



Assignment3

CS5250 – Advanced Operating Systems AY2021/2022 Semester 2

Assignment 3

Deadline: Monday, 21 Mar 2022• 11.59pm

Part A:

1. 12 marks, about 1 hour Answer the following questions:

- 2 marks. When will `module_init` and `module_exit` be loading/called?
- 3 marks. What are the commands in installing the module and removing the module from a **running kernel**? Hint : there is no need to reboot.
- 3 marks. 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.
- 4 marks. Add a `<myID>` parameter to your module so that your module will show hello `<myID>` during init stage. Give the added lines which implements the function and give the screenshot of the new `printk`.

2. 8 marks, about 2 hours to 4 hours

- 1 mark. Give the `mknod` command you use. Hint: Careful what major number you use.
- 1 marks. Give the screenshot of your device with `"ls -l /dev"` command and highlight your device.
- 6 marks. Give the codes of read and write functions that you implemented and the screenshots of the four testing cases. Hint : Do not run other commands on your device before `cat /dev/<name>`

3. 5 marks, about an hour

I know I said that tasklets are on the way out but compared to the other means, this is the easiest “bottom half” to use – and it doesn't have to rely on some hardware IRQ. Implement tasklets that are launched at each of the driver calls above that just use `printk` to log a message giving evidence that the tasklet was executed. Use `dmesg` to show the messages. Hint: Google is your friend The actual work is less than 10 lines of code in the right places. The trick is to explore the Internet and find out what to do.

Part B:

1. Let's assume that we have the following processes entering the system (in the given order):

- (a) What is the execution schedule and the completion time for each if we schedule the processes using preemptive shortest remaining time first with a context switching overhead of 1 time unit?

- (b) 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?
2. On Slide 45 of the “All about processes” PPT deck, I stated that if all jobs are of the same length, then:
- (a) Shortest Job First (SJF) is the optimal for minimizing average response time in the nonpreemptive case, and
- (b) Shortest Remaining Time First (SRTF) is the same as First Come First Serve (FCFS)
- Show why these two statements are true.
3. 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.

Part A:

1. 12 marks, about 1 hour Answer the following questions:

a. 2 marks. When will `module_init` and `module_exit` be loading/called?

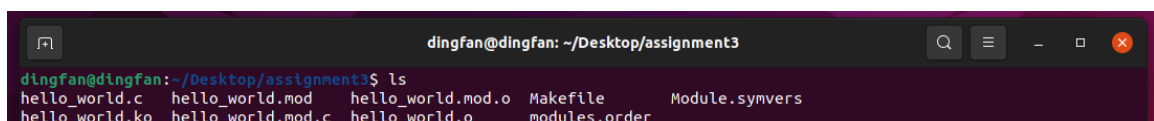
- `module_init` is called when the module is loaded or installed into the Linux kernel.
 - when we use `insmod` command insert the module into the Linux kernel
- `module_exit` is called when the module is removed from the Linux kernel
 - when we use `rmmod` command remove the module from the Linux kernel

b. 3 marks. What are the commands in installing the module and removing the module from a **running kernel**? Hint : there is no need to reboot.

1. `sudo make`
2. `sudo insmod hello_world.ko`
3. `sudo rmmod hello_world.ko`

c. 3 marks. 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.

1. `sudo make`



```
dingfan@dingfan: ~/Desktop/assignment3
dingfan@dingfan:~/Desktop/assignment3$ ls
hello_world.c  hello_world.mod  hello_world.mod.o  Makefile  Module.symvers
hello_world.ko  hello_world.mod.c  hello_world.o  modules.order
```

2. `sudo insmod hello_world.ko`
`sudo dmesg | tail`

```
dingfan@dingfan: ~/Desktop/assignment3
dingfan@dingfan:~/Desktop/assignment3$ sudo insmod hello_world.ko
dingfan@dingfan:~/Desktop/assignment3$ sudo dmesg | tail
[ 151.440669] Bluetooth: RFCOMM TTY layer initialized
[ 151.440677] Bluetooth: RFCOMM socket layer initialized
[ 151.440683] Bluetooth: RFCOMM ver 1.11
[ 159.708515] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 159.709087] ISO 9660 Extensions: RRIP_1991A
[ 160.031026] rfkill: input handler disabled
[ 171.327336] input: VMware DnD UInput pointer as /devices/virtual/input/input6
[ 851.095187] hello_world: loading out-of-tree module taints kernel.
[ 851.095220] hello_world: module verification failed: signature and/or required key missing - tainting kernel
[ 851.096327] Hello, world
```

