontaza

Deadline:
2021-12-29 (Wed) at 23:50:00

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** (5-10 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 1.4)**
   b. **calculatorModule.c (Section 1.4)**
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.

Projects Distribution:

1. Project 01: Linux Kernel Download, Patch, Compilation, Debugging and Profiling
   (Ubuntu 16.04.7 **Server** - Kernel **4.4.101** + Ubuntu 16.04.7 **Desktop** – Kernel **4.15.0-142**)
2. Project 02: Linux Kernel System Calls
   (Ubuntu **20.04.3 Desktop** – Kernel **4.19.148** + Kernel **5.13.19**)
3. **Project 03: Dynamically-Loadable Kernel Modules (DLKMs)**
   (Ubuntu **20.04.3 Desktop** – Kernel **4.19.148** + Kernel **5.13.19**)

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 1.2). 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 1.2 – 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 1.2 (paramsModule.c), the variable **charparameter** is of type **charp**. What is charp?

Additional questions (also answer in the report):

11. Which project (01 / 02 / 03) did you like the most? Why?
12. Which project (01 / 02 / 03) did you like the least? Why?
13. Did you learn anything new with these three projects? What did you learn?
14. Do you think these projects can help you in the future, if you look for a job in the industry?

Table of contents:

2. <u>Example 2 - Sending parameters to a module and reading and modifying module variables:</u>
    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. <u>Example 3 - Dynamically loading and unloading a module by user-space application:</u>
    a. Module creation.
    b. (Dynamically) module load and unload by c program.
    c. Send parameters to module from c program.
4. <u>Final exercise – Simple calculator</u>

Operating Systems Project Report

Project Number (01 / 02 / 03):	
Name:	
Student ID:	
YouTube link (Format youtube.com/watch?v=[key]):	https://www.youtube.com/watch?v=
Date (YYYY-MM-DD):	
Names of the files uploaded to E3:	
Physical Machine Total RAM (Example: 8.0 GB):	
Physical Machine CPU (Example: Intel i7-2600K):	

Checklist	
Yes/No	**Item**
	The report name follows the format "OS_Project03_StudentID.pdf".
	The report was uploaded to E3 before the deadline.
	The YouTube video is public, and anyone with the link can watch it.
	The audio of the video has a good volume.
	The pictures in your report and video have a good quality.
	All the questions and exercises were answered inside the report.
	I understand that late submission is late submission, regardless of the time uploaded.
	I understand that any cheating in my report / video / code will not be tolerated.

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

Section 1: Dynamically-Loadable Kernel Modules

Section 1.1: 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 virtual machine you used for Project 02.

1. In the Desktop, 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>
#include <linux/init.h>

#define DRIVER_AUTHOR "Ricardo Pontaza - OS TA 2021" // Replace with your name and
student ID
#define DRIVER_DESC   "A sample driver - OS Project 03"


static int studentId = 12345;

static int initialize(void)
{
      printk(KERN_INFO "[%d] : Function [%s] - Hello from OS Project 03!\n", stu-
dentId,__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);

MODULE_LICENSE("GPL");

MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
```

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 you created.]

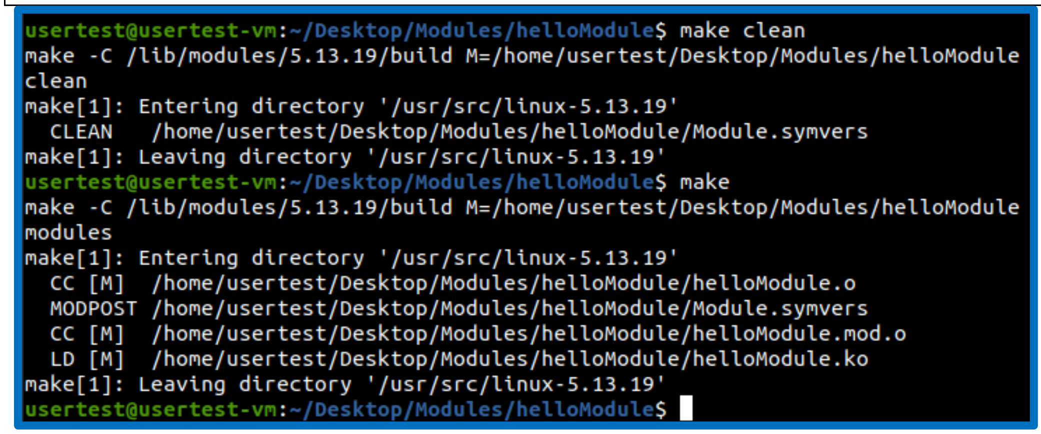
Important Note: To compile, load and unload a kernel module it is not necessary to re-compile the whole kernel.

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 **(without sudo)**

```
$ make clean
$ make
```



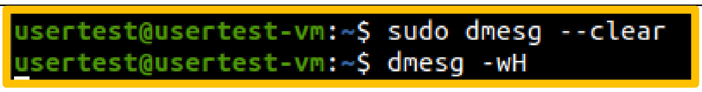
```
usertest@usertest-vm:~/Desktop/Modules/helloModule$ make clean
make -C /lib/modules/5.13.19/build M=/home/usertest/Desktop/Modules/helloModule
clean
make[1]: Entering directory '/usr/src/linux-5.13.19'
  CLEAN   /home/usertest/Desktop/Modules/helloModule/Module.symvers
make[1]: Leaving directory '/usr/src/linux-5.13.19'
usertest@usertest-vm:~/Desktop/Modules/helloModule$ make
make -C /lib/modules/5.13.19/build M=/home/usertest/Desktop/Modules/helloModule
modules
make[1]: Entering directory '/usr/src/linux-5.13.19'
  CC [M]  /home/usertest/Desktop/Modules/helloModule/helloModule.o
  MODPOST /home/usertest/Desktop/Modules/helloModule/Module.symvers
  CC [M]  /home/usertest/Desktop/Modules/helloModule/helloModule.mod.o
  LD [M]  /home/usertest/Desktop/Modules/helloModule/helloModule.ko
make[1]: Leaving directory '/usr/src/linux-5.13.19'
usertest@usertest-vm:~/Desktop/Modules/helloModule$
```

T1

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

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

```
$ sudo dmesg --clear
$ dmesg -wH
```



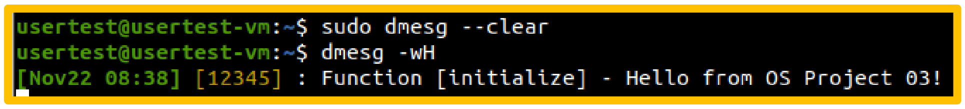
```
usertest@usertest-vm:~$ sudo dmesg --clear
usertest@usertest-vm:~$ dmesg -wH
```

T2

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

```
$ sudo insmod helloModule.ko
```

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



```
usertest@usertest-vm:~$ sudo dmesg --clear
usertest@usertest-vm:~$ dmesg -wH
[Nov22 08:38] [12345] : Function [initialize] - Hello from OS Project 03!
```

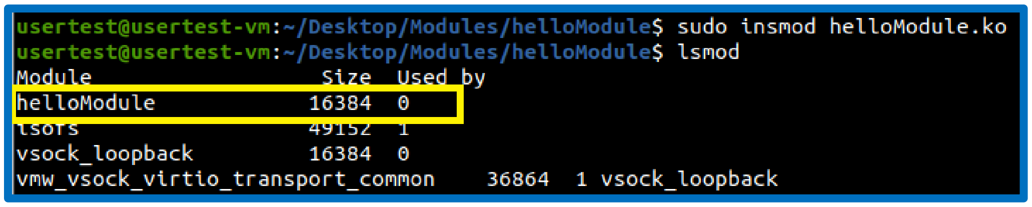
T2

[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 in **Terminal 1**

