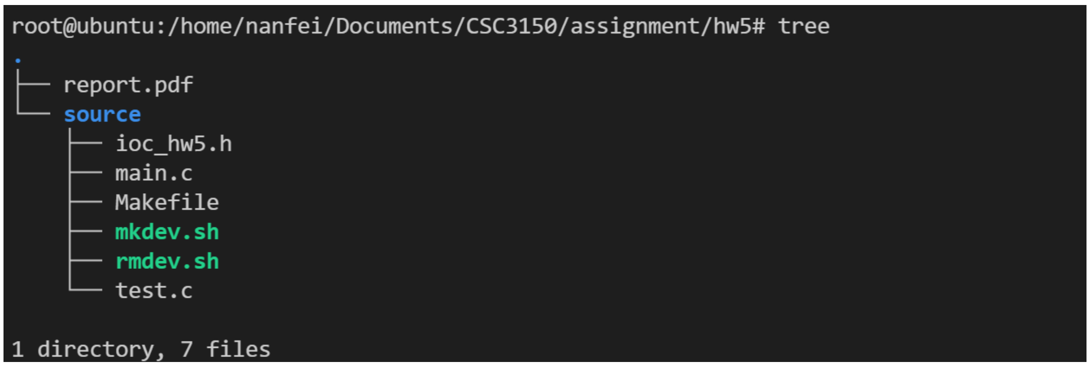
Assignment5 Report

Name: Huang Yingyi

Student ID: 119010114

File Tree (Bonus implemented in main.c)



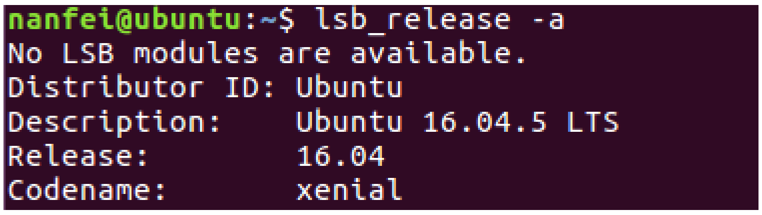
```
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5# tree
.
├── report.pdf
└── source
    ├── ioc_hw5.h
    ├── main.c
    ├── Makefile
    ├── mkdev.sh
    ├── rmdev.sh
    └── test.c

1 directory, 7 files
```

```
.
├── report.pdf
└── source
    ├── ioc_hw5.h
    ├── main.c
    ├── Makefile
    ├── mkdev.sh
    ├── rmdev.sh
    └── test.c

1 directory, 7 files
```

1. Running Environment

Version of OS: Ubuntu **16.04.5 LTS**

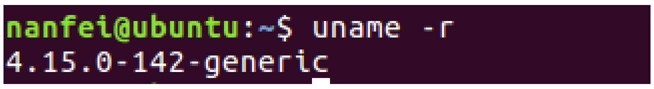
```
lsb_release -a
```



```
nanfei@ubuntu:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.5 LTS
Release:        16.04
Codename:       xenial
```

OS Kernel Version: **4.15.0**

```
uname -r
```



```
nanfei@ubuntu:~$ uname -r
4.15.0-142-generic
```

2. Execution Steps

Preparations

- Unzip the file and come to the directory of source file.

Execution

- Compile the code and install the kernel module

-
  ```
  make all
  ```
- Get the MAJOR & MINOR number

-
  ```
  dmesg | tail -5
  ```
- Make your own device.

-
  ```
  ./mkdev.sh MAJOR MINOR
  ```
- Run the test program / Perform some keyboard interrupts.

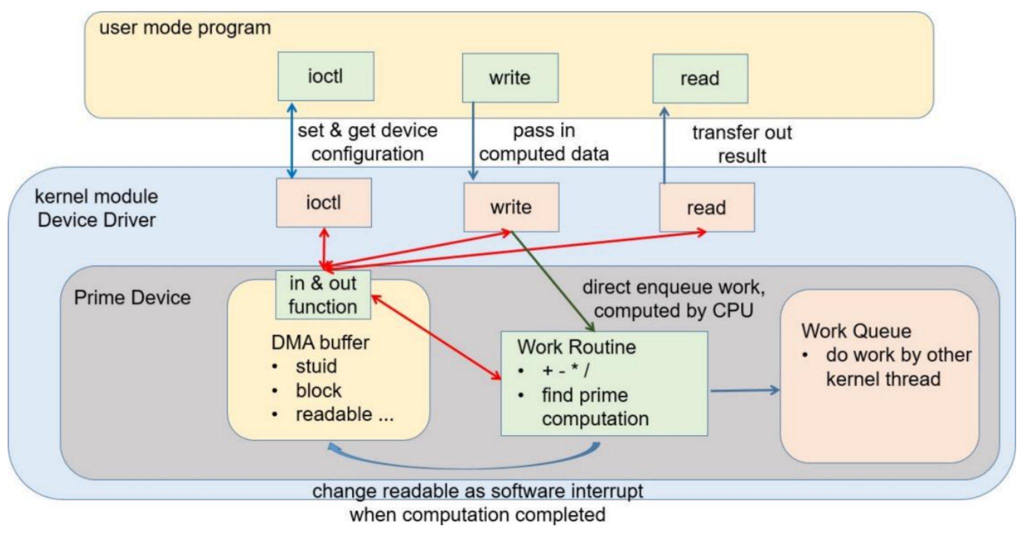
-
  ```
  ./test
  ```
- Clean the compiled files and see the results.

-
  ```
  make clean
  ```
- Clean the kernel message to clean all outputs in this execution.

-
  ```
  dmesg -c
  ```

