CSC3150 Assignment 5

Homework Requirements

Environment

- **WARNING!!!** Before starting on this assignment, make sure you have set up your VM properly. We would test all students' homework using the following environment. You can type the following command in terminal on your VM to see if your configuration matches the test environment. If not, you are still good to go, but please try to test your program with the following environment for at least once. Because you may be able to run your program on your environment, but not on TAs' environment, causng inconvenience or even grade deduction.

  If you follow the tutorials then your VM setting should be fine, though verify your environment again is recommended.

- **Linux Version**

```
main@ubuntu:/$ cat /etc/issue
Ubuntu 16.04.5 LTS \n \l
```

- **GCC Version**

```
main@ubuntu:/$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Submission

- **Due on 23:59, 8 Dec, 2021**
- Please note that, teaching assistants may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we would check whether your program is too similar to your fellow students' code using plagiarism detectors.
- Violation against the format requirements will lead to grade deduction.

Here is the format guide. The project structure is illustrated as below. You can also use `tree` command to check if your structure is fine. Structure mismatch would cause grade deduction.

Note that **NO EXTRA BONUS FILE/FOLDER** is needed for this assignment. Please mark whether you have finished bonus **in your report**.

```
main@ubuntu:~/Desktop/Assignment_5_<student_id>$ tree
.
├── report.pdf
└── source
    ├── ioc_hw5.h
    ├── main.c
    ├── Makefile
    ├── mkdev.sh
    ├── rmdev.sh
```

```
    └── test.c

1 directory, 7 files
```

Please compress all files in the file structure root folder into a single zip file and **name it using your student id as the code showing below and above, for example, Assignment_5_119010001.zip.** The report should be submitted in the format of **pdf**, together with your source code. Format mismatch would cause grade deduction. Here is the sample step for compress your code.

```
main@ubuntu:~/Desktop$ zip -q -r Assignment_5_<student_id>.zip Assignment_5_<student_id>
main@ubuntu:~/Desktop$ ls
Assignment_5_<student_id>                Assignment_5_<student_id>.zip
```

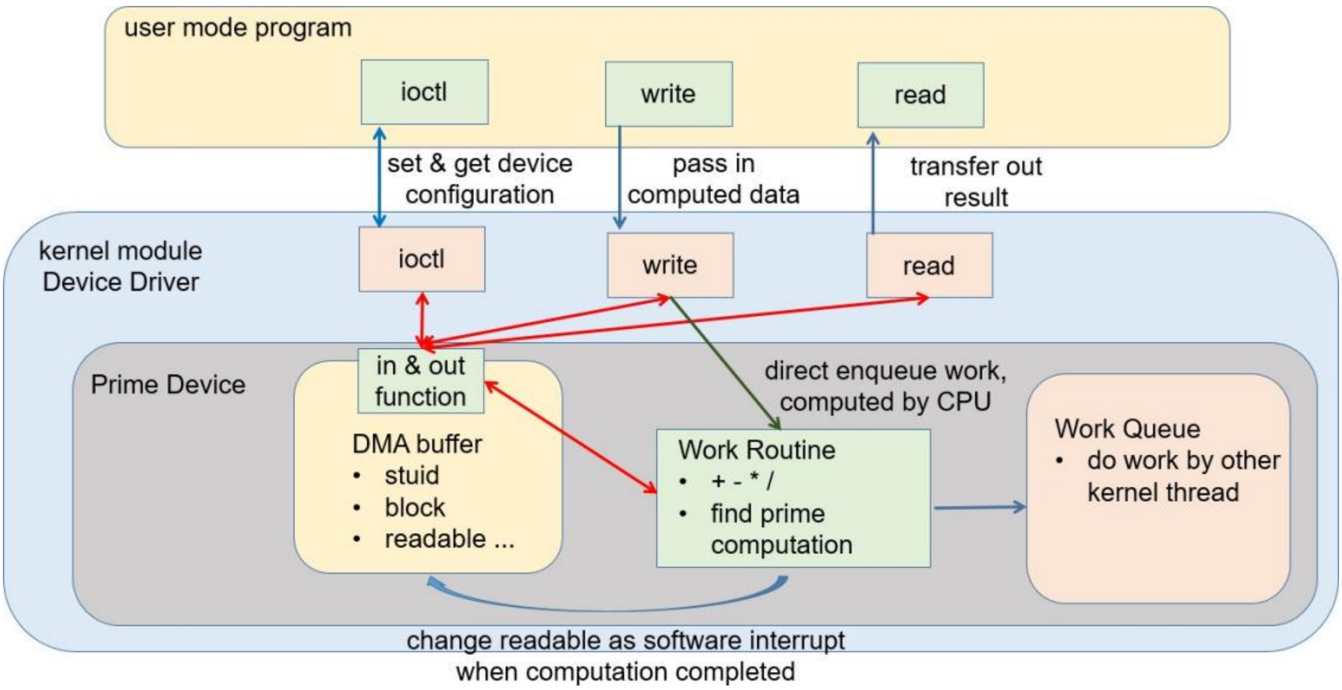
Task Description

In Assignment 5, you are required to make a prime device in Linux, and implement file operations in kernel module to control this device.

Outline:

- We will make a device under /dev by mknod command.
- This device can find n-th prime number.
- You will implement file operations in a kernel module to control this device.
- And implement ioctl function to change the device configuration.
- Simulate registers on device by allocating a memory region.

Global View:



Specification:

- **Register character device and make it live:**
    - You can use alloc_chrdev_region() to allocate a range of char device numbers.
    - And get available number by MAJOR() and MINOR() macro.

    - In your kernel module, you could allocate a cdev structure by cdev_alloc() and initialize it by cdev_init() to bind cdev file_operations.
    - Add a device by cdev_add() to make it live.

- **Test program and printk**:

    - Before write module, we need to know what this module do. So we provide a test program to test this device.
    - You can modify test program and test your module.
    - We will check with our test cases.
    - In kernel module, you need to write printk to help debug and use dmesg command to show message.
    - To help demo program, your printk must be started with "OS_AS5: function_name: message".

