CSC3150 Project 5 Report

119010341 WU Yifan

Running Environment

Linux Version

```
root@ubuntu:/home/wyf/Desktop/source# cat /etc/issue
Ubuntu 16.04.5 LTS \n \l
```

GCC Version

```
root@ubuntu:/home/wyf/Desktop/source# gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 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.
```

Execution Guide

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

> 1. Run make
> 2. Run dmesg to check available device number
> 3. Run sudo ./mkdev.sh MAJOR MINOR to build file node (MAJOR and MINOR are the available device number checked from previous step)
> 4. Run ./test to start testing
> 5. Run make clean to remove the module and check the messages
> 6. Run sudo ./rmdev.sh to remove the file node

Sample Output

```
root@ubuntu:/home/wyf/Desktop/source# make
```

```
root@ubuntu:/home/wyf/Desktop/source# make
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/wyf/Desktop/source/main.o
/home/wyf/Desktop/source/main.c: In function ‘init_modules’:
/home/wyf/Desktop/source/main.c:262:2: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
  dev_t dev;
  ^
  LD [M]  /home/wyf/Desktop/source/mydev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/wyf/Desktop/source/mydev.mod.o
  LD [M]  /home/wyf/Desktop/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
```

```
root@ubuntu:/home/wyf/Desktop/source# dmesg
```

```
[ 7825.125264] OS_AS5:init_modules():...............Start...............
[ 7825.125265] OS_AS5:init_modules():register chrdev(243,0)
[ 7825.125269] OS_AS5:init_modules():request_irq 1 returns 0
[ 7825.125269] OS_AS5:init_modules():allocate dma buffer
```

```
root@ubuntu:/home/wyf/Desktop/source# sudo ./mkdev.sh 243 0
```

```
root@ubuntu:/home/wyf/Desktop/source# sudo ./mkdev.sh 243 0
crw-rw-rw- 1 root root 243, 0 Dec  6 03:04 /dev/mydev
```

```
root@ubuntu:/home/wyf/Desktop/source# ./test
```

```
root@ubuntu:/home/wyf/Desktop/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
```

```
root@ubuntu:/home/wyf/Desktop/source# make clean
```

```
root@ubuntu:/home/wyf/Desktop/source# make clean
make -C /lib/modules/`uname -r`/build M=`pwd` clean
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-142-generic'
  CLEAN   /home/wyf/Desktop/source/.tmp_versions
  CLEAN   /home/wyf/Desktop/source/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-142-generic'
sudo rmmod mydev
rm test
dmesg | grep OS_AS5
[   53.343849] OS_AS5:init_modules():...............Start..............
[   53.343850] OS_AS5:init_modules():register chrdev(243,0)
[   53.343855] OS_AS5:init_modules():request_irq 1 returns 0
[   53.343856] OS_AS5:init_modules():allocate dma buffer
[   76.258137] OS_AS5:drv_open(): device open
[   76.258140] OS_AS5,drv_ioctl(): My STUID is = 123456789
[   76.258141] OS_AS5,drv_ioctl(): RW OK
[   76.258141] OS_AS5,drv_ioctl(): IOC OK
[   76.258142] OS_AS5,drv_ioctl(): IRQ OK
[   76.258150] OS_AS5,drv_ioctl(): Blocking IO
[   76.258151] OS_AS5:drv_write(): queue work
[   76.258151] OS_AS5:drv_write(): block
[   76.258175] OS_AS5,drv_arithmetic_routine(): 100 + 10 = 110
[   76.258195] OS_AS5:drv_read(): ans=110
[   76.258200] OS_AS5,drv_ioctl(): Non-Blocking IO
[   76.258202] OS_AS5:drv_write(): queue work
[   76.258204] OS_AS5,drv_ioctl(): wait readable 1
[   76.258206] OS_AS5,drv_arithmetic_routine(): 100 + 10 = 110
[   78.284551] OS_AS5:drv_read(): ans=110
[   78.284558] OS_AS5,drv_ioctl(): Blocking IO
[   78.284560] OS_AS5:drv_write(): queue work
[   78.284560] OS_AS5:drv_write(): block
[   78.284609] OS_AS5,drv_arithmetic_routine(): 100 - 10 = 90
[   78.284613] OS_AS5:drv_read(): ans=90
[   78.284618] OS_AS5,drv_ioctl(): Non-Blocking IO
[   78.284620] OS_AS5:drv_write(): queue work
[   78.284622] OS_AS5,drv_ioctl(): wait readable 1
[   78.284624] OS_AS5,drv_arithmetic_routine(): 100 - 10 = 90
[   80.304154] OS_AS5:drv_read(): ans=90
[   80.304161] OS_AS5,drv_ioctl(): Blocking IO
[   80.304163] OS_AS5:drv_write(): queue work
[   80.304163] OS_AS5:drv_write(): block
[   80.304197] OS_AS5,drv_arithmetic_routine(): 100 * 10 = 1000
[   80.304200] OS_AS5:drv_read(): ans=1000
[   80.304204] OS_AS5,drv_ioctl(): Non-Blocking IO
[   80.304205] OS_AS5:drv_write(): queue work
[   80.304207] OS_AS5,drv_ioctl(): wait readable 1
```