3. Program Design

Global View:



Basic Task

In this program, we **follow the global** view provided as the homework requirements.

Initialization

There are **many steps to perform** in the initialization (basically as the same as the instructions in the comment).

- Register the character device

```
ret = alloc_chrdev_region(&dev, 0, 1, "mydev");
```

- Initialize the character device and make it alive

```
dev_cdevp = cdev_alloc();
cdev_init(dev_cdevp, &fops);
dev_cdevp->owner = THIS_MODULE;
ret = cdev_add(dev_cdevp, MKDEV(dev_major, dev_minor), 1);
```

- Allocate DMA buffer

```
dma_buf = kzalloc(DMA_BUFSIZE, GFP_KERNEL); // simulate register and
memory on device, kmalloc a dma buffer
```

- Allocate work routine

```
work_routine = kmalloc(sizeof(typeof(*work_routine)), GFP_KERNEL);
```

Exit

In the module exit, we should free the space we have allocated and clean the device we have made.

- Free DMA buffer

```
kfree(dma_buf);
```

- Delete character device

```
cdev_del(dev_cdevp);
unregister_chrdev_region(dev, 1);
```

- Free work routine

```
kfree(work_routine);
```

IO control from user

In this function (drv_ioctl),we should pass the arguments from users to the corresponding address of the DMA buffer. Therefore, we distinguish the **cmd** parameter to decide where to place the users' argument.

- Get argument from users

```
get_user(cur_arg, (int *)arg);
```

- Decide where to place the users' arguments

```
switch (cmd)
    {
    case HW5_IOCSETSTUID:
        ...
    case HW5_IOCSETRWOK:
        ...
    case HW5_IOCSETIOCOK:
        ...
    case HW5_IOCSETIRQOK:
        ...
    case HW5_IOCSETBLOCK:
        ...
    case HW5_IOCWAITREADABLE:
        ...
    default:
        break;
    }
```

- Do the argument assignment

```
myouti(cur_arg, TARGET_ADDRESS);
```

- Retrieve from the target address to validate

```
res = myini(TARGET_ADDRESS);
printk("result as %d\n", TARGET_ADDRESS)
```

Read

In this function (drv_read()), we get the result from the **ANS** address of DMA buffer and set the readable flag as **false**.

- Get the result and send to user space memory

```
unsigned int res = myini(DMAANSADDR);
put_user(res, (unsigned int *)buffer);
```

- Clean the result and the readable flag

```
myouti(0, DMAREADABLEADDR);
myouti(0, DMAANSADDR);
```

Write

In this function (drv_write()), we need to perform the user work in two manner: blocking and non-blocking.

- Get operand from user (a, b, c) and store in DMA buffer

```
dataIn = kmalloc(sizeof(typeof(*dataIn)), GFP_KERNEL);
copy_from_user(dataIn, buffer, ss);
myoutc((unsigned char)dataIn->a, DMAOPCODEADDR);
myouti((unsigned int)dataIn->b, DMAOPERANDBADDR);
myouti((unsigned short)dataIn->c, DMAOPERANDCADDR);
```

- Initialize a work routine to perform arithmetic

```
INIT_WORK(work_routine, drv_arithmetic_routine);
```

- Do in the blocking manner if isBlocking = true

```
schedule_work(work_routine); // put the work task in global workqueue
flush_scheduled_work();
```

- Do in the non-blocking manner if isBlocking = false

```
schedule_work(work_routine);
```

Arithmetic Routine

This function (static void drv_arithmetic_routine(struct work_struct *ws)) is actually the work routine invoked in the write operation. We need to distinguish the operations to perform, store the results, and set the readable flag for users to get the result back.