```
OS_AS5:init_modules():...............Start...............
OS_AS5:init_modules(): requset_irq 1 return 0
OS_AS5:init_modules(): register chrdev(245,0)
OS_AS5:init_modules(): allocate dma buffer
OS_AS5:drv_open(): device open
```

- **File operations**:

    - You should write a struct **file_operations** to map the operations to functions in this module.
    - And use cdev_init() at module init to bind cdev and file_ operations.

```
static struct file_operations fops = {
        owner: THIS_MODULE,
        read: drv_read,
        write: drv_write,
        unlocked_ioctl: drv_ioctl,
        open: drv_open,
        release: drv_release,
};
```

- **ioctl:**

    - In Linux, device provide user mode program `ioctl` function to change the device configuration.
    - `ioctl` define many types of operation with switch case to do coordinated work.
    - And `ioctl` use mask to get value from these operation label.
    - Here we provide `ioc_hw5.h` to define 6 works.

        a. (HW5_IOC_SETSTUID) Set student ID: printk your student ID
        b. (HW5_IOCSETRWOK) Set if RW OK: printk OK if you complete R/W function
        c. (HW5_IOCSETIOCOK) Set if ioctl OK: printk OK if you complete ioctl function
        d. (HW5_IOCSETIRQOK) Set if IRQ OK: printk OK if you complete bonus
        e. (HW5_IOCSETBLOCK) Set blocking or non-blocking: set write function mode
        f. (HW5_IOCWAITREADABLE) Wait if readable now (synchronize function): used before read to confirm it can read answer now when use non-blocking write mode.

    - ioctl lables defined in `ioc_hw5.h` .

- `_IOW(type, nr, size )` is used for an `ioctl` to write data to the driver. It is to generate command numbers.

    ```
    #ifndef IOC_HW5_H
    #define IOC_HW5_H

    #define HW5_IOC_MAGIC          'k'
    #define HW5_IOCSETSTUID        _IOW(HW5_IOC_MAGIC, 1, int)
    #define HW5_IOCSETRWOK         _IOW(HW5_IOC_MAGIC, 2, int)
    #define HW5_IOCSETIOCOK        _IOW(HW5_IOC_MAGIC, 3, int)
    #define HW5_IOCSETIRQOK        _IOW(HW5_IOC_MAGIC, 4, int)
    #define HW5_IOCSETBLOCK        _IOW(HW5_IOC_MAGIC, 5, int)
    #define HW5_IOCWAITREADABLE    _IOR(HW5_IOC_MAGIC, 6, int)
    #define HW5_IOC_MAXNR          6

    #endif
    ```

- Demo for ioctl call in user mode: `ioctl(fd, HW5_IOCSETBLOCK, &ret)`
    - `fd` : an open file descriptor
    - `HW5_IOCSETBLOCK` : device-dependent request code.
    - `&ret` : an untyped pointer to memory

- **write**:
    - Define a data struct that is passed in write function.

    ```
    struct dataIn {
            char a;
            int b;
            short c;
    };
    ```

    - `a` is operator: '**+**', '**-**', '*****', '**/**', or '**p**' ('p' means find prime number)
    - `b` is operand 1
    - `c` is operand 2
    - Use INIT_WORK() and schedule_work() to queue work to system queue.

- **Find Prime operation:**
    - It finds c-th prime number bigger than b.

      (e.g, "1 p 3" means to find 3rd prime number which is bigger than 1, then it should be 5.)

    - And you will feel the I/O latency when execute test program for "100 p 10000" and "100 p 20000".
    - We will check your blocking and non-blocking IO by observing the delay of the message printed by test program.
    - R/W function packaged in arithmetic function in user mode program.

    `arithmetic(fd, 'p', 100, 10000);`

    - `fd` : an open file descriptor
    - `p` : operator
    - `100` : operand1
    - `10000` : operand2

- **Work Routine:**

- The work you enqueued should be written in a work routine function in module.
- These work will be processed by another kernel thread.
- computation is written in a work routine in module