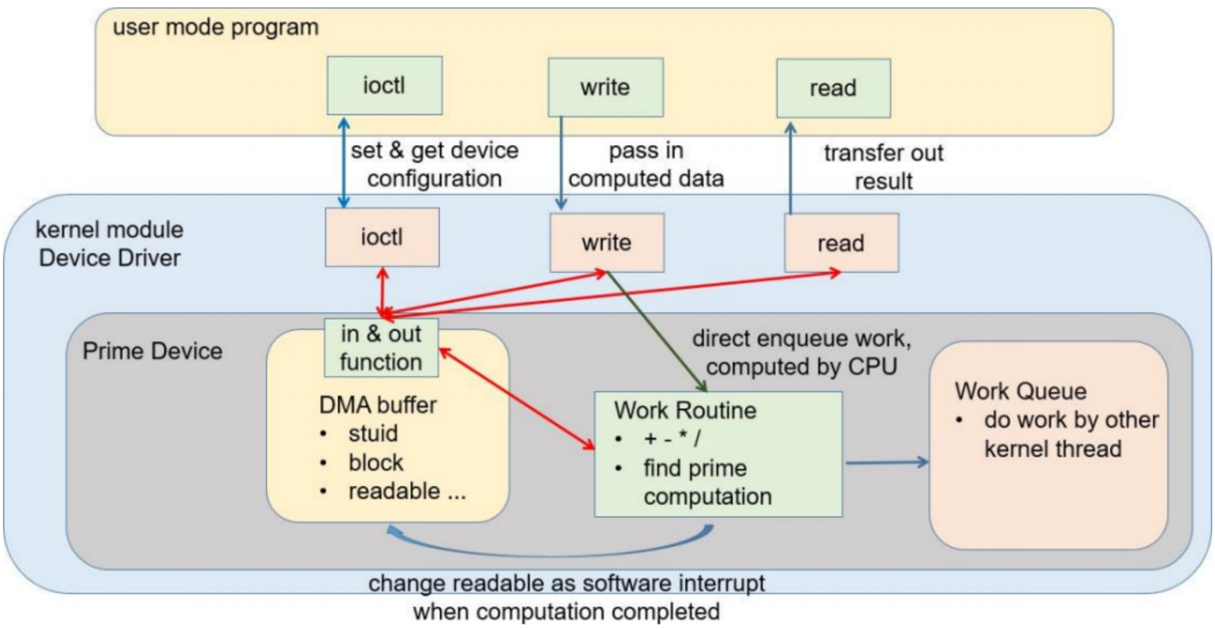
```
[   80.304209] OS_AS5,drv_arithmetic_routine(): 100 * 10 = 1000
[   82.323003] OS_AS5:drv_read(): ans=1000
[   82.323009] OS_AS5,drv_ioctl(): Blocking IO
[   82.323010] OS_AS5:drv_write(): queue work
[   82.323010] OS_AS5:drv_write(): block
[   82.323050] OS_AS5,drv_arithmetic_routine(): 100 / 10 = 10
[   82.323055] OS_AS5:drv_read(): ans=10
[   82.323059] OS_AS5,drv_ioctl(): Non-Blocking IO
[   82.323061] OS_AS5:drv_write(): queue work
[   82.323063] OS_AS5,drv_ioctl(): wait readable 1
[   82.323065] OS_AS5,drv_arithmetic_routine(): 100 / 10 = 10
[   84.342262] OS_AS5:drv_read(): ans=10
[   85.002274] OS_AS5,drv_ioctl(): Blocking IO
[   85.002277] OS_AS5:drv_write(): queue work
[   85.002277] OS_AS5:drv_write(): block
[   85.499443] OS_AS5,drv_arithmetic_routine(): 100 p 10000 = 105019
[   85.499622] OS_AS5:drv_read(): ans=105019
[   85.499631] OS_AS5,drv_ioctl(): Non-Blocking IO
[   85.499633] OS_AS5:drv_write(): queue work
[   85.499635] OS_AS5,drv_ioctl(): wait readable 1
[   86.003162] OS_AS5,drv_arithmetic_routine(): 100 p 10000 = 105019
[   87.514386] OS_AS5:drv_read(): ans=105019
[   90.337084] OS_AS5,drv_ioctl(): Blocking IO
[   90.337086] OS_AS5:drv_write(): queue work
[   90.337087] OS_AS5:drv_write(): block
[   92.530343] OS_AS5,drv_arithmetic_routine(): 100 p 20000 = 225077
[   92.530471] OS_AS5:drv_read(): ans=225077
[   92.530486] OS_AS5,drv_ioctl(): Non-Blocking IO
[   92.530489] OS_AS5:drv_write(): queue work
[   92.530491] OS_AS5,drv_ioctl(): wait readable 1
[   94.686067] OS_AS5,drv_arithmetic_routine(): 100 p 20000 = 225077
[   96.580067] OS_AS5:drv_read(): ans=225077
[   96.580132] OS_AS5:drv_release(): device close
[  102.963031] OS_AS5:exit_modules():free dma buffer
[  102.963033] OS_AS5:exit_modules():unregister chrdev
[  102.963042] OS_AS5:exit_modules(): interrupt count = 92
[  102.963042] OS_AS5:exit_modules():..............End..............
```

```
root@ubuntu:/home/wyf/Desktop/source# sudo ./rmdev.sh
```

```
root@ubuntu:/home/wyf/Desktop/source# sudo ./rmdev.sh
ls: cannot access '/dev/mydev': No such file or directory
```