- Distinguish the operation

```
switch (opcode)
    {
    case '+':
        ...
    case '-':
        ...
    case '*':
        ...
    case '/':
        ...
    case 'p':
        ...
    default:

    }
```

- Do the operations and write result to DMA buffer

```
ans = myini(DMAOPERANDBADDR) [operator] (unsigned
int)myins(DMAOPERANDCADDR);
// or
ans = prime(myini(DMAOPERANDBADDR), myins(DMAOPERANDCADDR));
myouti(ans, DMAANSADDR);
```

- Set the readable flag

  If the operations are in the non-blocking manner, the user should wait for its completion by checking the **readable** flag. Therefore, we should set the **readable** as true to let users read the answer.

```
if (isBlocking == 0)
        myouti(1, DMAREADABLEADDR);
```

Bonus Task

Initialization

In the initialization of bonus part, we should initialize the IRQ by request_irq():

```
#define IRQ_NUM 1
typedef irqreturn_t (*irq_handler_t)(int, void *);
printk("%s:%s(): request_irq %d returns %d\n", PREFIX_TITLE, __func__,
IRQ_NUM, request_irq(IRQ_NUM, (irq_handler_t)handler, IRQF_SHARED,
"OS_ASS5 DEVICE", (void *)dev_cdevp));
```

(We set the IRQ_NUM as the same as the tutorial).

In our interrupt handler, we add one to the interrupt count, which is stored in the DMA buffer.

```
myouti(myini(DMACOUNTADDR) + 1, DMACOUNTADDR);
```

At last, when we are going to exit the module, we should free the interrupt allocated by request_irq():

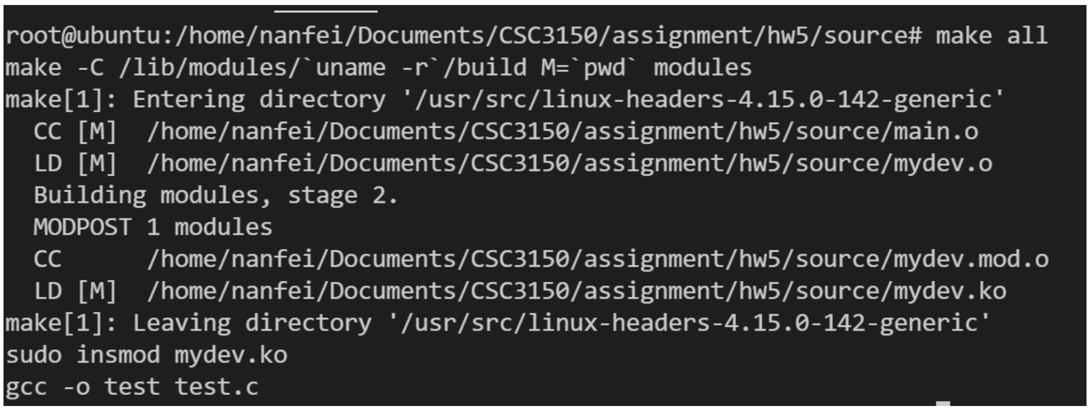
```
free_irq(IRQ_NUM, (void *)dev_cdevp);
```

4. Execution Results Demonstration

All the tests follow the execution steps above, and the test results are screenshots of output.

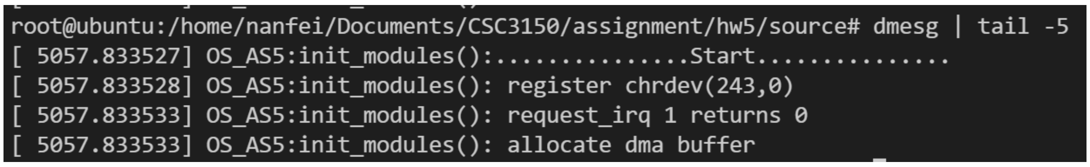
Execution Steps - Test

```
make all
```



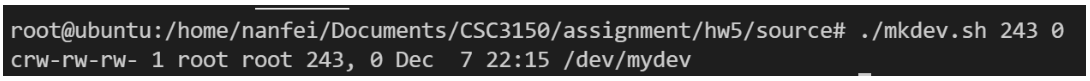
```
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# make all
make -C /lib/modules/`uname -r`/build M=`pwd` modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-142-generic'
  CC [M]  /home/nanfei/Documents/CSC3150/assignment/hw5/source/main.o
  LD [M]  /home/nanfei/Documents/CSC3150/assignment/hw5/source/mydev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/nanfei/Documents/CSC3150/assignment/hw5/source/mydev.mod.o
  LD [M]  /home/nanfei/Documents/CSC3150/assignment/hw5/source/mydev.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-142-generic'
sudo insmod mydev.ko
gcc -o test test.c
```

```
dmesg | tail -5
```



```
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# dmesg | tail -5
[ 5057.833527] OS_AS5:init_modules():..............Start..............
[ 5057.833528] OS_AS5:init_modules(): register chrdev(243,0)
[ 5057.833533] OS_AS5:init_modules(): request_irq 1 returns 0
[ 5057.833533] OS_AS5:init_modules(): allocate dma buffer
```

```
./mkdev.sh  243 0
```



```
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# ./mkdev.sh 243 0
crw-rw-rw- 1 root root 243, 0 Dec  7 22:15 /dev/mydev
```

```
./test
```

```
# click your keyboard
```

```
make clean
```