```
$ lsmod
```

we can see that there is a module called **helloModule** now in the modules list.

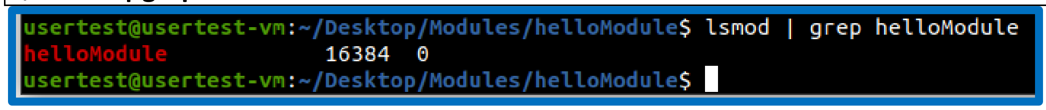


```
usertest@usertest-vm:~/Desktop/Modules/helloModule$ sudo insmod helloModule.ko
usertest@usertest-vm:~/Desktop/Modules/helloModule$ lsmod
Module                  Size  Used by
helloModule            16384  0
isofs                  49152  1
vsock_loopback         16384  0
vmw_vsock_virtio_transport_common    36864  1 vsock_loopback
```

T1

9. In **Terminal 1,** you can also check if the module was loaded or not by running the command

$ lsmod | grep helloModule



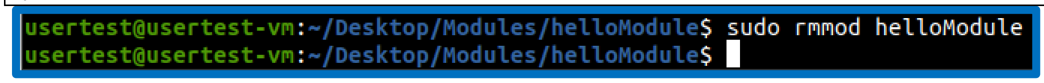
```
usertest@usertest-vm:~/Desktop/Modules/helloModule$ lsmod | grep helloModule
helloModule            16384  0
usertest@usertest-vm:~/Desktop/Modules/helloModule$ 
```

T1

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

10. To unload the module, in **Terminal 1** type

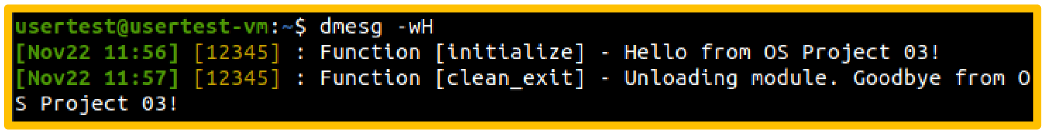
$ sudo rmmod helloModule



```
usertest@usertest-vm:~/Desktop/Modules/helloModule$ sudo rmmod helloModule
usertest@usertest-vm:~/Desktop/Modules/helloModule$ 
```

T1

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

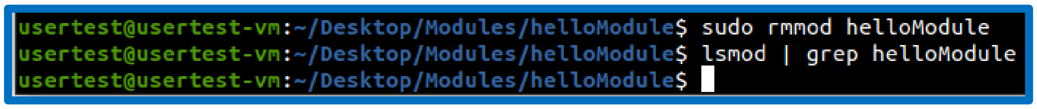


```
usertest@usertest-vm:~$ dmesg -wH
[Nov22 11:56] [12345] : Function [initialize] - Hello from OS Project 03!
[Nov22 11:57] [12345] : Function [clean_exit] - Unloading module. Goodbye from O
S Project 03!
```

T2

[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@usertest-vm:~/Desktop/Modules/helloModule$ sudo rmmod helloModule
usertest@usertest-vm:~/Desktop/Modules/helloModule$ lsmod | grep helloModule
usertest@usertest-vm:~/Desktop/Modules/helloModule$ 
```

T1

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

IMPORTANT NOTE: When you run the commands **$ make** and **$ make clean**, **DO NOT USE $ sudo**. If you do it, there is a possibility that you end up deleting part of your kernel and you will have to recompile the kernel all over again. (If you do it, when you run **$ make,** it might take several hours to finish).

Section 1.2: 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 **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 DRIVER_AUTHOR "Ricardo Pontaza - OS TA 2021" // Replace with your name and
student ID
#define DRIVER_DESC   "Example of how to send parameters to Module when loading - OS
Project 03"

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

static int studentId = 12345; // real studentId = 012345, removed 0 for display pur-
poses
module_param(studentId, int, 0644);
MODULE_PARM_DESC(studentId, "Parameter for student Id. (Leading zeros are omit-
ted)");

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

static char *charparameter = "Hello world! Project 03 - 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);

MODULE_LICENSE("GPL");
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
```

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**, clean dmesg and open dmesg with **$ dmesg -wH.**
6. Mount the module in **Terminal 1** with

```
$ sudo insmod paramsModule.ko
```

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

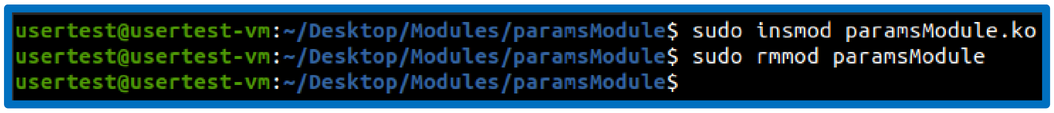
- studentId (12345),
- secretValue (987654321), and
- charparameter (“Hello world! Project 03 – Example 02”)

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

```
$ sudo rmmod paramsModule.ko
```

We see that the values studentId, secretValue and charparameter in **Terminal 2** **do not change**. (Please read the code in **paramsModule.c** to see where and how these values are defined).

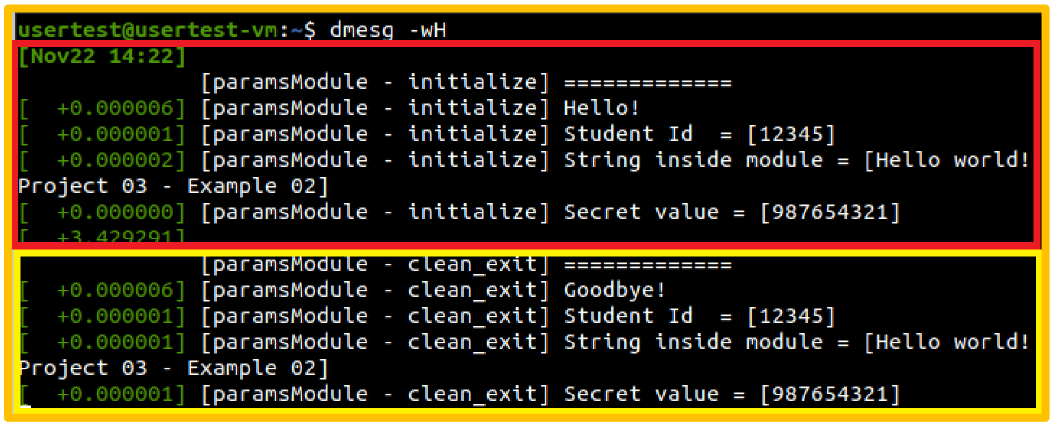
Terminal 1:



```
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ sudo insmod paramsModule.ko
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ sudo rmmod paramsModule
usertest@usertest-vm:~/Desktop/Modules/paramsModule$
```

T1

Terminal 2:



```
usertest@usertest-vm:~$ dmesg -wH
[Nov22 14:22]
                [paramsModule - initialize] =============
[  +0.000006] [paramsModule - initialize] Hello!
[  +0.000001] [paramsModule - initialize] Student Id  = [12345]
[  +0.000002] [paramsModule - initialize] String inside module = [Hello world!
Project 03 - Example 02]
[  +0.000000] [paramsModule - initialize] Secret value = [987654321]
[  +3.429291]
                [paramsModule - clean_exit] =============
[  +0.000006] [paramsModule - clean_exit] Goodbye!
[  +0.000001] [paramsModule - clean_exit] Student Id  = [12345]
[  +0.000001] [paramsModule - clean_exit] String inside module = [Hello world!
Project 03 - Example 02]
[  +0.000001] [paramsModule - clean_exit] Secret value = [987654321]
```

T2

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 in Terminal 2, and the commands you used to do so in Terminal 1].

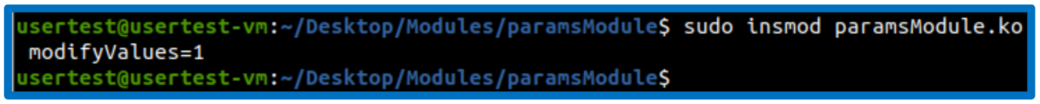
7. We can execute code when the module is loaded (and also when it is unloaded). Inside the function **initialize** we have the following piece of code

