Manpreet Singh (A0191418A) Assignment 3.pdf

Assignment

Kernel : 4.20.16 STABLE

OS : Ubuntu 18.04.2 desktop amd64

**Task 1: Build and run modules**

*linux/init.h* defines the macros used for free up kernal memory.*\_\_init, \_\_exit* and *\_\_initdata. \_\_init* and *\_\_initdata* are used to clear up functions and variables respectively. These are only need for built-in modules and it does nothing for loadable modules. *\_\_exit* macro help in omission of the function when the module is built-in because a built-in function does not need a exit function but a loadable function needs it.

*linux/module.h* help us in dynamic loading of modules into the kernel. And provides access to these methods:

*module\_init*  
 *module\_exit*

*MODULE\_LICENSE("GPL")* is used to inform the user that the code is not open source.

*module\_init* and *module\_exit* is use to point to the function pointer that will be executed when the module is loaded and when the module is unloaded receptively.

|  |
| --- |
| Makefile |
| obj-m := hello.o  KDIR := /lib/modules/$(shell uname -r)/build  all:      $(MAKE) -C $(KDIR) M=$(shell pwd) modules  clean:      $(MAKE) -C $(KDIR) M=$(shell pwd) clean      $(RM) Module.markers modules.order |

**Running of a hello.c**

Command used to build the module.

*sudo make*

Command used to insert the module into the running kernel

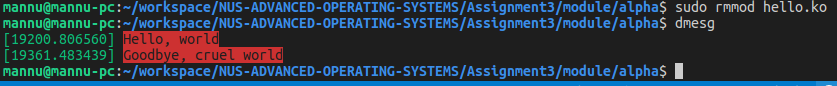
*sudo insmod hello.ko*

When the module is inserted the *dmseg* shows the output ofthe *printk().*



Command used to remove the module from the running kernel.

*sudo rmmod hello.ko*

When the module is removed the *dmseg* shows the output ofthe *printk().*

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

**module\_init** is the driver initialization entry point. Hence, it will either be called during *do\_initcalls* (if builtin) or at module insertion time (if a module). There can only be one *module\_init* per module.

**module\_exit** is the driver exit entry point. Hence, it will call the function to be run when driver is removed. Module\_exit also includes the *cleanup\_module*, which was the older approach of clean-up. Will call the function when used with *rmmod* and the driver is a module. However, if the driver is statically compiled into the kernel, *module\_exit* has no effect. There can only be one *module\_exit* per module

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

Command to build the module:

Inorder to build the module we need a code file *hello.c* and Makefile

|  |
| --- |
| Hello.c |
| #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/module.h>  MODULE\_LICENSE("GPL");  static int hello\_init(void)  {  printk(KERN\_ALERT "Hello, world\n");  return 0;  }  static void hello\_exit(void)  {  printk(KERN\_ALERT "Goodbye, cruel world\n");  }  module\_init(hello\_init);  module\_exit(hello\_exit); |

|  |
| --- |
| Makefile |
| obj-m := hello.o  KDIR := /lib/modules/$(shell uname -r)/build  all:      $(MAKE) -C $(KDIR) M=$(shell pwd) modules  clean:      $(MAKE) -C $(KDIR) M=$(shell pwd) clean      $(RM) Module.markers modules.order |

Then call the following command in the directory where the Makefile and the Code file is.

*sudo make*

Command to install the module:

*sudo* ***insmod*** *<module\_name>.ko*

Command to remove the module:

*sudo* ***rmmod*** *<module\_name>.ko*

1 c) Give the screenshot of the previous three commands and their results if any in the shell. If the output of *printk* doesn’t show in the shell, take a screenshot with ‘*dmesg | tail*’ or any other command to show the *printk* of hello world module.

**Screenshot of building the module:**

|  |
| --- |
| Makefile |
|  |

|  |
| --- |
| Inputs files |
|  |

|  |
| --- |
| *sudo make* |
|  |

|  |
| --- |
| Outputs files |
|  |

**Screenshot of installing the module:**

|  |
| --- |
| Installing module |
|  |

dmseg was cleared before inserting the mod.

|  |
| --- |
| Checking dmesg |
|  |

**Screenshot of removal the module:**

|  |
| --- |
| Removing module |
|  |

|  |
| --- |
| Checking dmesg |
|  |

1 d) Add a *<who>* parameter to your module so that your module will show *hello <who>* during init stage. Give the added lines which implements the function and give the screenshot of the new *printk*.