The results of the 3 steps above are shown in screenshots in the next part, which is to validate the functionality of the program.

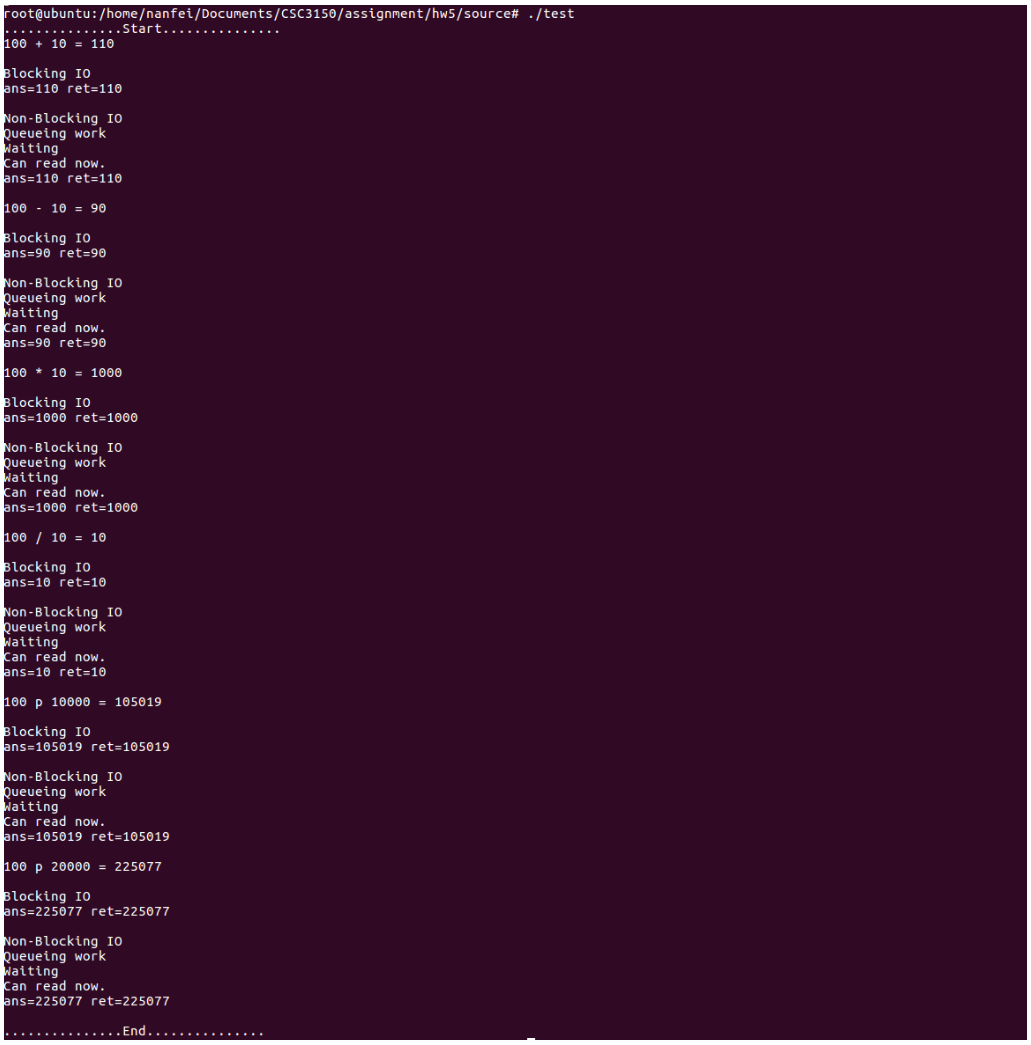
Screenshot Result for Basic and Bonus Task

Basic (Not testing the keyboard now)

- Results after running the test program:

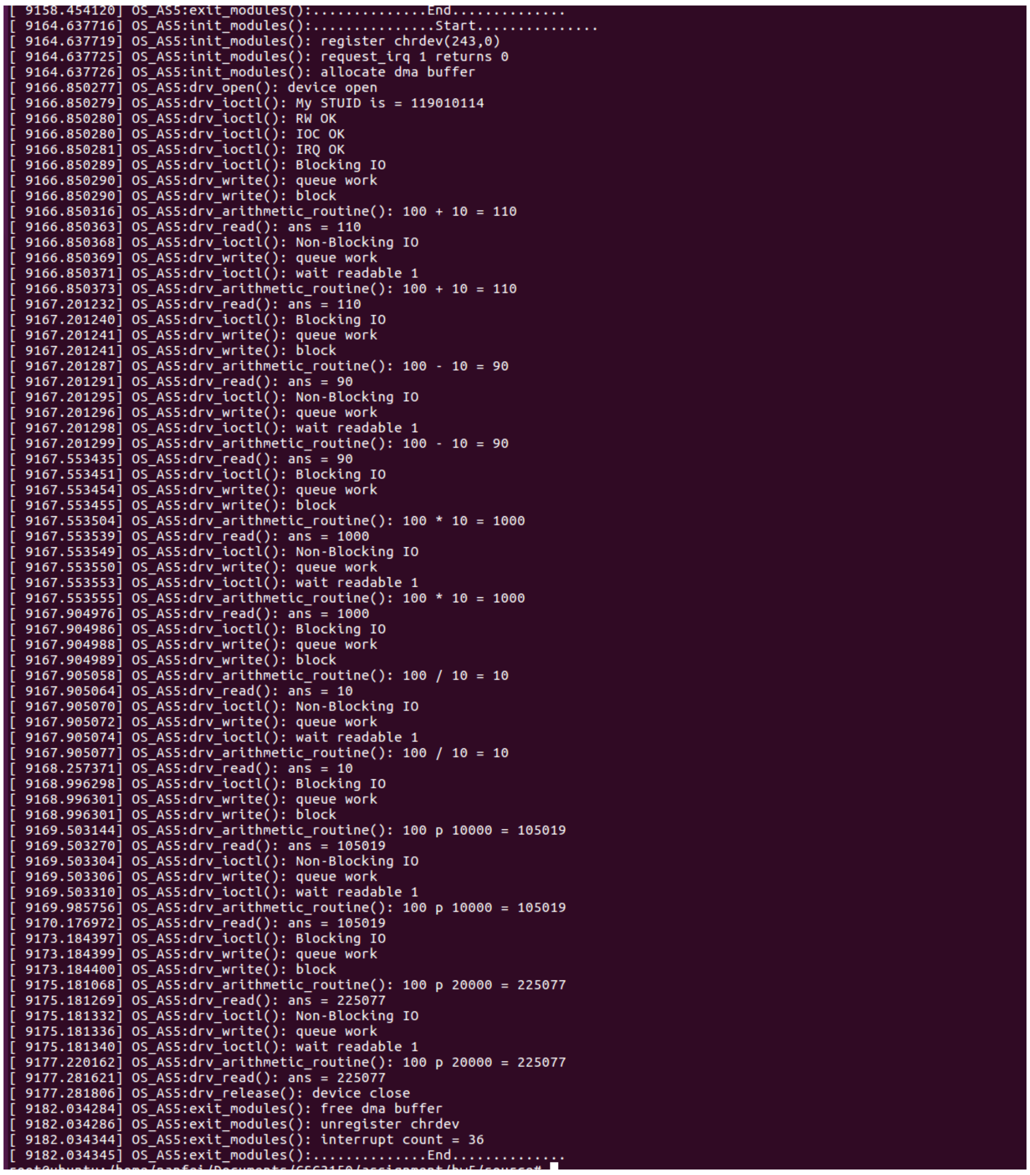
Test1

-
  ```
  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);
  ```
-



```
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# ./test
..............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..............
```

