

Operating Systems Homework 3 Report

Andres Ponce, 0616110

2020-12-07

Video Link

Click here

If it doesn't work here is the link: https://youtu.be/r1w03z_QpR4

Questions

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)?**

A kernel module extends the functionality of the kernel without being part of it. These modules are loaded by the main kernel, so they are not part of the main kernel. We can load and unload these modules by demand by using the `insmod` and `rmmod` commands. Dynamic kernel modules, also known as **loadable kernel modules** can be loaded by the user to add custom functionality to the kernel. Static kernel functions, on the other hand, are loaded at boot time after the initial ram file system loads the kernel, and are required by the system to boot.

There are some differences between kernel modules and system calls, which we implemented in our previous project. Firstly, system calls require us to recompile the entire kernel, since they can access some kernel resources. Kernel modules are also located outside the kernels, so modifying, inserting, or removing kernel modules does not require a recompile.

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

System calls are part of the kernel. This means that we cannot dynamically load new system kernels during execution of our operating system. System calls would be like adding a new function to a C source code file. In order for the function to be available we have to save the source file and recompile the program.

Kernel modules, on the other hand, are different because they are loaded in from the outside. They don't require the kernel to be recompiled because their source code is not directly inside the kernel. Continuing our C source code example, this would be like having a C program that reads variables from an external file and then adding a variable to that external file. Since we are not changing the source code of the program and instead adding modifications from outside, our program does not require a recompile. 3. **What are the commands `insmod`, `rmmod`, and `modinfo` for? How do you use them?**

These commands all deal with kernel modules. `insmod` inserts a module into the Linux kernel. The `rmmod` command removes a module from the Linux kernel, and the `modinfo` command shows the info associated with a particular kernel module.

To use these kernel modules, we need to know the name of the module, as well as what program options we would like to provide.

4. Write the usage of the following commands:

- `module_init`

The `module_init()` system call will load the module into the kernel space. Once it allocates space and can initialize the module parameters, it will call the module's `initialize()` function, which we pass as an argument. In our module, we print the values of our kernel parameters out to the kernel ring buffer.

- `module_exit`

In contrast to the `module_init()` system call, the `module_exit()` function takes care of wrapping up our module when the module is removed or otherwise terminated. The parameter is a function pointer to the cleanup routine we want to execute. In our case we again reprint the values to the kernel ring buffer, some of which might be changed.

- `MODULE_LICENSE`

The `MODULE_LICENSE()` specifies the license under which our module exists. The "GPL" string represents the "GNU Public License", specifically v2 of that license. This license enables the software to be distributed freely according to the philosophy of the Free Software Foundation. In our project, we use it by placing it at the top of the file.

- `module_param`

The `module_param()` command adds a parameter to our module, and we specify three parameters: the value to be modified, the type, and a mask for the permission. The last value is explained more in detail later.

- `MODULE_PARM_DESC`

This command specifies how to use the module parameter, what it does and how to use it. It is printed out when we inspect the module using `modinfo`, since that will print the descriptions for all the values.

5. What do the following terminal commands mean? (Explain what they do and what does the `-x` mean in each case):

- `cat`

The `cat` command prints out the contents of a file to standard output. It can also concatenate files.

- `ls -l`

The `ls` command is one of the most fundamental commands in Linux, and it lists the contents in a directory. When we give it the `-l` option, it will use a “long listing” format, where it indicates the permissions on the directory contents, the user which created them, the group the user belongs to, the size, and the time each was created.

- `dmesg -wH`

The `dmesg` command prints the contents of the kernel ring buffer. This means that any message we write in the `printk()` function will appear in this buffer. The `-wH` options signify two things: that the buffer should wait for new messages (`-w`), and that the contents should be printed out in human readable format (`-H`).

- `lsmod`

The `lsmod` command will list all the currently loaded kernel modules and their status. It formats and prints out the contents of `/proc/modules`.

- `lsmod | grep`

The first command in this sequence is the same as the previous command, however, we “pipe”, or send the results of `lsmod` to `grep`. `grep` looks for patterns in standard input, so we can use it to search whether our own kernel modules are running.

6. There is a -644 in the line

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

inside `paramsModule.c`. What does 0644 mean?

The 0644 in the function call represents a mask for the parameter’s visibility in `sysfs`. `sysfs` is a pseudo file-system which provides an interface to the kernel data structures. 0644 specifically means that our parameter will be “root-writeable”. We could also specify 0444 if we wanted our parameter to be writable by any user.

More generally, a mask is a value that is used in bitwise operations. So when the value 0644 is used in a bitwise operation with some other value, it will result in the permissions we can use.

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

This error means that the module was not able to be initialized. Normally, the function will return 0 in case that the module terminates normally. Maybe there was an error in the format of the function or the module was already loaded and thus the initialization function cannot run.

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?

In the `paramsModule.c` file, the `initialize()` function runs as soon as we load the module. Inside this function, we print out the values of the module, and we only print out some of the kernel module values. The reason we don't see all the variables in the debug buffer is because those values are not print out when our module initializes.

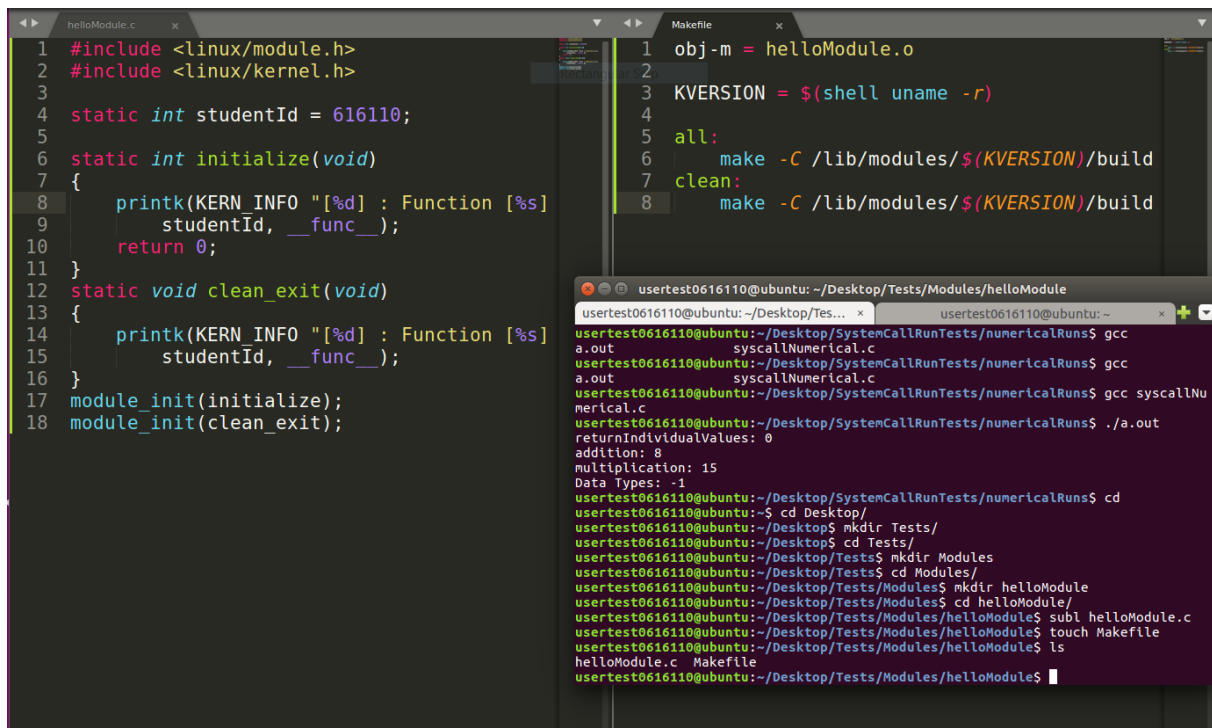
9. What is the `/sys/module` folder for?