3. `sudo rmmod hello_world.ko`

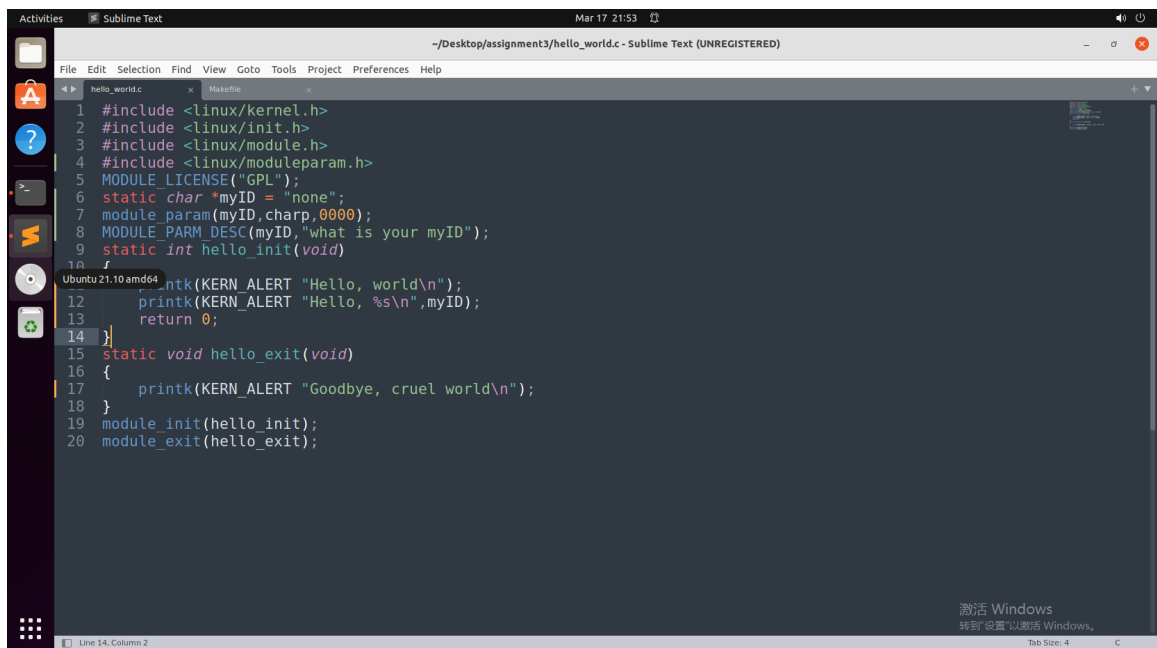
`sudo dmesg | tail`

```
dingfan@dingfan: ~/Desktop/assignment3
[ 851.095220] hello_world: module verification failed: signature and/or required key missing - tainting kernel
[ 851.096327] Hello, world
dingfan@dingfan:~/Desktop/assignment3$ sudo rmmod hello_world.ko
[sudo] password for dingfan:
dingfan@dingfan:~/Desktop/assignment3$ sudo dmesg | tail
[ 151.440677] Bluetooth: RFCOMM socket layer initialized
[ 151.440683] Bluetooth: RFCOMM ver 1.11
[ 159.708515] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 159.709087] ISO 9660 Extensions: RRIP_1991A
[ 160.031026] rfkill: input handler disabled
[ 171.327336] input: VMware DnD UInput pointer as /devices/virtual/input/input6
[ 851.095187] hello_world: loading out-of-tree module taints kernel.
[ 851.095220] hello_world: module verification failed: signature and/or required key missing - tainting kernel
[ 851.096327] Hello, world
[ 2990.368959] Goodbye, cruel world
dingfan@dingfan:~/Desktop/assignment3$
```

d. 4 marks. Add a `<myID>` parameter to your module so that your module will show hello `<myID>` during init stage. Give the added lines which implements the function and give the screenshot of the new `printk`.

1. The code of module, after changing

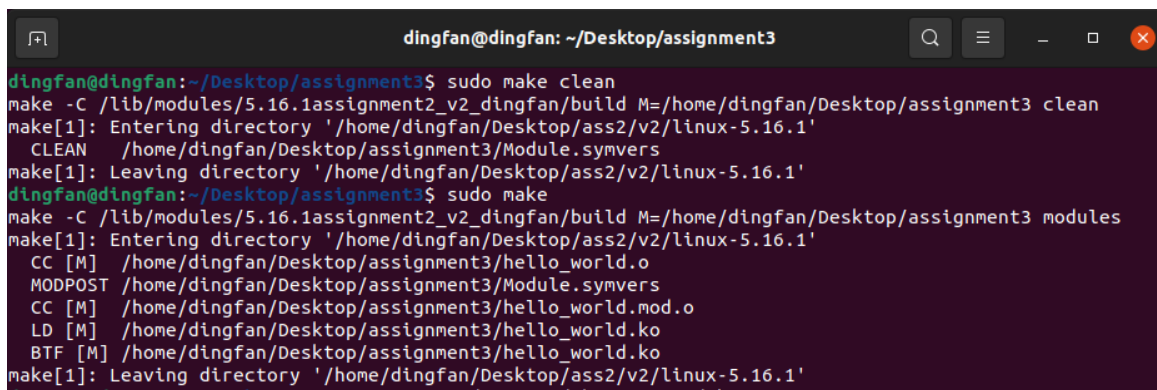
```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
MODULE_LICENSE("GPL");
static char *myID = "none";
module_param(myID, charp, 0000);
MODULE_PARM_DESC(myID, "what is your myID");
static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world\n");
    printk(KERN_ALERT "Hello, %s\n", myID);
    return 0;
}
static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```