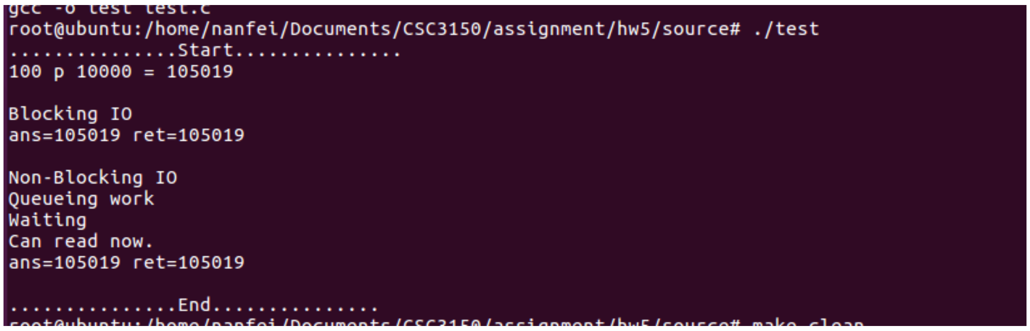
- Results after make clean to display kernel message
-



```
[ 9158.454120] OS_AS5:exit_modules():..............End..............
[ 9164.637716] OS_AS5:init_modules():..............Start..............
[ 9164.637719] OS_AS5:init_modules(): register chrdev(243,0)
[ 9164.637725] OS_AS5:init_modules(): request_irq 1 returns 0
[ 9164.637726] OS_AS5:init_modules(): allocate dma buffer
[ 9166.850277] OS_AS5:drv_open(): device open
[ 9166.850279] OS_AS5:drv_ioctl(): My STUID is = 119010114
[ 9166.850280] OS_AS5:drv_ioctl(): RW OK
[ 9166.850280] OS_AS5:drv_ioctl(): IOC OK
[ 9166.850281] OS_AS5:drv_ioctl(): IRQ OK
[ 9166.850289] OS_AS5:drv_ioctl(): Blocking IO
[ 9166.850290] OS_AS5:drv_write(): queue work
[ 9166.850290] OS_AS5:drv_write(): block
[ 9166.850316] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[ 9166.850363] OS_AS5:drv_read(): ans = 110
[ 9166.850368] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 9166.850369] OS_AS5:drv_write(): queue work
[ 9166.850371] OS_AS5:drv_ioctl(): wait readable 1
[ 9166.850373] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[ 9167.201232] OS_AS5:drv_read(): ans = 110
[ 9167.201240] OS_AS5:drv_ioctl(): Blocking IO
[ 9167.201241] OS_AS5:drv_write(): queue work
[ 9167.201241] OS_AS5:drv_write(): block
[ 9167.201287] OS_AS5:drv_arithmetic_routine(): 100 - 10 = 90
[ 9167.201291] OS_AS5:drv_read(): ans = 90
[ 9167.201295] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 9167.201296] OS_AS5:drv_write(): queue work
[ 9167.201298] OS_AS5:drv_ioctl(): wait readable 1
[ 9167.201299] OS_AS5:drv_arithmetic_routine(): 100 - 10 = 90
[ 9167.553435] OS_AS5:drv_read(): ans = 90
[ 9167.553451] OS_AS5:drv_ioctl(): Blocking IO
[ 9167.553454] OS_AS5:drv_write(): queue work
[ 9167.553455] OS_AS5:drv_write(): block
[ 9167.553504] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[ 9167.553539] OS_AS5:drv_read(): ans = 1000
[ 9167.553549] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 9167.553550] OS_AS5:drv_write(): queue work
[ 9167.553553] OS_AS5:drv_ioctl(): wait readable 1
[ 9167.553555] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[ 9167.904976] OS_AS5:drv_read(): ans = 1000
[ 9167.904986] OS_AS5:drv_ioctl(): Blocking IO
[ 9167.904988] OS_AS5:drv_write(): queue work
[ 9167.904989] OS_AS5:drv_write(): block
[ 9167.905058] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[ 9167.905064] OS_AS5:drv_read(): ans = 10
[ 9167.905070] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 9167.905072] OS_AS5:drv_write(): queue work
[ 9167.905074] OS_AS5:drv_ioctl(): wait readable 1
[ 9167.905077] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[ 9168.257371] OS_AS5:drv_read(): ans = 10
[ 9168.996298] OS_AS5:drv_ioctl(): Blocking IO
[ 9168.996301] OS_AS5:drv_write(): queue work
[ 9168.996301] OS_AS5:drv_write(): block
[ 9169.503144] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[ 9169.503270] OS_AS5:drv_read(): ans = 105019
[ 9169.503304] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 9169.503306] OS_AS5:drv_write(): queue work
[ 9169.503310] OS_AS5:drv_ioctl(): wait readable 1
[ 9169.985756] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[ 9170.176972] OS_AS5:drv_read(): ans = 105019
[ 9173.184397] OS_AS5:drv_ioctl(): Blocking IO
[ 9173.184399] OS_AS5:drv_write(): queue work
[ 9173.184400] OS_AS5:drv_write(): block
[ 9175.181068] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
[ 9175.181269] OS_AS5:drv_read(): ans = 225077
[ 9175.181332] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 9175.181336] OS_AS5:drv_write(): queue work
[ 9175.181340] OS_AS5:drv_ioctl(): wait readable 1
[ 9177.220162] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
[ 9177.281621] OS_AS5:drv_read(): ans = 225077
[ 9177.281806] OS_AS5:drv_release(): device close
[ 9182.034284] OS_AS5:exit_modules(): free dma buffer
[ 9182.034286] OS_AS5:exit_modules(): unregister chrdev
[ 9182.034344] OS_AS5:exit_modules(): interrupt count = 36
[ 9182.034345] OS_AS5:exit_modules():..............End..............
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source#
```