```
// Arithmetic function
static void drv_arithmetic_routine(struct work_struct* ws);
```

- **Blocking and Non-Blocking IO:**
  - The test program can use ioctl to set blocking or non-blocking.
  - Your write function in module can be blocking or non-blocking.
  - Blocking write need to wait computation completed.
  - Non-blocking write just return after queuing work.
  - Read function only has blocking, because not queuing work.
- **Blocking Write:**
  - In test program, we just need a write function.
  - Do not need another synchronize function.
  - But block when writing.
  - Blocking write in test program:

```
/******************Blocking IO******************/
printf("Blocking IO\n");
ret = 1;
if (ioctl(fd, HW5_IOCSETBLOCK, &ret) < 0) {
    printf("set blocking failed\n");
    return -1;
}

write(fd, &data, sizeof(data));

//Do not need to synchronize
//But need to wait computation completed

read(fd, &ret, sizeof(int));

printf("ans=%d ret=%d\n\n", ans, ret);
/***********************************************/
```

- **Non- Blocking Write:**
  - In test program, we can do something after write function.
  - Write function return after queueing work, it is non-blocking.
  - But need another synchronize function to wait work completed.
  - Non-blocking write in test program:

```
/****************Non-Blocking IO****************/
printf("Non-Blocking IO\n");
ret = 0;
if (ioctl(fd, HW5_IOCSETBLOCK, &ret) < 0) {
    printf("set non-blocking failed\n");
```

```
        return -1;
    }

    printf("Queueing work\n");
    write(fd, &data, sizeof(data));

    //Can do something here
    //But cannot confirm computation completed

    printf("Waiting\n");
    //synchronize function

    while (1) {
        if (ioctl(fd, HW5_IOCWAITREADABLE, &readable), readable)
            break;
        sleep(1);
    }

    printf("Can read now.\n");
    read(fd, &ret, sizeof(int));
    printf("ans=%d ret=%d\n\n", ans, ret);
    /************************************************/
```

- **Interrupt driven IO:**
    - When implementing blocking write and synchronize function, they use a while loop busy waiting the interrupt.
    - You can use a variable to simulate the interrupt.
    - At the final of the work routine function, change this variable as triggering the interrupt.
    - And then, blocking write and synchronize function can exit the while loop.
- **DMA Buffer:**
    - To simulate register and memory on device, you need to kmalloc a dma buffer.
    - This buffer is as I/O port mapping in main memory.
    - What device do is written in work routine function. This function get data from this buffer.
    - Defined value written into dma buffer:

```
// DMA
#define DMA_BUFSIZE 64
#define DMA_STUID_ADDR 0x0        // Student ID
#define DMA_RWOK_ADDR 0x4         // RW function complete
#define DMA_IOCOK_ADDR 0x8        // ioctl function complete
#define DMA_IRQOK_ADDR 0xc        // ISR function complete
#define DMA_COUNT_ADDR 0x10       // interrupt count function complete
#define DMA_ANS_ADDR 0x14         // Computation answer
#define DMA_READABLE_ADDR 0x18    // READABLE variable for synchronize
#define DMA_BLOCK_ADDR 0x1c       // Blocking or non-blocking IO
#define DMA_OPCODE_ADDR 0x20      // data.a opcode
#define DMA_OPERAND_B_ADDR 0x21    // data.b operand1
#define DMA_OPERAND_C_ADDR 0x25    // data.c operand2
static void *dma_buf;
```

- **In and out functions:**
    - You need to implement in & out function to access dma buffer just like physical device.
    - Out function is used to output data to dma buffer.

- On function is used to input data from dma buffer.
- The 6 in & out functions are definded in module to operate `dma_buf` :

  (`c` , `s` and `i` maps with data type `char` , `short` and `int`)

  ```
  // in and out function
  static void myoutc(unsigned char data,unsigned short int port);
  static void myouts(unsigned short data,unsigned short int port);
  static void myouti(unsigned int data,unsigned short int port);
  static unsigned char myinc(unsigned short int port);
  static unsigned short myins(unsigned short int port);
  static unsigned int myini(unsigned short int port);
  ```

- Demo usage of in and out functions:

  `myouti(value, DMAIOCOKADDR)`

  - `value` : data you want to write into `dma_buffer`
  - `DMAIOCOKADDR` : port in `dma_buffer`

- **Data transfer between kernel and user space:**
  - `get_user(x, ptr)`
    - Get a simple variable from user space.
    - `x` : Variable to store result.
    - `ptr` : Source address, in user space.
  - `put_user(x, ptr)`
    - Write a simple value into user space.
    - `x` : Value to copy to user space.
    - `ptr` : Destination address, in user space.

Template structure:

- Makefile is provided:
  - Command: `make`

    (It will firstly build your main.c as kernel module "mydev.ko", insert "mydev.ko", and then build "test.c" as executable file "test".)

    ```
    mname := mydev
    $(mname)-objs := main.o
    obj-m := $(mname).o

    KERNELDIR := /lib/modules/`uname -r`/build

    all:
    	$(MAKE) -C $(KERNELDIR) M=`pwd` modules
    	sudo insmod mydev.ko
    	gcc -o test test.c
    ```

- Command: `make clean`

  (It will remove "mydev.ko" and use "dmesg" to list kernel logs that includes keyword "OS_AS5")

  ```
  clean:
  	$(MAKE) -C $(KERNELDIR) M=`pwd` clean
  ```

```
sudo rmmod mydev
rm test
dmesg | grep OS_AS5
```

- `mknod` script is provided:

  ```bash
  #!/bin/bash
  mknod /dev/mydev c $1 $2
  chmod 666 /dev/mydev
  ls -l /dev/mydev
  ```

  - In mknod command: c means character device. Followed two number are Major and Minor number to specify device.
  - You can get available number by MAJOR() and MINOR() macro after alloc_chrdev_region() in module_init() function.
  - How to use mknod script is shown on Tutorial_11 Slide 3 to 6.

- **Steps you need to run the template:**
  - Run `make`
  - Run `dmesg` to check available device number
  - Run `sudo ./mkdev.sh MAJOR MINOR` to build file node (MAJOR and MINOR are the available device number checked from previous step)
  - Run `./test` to start testing
  - Run `make clean` to remove the module and check the messages
  - Run `sudo ./rmdev.sh` to remove the file node

- You should complete module init and exit functions:

  ```c
  static int __init init_modules(void) {

      printk("%s:%s():..............Start..............\n", PREFIX_TITLE, __func__);

      /* Register chrdev */

      /* Init cdev and make it alive */

      /* Allocate DMA buffer */

      /* Allocate work routine */

      return 0;
  }
  static void __exit exit_modules(void) {

      /* Free DMA buffer when exit modules */

      /* Delete character device */

      /* Free work routine */
  ```

```
printk("%s:%s():..............End..............\n", PREFIX_TITLE, __func__);
```

```
}
```

````
- Implement read/write/ioctl operations and arithmetic routine:

```C
static ssize_t drv_read(struct file *filp, char __user *buffer, size_t ss, loff_t* lo) {
    /* Implement read operation for your device */
    return 0;
}
static ssize_t drv_write(struct file *filp, const char __user *buffer, size_t ss, loff_t* lo) {
    /* Implement write operation for your device */
    return 0;
}
static long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) {
    /* Implement ioctl setting for your device */
    return 0;
}

static void drv_arithmetic_routine(struct work_struct* ws) {
    /* Implement arthemetic routine */
}
````