The `/sys/module` folder stores the information regarding all the currently loaded kernel modules. Inside the folder of each kernel module, we can find the current values used by the module. We can read and change the values, if we have the correct permissions. For the `paramsModule02` and `calculatorModule`, we are mostly interested in the parameter values which the module stores in the `/sys/module/$(module name)/$(parameter name)`.

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

`charp` in the module parameter declaration refers to a `char *`, or a pointer to a `char`. The macros used by the Linux kernel to check the validity of the parameters will eventually expand the value to a `char` pointer, which in C functions as a string.

Screenshot discussion



```
helloModule.c
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3
4 static int studentId = 616110;
5
6 static int initialize(void)
7 {
8     printk(KERN_INFO "[%d] : Function [%s]\n", studentId, __func__);
9     return 0;
10 }
11
12 static void clean_exit(void)
13 {
14     printk(KERN_INFO "[%d] : Function [%s]\n", studentId, __func__);
15 }
16
17 module_init(initialize);
18 module_exit(clean_exit);

Makefile
1 obj-m = helloModule.o
2
3 KVERSION = $(shell uname -r)
4
5 all:
6     make -C /lib/modules/$(KVERSION)/build
7 clean:
8     make -C /lib/modules/$(KVERSION)/build

Terminal Output:
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ gcc syscallNumerical.c
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ gcc syscallNumerical.c
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ gcc syscallNumerical.c
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ ./a.out
returnIndividualValues: 0
addition: 8
multiplication: 15
Data Types: -1
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ cd ~/Desktop/
usertest0616110@ubuntu: ~/Desktop$ mkdir Tests/
usertest0616110@ubuntu: ~/Desktop$ cd Tests/
usertest0616110@ubuntu: ~/Desktop/Tests$ mkdir Modules
usertest0616110@ubuntu: ~/Desktop/Tests$ cd Modules/
usertest0616110@ubuntu: ~/Desktop/Tests/Modules$ mkdir helloModule
usertest0616110@ubuntu: ~/Desktop/Tests/Modules$ cd helloModule/
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ touch Makefile
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$ ls
helloModule.c Makefile
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule$
```

Figure 1: Files associated with helloModule.

The first module we created in the project involves just printing a value to the kernel ring buffer when we load and remove the kernel module. The `initialize` function will execute whenever the kernel module is loaded and the `clean_exit` function will be executed when the module is removed.

The `Makefile` on the top right corner specifies the rules for how to build and compile the files. The first line specifies that the target file is in module form. The `all` and `clean` are different rules for where and how to compile certain files.

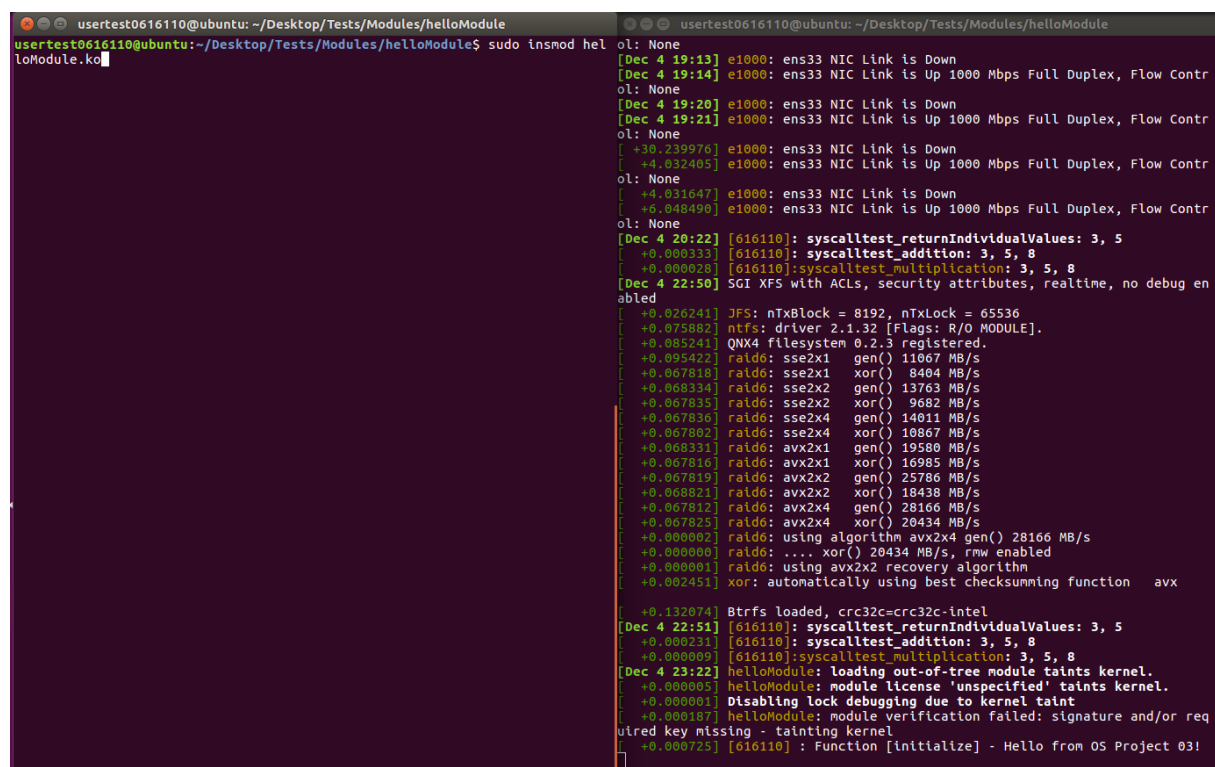
```

usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ make clean
make -C /lib/modules/4.19.148/build M=/home/usertest0616110/Desktop/Tests/Modules/helloModule clean
make[1]: Entering directory '/usr/src/linux-4.19.148'
  CLEAN    /home/usertest0616110/Desktop/Tests/Modules/helloModule/.tmp_versions
make[1]: Leaving directory '/usr/src/linux-4.19.148'
usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ make
make -C /lib/modules/4.19.148/build M=/home/usertest0616110/Desktop/Tests/Modules/helloModule modules
make[1]: Entering directory '/usr/src/linux-4.19.148'
  CC [M]    /home/usertest0616110/Desktop/Tests/Modules/helloModule/helloModule.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/usertest0616110/Desktop/Tests/Modules/helloModule/helloModule.o
see include/linux/module.h for more information
  CC        /home/usertest0616110/Desktop/Tests/Modules/helloModule/helloModule.mod.o
  LD [M]    /home/usertest0616110/Desktop/Tests/Modules/helloModule/helloModule.ko
make[1]: Leaving directory '/usr/src/linux-4.19.148'
usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$

```

Figure 2: The output of the `make` command.