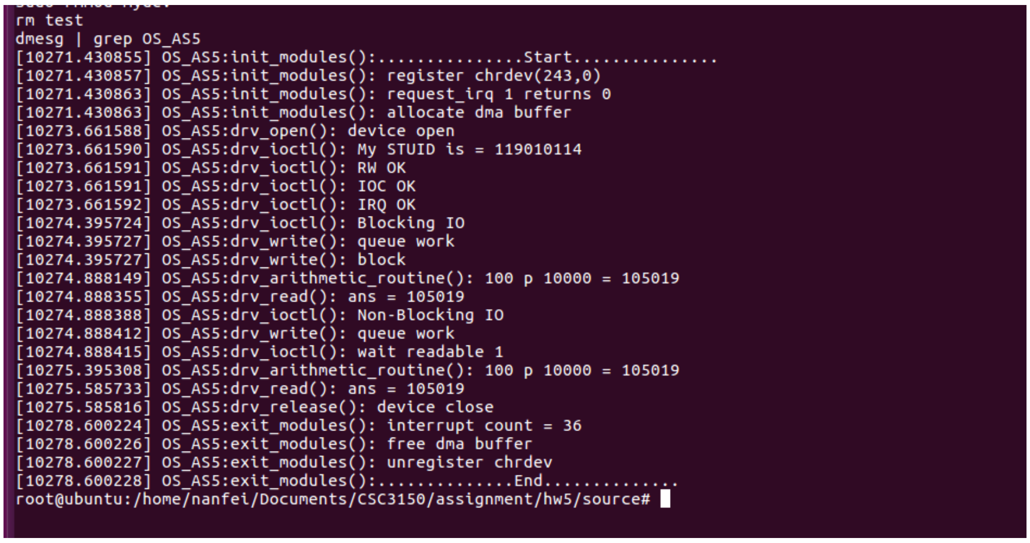
Test2

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

- 

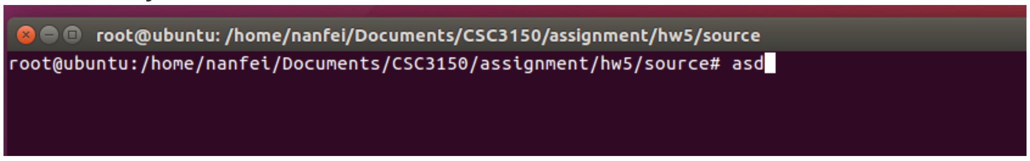
  ```
  gcc -o test test.c
  root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# ./test
  ...............Start...............
  100 p 10000 = 105019

  Blocking IO
  ans=105019 ret=105019

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

  ...............End...............
  root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# make clean
  ```

- Results after make clean to display kernel message
-

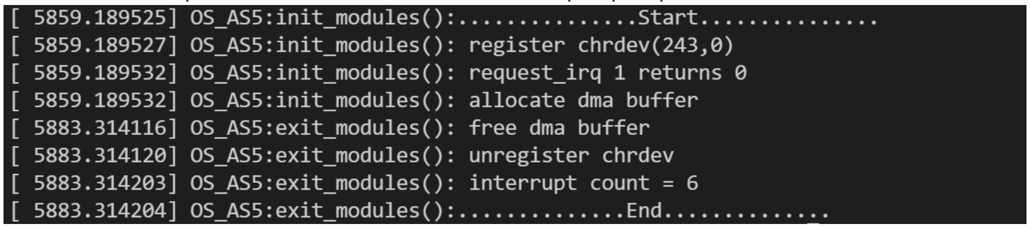


```
rm test
dmesg | grep OS_AS5
[10271.430855] OS_AS5:init_modules():..............Start..............
[10271.430857] OS_AS5:init_modules(): register chrdev(243,0)
[10271.430863] OS_AS5:init_modules(): request_irq 1 returns 0
[10271.430863] OS_AS5:init_modules(): allocate dma buffer
[10273.661588] OS_AS5:drv_open(): device open
[10273.661590] OS_AS5:drv_ioctl(): My STUID is = 119010114
[10273.661591] OS_AS5:drv_ioctl(): RW OK
[10273.661591] OS_AS5:drv_ioctl(): IOC OK
[10273.661592] OS_AS5:drv_ioctl(): IRQ OK
[10274.395724] OS_AS5:drv_ioctl(): Blocking IO
[10274.395727] OS_AS5:drv_write(): queue work
[10274.395727] OS_AS5:drv_write(): block
[10274.888149] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[10274.888355] OS_AS5:drv_read(): ans = 105019
[10274.888388] OS_AS5:drv_ioctl(): Non-Blocking IO
[10274.888412] OS_AS5:drv_write(): queue work
[10274.888415] OS_AS5:drv_ioctl(): wait readable 1
[10275.395308] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[10275.585733] OS_AS5:drv_read(): ans = 105019
[10275.585816] OS_AS5:drv_release(): device close
[10278.600224] OS_AS5:exit_modules(): interrupt count = 36
[10278.600226] OS_AS5:exit_modules(): free dma buffer
[10278.600227] OS_AS5:exit_modules(): unregister chrdev
[10278.600228] OS_AS5:exit_modules():..............End..............
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source#
```