Function Requirements (90 points):

- Register a character device when module initialized. (5 points)
- Initialized a cdev and add it to make it alive. (5 points)
- Allocate DMA buffer. (5 points)
- Allocate work routine. (5 points)
- Implement read operation for your device. (10 points)
- Implement write operation for your device. (20 points)
- Implement ioctl setting for your device. (15 points)
- Implement arithmetic routine for your device. (10 points)
- Complete module exit functions. (5 points)
- Update your student ID in test case and make it be print in kernel ioctl. (5 points)
- Run test cases to check write and read operations. (5 points)

Demo Output

- Test case: (: `+` , `-` , `*` , `/` is for your testing, we will mainly test `p` operation)

```
arithmetic(fd, '+', 100, 10);
arithmetic(fd, '-', 100, 10);
arithmetic(fd, '*', 100, 10);
arithmetic(fd, '/', 100, 10);
arithmetic(fd, 'p', 100, 10000);
arithmetic(fd, 'p', 100, 20000);
```

- User mode output:

```
...............Start..............
100 + 10 = 110

Blocking IO
ans=110 ret=110

Non-Blocking IO
Queueing work
Waiting
```
```

```
Can read now.
ans=110 ret=110

100 - 10 = 90

Blocking IO
ans=90 ret=90

Non-Blocking IO
Queueing work
Waiting
Can read now.
ans=90 ret=90

100 * 10 = 1000

Blocking IO
ans=1000 ret=1000

Non-Blocking IO
Queueing work
Waiting
Can read now.
ans=1000 ret=1000

100 / 10 = 10

Blocking IO
ans=10 ret=10

Non-Blocking IO
Queueing work
Waiting
Can read now.
ans=10 ret=10

100 p 10000 = 105019

Blocking IO
ans=105019 ret=105019

Non-Blocking IO
Queueing work
Waiting
Can read now.
ans=105019 ret=105019

100 p 20000 = 225077

Blocking IO
ans=225077 ret=225077

Non-Blocking IO
Queueing work
Waiting
Can read now.
ans=225077 ret=225077

...............End...............
```

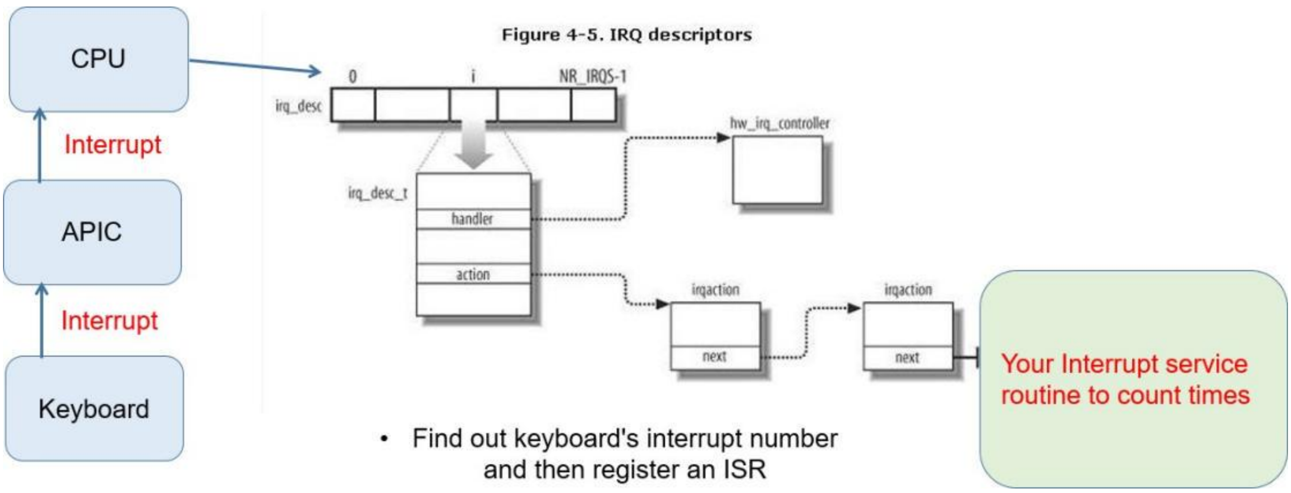
- Kernel mode output:

```
[35962.261080] OS_AS5:init_modules():...............Start...............
[35962.261081] OS_AS5:init_modules(): register chrdev(244,0)
[35962.261087] OS_AS5:init_modules(): allocate dma buffer
```

```
[35984.556209] OS_AS5:drv_open(): device open
[35984.556212] OS_AS5,drv_ioctl(): My STUID is = <student_id>
[35984.556213] OS_AS5,drv_ioctl(): RM OK
[35984.556213] OS_AS5,drv_ioctl(): IOC OK
[35990.337307] OS_AS5,drv_ioctl(): Blocking IO
[35990.337310] OS_AS5:drv_write(): queue work
[35990.337311] OS_AS5:drv_write(): block
[35993.161033] OS_AS5,drv_arithmetic_routine(): 100 p 20000 = 225077
[35993.161125] OS_AS5:drv_read(): ans = 225077
[35993.161140] OS_AS5,drv_ioctl(): Non-Blocking IO
[35993.161142] OS_AS5:drv_write(): queue work
[35993.161145] OS_AS5,drv_ioctl(): wait readable 1
[35996.012354] OS_AS5,drv_arithmetic_routine(): 100 p 20000 = 225077
[35996.116977] OS_AS5:drv_read(): ans = 225077
[35996.117106] OS_AS5:drv_release(): device close
[36010.064424] OS_AS5:exit_modules(): free dma buffer
[36010.064426] OS_AS5:exit_modules(): unregister chrdev
[36010.064426] OS_AS5:exit_modules():..............End..............
```

Bonus Task (10 pionts)

- Global View (Bonus)



- Count the interrupt times of input device like keyboard.
- Hint: watch -n 1 cat /proc/interrupts
- Use request_irq() in module_init to add an ISR into an IRQ number's action list.
- And free_irq() when module_ exit, otherwise kernel panic.
- Please define IRQ_NUM at head of code.
- Demo output:

```
[35962.261080] OS_AS5:init_modules():..............Start..............
[35962.261081] OS_AS5:init_modules(): register chrdev(244,0)
[35962.261087] OS_AS5:init_modules(): request_irq 1 returns 0
[35962.261087] OS_AS5:init_modules(): allocate dma buffer
[35984.556209] OS_AS5:drv_open(): device open
[35984.556212] OS_AS5,drv_ioctl(): My STUID is = <student_id>
[35984.556213] OS_AS5,drv_ioctl(): RM OK
[35984.556213] OS_AS5,drv_ioctl(): IOC OK
[35984.556214] OS_AS5,drv_ioctl(): IRQ OK
[35990.337307] OS_AS5,drv_ioctl(): Blocking IO
[35990.337310] OS_AS5:drv_write(): queue work
```

```
[35990.337311] OS_AS5:drv_write(): block
[35993.161033] OS_AS5,drv_arithmetic_routine(): 100 p 20000 = 225077
[35993.161125] OS_AS5:drv_read(): ans = 225077
[35993.161140] OS_AS5,drv_ioctl(): Non-Blocking IO
[35993.161142] OS_AS5:drv_write(): queue work
[35993.161145] OS_AS5,drv_ioctl(): wait readable 1
[35996.012354] OS_AS5,drv_arithmetic_routine(): 100 p 20000 = 225077
[35996.116977] OS_AS5:drv_read(): ans = 225077
[35996.117106] OS_AS5:drv_release(): device close
[36010.064423] OS_AS5:exit_modules(): interrupt count = 90
[36010.064424] OS_AS5:exit_modules(): free dma buffer
[36010.064426] OS_AS5:exit_modules(): unregister chrdev
[36010.064426] OS_AS5:exit_modules():..............End..............
```

Report (10 points)

Write a report for your assignment, which should include main information as below:

- Your name and student id.
- How did you design your program?
- The environment of running your program. (E.g., version of OS and kernel)
- The steps to execute your program.
- Screenshot of your program output.
- What did you learn from the tasks?

Grading rules

Here is a sample grading scheme. Different from the points specified above, this is the general guide when TA's grading.

Completion	Marks
Bonus	10 points
Report	10 points
Completed with good quality	80 ~ 90
Completed accurately	80 +
Fully Submitted (compile successfully)	60 +
Partial submitted	0 ~ 60
No submission	0
Late submission	Not allowed