Assignment 5

CSC3150

Student Number: 117010032

Student Name: 陈雅茜

香 港 中 文 大 學（深 圳）

The Chinese University of Hong Kong, Shenzhen

Introduction: Environment and Steps to Compile

This program is written to simulate the prime device in linux. The device can find the nth prime number and do the simple calculation. The program is implemented on C programming language on ubuntu (ubuntu version 16.04 and kernel version v4.8.0-36-generic). The user should input the test external file and the output will be the process of device control.

The command to run the program should be (under the source file directory):
sudo make
dmesg
sudo ./mkdev.sh Major Minor (the major and minor number should be given in the message part shown in the above step)
./test
make clean
sudo ./rmdev.sh

The input program should be like

```c
int main()
{
    printf("..............Start..............\n");

    //open my char device:
    int fd = open("/dev/mydev", O_RDWR);
    if(fd == -1) {
        printf("can't open device!\n");
        return -1;
    }

    int ret;

    ret = 117010032;
    if (ioctl(fd, HW5_IOCSETSTUID, &ret) < 0) {
        printf("set stuid failed\n");
        return -1;
    }

    ret = 1;
    if (ioctl(fd, HW5_IOCSETRWOK, &ret) < 0) {
        printf("set rw failed\n");
        return -1;
    }

    ret = 1;
    if (ioctl(fd, HW5_IOCSETIOCOK, &ret) < 0) {
        printf("set ioc failed\n");
        return -1;
    }

    ret = 1;
    if (ioctl(fd, HW5_IOCSETIRQOK, &ret) < 0) {
        printf("set irq failed\n");
        return -1;
    }

    arithmetic(fd, '+', 100, 10);
```

The output screen should be like

For the kernel interface (when displaying the message)

```
[ 6360.223260] OS_AS5:init_modules():..............Start..............
[ 6360.223263] OS_AS5:init_modules(): register chrdev(245,0)
[ 6360.223264] OS_AS5:init_modules(): allocate dma buffer
[ 6442.772448] OS_AS5:drv_open(): device open
[ 6442.772451] OS_AS5:drv_ioctl(): My student id: 117010032
[ 6442.772452] OS_AS5:drv_ioctl(): RW OK
[ 6442.772452] OS_AS5:drv_ioctl(): IOC OK
[ 6442.772453] OS_AS5:drv_ioctl(): IRQ OK
[ 6443.610894] OS_AS5:drv_ioctl(): Blocking IO
[ 6443.610897] OS_AS5:drv_write():queue work
[ 6443.610897] OS_AS5:drv_write():block
[ 6444.181925] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[ 6444.183719] OS_AS5:drv_read(): ans = 105019
[ 6444.183729] OS_AS5:drv_ioctl(): Non-blocking IO
[ 6444.183731] OS_AS5:drv_write():queue work
[ 6444.183731] OS_AS5,drv_write(): non-blocking
[ 6444.754574] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[ 6449.305405] OS_AS5:drv_ioctl(): wait readable 1
[ 6449.305437] OS_AS5:drv_read(): ans = 105019
[ 6449.305761] OS_AS5:drv_release(): device close
[ 6465.659342] OS_AS5:exit_modules(): free dma buffer
[ 6465.659344] OS_AS5:exit_modules():unregister chrdev
[ 6465.659344] OS_AS5:exit_modules():..............End..............
```

For the user interface (when running the test program)

```
[11/30/19]seed@VM:~/.../os5150_AS$ ./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..............
```

Flow: How Did I Design My Program

1. In the test file, the user program will firstly open the device using open, if the open failed, the program will print the “can’t open device” signal and return -1.

```
//open my char device:
int fd = open("/dev/mydev", O_RDWR);
if(fd == -1) {
    printf("can't open device!\n");
    return -1;
}

int ret;
```

2. Then the user program will pass the data to the kernel mode by calling ioctl function

```
int ret;

ret = 117010032;
if (ioctl(fd, HW5_IOCSETSTUID, &ret) < 0) {
    printf("set stuid failed\n");
    return -1;
}

ret = 1;
if (ioctl(fd, HW5_IOCSETRWOK, &ret) < 0) {
    printf("set rw failed\n");
    return -1;
}

ret = 1;
if (ioctl(fd, HW5_IOCSETIOCOK, &ret) < 0) {
    printf("set ioc failed\n");
    return -1;
}

ret = 1;
if (ioctl(fd, HW5_IOCSETIRQOK, &ret) < 0) {
    printf("set irq failed\n");
    return -1;
}
```