Program Design--Basic Task

Task Outline



In this assignment, we are required to **implement an arithmetic device and simulate a device driver in kernel mode to control this device**

1. initiate a device in kernel mode
2. allocate a block of memory to simulate registers for device
3. use ioctl functions to configure the device
4. use file operations in kernel mode to control the device
5. the device should be able to conduct some arithmetic operations

Initiate Device

First, we use `alloc_chrdev_region(&dev, DEV_BASEMINOR, DEV_COUNT, DEV_NAME)` to allocate a range of character devices. Here we only need one character device, so we simply set the `DEV_BASEMINOR` to be 0 and `DEV_COUNT` to be 1.

The function will output the device number into the variable `dev`. Note that we will need to create a file system node for the character device, so we use this `dev` together with `MAJOR()` and `MINOR()` to get the major device number and the minor device number of our character device and output them in kernel log.

Next, we perform the following steps to initiate the device

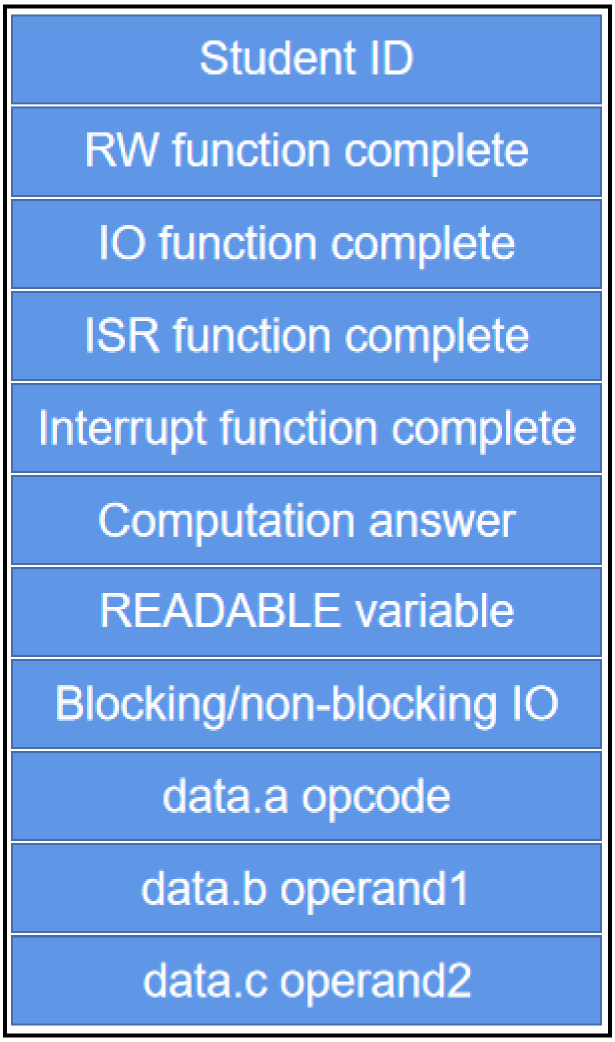
1. use `cdev_alloc()` to allocate the character device structure, the function will return the character device structure `dev_cdev`
2. use `cdev_init()` to initiate the device and match it with our file operations `fops`. The file operations `fops` map our designed file operation functions to the system API
3. use `cdev_add()` to add the device to the system

Allocate Registers

We design a DMA buffer to store the states of our character device.

After that, we can use `kzalloc()` to allocate a block of memory in the kernel for our DMA buffer.

For each variable in the DMA buffer, we can use `myini()`, `myins()`, and `myinc()` to retrieve the data and `myouts`, `myouti`, `myoutc` to modify the data.



DMA Buffer

ioctl Function