```
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

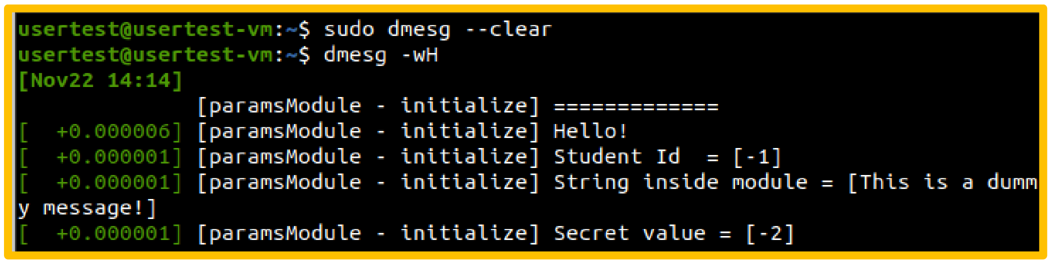
```
$ sudo insmod paramsModule.ko modifyValues=1
```



```
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ sudo insmod paramsModule.ko
 modifyValues=1
usertest@usertest-vm:~/Desktop/Modules/paramsModule$
```

T1

And in Terminal 2 you will see the modified values.

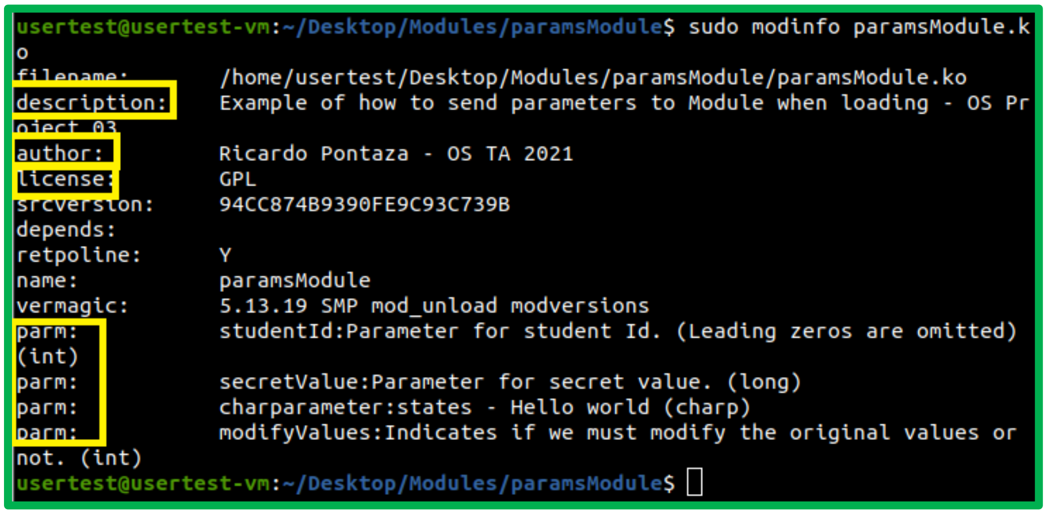


```
usertest@usertest-vm:~$ sudo dmesg --clear
usertest@usertest-vm:~$ dmesg -wH
[Nov22 14:14]
              [paramsModule - initialize] =============
[  +0.000006] [paramsModule - initialize] Hello!
[  +0.000001] [paramsModule - initialize] Student Id  = [-1]
[  +0.000001] [paramsModule - initialize] String inside module = [This is a dumm
y message!]
[  +0.000001] [paramsModule - initialize] Secret value = [-2]
```

T2

8. Point **Terminal 3** to the folder where you have the module, and run the command

```
$ sudo modinfo paramsModule.ko
```



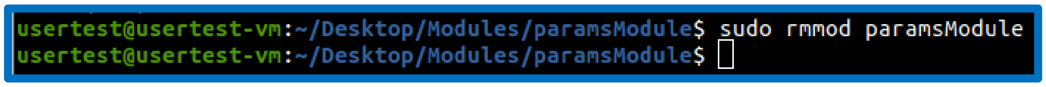
```
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ sudo modinfo paramsModule.k
o
filename:       /home/usertest/Desktop/Modules/paramsModule/paramsModule.ko
description:    Example of how to send parameters to Module when loading - OS Pr
oject 03
author:         Ricardo Pontaza - OS TA 2021
license:        GPL
srcversion:     94CC874B9390FE9C93C739B
depends:
retpoline:      Y
name:           paramsModule
vermagic:       5.13.19 SMP mod_unload modversions
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@usertest-vm:~/Desktop/Modules/paramsModule$
```

T3

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**).

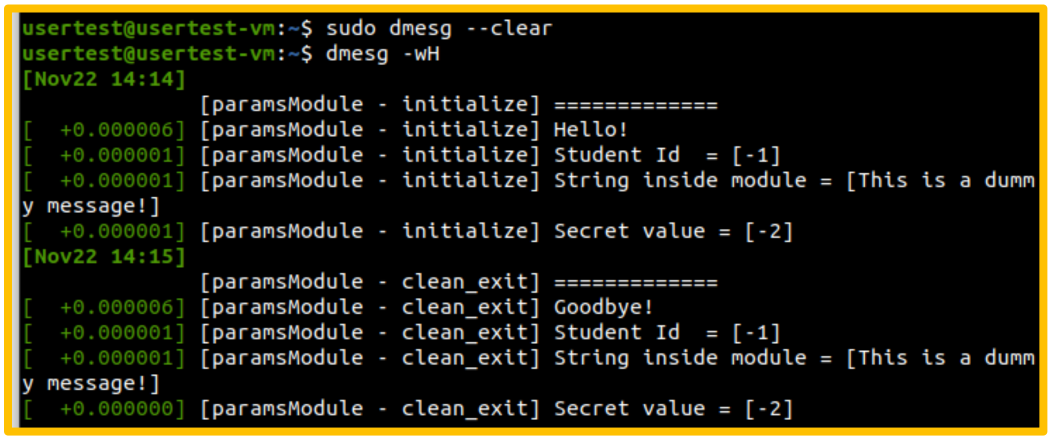
IMPORTANT NOTE: Note that **paramsModule.c** has two variables (**dummyStudentId** and **dummySecretValue** that are not displayed by **modinfo**.

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



```
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ sudo rmmod paramsModule
usertest@usertest-vm:~/Desktop/Modules/paramsModule$
```

T1

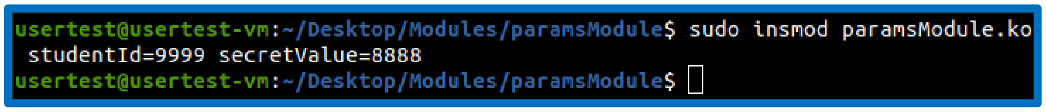


```
usertest@usertest-vm:~$ sudo dmesg --clear
usertest@usertest-vm:~$ dmesg -wH
[Nov22 14:14]
              [paramsModule - initialize] =============
[  +0.000006] [paramsModule - initialize] Hello!
[  +0.000001] [paramsModule - initialize] Student Id  = [-1]
[  +0.000001] [paramsModule - initialize] String inside module = [This is a dumm
y message!]
[  +0.000001] [paramsModule - initialize] Secret value = [-2]
[Nov22 14:15]
              [paramsModule - clean_exit] =============
[  +0.000006] [paramsModule - clean_exit] Goodbye!
[  +0.000001] [paramsModule - clean_exit] Student Id  = [-1]
[  +0.000001] [paramsModule - clean_exit] String inside module = [This is a dumm
y message!]
[  +0.000000] [paramsModule - clean_exit] Secret value = [-2]
```

T2

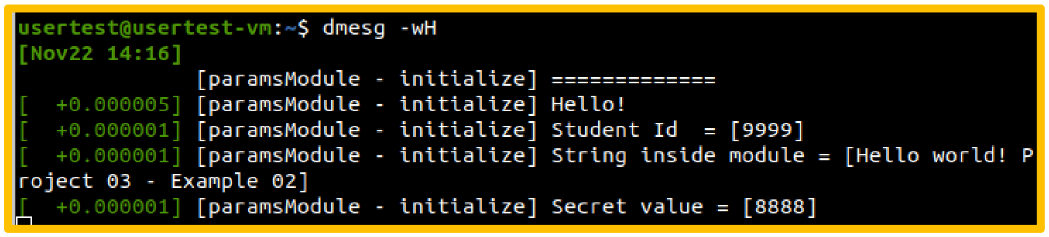
[Screenshot #9-10: Create two screenshots explaining steps 7 and 8.]