```
1 #include <linux/kernel.h>
2 #include <linux/init.h>
3 #include <linux/module.h>
4 #include <linux/moduleparam.h>
5 MODULE_LICENSE("GPL");
6 static char *myID = "none";
7 module_param(myID, charp, 0000);
8 MODULE_PARM_DESC(myID, "what is your myID");
9 static int hello_init(void)
10 {
11     printk(KERN_ALERT "Hello, world\n");
12     printk(KERN_ALERT "Hello, %s\n", myID);
13     return 0;
14 }
15 static void hello_exit(void)
16 {
17     printk(KERN_ALERT "Goodbye, cruel world\n");
18 }
19 module_init(hello_init);
20 module_exit(hello_exit);
```

2. `sudo make clean`

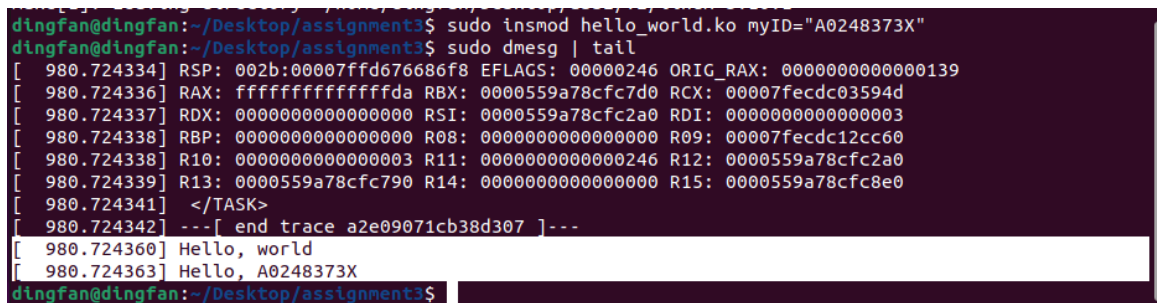
`sudo make`



```
dingfan@dingfan: ~/Desktop/assignment3
dingfan@dingfan:~/Desktop/assignment3$ sudo make clean
make -C /lib/modules/5.16.1/assignment2_v2_dingfan/build M=/home/dingfan/Desktop/assignment3 clean
make[1]: Entering directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
CLEAN /home/dingfan/Desktop/assignment3/Module.symvers
make[1]: Leaving directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
dingfan@dingfan:~/Desktop/assignment3$ sudo make
make -C /lib/modules/5.16.1/assignment2_v2_dingfan/build M=/home/dingfan/Desktop/assignment3 modules
make[1]: Entering directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
CC [M] /home/dingfan/Desktop/assignment3/hello_world.o
MODPOST /home/dingfan/Desktop/assignment3/Module.symvers
CC [M] /home/dingfan/Desktop/assignment3/hello_world.mod.o
LD [M] /home/dingfan/Desktop/assignment3/hello_world.ko
BTF [M] /home/dingfan/Desktop/assignment3/hello_world.ko
make[1]: Leaving directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
```

3. `sudo insmod hello_world.ko myID="A0248373X"`

`sudo dmesg | tail`



```
dingfan@dingfan:~/Desktop/assignment3$ sudo insmod hello_world.ko myID="A0248373X"
dingfan@dingfan:~/Desktop/assignment3$ sudo dmesg | tail
[ 980.724334] RSP: 002b:00007ffd676686f8 EFLAGS: 00000246 ORIG_RAX: 0000000000000139
[ 980.724336] RAX: ffffffffda RBX: 0000559a78cfc7d0 RCX: 00007fecdc03594d
[ 980.724337] RDX: 0000000000000000 RSI: 0000559a78cfc2a0 RDI: 0000000000000003
[ 980.724338] RBP: 0000000000000000 R08: 0000000000000000 R09: 00007fecdc12cc60
[ 980.724338] R10: 0000000000000003 R11: 0000000000000246 R12: 0000559a78cfc2a0
[ 980.724339] R13: 0000559a78cfc790 R14: 0000000000000000 R15: 0000559a78cfc8e0
[ 980.724341] </TASK>
[ 980.724342] ---[ end trace a2e09071cb38d307 ]---
[ 980.724360] Hello, world
[ 980.724363] Hello, A0248373X
dingfan@dingfan:~/Desktop/assignment3$
```

`sudo rmmod hello_world.ko`

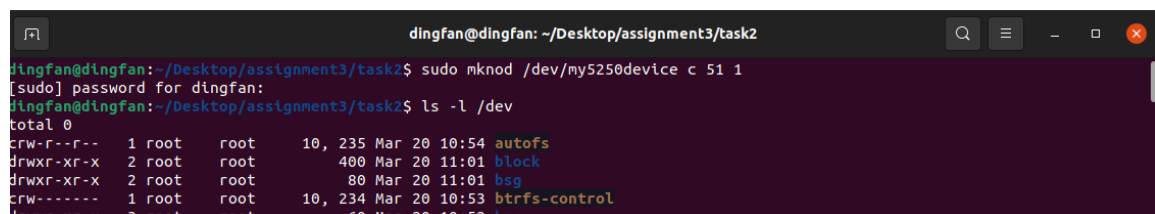
`sudo dmesg | tail`