In our file operations structure `fops`, the IO configuration functions is named as `drv_ioctl()`. There are 2 main arguments for this function

> - configuration command `unsigned int cmd`
> - user input `unsigned long arg`

`arg` is the input from the user mode, its value equals to the address of the input variable, and we can retrieve the value of this input variable by `get_from_user()`.

`cmd` is the configuration command. In total there are 6 works for the ioctl function: `HW5_IOCSETSTUID`, `HW5_IOCSETRWOK`, `HW5_IOCSETIOCOK`, `HW5_IOCSETIRQOK`, `HW5_IOCSETBLOCK`, and `HW5_IOCWAITREADABLE`. Different command leads to different operations, so we can just use a `switch` statement to implement the structure.

1. For `HW5_IOCSETSTUID`, we retrieve the value of the input variable from user mode and print it in the kernel log

2. For `HW5_IOCSETRWOK`, `HW5_IOCSETIOCOK`, and `HW5_IOCSETIRQOK` we can just print "xx OK"
3. For `HW5_IOCSETBLOCK`, we need to justify the value of the input variable to determine the current IO mode, and then print the IO mode in the kernel log
4. `HW5_IOCWAITREADABLE` is for synchronizing. We should sleep the kernel until the arithmetic function completes the computaion and write back the answer.

Write Function

Firstly, we need to determine the current IO mode through the value of `Blocking/non-Blocking IO` in DMA .

Secondly, the `drv_write()` function contains an argument `char __user *buffer` which is the address of the input data from user mode. We can retrieve the input data from user mode by using `copy_from_user()` and temporarily store the data in our structure `dataIn`.

To formally store the input data, we need to transfer the value of `dataIn->a`, `dataIn->b`, and `dataIn->c` into our DMA buffer.

After that, we may inititate the work routine and add it to our system. Note that for Blocking IO we need to use `flush_scheduled_work()` to flush global work queue to block until its completion.