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@usertest-vm:~/Desktop/Modules/paramsModule$ sudo insmod paramsModule.ko
 studentId=9999 secretValue=8888
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ 
```

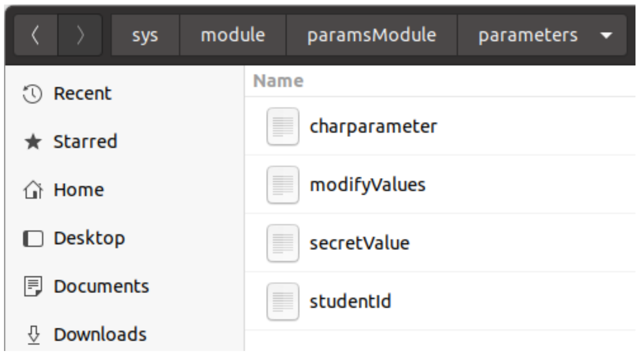
T1



```
usertest@usertest-vm:~$ dmesg -wH
[Nov22 14:16]
              [paramsModule - initialize] =============
[  +0.000005] [paramsModule - initialize] Hello!
[  +0.000001] [paramsModule - initialize] Student Id  = [9999]
[  +0.000001] [paramsModule - initialize] String inside module = [Hello world! P
roject 03 - Example 02]
[  +0.000001] [paramsModule - initialize] Secret value = [8888]
```

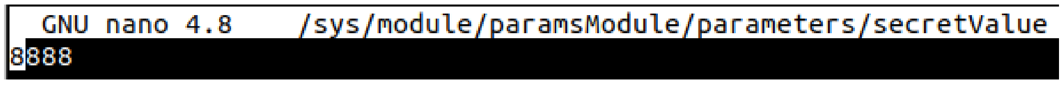
T2

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



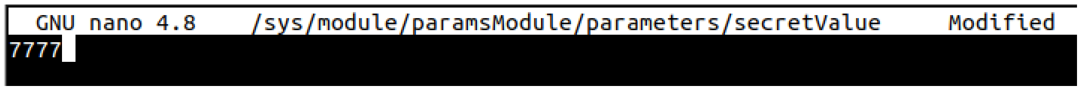
In Terminal 3, we can modify the value of those variables with nano:

```
$ sudo nano /sys/module/paramsModule/parameters/secretValue
```



```
  GNU nano 4.8    /sys/module/paramsModule/parameters/secretValue
8888
```

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

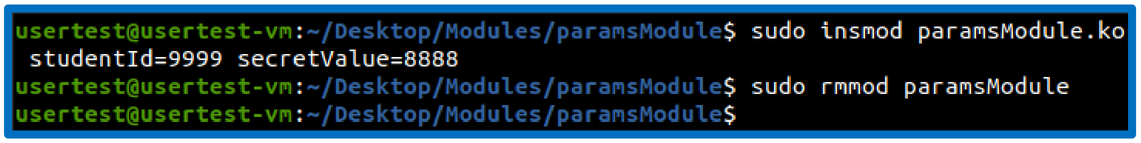


```
  GNU nano 4.8     /sys/module/paramsModule/parameters/secretValue     Modified
7777
```

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.

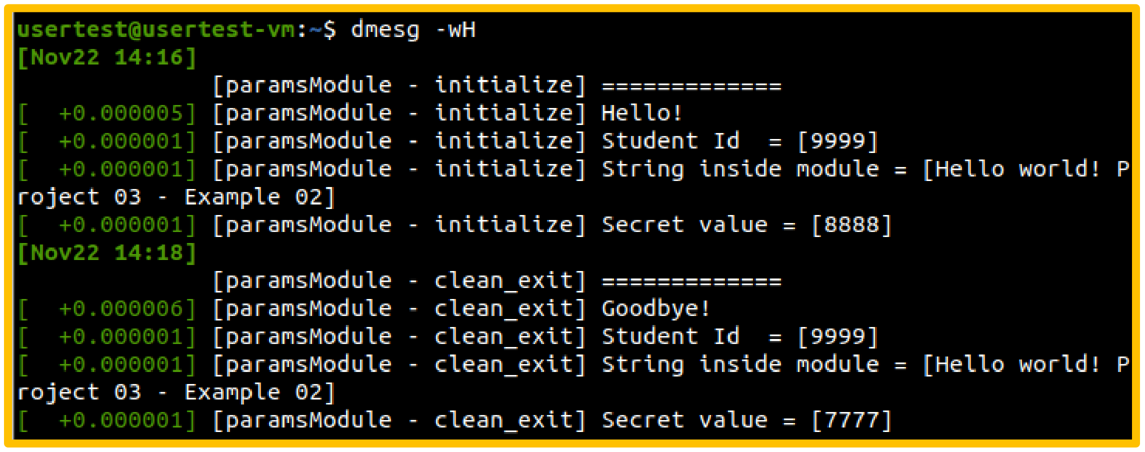
If you get the **ERROR: Module <name> is in use**, then you must fix the value you just saved, and restart your machine.

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@usertest-vm:~/Desktop/Modules/paramsModule$ sudo insmod paramsModule.ko
 studentId=9999 secretValue=8888
usertest@usertest-vm:~/Desktop/Modules/paramsModule$ sudo rmmod paramsModule
usertest@usertest-vm:~/Desktop/Modules/paramsModule$
```

T1



```
usertest@usertest-vm:~$ dmesg -wH
[Nov22 14:16]
              [paramsModule - initialize] =============
[  +0.000005] [paramsModule - initialize] Hello!
[  +0.000001] [paramsModule - initialize] Student Id  = [9999]
[  +0.000001] [paramsModule - initialize] String inside module = [Hello world! P
roject 03 - Example 02]
[  +0.000001] [paramsModule - initialize] Secret value = [8888]
[Nov22 14:18]
              [paramsModule - clean_exit] =============
[  +0.000006] [paramsModule - clean_exit] Goodbye!
[  +0.000001] [paramsModule - clean_exit] Student Id  = [9999]
[  +0.000001] [paramsModule - clean_exit] String inside module = [Hello world! P
roject 03 - Example 02]
[  +0.000001] [paramsModule - clean_exit] Secret value = [7777]
```

T2

[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.

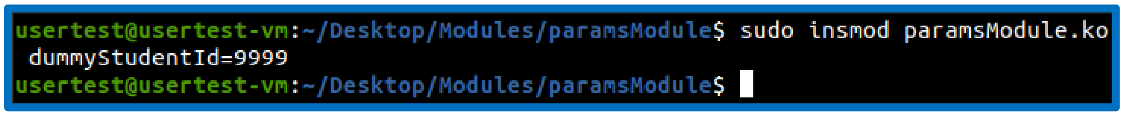
In **Terminal 1** mount the module with

$ sudo insmod paramsModule.ko dummyStudentId=9999

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

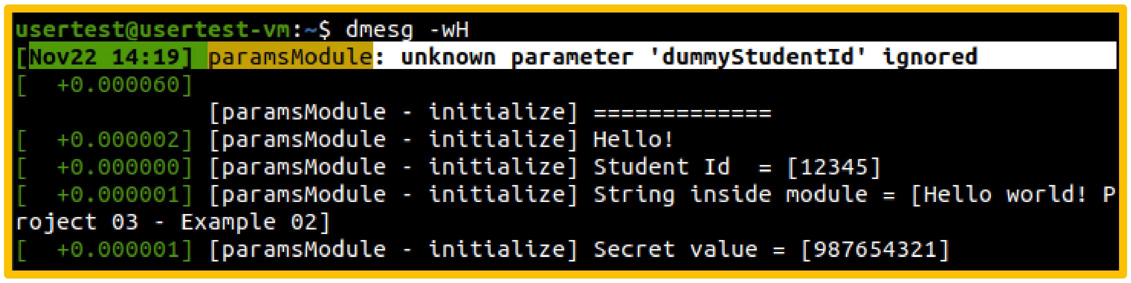
paramsModule: unknown parameter 'dummyStudentId' ignored

appears.