```
dingfan@dingfan:~/Desktop/assignment3$ sudo rmmod hello_world.ko
dingfan@dingfan:~/Desktop/assignment3$ sudo dmesg | tail
[ 980.724336] RAX: ffffffffda RBX: 0000559a78cfc7d0 RCX: 00007fecdc03594d
[ 980.724337] RDX: 0000000000000000 RSI: 0000559a78cfc2a0 RDI: 0000000000000003
[ 980.724338] RBP: 0000000000000000 R08: 0000000000000000 R09: 00007fecdc12cc60
[ 980.724338] R10: 0000000000000003 R11: 0000000000000246 R12: 0000559a78cfc2a0
[ 980.724339] R13: 0000559a78cfc790 R14: 0000000000000000 R15: 0000559a78cfc8e0
[ 980.724341] </TASK>
[ 980.724342] ---[ end trace a2e09071cb38d307 ]---
[ 980.724360] Hello, world
[ 980.724363] Hello, A0248373X
[ 1193.692835] Goodbye, cruel world
dingfan@dingfan:~/Desktop/assignment3$
```

2. 8 marks, about 2 hours to 4 hours

a. 1 mark. Give the `mknod` command you use. Hint: Careful what major number you use.

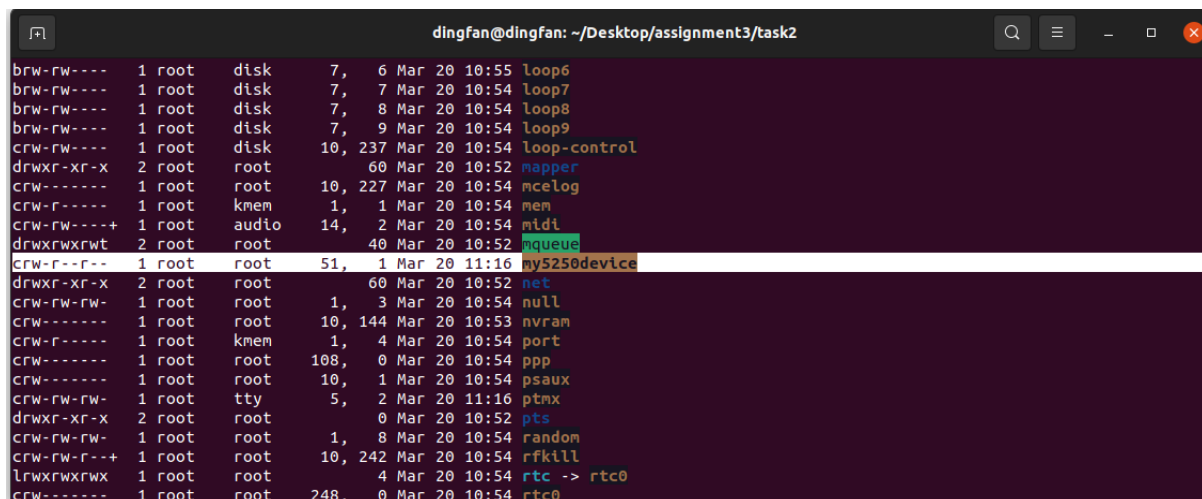
1. `sudo mknod /dev/my5250device c 51 1`



```
dingfan@dingfan: ~/Desktop/assignment3/task2
dingfan@dingfan:~/Desktop/assignment3/task2$ sudo mknod /dev/my5250device c 51 1
[sudo] password for dingfan:
dingfan@dingfan:~/Desktop/assignment3/task2$ ls -l /dev
total 0
crw-r--r--  1 root   root    10, 235 Mar 20 10:54 autofs
drwxr-xr-x  2 root   root    400 Mar 20 11:01 block
drwxr-xr-x  2 root   root     80 Mar 20 11:01 bsg
crw-----  1 root   root    10, 234 Mar 20 10:53 btrfs-control
drwxr-xr-x  3 root   root     60 Mar 20 10:52 bus
```

2. `sudo insmod my_oneByte_device.ko`
3. `sudo rmmod my_oneByte_device.ko`

b. 1 marks. Give the screenshot of your device with “`ls -l /dev`” command and highlight your device.