3. When the ioctl function been called, in the kernel mode it will check the command, the kernel mode will get the data from user program to the kernel mode by get_user() function and use the myouti() function to put the data into the DMA buffer.

```
static long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) {
    /* Implement ioctl setting for your device */
    int info;
    get_user(info,(int*)arg);
    int readable  = myini(DMAREADABLEADDR);
    if (cmd == HW5_IOCSETSTUID){
        myouti(info, DMASTUIDADDR);
        printk("%s:%s(): My student id: %i\n", PREFIX_TITLE, __func__, info);
    }
    else if (cmd == HW5_IOCSETRWOK){
        myouti(info, DMARWOKADDR);
        if (info > 0) printk("%s:%s(): RW OK\n", PREFIX_TITLE, __func__);
        else if (info < 0) printk("%s:%s(): RW not complete\n", PREFIX_TITLE, __func__);
    }
    else if (cmd == HW5_IOCSETIOCOK){
        myouti(info, DMAIOCOKADDR);
        if (info > 0) printk("%s:%s(): IOC OK\n", PREFIX_TITLE, __func__);
        else if (info < 0) printk("%s:%s(): IOC not complete\n", PREFIX_TITLE, __func__);
    }
    else if(cmd == HW5_IOCSETIRQOK){
        myouti(info, DMAIRQOKADDR);
        if (info > 0) printk("%s:%s(): IRQ OK\n", PREFIX_TITLE, __func__);
        else if (info < 0) printk("%s:%s(): IRQ not complete\n", PREFIX_TITLE, __func__);
    }
    else if(cmd == HW5_IOCSETBLOCK){
        myouti(info, DMABLOCKADDR);
        if (info == 1) printk("%s:%s(): Blocking IO\n", PREFIX_TITLE, __func__);
        else if (info == 0) printk("%s:%s(): Non-blocking IO\n", PREFIX_TITLE, __func__);
    }
    else if(cmd == HW5_IOCWAITREADABLE){
        while (readable == 0){
            msleep(5000);
            readable = myini(DMAREADABLEADDR);
        }
        put_user(readable,(int *)arg);
        printk("%s:%s(): wait readable 1\n", PREFIX_TITLE, __func__);
    }
    else{
        printk("%s:%s(): no such operation error\n", PREFIX_TITLE, __func__);
    }
    return 0;
}
```

4. The user program will also call the arithmetic function, inside which the user program will firstly do the calculation itself and print the answer.

```c
int arithmetic(int fd, char operator, int operand1, short operand
{
    struct dataIn data;
    int ans;
    int readable;
    int ret;

    data.a = operator;
    data.b = operand1;
    data.c = operand2;

    switch(data.a) {
        case '+':
            ans=data.b+data.c;
            break;
        case '-':
            ans=data.b-data.c;
            break;
        case '*':
            ans=data.b*data.c;
            break;
        case '/':
            ans=data.b/data.c;
            break;
        case 'p':
            ans = prime(data.b, data.c);
            break;
        default:
            ans=0;
    }

    printf("%d %c %d = %d\n\n", data.b, data.a, data.c, ans);
```

5. Then the user program will ask to do the blocking I/O and non-blocking I/O. In the blocking I/O, the user program will firstly call the ioctl() function to check the info number. If info number is 0 or 1, set the DMABLOCKADDR to the info number otherwise -1 and print the “set non-blocking/blocking failed” signal.

```c
/**********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));
```