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

T1



```
usertest@usertest-vm:~$ dmesg -wH
[Nov22 14:19] paramsModule: unknown parameter 'dummyStudentId' ignored
[  +0.000060]
              [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 03 - Example 02]
[  +0.000001] [paramsModule - initialize] Secret value = [987654321]
```

T2

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

13. Finally, remove the module in **Terminal 1**.

<u>Section 1.3: Dynamically-Loadable Kernel Modules – Example 3:</u>
<u>Dynamically loading and unloading a module by user-space application</u>

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

1. In your Desktop, inside the **Modules/ folder**, create a new folder called **loadUnloadModule,** and inside it creates 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, mod-
ule_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: 012345

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 lead-
ing 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",moduleNameNoExten-
sion,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 exten-
sion>/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 **Modules/loadUnloadModule** create the **paramsModule02.c** file

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

#define DRIVER_AUTHOR "Ricardo Pontaza - OS TA 2021" // Replace with your name and
student ID
#define DRIVER_DESC   "Example of how to dynamically load and unload a module from
user space - OS Project 03"

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

static int studentId = 12345; // real studentId = 012345, removed 0 for display pur-
poses
module_param(studentId, int, 0644);
MODULE_PARM_DESC(studentId, "Parameter for student Id. (Leading zeros are omit-
ted)");

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);

MODULE_LICENSE("GPL");
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
```

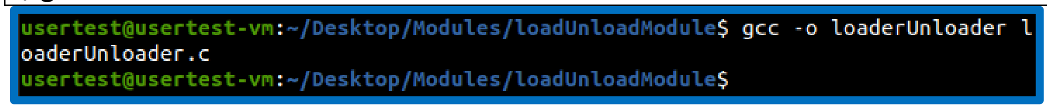
3. Also, inside the **Modules/loadUnloadModule** create a Makefile

```
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

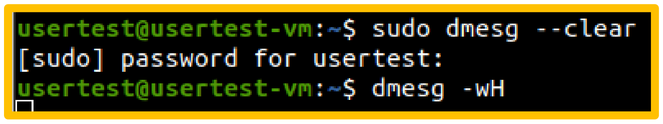


```
usertest@usertest-vm:~/Desktop/Modules/loadUnloadModule$ gcc -o loaderUnloader l
oaderUnloader.c
usertest@usertest-vm:~/Desktop/Modules/loadUnloadModule$
```

T1

Also, build the module (**$make clean** and **$make**).

5. In **Terminal 2,** open **$ dmesg -wH**

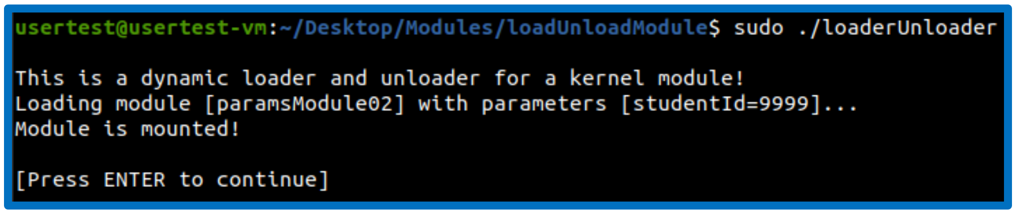


```
usertest@usertest-vm:~$ sudo dmesg --clear
[sudo] password for usertest:
usertest@usertest-vm:~$ dmesg -wH
```

T2

6. In **Terminal 1,** run the ./loaderUnloader application using sudo

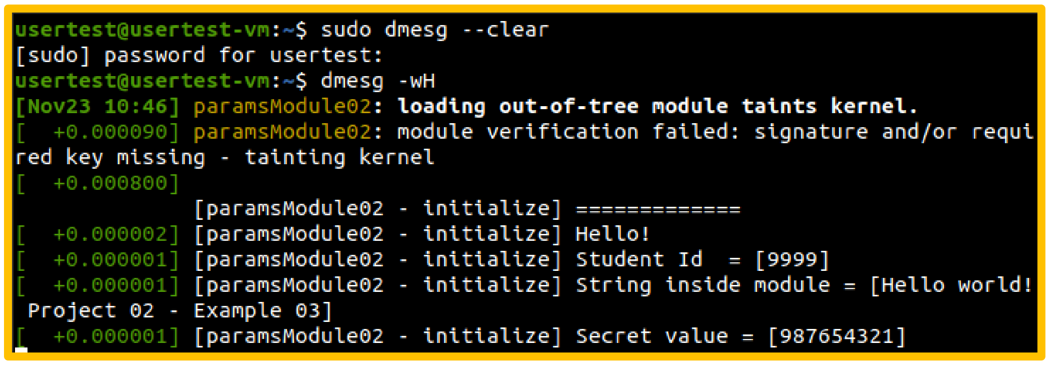
$ sudo ./loaderUnloader



```
usertest@usertest-vm:~/Desktop/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]
```

T1

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]**.

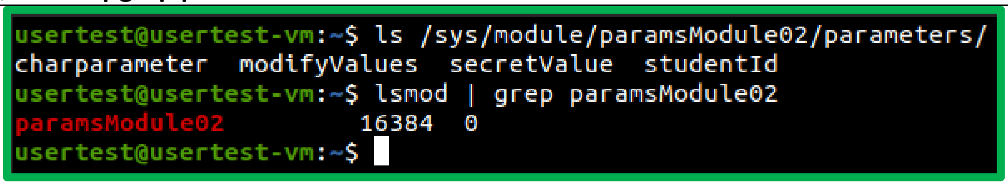


```
usertest@usertest-vm:~$ sudo dmesg --clear
[sudo] password for usertest:
usertest@usertest-vm:~$ dmesg -wH
[Nov23 10:46] paramsModule02: loading out-of-tree module taints kernel.
[  +0.000090] paramsModule02: module verification failed: signature and/or requi
red key missing - tainting kernel
[  +0.000800]
              [paramsModule02 - initialize] =============
[  +0.000002] [paramsModule02 - initialize] Hello!
[  +0.000001] [paramsModule02 - initialize] Student Id  = [9999]
[  +0.000001] [paramsModule02 - initialize] String inside module = [Hello world!
 Project 02 - Example 03]
[  +0.000001] [paramsModule02 - initialize] Secret value = [987654321]
```

T2

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:

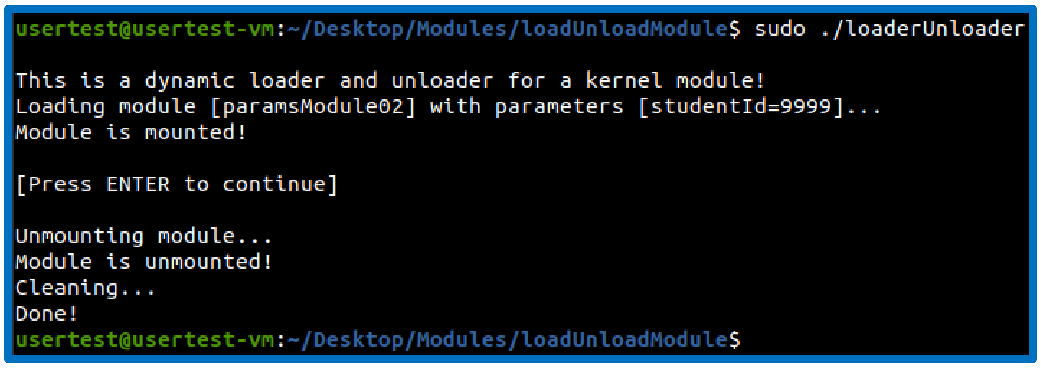
```
$ ls /sys/module/paramsModule02/parameters/
$ lsmod | grep paramsModule02
```