As specified previously, the `make` command specifies how the project or certain files are to be compiled. When the command is run in the command line, this is the output. In the output we see that the make program uses the `/usr/src/linux-4.19.148` directory to make the kernel module file, `helloModule.ko`. Afterwards this resulting file will be the one that we load into the kernel.



```

usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule
usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo insmod helloModule.ko
[Dec 4 19:13] e1000: ens33 NIC Link is Down
[Dec 4 19:14] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Contr
ol: None
[Dec 4 19:20] e1000: ens33 NIC Link is Down
[Dec 4 19:21] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Contr
ol: None
[ +30.239976] e1000: ens33 NIC Link is Down
[ +4.032405] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Contr
ol: None
[ +4.031647] e1000: ens33 NIC Link is Down
[ +5.040490] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Contr
ol: None
[Dec 4 20:22] [616110]: syscalltest_returnIndividualValues: 3, 5
[ +0.000333] [616110]: syscalltest_addition: 3, 5, 8
[ +0.000028] [616110]: syscalltest_multiplication: 3, 5, 8
[Dec 4 22:50] SGI XFS with ACLs, security attributes, realtime, no debug en
abled
[ +0.026241] JFS: nTxBlock = 8192, nTxLock = 65536
[ +0.075882] ntfs: driver 2.1.32 [Flags: R/O MODULE].
[ +0.085241] QNX4 filesystem 0.2.3 registered.
[ +0.095422] raid0: sse2x1 gen() 11067 MB/s
[ +0.067818] raid0: sse2x1 xor() 8404 MB/s
[ +0.068334] raid0: sse2x2 gen() 13763 MB/s
[ +0.067835] raid0: sse2x2 xor() 9682 MB/s
[ +0.067836] raid0: sse2x4 gen() 14011 MB/s
[ +0.067802] raid0: sse2x4 xor() 10867 MB/s
[ +0.068331] raid0: avx2x1 gen() 19580 MB/s
[ +0.067816] raid0: avx2x1 xor() 16985 MB/s
[ +0.067819] raid0: avx2x2 gen() 25786 MB/s
[ +0.068821] raid0: avx2x2 xor() 18438 MB/s
[ +0.067812] raid0: avx2x4 gen() 28166 MB/s
[ +0.067825] raid0: avx2x4 xor() 20434 MB/s
[ +0.000002] raid0: using algorithm avx2x4 gen() 28166 MB/s
[ +0.000000] raid0: .... xor() 20434 MB/s, rwm enabled
[ +0.000001] raid0: using avx2x2 recovery algorithm
[ +0.002451] xor: automatically using best checksumming function avx
[ +0.132074] Btrfs loaded, crc32c=crc32c-intel
[Dec 4 22:51] [616110]: syscalltest_returnIndividualValues: 3, 5
[ +0.000231] [616110]: syscalltest_addition: 3, 5, 8
[ +0.000000] [616110]: syscalltest_multiplication: 3, 5, 8
[Dec 4 23:22] helloModule: loading out-of-tree module taints kernel.
[ +0.000005] helloModule: module license 'unspecified' taints kernel.
[ +0.000001] Disabling lock debugging due to kernel taint
[ +0.000187] helloModule: module verification failed: signature and/or req
uired key missing - tainting kernel
[ +0.000725] [616110]: Function [initialize] - Hello from OS Project 03!

```

Figure 3: Message printed on the kernel ring buffer when we load the module.

In the `helloModule.c` file, we have a procedure specified when we initialize the module and when we remove the module. These functions get bound in the `module_init()` and `module_exit()`

function calls, to which we pass the function. The output in the screenshot is the output after the `initialize` function has been called.

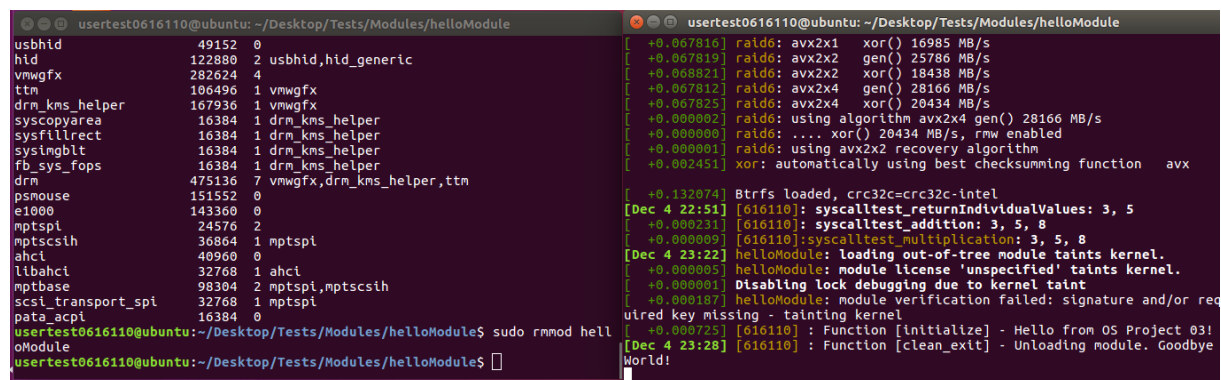
```

usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod | grep he
lloModule
helloModule          16384  0
usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod
Module                Size  Used by
helloModule           16384  0
btrfs                 1163264 0

```

Figure 4: Checking that our module is loaded with the `lsmod` command.

While our module is loaded from the previous step, we use the `lsmod` to list all the currently loaded kernel modules. We pipe this output to the `grep` command which will look for the name of our module. We do this to check if the module has been loaded.



```

usertest0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule
usbhid                49152  0
hid                   122880 2 usbhid,hid_generic
vmwgfx                282624 4
ttm                   186496 1 vmwgfx
drm_kms_helper        167936 1 vmwgfx
syscopyarea           16384 1 drm_kms_helper
sysfillrect           16384 1 drm_kms_helper
sysimgblt             16384 1 drm_kms_helper
fb_sys_fops           16384 1 drm_kms_helper
drm                   475136 7 vmwgfx,drm_kms_helper,ttm
psmouse               151552 0
e1000                 143360 0
mptspi                24576 2
mptscsih              36864 1 mptspi
ahci                   40960 0
libahci               32768 1 ahci
mptbase               98304 2 mptspi,mptscsih
scsi_transport_spi    32768 1 mptspi
pata_acpi             16384 0
usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo rmmod hell
oModule
usertest0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$