6. After setting, call the write function to do the I/O instruction. In the write function in kernel mode, the program will firstly fetch the data from user mode using get_user() function and schedule the work, call routine function and flush the work out of the queue in blocking mode.

```
static ssize_t drv_write(struct file *filp, const char __user *buffer, size_t ss, loff_t* lo) {
    /* Implement write operation for your device */
    struct DataIn data;
    get_user(data.a,(char*)buffer);
    get_user(data.b,(int*)buffer+1);
    get_user(data.c,(int*)buffer+2);
    myoutc(data.a, DMAOPCODEADDR);
    myouti(data.b,DMAOPERANDBADDR);
    myouti(data.c,DMAOPERANDCADDR);

    INIT_WORK(work,drv_arithmetic_routine);
    printk("%s:%s():queue work\n",PREFIX_TITLE,__func__);
    if (myini(DMABLOCKADDR) == 1){
        //block IO
        printk("%s:%s():block\n",PREFIX_TITLE,__func__);
        schedule_work(work);
        flush_scheduled_work();
    }
    else {
        // Non-blocking io
        printk("%s,%s(): non-blocking\n",PREFIX_TITLE,__func__);
        myouti(0, DMAREADABLEADDR);
        schedule_work(work);
    }
    return 0;
}
```

7. In the non-blocking mode, the user program will set the non-blocking information as mentioned in the above step and write function in the kernel mode to do the I/O instruction.

```
  /**********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
  ioctl(fd, HW5_IOCWAITREADABLE, &readable);

  if(readable==1){
      printf("Can read now.\n");
      read(fd, &ret, sizeof(int));
  }
  printf("ans=%d ret=%d\n\n", ans, ret);
  /**************************************/
```

8. In the kernel mode, the program needs to firstly set the DMAREADABLEADDR to 0 which means that it is not readable now until the computation complete. Then call the routine function to do the computation.

```
  else {
      // Non-blocking io
      printk("%s,%s(): non-blocking\n",PREFIX_TITLE,__func__);
      myouti(0, DMAREADABLEADDR);
      schedule_work(work);
  }
  return 0;
```

9. In the user mode, the program needs to wait until the computation complete by calling the ioctl function. Then read the data from the device.

```
if(readable==1){
    printf("Can read now.\n");
    read(fd, &ret, sizeof(int));
}
printf("ans=%d ret=%d\n\n", ans, ret);
/**************************************/

return ans;
```

Functions: How did I Design My Program

static ssize_t drv_read(struct file *filp, char __user *buffer, size_t ss, loff_t* lo): the function is used read the answer from the DMA buffer. It will also set the readable to 0 and clean all the data in the DMA.

```
static ssize_t drv_read(struct file *filp, char __user *buffer, size_t ss, loff_t* lo) {
    /* Implement read operation for your device */
    if (myini(DMAREADABLEADDR) == 1){
        printk("%s:%s(): ans = %i\n", PREFIX_TITLE, __func__, myini(DMAANSADDR));
        put_user(myini(DMAANSADDR),(int*)buffer);
        myouti(0, DMAREADABLEADDR);
        //clean the data in DMA
        myouti(0, DMASTUIDADDR);
        myouti(0, DMARWOKADDR);
        myouti(0, DMAIOCOKADDR);
        myouti(0, DMAIRQOKADDR);
        myouti(0, DMACOUNTADDR);
        myouti(0, DMAANSADDR);
        myouti(0, DMABLOCKADDR);
        myouti(0, DMAOPCODEADDR);
        myoutc(NULL, DMAOPCODEADDR);
        myouti(0, DMAOPERANDCADDR);
        myouti(0, DMAOPERANDBADDR);
    }
    return 0;
}
```

static ssize_t drv_write(struct file *filp, const char __user *buffer, size_t ss, loff_t* lo): the function is to get the data from the user mode and write the data to the DMA. There are: blocking and non-blocking mode, in the blocking mode, the it will put the execution function into the work queue, call the function and flush the work. In the non-blocking mode, the program will set the readable data to be 0 and schedule the work.

```
static ssize_t drv_write(struct file *filp, const char __user *buffer, size_t ss, loff_t* lo) {
    /* Implement write operation for your device */
    struct DataIn data;
    get_user(data.a,(char*)buffer);
    get_user(data.b,(int*)buffer+1);
    get_user(data.c,(int*)buffer+2);
    myoutc(data.a, DMAOPCODEADDR);
    myouti(data.b,DMAOPERANDBADDR);
    myouti(data.c,DMAOPERANDCADDR);

    INIT_WORK(work,drv_arithmetic_routine);
    printk("%s:%s():queue work\n",PREFIX_TITLE,__func__);
    if (myini(DMABLOCKADDR) == 1){
        //block IO
        printk("%s:%s():block\n",PREFIX_TITLE,__func__);
        schedule_work(work);
        flush_scheduled_work();
    }
    else {
        // Non-blocking io
        printk("%s,%s(): non-blocking\n",PREFIX_TITLE,__func__);
        myouti(0, DMAREADABLEADDR);
        schedule_work(work);
    }
    return 0;
}
```