In order for your module to accept arguments, you need to first declare the variables as global that will take the values from the command line. With the use of the *module\_param* macro from *linux/moduleparam.h,* we are able the setup this.

|  |
| --- |
| hello.c [**bold** = new lines added or modified] |
| #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/module.h>  **#include <linux/moduleparam.h> // for module\_param()**  MODULE\_LICENSE("GPL");  static char \*who = "NOT DEFINED";  /\*  \* module\_param(foo, int, 0000)  \* The first param is the parameters name  \* The second param is it's data type  \* The third argument is the permissions bits,  \*/  **module\_param(who, charp, 0000);**  **MODULE\_PARM\_DESC(who, "Name of the user.");**  static int hello\_init(void)  {  **printk(KERN\_ALERT "Hello, %s\n", who);**  return 0;  }  static void hello\_exit(void)  {  printk(KERN\_ALERT "Goodbye, cruel world\n");  }  module\_init(hello\_init);  module\_exit(hello\_exit); |

**Screenshot of installing and removing the module:**

|  |
| --- |
| Insmod and rmmod |
|  |

**Task 2: Build a device module**

Understanding *ls -l /dev* command

|  |
| --- |
|  |

First column:



This is the permissions flags, the codes indicate the file’s type, as well as who may read, delete , modify or execute the file.

The first char ‘c’ is used to symbol char driver.

Second column:



This is this links column.This indicates the number of links the file has to other files on the system. Links provide a way to access content of one file by typing the name of another.

Third column:



This the Owner’s name.

Fourth column:



This is the group’s name the owner belongs to. If the owner belongs to no group then this will be same as the owner’s name.

Fifth column:



This indicates the file size in bytes for block devices and for char driver this will indicate the major number of 10 and minor number of 235.

Sixth column:



This indicates the the date of creation or last modified date.

Seventh column:



This indicates the file name.

2 a) Give the *mknod* command you use.

|  |
| --- |
|  |

*sudo mknod /dev/byteone c 61 1*

C is to specify the driver as char driver

61 is major number defined in the driver.

1 is minor number of the device

2 b)Give the screenshot of your device with “ls -l /dev” command and highlight your device.