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

T3

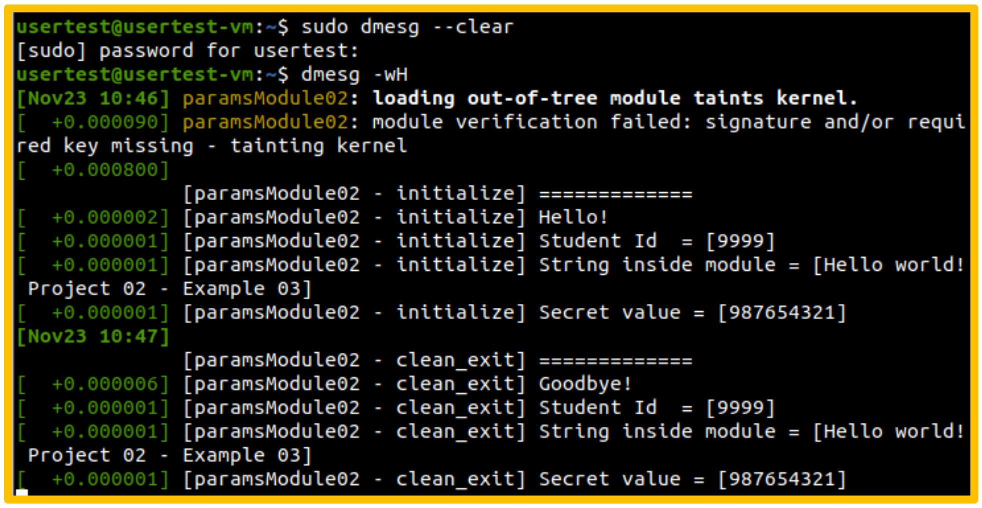
8. In **Terminal 1** press ENTER. This will continue the execution of **./loaderUnloader**, and it will proceed to unload the module.



```
usertest@usertest-vm:~/Desktop/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@usertest-vm:~/Desktop/Modules/loadUnloadModule$
```

T1

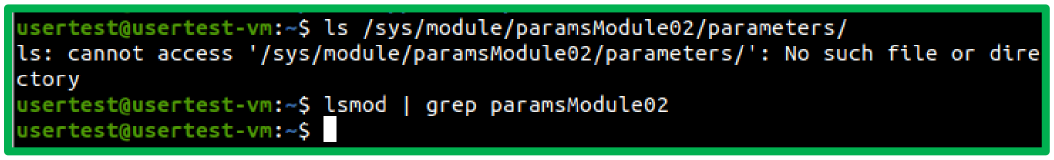
In **Terminal 2**, you will see that the module was unloaded



```
usertest@usertest-vm:~$ sudo dmesg --clear
[sudo] password for usertest:
usertest@usertest-vm:~$ dmesg -wH
[Nov23 10:46] paramsModule02: loading out-of-tree module taints kernel.
[  +0.000090] paramsModule02: module verification failed: signature and/or requi
red key missing - tainting kernel
[  +0.000800]
              [paramsModule02 - initialize] =============
[  +0.000002] [paramsModule02 - initialize] Hello!
[  +0.000001] [paramsModule02 - initialize] Student Id  = [9999]
[  +0.000001] [paramsModule02 - initialize] String inside module = [Hello world!
 Project 02 - Example 03]
[  +0.000001] [paramsModule02 - initialize] Secret value = [987654321]
[Nov23 10:47]
              [paramsModule02 - clean_exit] =============
[  +0.000006] [paramsModule02 - clean_exit] Goodbye!
[  +0.000001] [paramsModule02 - clean_exit] Student Id  = [9999]
[  +0.000001] [paramsModule02 - clean_exit] String inside module = [Hello world!
 Project 02 - Example 03]
[  +0.000001] [paramsModule02 - clean_exit] Secret value = [987654321]
```

T2

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@usertest-vm:~$ ls /sys/module/paramsModule02/parameters/
ls: cannot access '/sys/module/paramsModule02/parameters/': No such file or dire
ctory
usertest@usertest-vm:~$ lsmod | grep paramsModule02
usertest@usertest-vm:~$ 
```

T3

[Screenshot #17-18: Create two screenshots explaining steps 4,5,6,7 and 8.]

Section 1.4: 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 **Modules/calculatorModule** and work inside it):

1. You are given a **calculator.c** file 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>
#include <string.h>

//ADD ADDITIONAL DEFINES HERE

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

int LoadModule(const char* params);
int UnLoadModule();
void SetParamString(char* parm, int input1, int input2, const char* op);
long GetResult();

// StudentID = 012345 // replace with your student ID

// ADD ADDITIONAL VARIABLES HERE
// You must load and unload the "calculatorModule.ko" module.

int main(int argc, char** argv)
{
	long result = -9999999;
	long resultError = -9999999;

	if (argc != 4) {
		printf("%ld\n", resultError);
		return resultError;
	}

	int param01 = atoi(argv[1]);
	int param02 = atoi(argv[2]);
	int operationError = 1;
	if (strcmp(argv[3], "add") == 0) {
		operationError = addition(&result, param01, param02);
	}
	else if (strcmp(argv[3], "sub") == 0) {
		operationError = substraction(&result, param01, param02);
	}
	else if (strcmp(argv[3], "mul") == 0) {
		operationError = multiplication(&result, param01, param02);
	}

	if (operationError == EXIT_SUCCESS) {
		printf("%ld\n", result);
	}
	else {
		printf("%ld\n", resultError);
	}

	return 0;
}

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

	//INSERT YOUR CODE HERE
	// Your code must call SetParamString, LoadModule and UnLoadModule.
	// It also must return 0 if success, or EXIT_FAILURE if failure.
	// The result of the operation must be stored in the variable *result.

	return operationError;
}

int substraction(long* result, int input1, int input2)
{
	*result = 0;
	int operationError = 0;

	//INSERT YOUR CODE HERE
	// Your code must call SetParamString, LoadModule and UnLoadModule.
	// It also must return 0 if success, or EXIT_FAILURE if failure.
	// The result of the operation must be stored in the variable *result.

	return operationError;
}

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

	//INSERT YOUR CODE HERE
	// Your code must call SetParamString, LoadModule and UnLoadModule.
	// It also must return 0 if success, or EXIT_FAILURE if failure.
	// The result of the operation must be stored in the variable *result.

	return operationError;
}
```

```
void SetParamString(char* parm, int input1, int input2, const char* op) {
	//INSERT YOUR CODE HERE
	// Your code must concatenate the parameters in the params variable.
	// Read calculatorModule.c to see how you must send the parameters.
}

int LoadModule(const char* params) {
	//INSERT YOUR CODE HERE
	// Your code must load the module "calculatorModule.ko" and send the parame-
ters params.
	// Read calculatorModule.c to see how you must send the parameters.
	// This function must return EXIT_FAILURE if failure, or 0 if success.

	return 0;
}

long GetResult() {
	long result = 0;

	//INSERT YOUR CODE HERE
	// Your code must read the result from the /sys/module/calculatorModule/
folder.
	// Read calculatorModule.c to know which variable to read.
	// This function must return the result as long.
	// REMEMBER TO CLOSE THE FILE AFTER USING IT.

	return result;
}

int UnLoadModule() {
	//INSERT YOUR CODE HERE
	// Your code must unload the module "calculatorModule.ko".
	// This function must return EXIT_FAILURE if failure, or 0 if success.

	return 0;
}
```