```

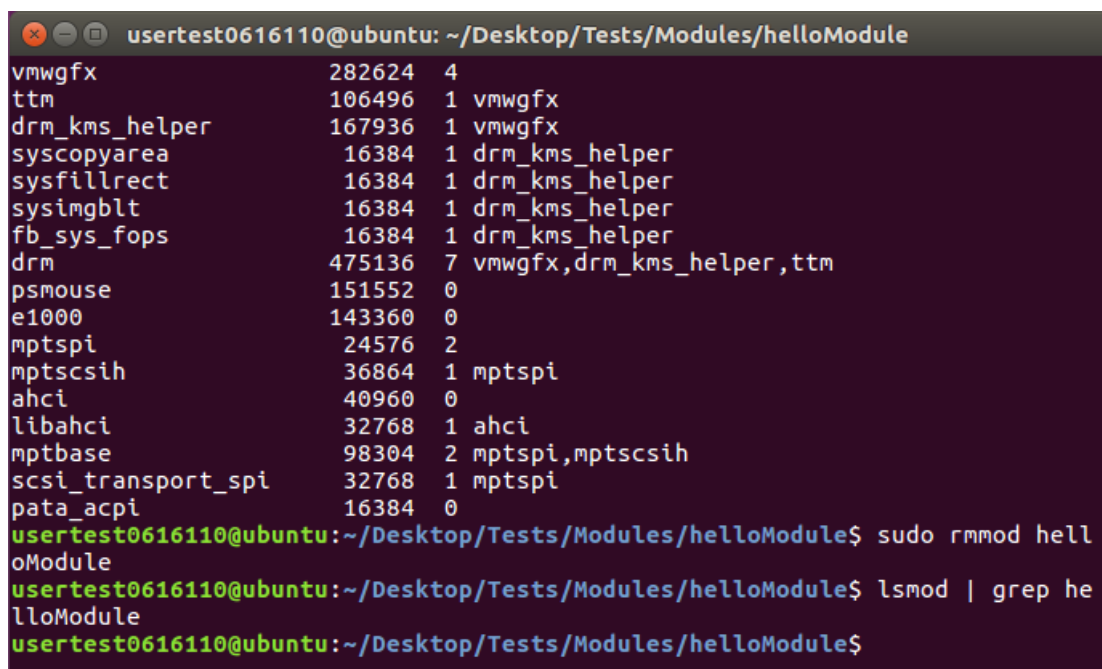
```

+0.067816] raid6: avx2x1  xor() 16985 MB/s
+0.067819] raid6: avx2x2  gen() 25786 MB/s
+0.068821] raid6: avx2x2  xor() 18438 MB/s
+0.067812] raid6: avx2x4  gen() 28166 MB/s
+0.067825] raid6: avx2x4  xor() 20434 MB/s
+0.000002] raid6: using algorithm avx2x4 gen() 28166 MB/s
+0.000000] raid6: .... xor() 20434 MB/s, rmw enabled
+0.000001] raid6: using avx2x2 recovery algorithm
+0.002451] xor: automatically using best checksumming function   avx
+0.132074] Btrfs loaded, crc32c=crc32c-intel
[Dec 4 22:51] [016110]: syscalltest_returnIndividualValues: 3, 5
+0.000231] [016110]: syscalltest_addition: 3, 5, 8
+0.000009] [016110]:syscalltest_multiplication: 3, 5, 8
[Dec 4 23:22] helloModule: loading out-of-tree module taints kernel.
+0.000005] helloModule: module license 'unspecified' taints kernel.
+0.000001] Disabling lock debugging due to kernel taint
+0.000187] helloModule: module verification failed: signature and/or required key missing - tainting kernel
+0.000725] [016110] : Function [initialize] - Hello from OS Project 03!
[Dec 4 23:28] [016110] : Function [clean_exit] - Unloading module. Goodbye
World!

```

Figure 5: Unloading `helloModule`.

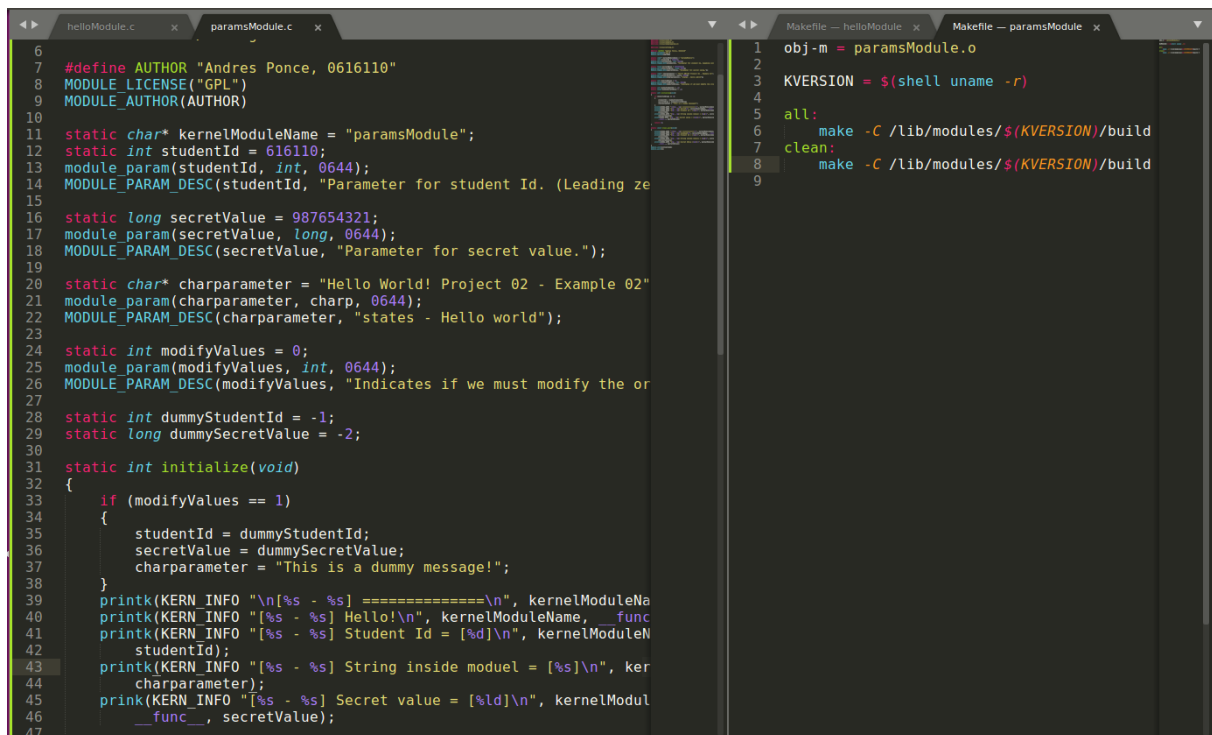
When we unload `helloModule`, we will see the state of the module parameters. In the future exercises we will change the values of the parameters in between the time of loading and unloading. Here, the code inside the `clean_exit()` function is executed.



```
user0616110@ubuntu: ~/Desktop/Tests/Modules/helloModule
vmwgfx                282624  4
ttm                   106496  1 vmwgfx
drm_kms_helper        167936  1 vmwgfx
syscopyarea           16384  1 drm_kms_helper
sysfillrect           16384  1 drm_kms_helper
sysimgblt             16384  1 drm_kms_helper
fb_sys_fops           16384  1 drm_kms_helper
drm                   475136  7 vmwgfx,drm_kms_helper,ttm
psmouse              151552  0
e1000                 143360  0
mptspi                24576  2
mptscsih              36864  1 mptspi
ahci                  40960  0
libahci               32768  1 ahci
mptbase               98304  2 mptspi,mptscsih
scsi_transport_spi    32768  1 mptspi
pata_acpi             16384  0
user0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ sudo rmmod hell
oModule
user0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$ lsmod | grep he
lloModule
user0616110@ubuntu:~/Desktop/Tests/Modules/helloModule$
```

Figure 6: Verifying that the module is unloaded.

In a previous screenshot, we searched for the loaded kernel module among the output of `lsmod`. Here we repeat the same command after using the `rmmod` command to remove the module. The `lsmod | grep helloModule` command returns nothing since `helloModule` has already been unloaded.