static long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg): in this function, the program needs to do the corresponding instruction according to the input command. Get the data from the user mode and store the data in the DMA buffer. The info parameters can only be 0 or 1 for the setting function to show the status. If the info number is not 0 nor 1, it will return -1 and print the error signal in the user mode.

```c
static long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) {
    /* Implement ioctl setting for your device */
    int info;
    get_user(info,(int*)arg);
    int readable  = myini(DMAREADABLEADDR);
    if (cmd == HW5_IOCSETSTUID){
        myouti(info, DMASTUIDADDR);
        printk("%s:%s(): My student id: %i\n", PREFIX_TITLE, __func__, info);
    }
    else if (cmd == HW5_IOCSETRWOK){
        if (info == 0 || info == 1){
            printk("%s:%s(): RW OK\n", PREFIX_TITLE, __func__);
            myouti(info, DMARWOKADDR);
        }
        else{
            printk("%s:%s(): RW not complete\n", PREFIX_TITLE, __func__);
            return -1;
        }
    }
    else if (cmd == HW5_IOCSETIOCOK){
        if (info == 0 || info == 1){
            myouti(info, DMAIOCOKADDR);
            printk("%s:%s(): IOC OK\n", PREFIX_TITLE, __func__);
        }
        else {
            printk("%s:%s(): IOC not complete\n", PREFIX_TITLE, __func__);
            return -1;
        }
    }
    else if(cmd == HW5_IOCSETIRQOK){
        if (info == 0 || info == 1){
            myouti(info, DMAIRQOKADDR);
            printk("%s:%s(): IRQ OK\n", PREFIX_TITLE, __func__);
        }
        else {
            printk("%s:%s(): IRQ not complete\n", PREFIX_TITLE, __func__);
            return -1;
        }
    }
    else if(cmd == HW5_IOCSETBLOCK){
        if (info == 0 || info == 1) myouti(info, DMABLOCKADDR);
        if (info == 1) printk("%s:%s(): Blocking IO\n", PREFIX_TITLE, __func__);
        else if (info == 0) printk("%s:%s(): Non-blocking IO\n", PREFIX_TITLE, __func__);
        else return -1;
    }
    else if(cmd == HW5_IOCWAITREADABLE){
        while (readable == 0){
            msleep(5000);
            readable = myini(DMAREADABLEADDR);
        }
        put_user(readable,(int *)arg);
        printk("%s:%s(): wait readable 1\n", PREFIX_TITLE, __func__);
    }
    else{
        printk("%s:%s(): no such operation error\n", PREFIX_TITLE, __func__);
    }
    return 0;
}
```

int prime(int base, short nth): the function is used to find nth prime number for the drv_arithmetic_routine function.

```
int prime(int base, short nth)
{
    int fnd=0;
    int i, num, isPrime;

    num = base;
    while(fnd != nth) {
        isPrime=1;
        num++;
        for(i=2;i<=num/2;i++) {
            if(num%i == 0) {
                isPrime=0;
                break;
            }
        }

        if(isPrime) {
            fnd++;
        }
    }
    return num;
}
```

static void drv_arithmetic_routine(struct work_struct* ws): this function is used to do the computation according to the input operand in the user mode and store the answer in DMA.

```
static void drv_arithmetic_routine(struct work_struct* ws) {
    /* Implement arthemetic routine */
    struct DataIn data;
    int ans;

    data.a = myinc(DMAOPCODEADDR);
    data.b = myini(DMAOPERANDBADDR);
    data.c = myini(DMAOPERANDCADDR);

    switch(data.a) {
        case '+':
            ans=data.b+data.c;
            break;
        case '-':
            ans=data.b-data.c;
            break;
        case '*':
            ans=data.b*data.c;
            break;
        case '/':
            ans=data.b/data.c;
            break;
        case 'p':
            ans = prime(data.b, data.c);
            break;
        default:
            ans=0;
    }

    myouti(ans,DMAANSADDR);
    myouti(1, DMAREADABLEADDR);
    printk("%s:%s(): %i %c %i = %i\n", PREFIX_TITLE,__func__,data.b, data.a, data.c, ans);
}
```

static int __init init_modules(void): the function is used to initialize the module, it will do the registration, initialize the module, make it alive, allocate the DMA buffer and allocate the work routine.