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 DRIVER_AUTHOR "Ricardo Pontaza - OS TA 2021" // Replace with your name and
student ID
#define DRIVER_DESC   "Dynamic calculator - OS Project 03"

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: 'add' - 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",kernelModu-
leName,__func__);

    // INSERT YOUR CODE HERE
    // Perfom addition, substraction or multiplication of firstParam and secondParam
depending on the value of operationParam.
    // Operation:
    //  operationParam equals "add": addition
    //  operationParam equals "sub": substraction
    //  operationParam equals "mul": multiplication
    // If operationParam has an invalid value, return 0.

    printk(KERN_INFO "[%s - %s] Operation = %s\n", kernelModuleName, __func__, oper-
ationParam);
    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__,result-
Param);

    return 0;
}
```

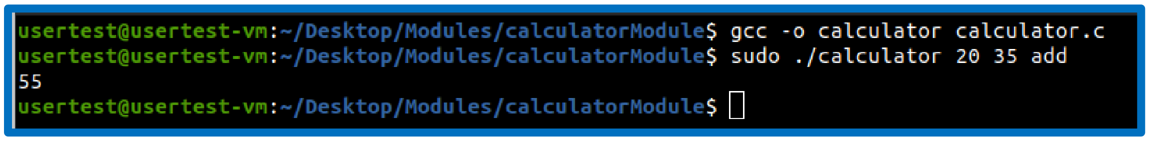
```
static void clean_exit(void){
    printk(KERN_INFO "\n[%s - %s] =============\n",kernelModuleName,__func__);
    printk(KERN_INFO "[%s - %s] Goodbye from calculatorModule!\n",kernelModule-
Name,__func__);

    printk(KERN_INFO "[%s - %s] Operation = %s\n", kernelModuleName,__func__,opera-
tionParam);
    printk(KERN_INFO "[%s - %s] First parameter = %d\n", kernelModule-
Name,__func__,firstParam);
    printk(KERN_INFO "[%s - %s] Second parameter = %d\n", kernelModule-
Name,__func__,secondParam);
    printk(KERN_INFO "[%s - %s] Result = %ld\n", kernelModuleName,__func__,resultPa-
ram);
}

module_init(initialize);
module_exit(clean_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
```

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
```

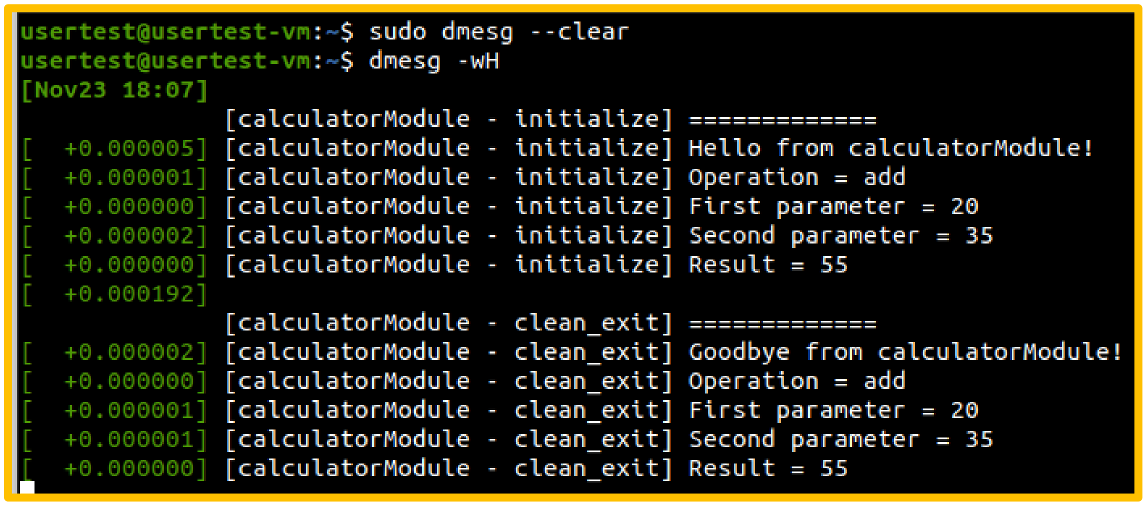
4. Your objective is simple. Finish both the **calculator.c** and **calculatorModule.c** files, so the following happens:
    a. The **calculatorModule.ko** is able to calculate additions, substractions and multiplications of **firstParam** and **secondParam**, depending on the value of **operationParam.**
    b. The valid values of **operationParam** are:
        i. add = addition
        ii. sub = subtraction
        iii. mul = multiplication
        iv. (other) = return -9999999
    c. The generated application ./**calculator** must take parameters as follows:
        i. Ask for operation (add – sub – mul)
        ii. Ask for two integer operands separated by space
        iii. Return **in terminal** the result.
        iv. Output the selected operation, operands and result in **dmesg** (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. Unload the module, and
v. Display the result in terminal and in **dmesg**



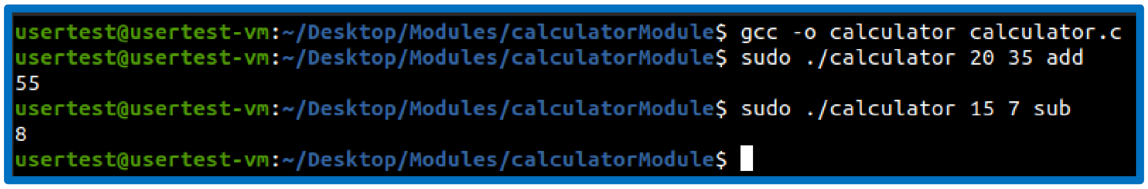
```
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ gcc -o calculator calculator.c
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 20 35 add
55
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$
```

T1



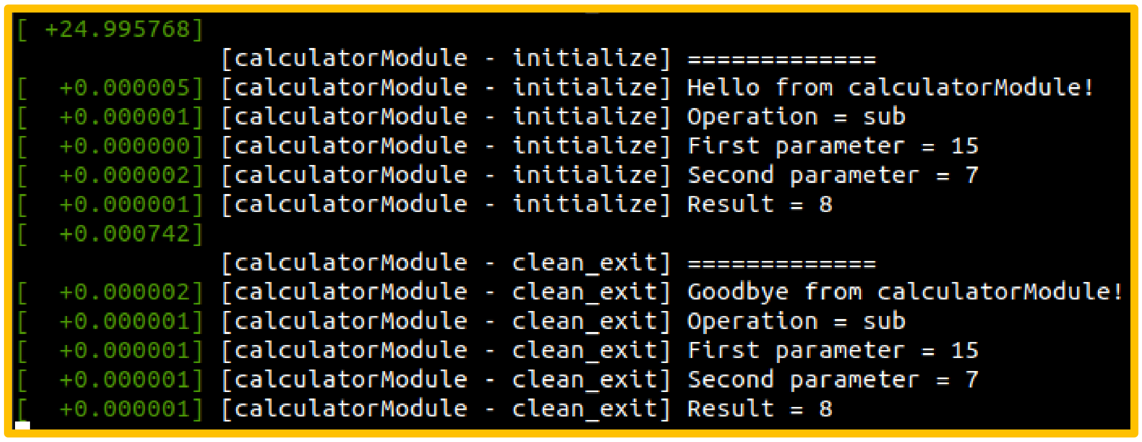
```
usertest@usertest-vm:~$ sudo dmesg --clear
usertest@usertest-vm:~$ dmesg -wH
[Nov23 18:07]
              [calculatorModule - initialize] =============
[  +0.000005] [calculatorModule - initialize] Hello from calculatorModule!
[  +0.000001] [calculatorModule - initialize] Operation = add
[  +0.000000] [calculatorModule - initialize] First parameter = 20
[  +0.000002] [calculatorModule - initialize] Second parameter = 35
[  +0.000000] [calculatorModule - initialize] Result = 55
[  +0.000192]
              [calculatorModule - clean_exit] =============
[  +0.000002] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[  +0.000000] [calculatorModule - clean_exit] Operation = add
[  +0.000001] [calculatorModule - clean_exit] First parameter = 20
[  +0.000001] [calculatorModule - clean_exit] Second parameter = 35
[  +0.000000] [calculatorModule - clean_exit] Result = 55
```