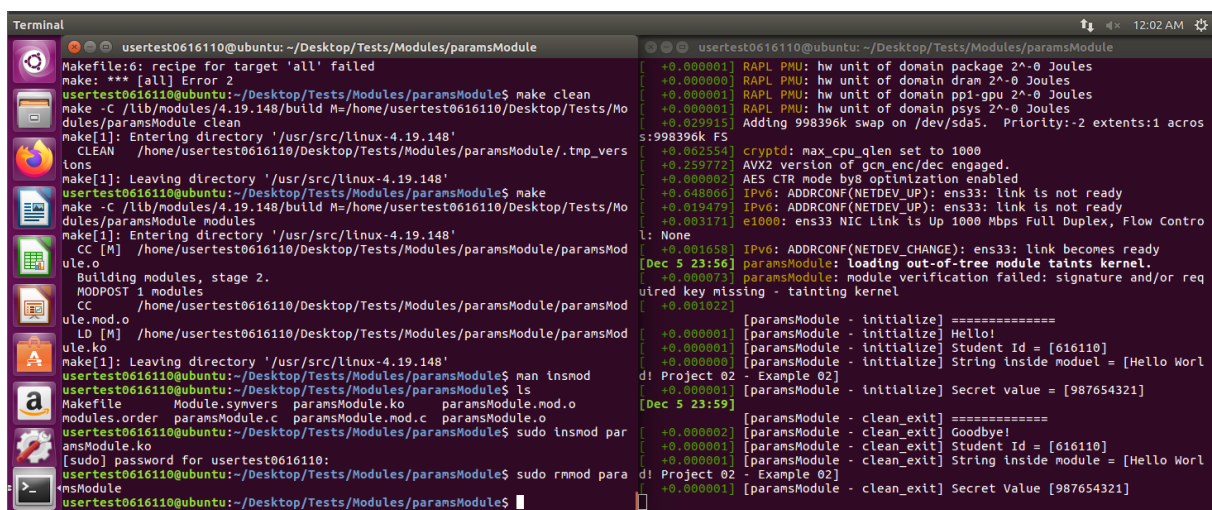
```

6
7 #define AUTHOR "Andres Ponce, 0616110"
8 MODULE_LICENSE("GPL")
9 MODULE_AUTHOR(AUTHOR)
10
11 static char* kernelModuleName = "paramsModule";
12 static int studentId = 616110;
13 module_param(studentId, int, 0644);
14 MODULE_PARAM_DESC(studentId, "Parameter for student Id. (Leading ze
15
16 static long secretValue = 987654321;
17 module_param(secretValue, long, 0644);
18 MODULE_PARAM_DESC(secretValue, "Parameter for secret value.");
19
20 static char* charparameter = "Hello World! Project 02 - Example 02"
21 module_param(charparameter, charp, 0644);
22 MODULE_PARAM_DESC(charparameter, "states - Hello world");
23
24 static int modifyValues = 0;
25 module_param(modifyValues, int, 0644);
26 MODULE_PARAM_DESC(modifyValues, "Indicates if we must modify the or
27
28 static int dummyStudentId = -1;
29 static long dummySecretValue = -2;
30
31 static int initialize(void)
32 {
33     if (modifyValues == 1)
34     {
35         studentId = dummyStudentId;
36         secretValue = dummySecretValue;
37         charparameter = "This is a dummy message!";
38     }
39     printk(KERN_INFO "\n[%s - %s] =====\n", kernelModuleNa
40     printk(KERN_INFO "[%s - %s] Hello!\n", kernelModuleName, __func
41     printk(KERN_INFO "[%s - %s] Student Id = [%d]\n", kernelModuleN
42     studentId);
43     printk(KERN_INFO "[%s - %s] String inside module = [%s]\n", ker
44     charparameter);
45     printk(KERN_INFO "[%s - %s] Secret value = [%ld]\n", kernelModul
46     __func__, secretValue);
47
1
2 obj-m = paramsModule.o
3
4 KVERSION = $(shell uname -r)
5
6 all:
7     make -C /lib/modules/$(KVERSION)/build
8
9 clean:
10    make -C /lib/modules/$(KVERSION)/build

```

Figure 7: The paramsModule and its Makefile.