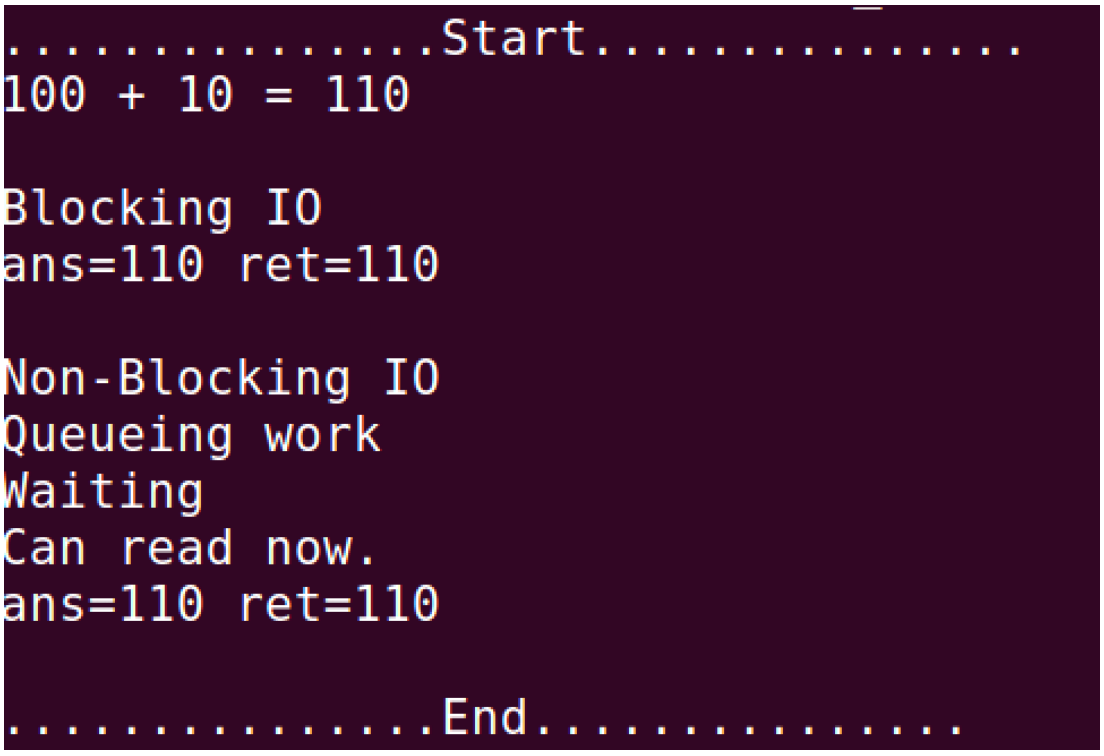
```
static int __init init_modules(void) {

    dev_t dev;

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

    /* Register chrdev */
    if (alloc_chrdev_region(&dev, DEV_BASEMINOR, DEV_COUNT, DEV_NAME) < 0){
        printk(KERN_ALERT"Register chrdev failed!\n");
        return -1;
    }else {
        printk("%s:%s(): register chrdev(%i,%i)\n",PREFIX_TITLE,__func__,MAJOR(dev),MINOR(dev));
    }

    dev_major = MAJOR(dev);
    dev_minor = MINOR(dev);

    /* Init cdev and make it alive */
    dev_cdev = cdev_alloc();
    cdev_init(dev_cdev, &fops);
    dev_cdev->ops = &fops;
    dev_cdev->owner = THIS_MODULE;

    if(cdev_add(dev_cdev,dev,1) < 0){
        printk(KERN_ALERT"%s:%s():Add cdev failed!\n",PREFIX_TITLE,__func__);
        return -1;
    }

    /* Allocate DMA buffer */
    printk("%s:%s(): allocate dma buffer\n",PREFIX_TITLE,__func__);
    dma_buf = kmalloc(DMA_BUFSIZE, GFP_KERNEL);

    /* Allocate work routine */
    work = kmalloc(sizeof(typeof(*work)),GFP_KERNEL);

    return 0;
}
```