T2



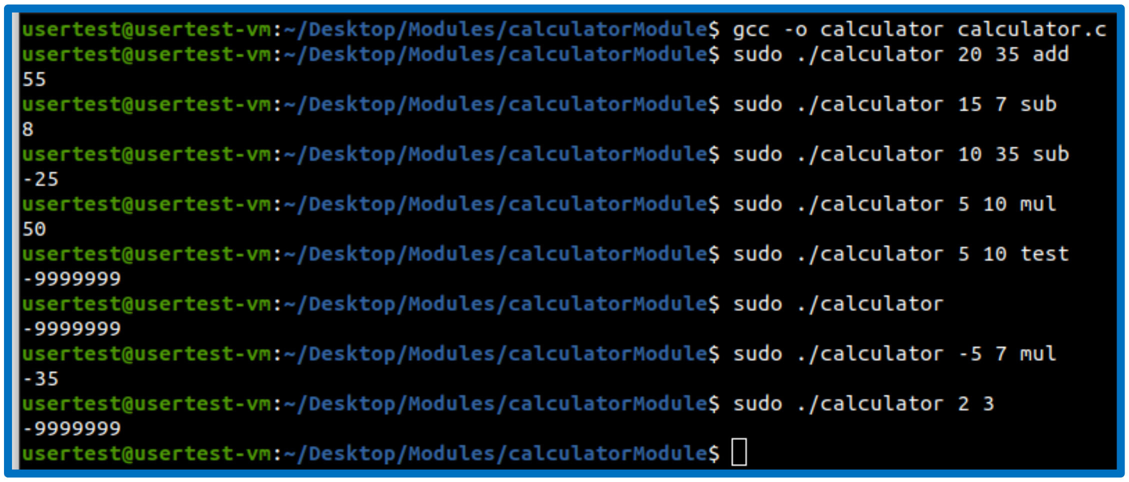
```
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ gcc -o calculator calculator.c
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 20 35 add
55
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 15 7 sub
8
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$
```

T1



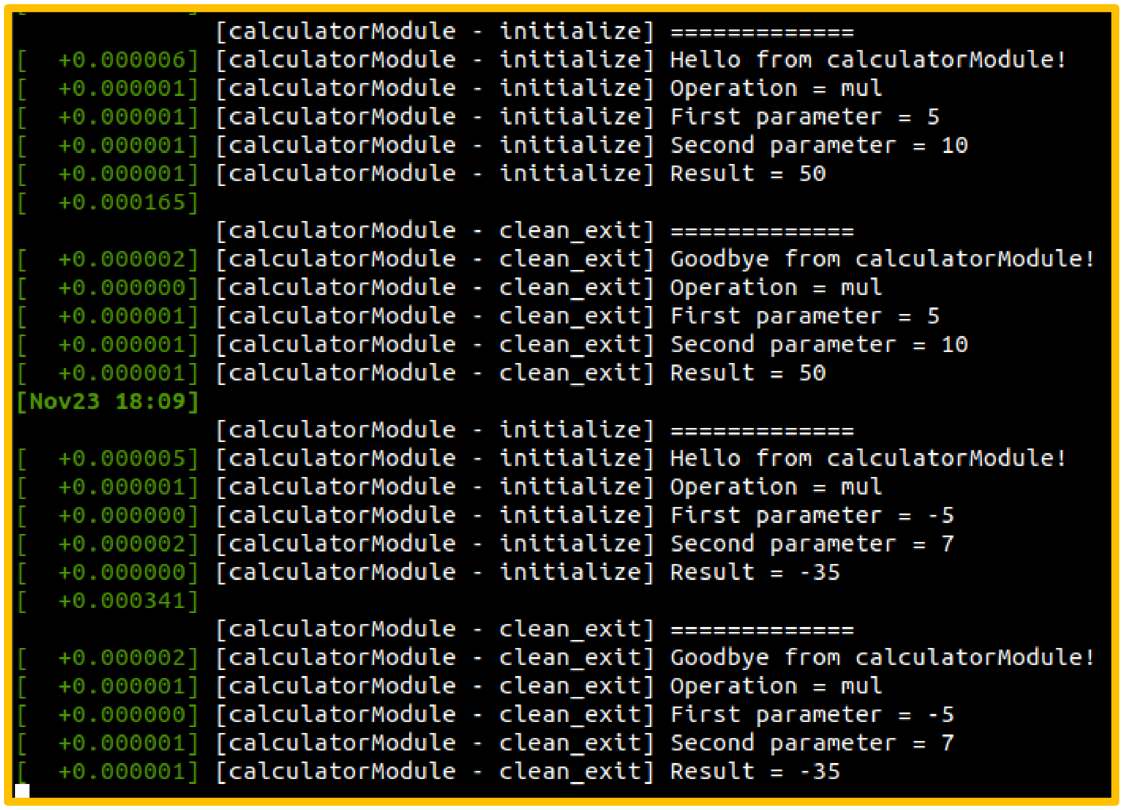
```
[ +24.995768]
              [calculatorModule - initialize] =============
[  +0.000005] [calculatorModule - initialize] Hello from calculatorModule!
[  +0.000001] [calculatorModule - initialize] Operation = sub
[  +0.000000] [calculatorModule - initialize] First parameter = 15
[  +0.000002] [calculatorModule - initialize] Second parameter = 7
[  +0.000001] [calculatorModule - initialize] Result = 8
[  +0.000742]
              [calculatorModule - clean_exit] =============
[  +0.000002] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[  +0.000001] [calculatorModule - clean_exit] Operation = sub
[  +0.000001] [calculatorModule - clean_exit] First parameter = 15
[  +0.000001] [calculatorModule - clean_exit] Second parameter = 7
[  +0.000001] [calculatorModule - clean_exit] Result = 8
```

T2



```
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ gcc -o calculator calculator.c
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 20 35 add
55
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 15 7 sub
8
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 10 35 sub
-25
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 5 10 mul
50
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 5 10 test
-9999999
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator
-9999999
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator -5 7 mul
-35
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$ sudo ./calculator 2 3
-9999999
usertest@usertest-vm:~/Desktop/Modules/calculatorModule$
```

T1



```
              [calculatorModule - initialize] =============
[  +0.000006] [calculatorModule - initialize] Hello from calculatorModule!
[  +0.000001] [calculatorModule - initialize] Operation = mul
[  +0.000001] [calculatorModule - initialize] First parameter = 5
[  +0.000001] [calculatorModule - initialize] Second parameter = 10
[  +0.000001] [calculatorModule - initialize] Result = 50
[  +0.000165]
              [calculatorModule - clean_exit] =============
[  +0.000002] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[  +0.000000] [calculatorModule - clean_exit] Operation = mul
[  +0.000001] [calculatorModule - clean_exit] First parameter = 5
[  +0.000001] [calculatorModule - clean_exit] Second parameter = 10
[  +0.000001] [calculatorModule - clean_exit] Result = 50
[Nov23 18:09]
              [calculatorModule - initialize] =============
[  +0.000005] [calculatorModule - initialize] Hello from calculatorModule!
[  +0.000001] [calculatorModule - initialize] Operation = mul
[  +0.000000] [calculatorModule - initialize] First parameter = -5
[  +0.000002] [calculatorModule - initialize] Second parameter = 7
[  +0.000000] [calculatorModule - initialize] Result = -35
[  +0.000341]
              [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 = -5
[  +0.000001] [calculatorModule - clean_exit] Second parameter = 7
[  +0.000001] [calculatorModule - clean_exit] Result = -35
```

T2

[Screenshot #19-20: Create two 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).