The next kernel module we designed involved passing parameters to the module. In `paramsModule02.c`, the lines with the `module_param()` function will add a parameter of a certain type to the module. If we have the `modifyValues` parameter toggled, then we will change some of the other parameters. We can toggle this value from the command line or dynamically from another file.



```

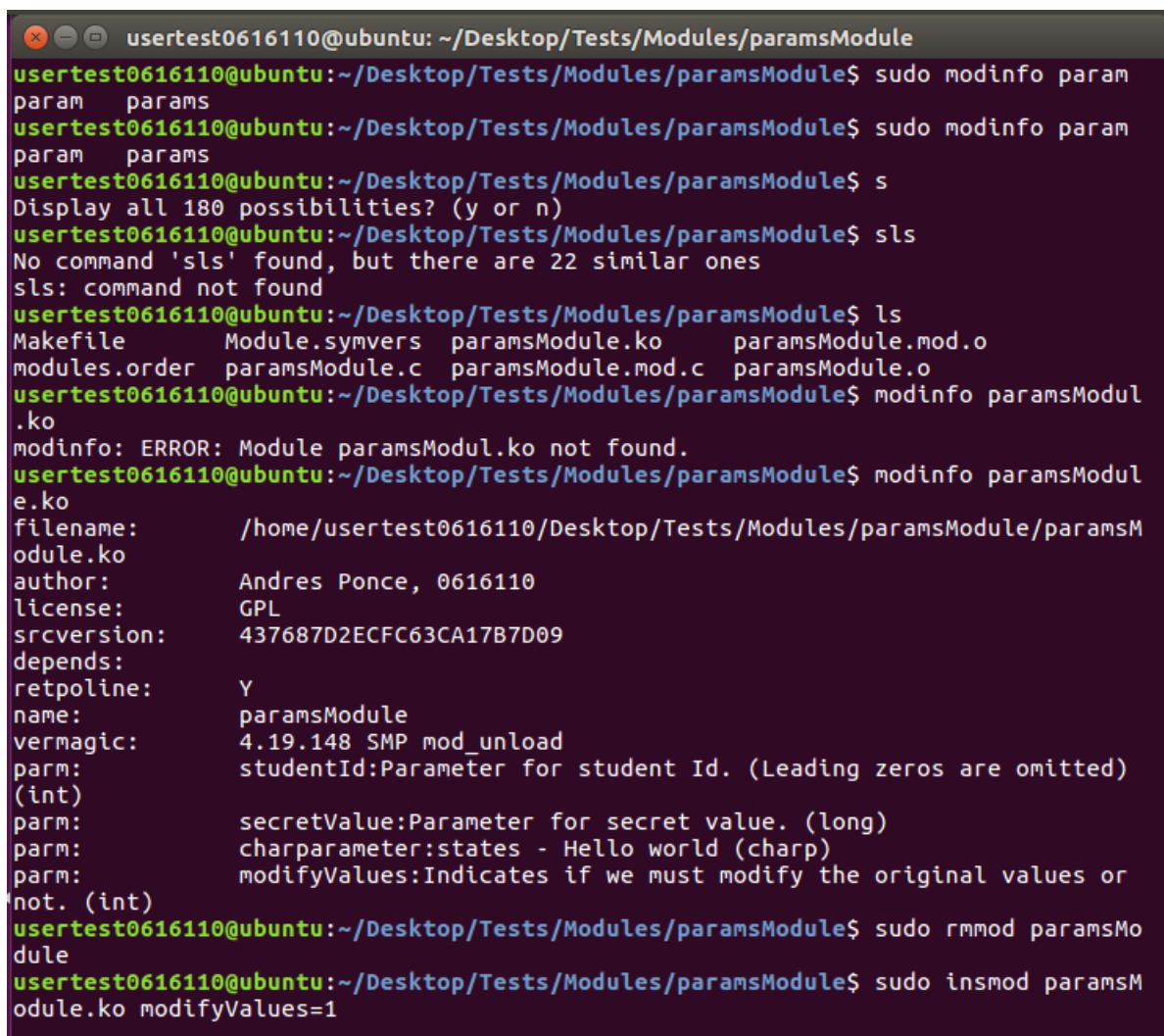
Terminal
userstest0616110@ubuntu: ~/Desktop/Tests/Modules/paramsModule
Makefile:6: recipe for target 'all' failed
make: *** [all] Error 2
userstest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ make clean
make -C /lib/modules/4.19.148/build M=/home/userstest0616110/Desktop/Tests/Modules/paramsModule clean
make[1]: Entering directory '/usr/src/linux-4.19.148'
CLEAN /home/userstest0616110/Desktop/Tests/Modules/paramsModule/.tmp_versions
make[1]: Leaving directory '/usr/src/linux-4.19.148'
userstest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ make
make -C /lib/modules/4.19.148/build M=/home/userstest0616110/Desktop/Tests/Modules/paramsModule modules
make[1]: Entering directory '/usr/src/linux-4.19.148'
CC [M] /home/userstest0616110/Desktop/Tests/Modules/paramsModule/paramsModule.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/userstest0616110/Desktop/Tests/Modules/paramsModule/paramsModule.mod.o
LD [M] /home/userstest0616110/Desktop/Tests/Modules/paramsModule/paramsModule.ko
make[1]: Leaving directory '/usr/src/linux-4.19.148'
userstest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ man insmod
Module: Module.symvers paramsModule.ko paramsModule.mod.o
modules.order paramsModule.c paramsModule.mod.c paramsModule.o
userstest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.ko
[sudo] password for userstest0616110:
userstest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule
userstest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$

[+0.000001] RAPL PMU: hw unit of domain package 2^0-0 Joules
[+0.000000] RAPL PMU: hw unit of domain dram 2^0-0 Joules
[+0.000001] RAPL PMU: hw unit of domain ppi-gpu 2^0-0 Joules
[+0.000001] RAPL PMU: hw unit of domain psys 2^0-0 Joules
[+0.000015] Adding 998396k swap on /dev/sda5. Priority:-2 extents:1 across:1996792k FS
[+0.002554] cryptd: max_cpu_qlen set to 1000
[+0.259772] AVX2 version of gcm enc/dec engaged.
[+0.000002] AES CTR mode by8 optimization enabled
[+0.648066] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[+0.019479] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[+0.003171] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[+0.001650] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[Dec 5 23:56] paramsModule: loading out-of-tree module taints kernel.
[+0.000073] paramsModule: module verification failed: signature and/or required key missing - tainting kernel
[+0.001022] [paramsModule - initialize] =====
[+0.000001] [paramsModule - initialize] Hello!
[+0.000001] [paramsModule - initialize] Student Id = [616110]
[+0.000001] [paramsModule - initialize] String inside module = [Hello World! Project 02 - Example 02]
[+0.000001] [paramsModule - initialize] Secret value = [987654321]
[Dec 5 23:59] [paramsModule - clean_exit] =====
[+0.000002] [paramsModule - clean_exit] Goodbye!
[+0.000001] [paramsModule - clean_exit] Student Id = [616110]
[+0.000001] [paramsModule - clean_exit] String inside module = [Hello World! Project 02 - Example 02]
[+0.000001] [paramsModule - clean_exit] Secret Value [987654321]

```

Figure 8: Module parameters printed when loaded and removed.

In terminal 1, we load and remove the module using the `insmod` and `rmmod` commands. This causes the module to execute the functions we specified in the `module_init()` and `module_exit()` functions. Thus the parameters are printed in the kernel ring buffer. In this step, we have not changed any of the parameters, so the values in the exit method will be the same as the ones in the input. Unless we specify the `modifyValues` parameter, we won't change their values..

A terminal window with a dark background and light-colored text. The window title is 'usertest0616110@ubuntu: ~/Desktop/Tests/Modules/paramsModule'. The terminal shows a series of commands and their outputs. The user runs 'sudo modinfo param', which outputs 'param params'. This is repeated. Then, the user runs 's', which prompts 'Display all 180 possibilities? (y or n)'. Next, the user runs 'sls', which outputs 'No command 'sls' found, but there are 22 similar ones' and 'sls: command not found'. Then, the user runs 'ls', which lists files: 'Makefile', 'Module.symvers', 'paramsModule.ko', 'paramsModule.mod.o', 'modules.order', 'paramsModule.c', 'paramsModule.mod.c', and 'paramsModule.o'. Then, the user runs 'modinfo paramsModul', which outputs an error: 'modinfo: ERROR: Module paramsModul.ko not found.'. Then, the user runs 'modinfo paramsModule.ko', which outputs detailed module information: filename, author (Andres Ponce, 0616110), license (GPL), srcversion (437687D2ECFC63CA17B7D09), depends, retpoline (Y), name (paramsModule), vermagic (4.19.148 SMP mod_unload), and several parameters: studentId, secretValue, charparameter, and modifyValues. Finally, the user runs 'sudo rmmod paramsModule' and 'sudo insmod paramsModule.ko modifyValues=1'.

```
usertest0616110@ubuntu: ~/Desktop/Tests/Modules/paramsModule
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo modinfo param
param    params
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo modinfo param
param    params
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ s
Display all 180 possibilities? (y or n)
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sls
No command 'sls' found, but there are 22 similar ones
sls: command not found
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ ls
Makefile      Module.symvers  paramsModule.ko  paramsModule.mod.o
modules.order  paramsModule.c  paramsModule.mod.c  paramsModule.o
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ modinfo paramsModul
.ko
modinfo: ERROR: Module paramsModul.ko not found.
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ modinfo paramsModule.ko
filename:       /home/usertest0616110/Desktop/Tests/Modules/paramsModule/paramsModule.ko
author:        Andres Ponce, 0616110
license:       GPL
srcversion:    437687D2ECFC63CA17B7D09
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)
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsModule
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsModule.ko modifyValues=1
```

Figure 9: Getting information about the module with the `modinfo` command.

```

[paramsModule - initialize] =====
[ +0.000001] [paramsModule - initialize] Hello!
[ +0.000002] [paramsModule - initialize] Student Id = [-1]
[ +0.000000] [paramsModule - initialize] String inside module = [This is a dumm
y message!]
[ +0.000001] [paramsModule - initialize] Secret value = [-2]
[Dec 6 00:10]
[paramsModule - clean_exit] =====
[ +0.000002] [paramsModule - clean_exit] Goodbye!
[ +0.000001] [paramsModule - clean_exit] Student Id = [-1]
[ +0.000000] [paramsModule - clean_exit] String inside module = [This is a dumm
y message!]
[ +0.000001] [paramsModule - clean_exit] Secret Value [-2]

```

Figure 10: Changing some of the parameters by toggling `modifyValues`

In the first screenshot, we first print the module information, such as the parameter names, their types, and a short description. Next, we load the module again, except this time when we load the module we use the command

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

which will cause the values of some of the parameters. Our code will check this value and change some of the parameters in accordance with the code.