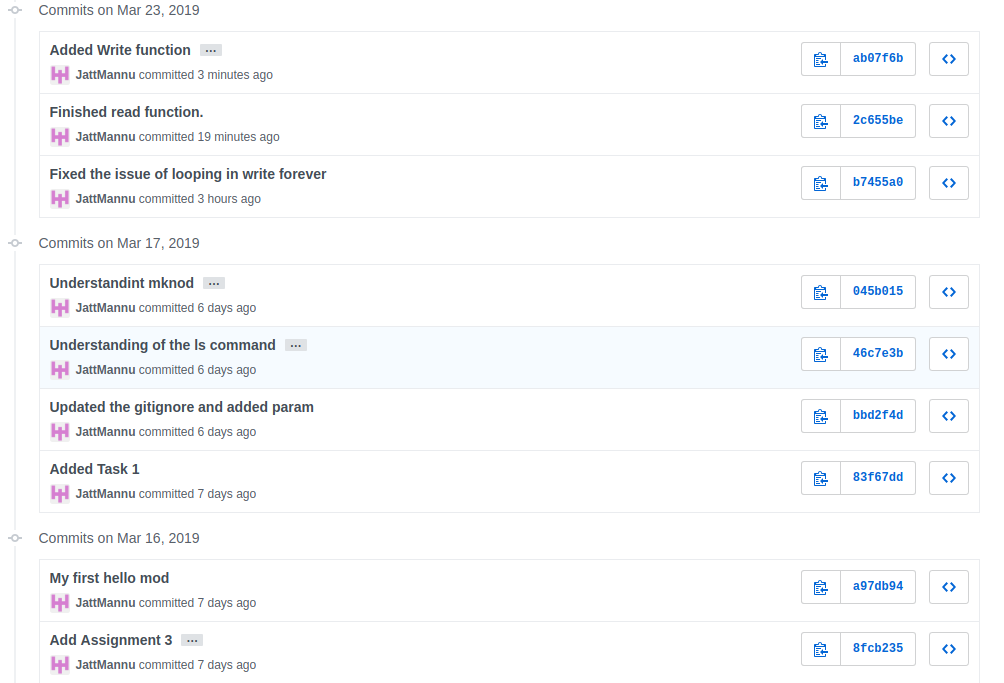
*ls -l /dev/byteone*

2 c) marks. Students are required to build or use their github accounts and sync their codes throughout the whole process of modifying codes and provide a screenshot of the commits. Give the codes of read and write functions that you

implemented and the screenshots of the four testing case

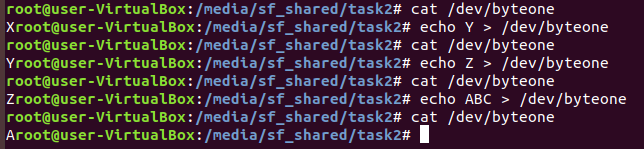
Added this *linux/uaccess.h* to use *copy\_from\_user* and *copy\_to\_user*

Github

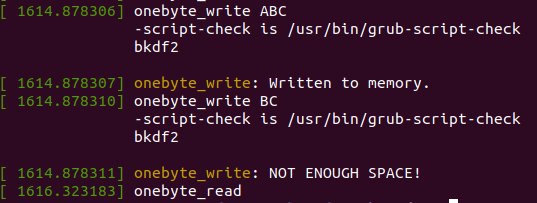


|  |
| --- |
|  |
|  |

Testing case:



dmesg for writing ABC



Part B

1. Let’s assume that we have the following processes entering the system (in the given

order):

1. P1 burst CPU time: 23, arrival time 0
2. P2 burst CPU time: 12, arrival time 5
3. P3 burst CPU time: 41, arrival time 10
4. P4 burst CPU time: 17, arrival time 15
5. P5 burst CPU time: 29, arrival time 40
6. What is the execution schedule and the completion time for each if we schedule the processes using pre-emptive shortest remaining time first with a context switching overhead of 1 time unit?

At time 0

Runnable Queue = {}

CPU = P1:23

At time 5

Runnable Queue = { P2:12}

CPU = P1:18

At time 6

Runnable Queue = { P1:18}

CPU = P2:12

At time 10

Runnable Queue = { P1:18 , P3:41}

CPU = P2:8

At time 15

Runnable Queue = { P1:18 , P3:41 , P4:17}

CPU = P2:3

At time 18

Runnable Queue = { P1:18 , P3:41 , P4:17}

CPU = P2:0

**P2 finished at 18th tick**

At time 19

Runnable Queue = { P1:18 , P3:41}

CPU = P4:17

At time 36

Runnable Queue = { P1:18 , P3:41}

CPU = P4:0

**P4 finished at 36th tick**

At time 37

Runnable Queue = { P3:41}

CPU = P1:18

At time 40

Runnable Queue = { P3:41 , P5:29}

CPU = P1:15

At time 55

Runnable Queue = { P3:41 , P5:29}

CPU = P1:0

**P1 finished at 55th tick**

At time 56

Runnable Queue = { P3:41}

CPU = P5:29

At time 85

Runnable Queue = { P3:41}

CPU = P5:0

**P5 finished at 85th tick**

At time 86

Runnable Queue = {}

CPU = P3:41

At time 127

Runnable Queue = {}

CPU = P3:0

**P3 finished at 127th tick**

1. What is the completion time for each task if we schedule the processes using round robin with a quantum of 10 with no overhead in context switching?