Arithmetic Function

Firstly, the arithmetic function needs to retrieve the input data from the DMA buffer. The input data includes the arithmetic operator and the 2 operands.

In this assignment, our arithmetic function should be able to complete 5 types of tasks inlcuding `+`, `-`, `*`, `/`, and finding-prime-number. (The detail of the arithmetic operations algorithms are omitted here.)

After complete the computation, the arithmetic function should write back the answer into the DMA buffer immediately. Furthermore, for the non-Blocking IO mode, we need to set the READABLE variable in the DMA buffer to be 1.

Read Function

Firstly, we retrieve the computation answer from the DMA buffer.

Next, the `drv_read()` function contains an argument `char __user *buffer` which is the address of the output variable from user mode. We can use `put_user()` to output our computation answer to the user mode.

We should reset the READABLE variable in DMA buffer to be 0 after the read operation.

Exit deivce

After we complete all the operations for the device, we also need to free the memory. We have allocated for the DMA buffer and the work routine, unregister the deivce we have added to the system, and free the IRQ line(bonus).

Program Design--Bonus

In the bonus task, we will try to count the number of interrupts of the keyboard.

Using the command line `watch -n 1 cat /proc/interrupts`, we can see different interrupt request lines, and we choose `IRQ_NUM` to be 1 which denotes the keyboard interrupts.

```
           CPU0       CPU1
  0:          4          0   IO-APIC    2-edge      timer
  1:          0        327   IO-APIC    1-edge      i8042
  8:          1          0   IO-APIC    8-edge      rtc0
  9:          0          0   IO-APIC    9-fasteoi   acpi
 12:         16      10178   IO-APIC   12-edge      i8042
 14:          0          0   IO-APIC   14-edge      ata_piix
 15:          0          0   IO-APIC   15-edge      ata_piix
 16:        437        449   IO-APIC   16-fasteoi   vmwgfx, snd_ens1371
 17:       9485      27051   IO-APIC   17-fasteoi   ehci_hcd:usb1, ioc0
 18:          0        168   IO-APIC   18-fasteoi   uhci_hcd:usb2
 19:          0       1612   IO-APIC   19-fasteoi   ens33
 24:          0          0   PCI-MSI 344064-edge      PCIe PME, pciehp
 25:          0          0   PCI-MSI 346112-edge      PCIe PME, pciehp
 26:          0          0   PCI-MSI 348160-edge      PCIe PME, pciehp
 27:          0          0   PCI-MSI 350208-edge      PCIe PME, pciehp
 28:          0          0   PCI-MSI 352256-edge      PCIe PME, pciehp
 29:          0          0   PCI-MSI 354304-edge      PCIe PME, pciehp
```

Allocate Interrput line

We use `request_irq(IRQ_NUM, irq_handler, IRQF_SHARED, "my_irq", dev_cdev)` to allocate a interrupt line.

> - `irq_handler` is the function we use to count the number of interrupts
> - To enable the IRQ function, we set the 3rd argument, IRQ flags to be `IRQF_SHARED`
> - The last argument is a cookie passed to the IRQ handler to differentiate interrupts, we can use our device number as parameter

Interrupt Handler

In the DMA buffer, we have a block named as `DMACOUNTADDR`, which can be used to store the interrupt count.

Everytime the interrupt handler is invoked, we just add 1 to the value of the interrupt count stored in the DMA buffer.

Print Out Total Counts of Interrupts

When we exit the device, we can print out the total interrupt count retrieved from the DMA buffer in the kernel log.

What I have learned

Through this assignment, I have had a deeper understanding about the IO control mechanisms. The device is allocated and added to the system when the user use it. Each device has a device controller which includes multiple registers to store the device states. And the most impressive part about IO control is the 2 different IO mode. Tradditionally, we use the non-blocking IO mode, which transfer the data by character and is very slow. We also need to do synchronization between the CPU and the device. However, the Blocking IO mode is much more convenient and efficient.

I also gain deeper understanding about the data trasferring between the use mode the the kernel mode. By using `copy_from_user()`, `get_user()` and `put_user()`, I can easily transfer data among different levels. The most important in this mechanism is using the address of the user variables to retrieve data as well as write back data.