The next screenshot shows the values after they are changed. The `modifyValues` parameter will replace the `studentId` and `message` with some dummy values, which then get printed to the kernel ring buffer.

```

usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsM
odule.ko studentId=616110 secretValue=8888

```

Figure 11: Loading the module again with our own custom parameters.

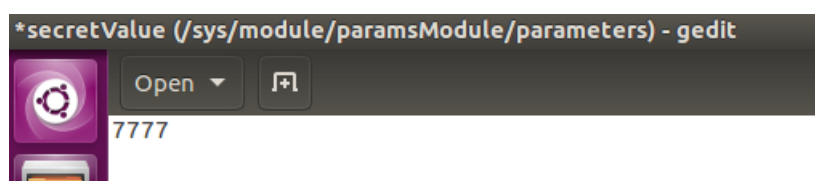


Figure 12: The parameter's value written in the correct file.

```

[paramsModule - initialize] =====
[ +0.000001] [paramsModule - initialize] Hello!
[ +0.000001] [paramsModule - initialize] Student Id = [616110]
[ +0.000001] [paramsModule - initialize] String inside module = [Hello World! P
roject 02 - Example 02]
[ +0.000001] [paramsModule - initialize] Secret value = [8888]
[Dec 6 00:44]
[paramsModule - clean_exit] =====
[ +0.000023] [paramsModule - clean_exit] Goodbye!
[ +0.000003] [paramsModule - clean_exit] Student Id = [616110]
[ +0.000002] [paramsModule - clean_exit] String inside module = [Hello World! P
roject 02 - Example 02]
[ +0.000002] [paramsModule - clean_exit] Secret Value [7777]

```

Figure 13: The value we updated in the parameter file shows once we remove the module.

In the previous section, we had changed some parameters of our module from within the module file. The first screenshot shows how we can change the parameters of the file directly from the command line. The module is then loaded with those parameters and we can thus see it on the kernel ring buffer.

The kernel keeps track of the module parameter's values inside of a file. Originally, the secret value was 8888 but we can change it by going into the file and just replacing the old value with 7777. When we remove the module, we should see the updated value being printed on the kernel ring buffer.

The last screenshot is again the values of the parameters once we remove the module and its exit function is executed. This shows that once we edit the values in the files, the module will use the new values.

```

** (gedit:10334): WARNING **: Set document metadata failed: Setting attribute me
tadata: gedit-position not supported
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsM
odule
sudo: rmmod: command not found
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo rmmod paramsMo
dule
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$ sudo insmod paramsM
odule.ko dummyStudentId=9999
usertest0616110@ubuntu:~/Desktop/Tests/Modules/paramsModule$
[Dec 6 00:48] [paramsModule: unknown parameter 'dummyStudentId' ignored]
[ +0.000385] [paramsModule - initialize] =====
[ +0.000004] [paramsModule - initialize] Hello!
[ +0.000003] [paramsModule - initialize] Student Id = [616110]
[ +0.000003] [paramsModule - initialize] String inside module = [Hello World! P
roject 02 - Example 02]
[ +0.000003] [paramsModule - initialize] Secret value = [987654321]

```

Figure 14: Loading an invalid parameter.

When we try to change a variable that cannot be changed, we will get a message in the kernel ring buffer that displays a message saying we cannot change that value. The reason that this parameter is not recognized is that we did not add it to the list of parameters. So, even though `dummyStudentId` is a variable inside our file, it is seen as a local variable rather than a valid module parameter.

```
//Loading the module
fd = open(moduleName, O_RDONLY);
printf("Loading module [%s] with parameters [%s]...\n", moduleNameNoExt, 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");
printf("\n[Press ENTER to continue]\n");
getchar();

// Unloading module
printf("Unloading module....\n");
if(delete_module(moduleNameNoExtension, 0_NONBLOCK) != 0)
{
    perror("delete_module");
    return EXIT_FAILURE;
}

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

free(image);
printf("Done!\n");
```

Figure 15: Code responsible for loading and unloading the module.

```
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, "\n[%s - %s] Hello!\n",kernelModuleName, __func__);
    printk(KERN_INFO, "\n[%s - %s] StudentId = [%d]\n",kernelModuleName, __t
        studentId);
    printk(KERN_INFO, "\n[%s - %s] String inside module = [%s]\n",kernelModu
        charparameter);
    printk(KERN_INFO, "\n[%s - %s] Secret Value = [%d]\n",kernelModuleName,

    return 0;
}

static void clean_exit(void)
{
    printk(KERN_INFO, "\n[%s - %s]=====\n",kernelModuleName, __func__);
    printk(KERN_INFO, "\n[%s - %s] Goodbye!\n",kernelModuleName, __func__);
    printk(KERN_INFO, "\n[%s - %s] StudentId = [%d]\n",kernelModuleName, __t
        studentId);
    printk(KERN_INFO, "\n[%s - %s] String inside module = [%s]\n",kernelModu
        charparameter);
    printk(KERN_INFO, "\n[%s - %s] Secret Value = [%d]\n",kernelModuleName,

}

module_init(initialize);
```

Figure 16: The functions that run on loading and removing the module, respectively.

The first creates the image for reading the module. We then pass it to the `init_module()` macro, which will attempt to load the module with the parameters passed in the `paramsNew` string. We also have to allocate some memory space for the memory image.

The next image shows the two methods we run on loading and removing the modules. The last two lines, `module_init(initialize)` and `module_exit(clean_exit)` are bound to the module, so these functions will execute on module loading and removing, respectively.


```

usertest0616110@ubuntu: ~/Des... x usertest0616110@ubuntu: ~/Des... x usertest0616110@ubuntu: /lib/m... x
usertest0616110@ubuntu:~/Desktop/Tests/Modules/loadUnLoadModule$ gcc -o loaderUnloader loaderUnlo
ader.c
usertest0616110@ubuntu:~/Desktop/Tests/Modules/loadUnLoadModule$ ls
a.out          Makefile      Module.symvers  paramsModule02.mod.c
loaderUnloader  Makefile.bak  paramsModule02.c  paramsModule02.mod.o
loaderUnloader.c modules.order  paramsModule02.ko  paramsModule02.o
usertest0616110@ubuntu:~/Desktop/Tests/Modules/loadUnLoadModule$

```

Figure 17: Compiling the loaderUnloader module.

```

[Dec13 07:59] [paramsModule02 - clean_exit] Secret Value = [0]
[paramsModule02 - initialize]=====
[ +0.000003] [paramsModule02 - initialize] Hello!
[ +0.000001] [paramsModule02 - initialize] StudentId = [616110]
[ +0.000001] [paramsModule02 - initialize] String inside module = [Hello World! Project 02 - Example 03]
[ +0.000001] [paramsModule02 - initialize] Secret Value = [987654321]

```

Figure 18: Dmesg output after we load the module

```

usertest0616110@ubuntu: ~/Des... x usertest0616110@ubuntu: ~/Des... x usertest0616110@ubuntu: /lib/m... x
usertest0616110@ubuntu:/lib/modules/4.19.148/build/include$ ls /sys/module/paramsModule02/paramet
ers/
charparameter modifyValues secretValue studentId
usertest0616110@ubuntu:/lib/modules/4.19.148/build/include$ lsmod | grep paramsModule02
paramsModule02      16384  0
usertest0616110@ubuntu:/lib/modules/4.19.148/build/include$

```

Figure 19: Checking that we loaded the module

```

[Dec13 08:05] [paramsModule02 - clean_exit] Secret Value = [987654321]
[paramsModule02 - clean_exit]=====
[ +0.000002] [paramsModule02 - clean_exit] Goodbye!
[ +0.000001] [paramsModule02 - clean_exit] StudentId = [616110]
[ +0.000001] [paramsModule02 - clean_exit] String inside module = [Hello World! Project 02 - Exa
mple 03]
[ +0.000001] [paramsModule02 - clean_exit] Secret Value = [6]

```

Figure 20: dmesg Values once we unload the module

Once we are done with writing the module code, we have to compile the program that loads and unloads the module. `loaderUnloader.c` contains the calls necessary to load the module and give it the parameters we need.