```
dingfan@dingfan: ~/Desktop/assignment3/task2
brw-rw----  1 root   disk     7,  6 Mar 20 10:55 loop6
brw-rw----  1 root   disk     7,  7 Mar 20 10:54 loop7
brw-rw----  1 root   disk     7,  8 Mar 20 10:54 loop8
brw-rw----  1 root   disk     7,  9 Mar 20 10:54 loop9
crw-rw----  1 root   disk    10, 237 Mar 20 10:54 loop-control
drwxr-xr-x  2 root   root     60 Mar 20 10:52 mapper
crw-----  1 root   root    10, 227 Mar 20 10:54 mcelog
crw-r-----  1 root   kmem     1,  1 Mar 20 10:54 mem
crw-rw----+  1 root   audio    14,  2 Mar 20 10:54 midi
drwxrwxrwt  2 root   root     40 Mar 20 10:52 mqueue
crw-r--r--  1 root   root     51,  1 Mar 20 11:16 my5250device
drwxr-xr-x  2 root   root     60 Mar 20 10:52 net
crw-rw-rw-  1 root   root     1,  3 Mar 20 10:54 null
crw-----  1 root   root    10, 144 Mar 20 10:53 nvram
crw-r-----  1 root   kmem     1,  4 Mar 20 10:54 port
crw-----  1 root   root    108,  0 Mar 20 10:54 ppp
crw-----  1 root   root     10,  1 Mar 20 10:54 psaux
crw-rw-rw-  1 root   tty       5,  2 Mar 20 11:16 ptmx
drwxr-xr-x  2 root   root     0 Mar 20 10:52 pts
crw-rw-rw-  1 root   root     1,  8 Mar 20 10:54 random
crw-rw-r--++  1 root   root    10, 242 Mar 20 10:54 rfkill
lrwxrwxrwx  1 root   root      4 Mar 20 10:54 rtc -> rtc0
crw-----  1 root   root    248,  0 Mar 20 10:54 rtc0
```

c. 6 marks. Give the codes of read and write functions that you implemented and the screenshots of the four testing cases. Hint : Do not run other commands on your device before `cat /dev/<name>`

- the codes of read and write functions that I implemented

```

ssize_t onebyte_read(struct file *filep, char *buf, size_t count, loff_t *f_pos)
{
    /*please complete the function on your own*/
    if(*onebyte_data == 0){
        onebyte_data --;
        return 0;
    }

    if(count>1) // sizeof(char) = 1 (1byte)
    {
        count = 1;
    }

    if(copy_to_user(buf, onebyte_data++, count))
    {
        return -EFAULT;
    }
    printk("read %s\n", onebyte_data);
    return 1;
}

ssize_t onebyte_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos)
{
    /*please complete the function on your own*/
    if(copy_from_user(onebyte_data, buf, 1))
    {
        return -EFAULT;
    }

    printk("write %s\n", onebyte_data);

    if(count>1) // sizeof(char) = 1 (1byte)
    {
        printk(KERN_ALERT "NO SPACE LEFT! Only Write Onebyte!\n");
        return -ENOSPC;
    }

    return count;
}

```

- the screenshots of the four testing cases

```

root@dingfan: /home/dingfan/Desktop/assignment3/task2
dingfan@dingfan:~/Desktop/assignment3/task2$ sudo make clean
make -C /lib/modules/5.16.1/assignment2_v2_dingfan/build M=/home/dingfan/Desktop/assignment3/task2 clean
make[1]: Entering directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
CLEAN /home/dingfan/Desktop/assignment3/task2/Module.symvers
make[1]: Leaving directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
dingfan@dingfan:~/Desktop/assignment3/task2$ sudo make
make -C /lib/modules/5.16.1/assignment2_v2_dingfan/build M=/home/dingfan/Desktop/assignment3/task2 modules
make[1]: Entering directory '/home/dingfan/Desktop/ass2/v2/linux-5.16.1'
CC [M] /home/dingfan/Desktop/assignment3/task2/my_oneByte_device.o
MODPOST /home/dingfan/Desktop/assignment3/task2/Module.symvers
CC [M] /home/dingfan/Desktop/assignment3/task2/my_oneByte_device.mod.o
LD /home/dingfan/Desktop/assignment3/task2/my_oneByte_device.ko
BTF [M] /home/dingfan/Desktop/assignment3/task2/my_oneByte_device.ko
dingfan@dingfan:~/Desktop/assignment3/task2$ sudo insmod my_oneByte_device.ko
dingfan@dingfan:~/Desktop/assignment3/task2$ sudo su
root@dingfan:/home/dingfan/Desktop/assignment3/task2# cat /dev/my5250device
Xroot@dingfan:/home/dingfan/Desktop/assignment3/task2# printf a > /dev/my5250device
root@dingfan:/home/dingfan/Desktop/assignment3/task2# cat /dev/my5250device
a
root@dingfan:/home/dingfan/Desktop/assignment3/task2# printf b > /dev/my5250device
root@dingfan:/home/dingfan/Desktop/assignment3/task2# cat /dev/my5250device
b
root@dingfan:/home/dingfan/Desktop/assignment3/task2# printf abc > /dev/my5250device
bash: printf: write error: No space left on device
root@dingfan:/home/dingfan/Desktop/assignment3/task2# cat /dev/my5250device
a
root@dingfan:/home/dingfan/Desktop/assignment3/task2#

```

3. 5 marks, about an hour

I know I said that tasklets are on the way out but compared to the other means, this is the easiest “bottom half” to use – and it doesn’t have to rely on some hardware IRQ. Implement tasklets that are launched at each of the driver calls above that just use `printk` to log a message giving evidence that the tasklet was executed. Use `dmesg` to show the messages. Hint: Google is your friend The actual work is less than 10 lines of code in the right places. The trick is to explore the Internet and find out what to do.

1. based on the previous question, add codes to the right places to implement the tasklet.

- adding code at the beginning part of the file