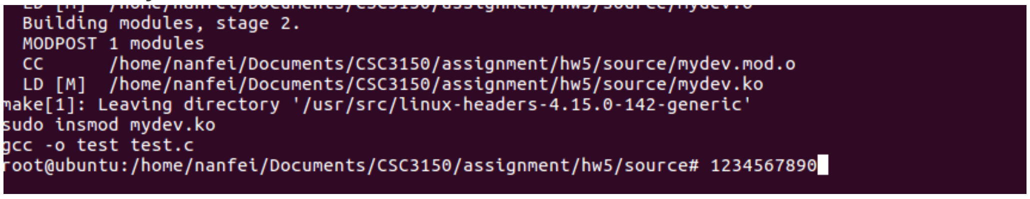
Bonus (Testing on the keyboard interrupt)

- Results after make clean to display kernel message
- Push the keyboards third times.
- 

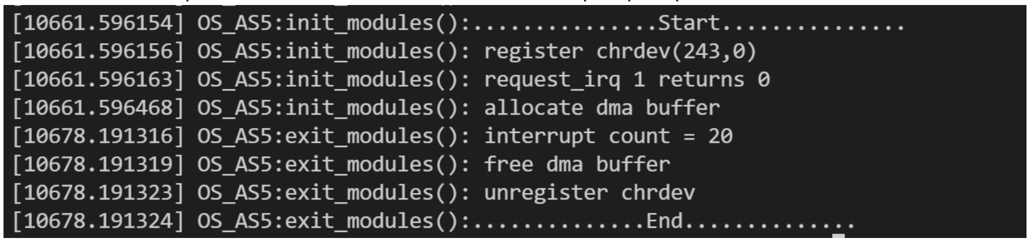
```
root@ubuntu: /home/nanfei/Documents/CSC3150/assignment/hw5/source
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# asd
```

- Get the interrupt count result as 6, two interrupts per push.
- 

```
[ 5859.189525] OS_AS5:init_modules():..............Start..............
[ 5859.189527] OS_AS5:init_modules(): register chrdev(243,0)
[ 5859.189532] OS_AS5:init_modules(): request_irq 1 returns 0
[ 5859.189532] OS_AS5:init_modules(): allocate dma buffer
[ 5883.314116] OS_AS5:exit_modules(): free dma buffer
[ 5883.314120] OS_AS5:exit_modules(): unregister chrdev
[ 5883.314203] OS_AS5:exit_modules(): interrupt count = 6
[ 5883.314204] OS_AS5:exit_modules():.............End.............
```

- Push the keyboards ten times.
- 

```
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/nanfei/Documents/CSC3150/assignment/hw5/source/mydev.mod.o
  LD [M]  /home/nanfei/Documents/CSC3150/assignment/hw5/source/mydev.ko
nake[1]: Leaving directory '/usr/src/linux-headers-4.15.0-142-generic'
sudo insmod mydev.ko
gcc -o test test.c
root@ubuntu:/home/nanfei/Documents/CSC3150/assignment/hw5/source# 1234567890
```

- Get the interrupt count result as 20, two interrupts per push.
- 

```
[10661.596154] OS_AS5:init_modules():..............Start..............
[10661.596156] OS_AS5:init_modules(): register chrdev(243,0)
[10661.596163] OS_AS5:init_modules(): request_irq 1 returns 0
[10661.596468] OS_AS5:init_modules(): allocate dma buffer
[10678.191316] OS_AS5:exit_modules(): interrupt count = 20
[10678.191319] OS_AS5:exit_modules(): free dma buffer
[10678.191323] OS_AS5:exit_modules(): unregister chrdev
[10678.191324] OS_AS5:exit_modules():.............End.............
```

5. Conclusion

According to the tests results, we can conclude that the **design and code implementation** of these three programs are **successful**.

In these tasks, I learnt:

- User program can control the IO device / controller with the help of kernel.
- There is data transfer between user space and kernel space when doing the IO operations.
- The blocking and non-blocking IO should be performed in different manner, in which the work queue design is useful.