We then load the module with our student Id as parameter, which is displayed in the kernel ring buffer once we call `dmesg -wH`. There are also the secret value and the string inside the module which we print to the kernel.

So far we have compiled our module, loaded it to the kernel, and we saw the result of the `initialize()` function being printed to the ring buffer. To indeed check whether `paramsModule`, or any other module, is loaded, we can run the `lsmod` command in combination with `grep` to check whether the module is loaded.

Lastly, we unload the module when we allow `loaderUnloader` to finish executing. The second part of the program will unload the module and again print the values to the kernel ring buffer.

```
make -C /lib/modules/4.19.148/build M=/home/user0616110/Desktop/Tests/Modules/calculatorModule mo
modules
make[1]: Entering directory '/usr/src/linux-4.19.148'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-4.19.148'
user0616110@ubuntu:~/Desktop/Tests/Modules/calculatorModule$ sudo ./calculator
[sudo] password for user0616110:
=====
Enter operation [sum - sub - mul -exit]: sum
Enter two operands(space separated): 1 7
Loading module [calculatorModule.ko] with parameters [operationParam=sum firstParam=1 secondParam=7]
Module is unmounted!
Cleaning...
Done.
Operation [sum] - Operands [1 7] - Result: [8]
=====
Enter operation [sum - sub - mul -exit]: sub
Enter two operands(space separated): 10 2
Loading module [calculatorModule.ko] with parameters [operationParam=sub firstParam=10 secondParam=2]
Module is unmounted!
Cleaning...
Done.
Operation [sub] - Operands [10 2] - Result: [8]
=====
Enter operation [sum - sub - mul -exit]: mul
Enter two operands(space separated): 2 8
Loading module [calculatorModule.ko] with parameters [operationParam=mul firstParam=2 secondParam=8]
Module is unmounted!
Cleaning...
Done.
Operation [mul] - Operands [2 8] - Result: [16]
```

Figure 21: stdout when we load the calculator module.

```

[ +0.000003] [calculatorModule - initialize] Hello from calculatorModule!
[ +0.000002] [calculatorModule - initialize] Operation = sum
[ +0.000003] [calculatorModule - initialize] First parameter = 1
[ +0.000001] [calculatorModule - initialize] Second parameter = 7
[ +0.000002] [calculatorModule - initialize] Result = 8
[ +0.000584] [calculatorModule - clean_exit] =====
[ +0.000001] [calculatorModule - clean_exit] \
[ +0.000001] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[ +0.000003] [calculatorModule - clean_exit] Operation = sum
[ +0.000001] [calculatorModule - clean_exit] First parameter = 1
[ +0.000002] [calculatorModule - clean_exit] second parameter = 7
[ +0.000001] [calculatorModule - clean_exit] Result = 8
[Dec14 02:37] [calculatorModule - initialize]=====
[ +0.000004] [calculatorModule - initialize] Hello from calculatorModule!
[ +0.000002] [calculatorModule - initialize] Operation = sub
[ +0.000003] [calculatorModule - initialize] First parameter = 10
[ +0.000001] [calculatorModule - initialize] Second parameter = 2
[ +0.000002] [calculatorModule - initialize] Result = 8
[ +0.000464] [calculatorModule - clean_exit] =====
[ +0.000002] [calculatorModule - clean_exit] \
[ +0.000002] [calculatorModule - clean_exit] Goodbye from calculatorModule!
[ +0.000002] [calculatorModule - clean_exit] Operation = sub
[ +0.000002] [calculatorModule - clean_exit] First parameter = 10
[ +0.000001] [calculatorModule - clean_exit] second parameter = 2
[ +0.000001] [calculatorModule - clean_exit] Result = 8
[ +9.215573] [calculatorModule - initialize]=====
[ +0.000004] [calculatorModule - initialize] Hello from calculatorModule!
[ +0.000003] [calculatorModule - initialize] Operation = mul

```

Figure 22: Kernel ring buffer when we pass our own parameters to our module.

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

    //INSERT YOUR CODE HERE TO LOAD/UNLOAD AND USE MODULE
    char args[100];
    prepare_args(args, "sum", input1, input2);
    result = apply_module(args);

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

    //INSERT YOUR CODE HERE TO LOAD/UNLOAD AND USE MODULE
    char args[100];
    prepare_args(args, "sub", input1, input2);
    result = apply_module(args);

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

    //INSERT YOUR CODE HERE TO LOAD/UNLOAD AND USE MODULE
    char args[100];
    prepare_args(args, "mul", input1, input2);
    result = apply_module(args);

    return result;
}
```

Figure 23: How we handle the parameter calculation.

```
//INSERT YOUR CODE HERE
// Perform addition, subtraction or multiplication of firstParam and secondParam
// depending on the value of the operationParam
// If operationParam has an invalid value, return
if(strcmp(operationParam, "sum") == 0)
{
    resultParam = firstParam + secondParam;
}
else if(strcmp(operationParam, "sub") == 0)
{
    resultParam = firstParam - secondParam;
}
else if(strcmp(operationParam, "mul") == 0)
{
    resultParam = firstParam * secondParam;
}
else{
    resultParam = 0;
}
```

Figure 24: Actual code implementing the arithmetic operation.

For the last part of the assignment, we had to write our own module which took in some parameters and called the module to perform some arithmetic operation. The final output on the terminal is shown in the first image, where we try out all the three operations along with two numbers. Before the module is loaded, the name of the module and the final string are shown on screen.

For the second screenshot, we show the dmesg output once we load the module. Here we essentially just print out the values that were passed on to the module at load time, and we see some of the initialization and unloading.

The third screenshot shows the structure of the `addition`, `subtraction`, and `multiplication` functions. The most important part is having a string which contains the name of the operation and the two arguments, each parameter separated by a space. Inside the `prepare_args()` function, we use the `snprintf()` C function to format the string accordingly. We then copy the string into the `args` buffer and return the string ready for use. The `apply_module()` function will mostly replicate the code for loading and unloading the module used for `paramsModule()`, with a couple important additions:

1. It will pass along our string as a parameter when calling `init_module` (also used in `paramsModule`).
2. It will return the arithmetic result of the operation after reading it from the parameter file.¹

The last screenshot shows the actual code responsible for the arithmetic operations in `calculatorModule.c`. At the moment we load the modules, the parameters will already be set to the values we need them at. Therefore, in the `initialize()` method all we need to do is just perform the correct

¹This parameter is found in `/sys/module/calculatorModule/parameters/resultParam`.

arithmetic operation based on the value of the `operationParam` string, which we determine with `strcmp`.