```
void tasklet_fn(unsigned long);
/* Tasklet by Dynamic Method */
struct tasklet_struct *tasklet = NULL;

/*Tasklet Function*/
void tasklet_fn(unsigned long arg)
{
    printk(KERN_INFO "Executing Tasklet Function!\n");
}
```

- adding code in the function `onebyte_init`

```
tasklet = kmalloc(sizeof(struct tasklet_struct), GFP_KERNEL);
if(tasklet == NULL) {
    printk(KERN_INFO "etx_device: cannot allocate Memory\n");
}
tasklet_init(tasklet, tasklet_fn, 0);
tasklet_schedule(tasklet);
```

2. The following is the code after modified:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/interrupt.h>

#define MAJOR_NUMBER 51

void tasklet_fn(unsigned long);
/* Tasklet by Dynamic Method */
struct tasklet_struct *tasklet = NULL;

/*Tasklet Function*/
```



```

void tasklet_fn(unsigned long arg)
{
    printk(KERN_INFO "Executing Tasklet Function!\n");
}

/* forward declaration */
int onebyte_open(struct inode *inode, struct file *filep);
int onebyte_release(struct inode *inode, struct file *filep);
ssize_t onebyte_read(struct file *filep, char *buf, size_t count, loff_t *f_pos);
ssize_t onebyte_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos);
static void onebyte_exit(void);

/* definition of file_operation structure */
struct file_operations onebyte_fops = {
    read: onebyte_read,
    write: onebyte_write,
    open: onebyte_open,
    release: onebyte_release
};

char *onebyte_data = NULL;

int onebyte_open(struct inode *inode, struct file *filep)
{
    return 0; // always successful
}

int onebyte_release(struct inode *inode, struct file *filep)
{
    return 0; // always successful
}

ssize_t onebyte_read(struct file *filep, char *buf, size_t count, loff_t *f_pos)
{
    /*please complete the function on your own*/
    if(*onebyte_data == 0){
        onebyte_data--;
        return 0;
    }

    if(count>1) // sizeof(char) = 1 (1byte)
    {
        count = 1;
    }

    if(copy_to_user(buf, onebyte_data++, count))
    {
        return -EFAULT;
    }
    printk("read %s\n", onebyte_data);
    return 1;
}

ssize_t onebyte_write(struct file *filep, const char *buf, size_t count, loff_t *f_pos)
{
    /*please complete the function on your own*/
    if(copy_from_user(onebyte_data, buf, 1))
    {
        return -EFAULT;
    }

    printk("write %s\n", onebyte_data);

    if(count>1) // sizeof(char) = 1 (1byte)

```



```

    {
        printk(KERN_ALERT "NO SPACE LEFT! Only Write Onebyte!\n");
        return -ENOSPC;
    }

    return count;
}

static int onebyte_init(void)
{
    int result;
    // register the device
    result = register_chrdev(MAJOR_NUMBER, "onebyte", &onebyte_fops);
    if (result < 0) {
        return result;
    }
    // allocate one byte of memory for storage
    // kmalloc is just like malloc, the second parameter is
    // the type of memory to be allocated.
    // To release the memory allocated by kmalloc, use kfree.
    onebyte_data = kmalloc(sizeof(char), GFP_KERNEL);
    if (!onebyte_data) {
        onebyte_exit();
        // cannot allocate memory
        // return no memory error, negative signify a failure
        return -ENOMEM;
    }
    // initialize the value to be X
    *onebyte_data = 'X';
    printk(KERN_ALERT "This is a onebyte device module\n");

    tasklet = kmalloc(sizeof(struct tasklet_struct), GFP_KERNEL);
    if (tasklet == NULL) {
        printk(KERN_INFO "etx_device: cannot allocate Memory\n");
    }
    tasklet_init(tasklet, tasklet_fn, 0);
    tasklet_schedule(tasklet);

    printk(KERN_INFO "Device Driver Insert...Done!!!\n");
    return 0;
}

static void onebyte_exit(void) {
    // if the pointer is pointing to something
    if (onebyte_data) {
        // free the memory and assign the pointer to NULL
        kfree(onebyte_data);
        onebyte_data = NULL;
    }
    // unregister the device
    unregister_chrdev(MAJOR_NUMBER, "onebyte");
    printk(KERN_ALERT "Onebyte device module is unloaded\n");
}

MODULE_LICENSE("GPL");
module_init(onebyte_init);
module_exit(onebyte_exit);