At time 0

Runnable Queue = {}

Next Epoch = {}

CPU = P1:23

At time 5

Runnable Queue = {P2:12}

Next Epoch = {}

CPU = P1:18

At time 10

Runnable Queue = {P3:41}

Next Epoch = {P1:13}

CPU = P2:12

At time 15

Runnable Queue = {P3:41, P4:17}

Next Epoch = {P1:13}

CPU = P2:7

At time 20

Runnable Queue = {P4:17}

Next Epoch = {P1:13, P2:2}

CPU = P3:41

At time 30

Runnable Queue = {}

Next Epoch = {P1:13, P2:2, P3:31}

CPU = P4:17

At time 40

Runnable Queue = {}

Next Epoch = {P1:13, P2:2, P3:31,P4:7}

CPU = P5:29

At time 50

Runnable Queue = {P2:2, P3:31, P4:7, P5:19}

Next Epoch = {}

CPU = P1:13

At time 60

Runnable Queue = {P3:31, P4:7, P5:19}

Next Epoch = {P1:3}

CPU = P2:2

At time 62

Runnable Queue = {P3:31, P4:7, P5:19}

Next Epoch = {P1:3}

CPU = P2:0

**P2 finished at 62th tick**

At time 62

Runnable Queue = {P4:7, P5:19}

Next Epoch = {P1:3}

CPU = P3:31

At time 72

Runnable Queue = {P5:19}

Next Epoch = {P1:3, P3:21}

CPU = P4:7

At time 79

Runnable Queue = {P5:19}

Next Epoch = {P1:3, P3:21}

CPU = P4:0

**P4 finished at 79th tick**

At time 79

Runnable Queue = {}

Next Epoch = {P1:3, P3:21}

CPU = P5:19

At time 89

Runnable Queue = {P3:21,P5:9}

Next Epoch = {}

CPU = P1:3

At time 92

Runnable Queue = {P3:21,P5:9}

Next Epoch = {}

CPU = P1:0

**P1 finished at 92th tick**

At time 92

Runnable Queue = {P5:9}

Next Epoch = {}

CPU = P3:21

At time 102

Runnable Queue = {}

Next Epoch = {P3:11}

CPU = P5:9

At time 111

Runnable Queue = {}

Next Epoch = {P3:11}

CPU = P5:0

**P5 finished at 111th tick**

At time 111

Runnable Queue = {}

Next Epoch = {}

CPU = P3:11

At time 121

Runnable Queue = {}

Next Epoch = {}

CPU = P3:1

At time 122

Runnable Queue = {}

Next Epoch = {}

CPU = P3:0

**P3 finished at 122th tick**

2. Draw the final red-black tree after the following insertions are completed. Use a single circle to represent a “red” node and a double concentric circle to represent a “black” node. If you use other notations, make sure you give a legend.

1. Insert 112
2. Insert 12
3. Insert 304
4. Insert 4
5. Insert 55
6. Insert 176
7. Insert 64
8. Insert 224
9. Insert 24

Red Black Tree

112

224

12

55

4

304

176

64

24

APPENDIX

sudo apt-get install linux-source

sudo apt-get install virtualbox

<https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html>

<http://tldp.org/LDP/lkmpg/2.6/html/x279.html>

<https://www.fsl.cs.sunysb.edu/kernel-api/re02.html>

<https://www.fsl.cs.sunysb.edu/kernel-api/re01.html>

<http://tldp.org/LDP/lkmpg/2.6/html/x323.html>

[https://books.google.com.sg/books?id=I2JLu34OFOUC&pg=PA404&lpg=PA404&dq=ls+-l+columns+explained&source=bl&ots=nGsbJVvsFa&sig=ACfU3U0Yt9\_fCCOAObPxeBK2nguOJNMuVw&hl=en&sa=X&ved=2ahUKEwje2OzZ0ojhAhUQgUsFHRk2AZ0Q6AEwB3oECAMQAQ#v=onepage&q=ls%20-l%20columns%20explained&f=false](https://books.google.com.sg/books?id=I2JLu34OFOUC&pg=PA404&lpg=PA404&dq=ls+-l+columns+explained&source=bl&ots=nGsbJVvsFa&sig=ACfU3U0Yt9_fCCOAObPxeBK2nguOJNMuVw&hl=en&sa=X&ved=2ahUKEwje2OzZ0ojhAhUQgUsFHRk2AZ0Q6AEwB3oECAMQAQ" \l "v=onepage&q=ls -l columns explained&f=false)