```

3. The screenshots of tasklet test result

- The `tasklet_schedule(tasklet)` is called in `onebyte_init` function, when we use command `sudo insmod my_oneByte_device.ko`
 - the function `onebyte_init` is called, and `tasklet_schedule(tasklet)` is called within the function `onebyte_init`

```

dingfan@dingfan: ~/Desktop/assignment3/task3
[sudo] password for dingfan:
make -C /lib/modules/5.16.1/assignment2_v2_dingfan/build M=/home/dingfan/Desktop/assignment3/task3 clean
make[1]: Entering directory '/home/dingfan/Desktop/assignment3/task3'
CLEAN /home/dingfan/Desktop/assignment3/task3/Module.symvers
make[1]: Leaving directory '/home/dingfan/Desktop/assignment3/task3'
dingfan@dingfan:~/Desktop/assignment3/task3$ sudo make
make -C /lib/modules/5.16.1/assignment2_v2_dingfan/build M=/home/dingfan/Desktop/assignment3/task3 modules
make[1]: Entering directory '/home/dingfan/Desktop/assignment3/task3'
CC [M] /home/dingfan/Desktop/assignment3/task3/my_oneByte_device.o
MODPOST /home/dingfan/Desktop/assignment3/task3/Module.symvers
CC [M] /home/dingfan/Desktop/assignment3/task3/my_oneByte_device.mod.o
LD [M] /home/dingfan/Desktop/assignment3/task3/my_oneByte_device.ko
BTF [M] /home/dingfan/Desktop/assignment3/task3/my_oneByte_device.ko
make[1]: Leaving directory '/home/dingfan/Desktop/assignment3/task3'
dingfan@dingfan:~/Desktop/assignment3/task3$ sudo insmod my_oneByte_device.ko
dingfan@dingfan:~/Desktop/assignment3/task3$ sudo dmesg | tail
[ 24.099411] Bluetooth: RFCOMM ver 1.11
[ 26.648044] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 26.648600] ISO 9660 Extensions: RRIP_1991A
[ 26.887335] rkill: input handler disabled
[ 28.448713] input: VMware DnD UInput pointer as /devices/virtual/input/input6
[ 412.514721] my_oneByte_device: loading out-of-tree module taints kernel.
[ 412.514752] my_oneByte_device: module verification failed: signature and/or required key missing - tainting kernel
[ 412.516050] This is a onebyte device module
[ 412.516050] Device Driver Insert...Done!!!
[ 412.516061] Executing Tasklet Function!
dingfan@dingfan:~/Desktop/assignment3/task3$

```

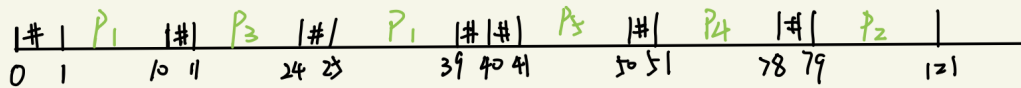
- As the screenshots show, the implemented tasklet is called and log a message giving evidence that the tasklet was executed

Part B:

1. Let's assume that we have the following processes entering the system (in the given order):

- P1 burst CPU time: 23, arrival time 0
- P2 burst CPU time: 42, arrival time 5
- P3 burst CPU time: 13, arrival time 10
- P4 burst CPU time: 27, arrival time 15
- P5 burst CPU time: 9, arrival time 40

(a) What is the execution schedule and the completion time for each if we schedule the processes using preemptive shortest remaining time first with a context switching overhead of 1 time unit?



- At time 0: P1 arrive, P1 is the only process in the ready queue. Begin context switch
- At time 1: P1 start to run in CPU
- At time 5: P2 arrive, the remaining time of P1 is 19, the remaining time of P2 is 42; P1 is already in CPU, P1 continue to run
- At time 10: P3 arrive, the remaining time of P1 is 14, the remaining time of P2 is 42, the remaining time of P3 is 13, P3 remaining time is smaller than P1, begin context switch
- At time 11: P3 begin to run in CPU
- At time 15: P4 arrive, the remaining time of P1 is 14, the remaining time of P2 is 42, the remaining time of P3 is 9, the remaining time of P4 is 27; P3 is already in CPU, P3 continue to run
- At time 24: P3 finished, the remaining time of P1 is 14, the remaining time of P2 is 42, the remaining time of P4 is 27, P1 has the shortest remaining time, begin context switch
- At time 25: P1 begin to run in CPU
- At time 39: P1 finished, the remaining time of P2 is 42, the remaining time of P4 is 27, P4 has the shortest remaining time, begin context switch
- At time 40: P5 arrive. the remaining time of P2 is 42, the remaining time of P4 is 27, the remaining time of P5 is 9, P5 remaining time is smaller than P4, begin context switch
- At time 41: P5 begin to run in process
- At time 50: P5 finished, the remaining time of P2 is 42, the remaining time of P4 is 27, P4 has the shortest remaining time, begin context switch
- At time 51: P4 begin to run in CPU
- At time 78: P4 finished, the remaining time of P2 is 42, begin context switch
- At time 79: P2 begin to run in CPU
- At time 121: P2 finished.

In conclusion: the completion time for each process:

- P1 = 39
- P2 = 121

- P3 = 24
- P4 = 78
- P5 = 50

(b) 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?

- P1 burst CPU time: 23, arrival time 0
- P2 burst CPU time: 42, arrival time 5
- P3 burst CPU time: 13, arrival time 10
- P4 burst CPU time: 27, arrival time 15
- P5 burst CPU time: 9, arrival time 40

time	CPU	queue	
0	P1:23		← P1 Arrive
5	P1:18	P2:42	← P2 Arrive
10	P2:42	P3:13 P1:13	← P3 Arrive ← context switch { P1 out of cpu P2 in to cpu
15	P2:37	P3:13 P1:13 P4:2	← P4 Arrive
20	P3:13	P1:13 P4:27 P2:32	← context switch { P2 out of cpu P3 into cpu
30	P1:13	P4:27 P2:32 P3:3	← context switch { P3 out of cpu P1 into cpu
40	P4:27	P2:32 P3:3 P5:9 P1:3	← P5 Arrive ← context switch { P1 out of cpu P4 into cpu
50	P2:32	P3:3 P5:9 P1:3 P4:17	← context switch { P4 out of cpu P2 into cpu
60	P3:3	P5:9 P1:3 P4:17 P2:22	← context switch { P2 out of cpu P3 into cpu
63	P5:9	P1:3 P4:17 P2:22	← P3 finished ← context switch { P3 out of cpu P5 into cpu
72	P1:3	P4:17 P2:22	← P5 finished ← context switch { P5 out of cpu P1 into cpu
75	P4:17	P2:22	← P1 finished ← context switch { P1 out of cpu P4 into cpu
85	P2:22	P4:7	← context switch { P4 out of cpu P2 into cpu
95	P4:7	P2:12	← context switch { P2 out of cpu P4 into cpu
102	P2:12		← P4 finished ← context switch { P4 out of cpu P2 into cpu
112	P2:2		← P2 continue to execute
114			← P2 finished

- In conclusion: the completion time for each process:
- P1 = 75
- P2 = 114
- P3 = 63
- P4 = 102

- $P5 = 72$

2. On Slide 45 of the “All about processes” PPT deck, I stated that if all jobs are of the same length, then:

(a) Shortest Job First (SJF) is the optimal for minimizing average response time in the nonpreemptive case, and

- If all jobs are of the same length, then the Shortest Job First (SJF) is the same as First Come First Serve (FCFS), in the nonpreemptive case
- Both the 2 algorithm has the same minimizing average response time.

(b) Shortest Remaining Time First (SRTF) is the same as First Come First Serve (FCFS)

Show why these two statements are true.

solution:

- Because the lengths of the task are same, the first arrival task will have the shortest remaining time than the other later arrival tasks. Because the first arrive task has executed for some time.
- For example, A B has the same burst time T . At time 0 , the task A arrive and begin to execute. At time T_1 , the Task B arrive, at this point the remaining time of Task A is $T - T_1$, the remaining time of task B is T .
- According to the Shortest Remaining Time First (SRTF), the shortest remaining task will execute first. Because the first arrival task will have the shortest remaining time. When new task arrive it will not happen preemptive case.
- The Task A has the shortest remaining time, it will continue to execute in CPU.
- So the earlier a task comes, the sooner it is executed. It is the same as First Come First Serve (FCFS).

3. 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.

- Insert 181
- Insert 30
- Insert 14
- Insert 6
- Insert 99

- Insert 123
- Insert 45
- Insert 85
- Insert 21
- Insert 62

solution:

- the final red-black tree:
 - a single circle to represent a “red” node
 - a double concentric circle to represent a “black” node