static void __exit exit_modules(void): the function is used to free the dma buffer and work routine. It will unregister and delete the character device.

```c
static void __exit exit_modules(void) {

    /* Free DMA buffer when exit modules */
    kfree(dma_buf);
    printk("%s:%s(): free dma buffer\n", PREFIX_TITLE, __func__);

    /* Delete character device */
    unregister_chrdev_region(MKDEV(dev_major,dev_minor),DEV_COUNT);
    cdev_del(dev_cdev);

    /* Free work routine */
    kfree(work);
    printk("%s:%s():unregister chrdev\n",PREFIX_TITLE,__func__);
    printk("%s:%s():.............End.............\n", PREFIX_TITLE, __func__);
}
```

Results: Screen Shot of My Output Result

The user mode for plus



The kernel mode for plus

```
[ 9390.379472] OS_AS5:init_modules():..............Start.............
[ 9390.379475] OS_AS5:init_modules(): register chrdev(245,0)
[ 9390.379477] OS_AS5:init_modules(): allocate dma buffer
[ 9515.224418] OS_AS5:drv_open(): device open
[ 9515.224421] OS_AS5:drv_ioctl(): My student id: 117010032
[ 9515.224421] OS_AS5:drv_ioctl(): RW OK
[ 9515.224422] OS_AS5:drv_ioctl(): IOC OK
[ 9515.224422] OS_AS5:drv_ioctl(): IRQ OK
[ 9515.224494] OS_AS5:drv_ioctl(): Blocking IO
[ 9515.224495] OS_AS5:drv_write():queue work
[ 9515.224496] OS_AS5:drv_write():block
[ 9515.224499] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[ 9515.224501] OS_AS5:drv_read(): ans = 110
[ 9515.224562] OS_AS5:drv_ioctl(): Non-blocking IO
[ 9515.224587] OS_AS5:drv_write():queue work
[ 9515.224587] OS_AS5,drv_write(): non-blocking
[ 9515.224589] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[ 9515.224613] OS_AS5:drv_ioctl(): wait readable 1
[ 9515.224638] OS_AS5:drv_read(): ans = 110
[ 9515.224800] OS_AS5:drv_release(): device close
[ 9536.253598] OS_AS5:exit_modules(): free dma buffer
[ 9536.253600] OS_AS5:exit_modules():unregister chrdev
[ 9536.253600] OS_AS5:exit_modules():.............End.............
```

The user mode for finding the prime number

```
.................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.................
```

The kernel mode for finding the prime number

```
[ 6360.223260] OS_AS5:init_modules():..............Start..............
[ 6360.223263] OS_AS5:init_modules(): register chrdev(245,0)
[ 6360.223264] OS_AS5:init_modules(): allocate dma buffer
[ 6442.772448] OS_AS5:drv_open(): device open
[ 6442.772451] OS_AS5:drv_ioctl(): My student id: 117010032
[ 6442.772452] OS_AS5:drv_ioctl(): RW OK
[ 6442.772452] OS_AS5:drv_ioctl(): IOC OK
[ 6442.772453] OS_AS5:drv_ioctl(): IRQ OK
[ 6443.610894] OS_AS5:drv_ioctl(): Blocking IO
[ 6443.610897] OS_AS5:drv_write():queue work
[ 6443.610897] OS_AS5:drv_write():block
[ 6444.181925] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[ 6444.183719] OS_AS5:drv_read(): ans = 105019
[ 6444.183729] OS_AS5:drv_ioctl(): Non-blocking IO
[ 6444.183731] OS_AS5:drv_write():queue work
[ 6444.183731] OS_AS5,drv_write(): non-blocking
[ 6444.754574] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[ 6449.305405] OS_AS5:drv_ioctl(): wait readable 1
[ 6449.305437] OS_AS5:drv_read(): ans = 105019
[ 6449.305761] OS_AS5:drv_release(): device close
[ 6465.659342] OS_AS5:exit_modules(): free dma buffer
[ 6465.659344] OS_AS5:exit_modules():unregister chrdev
[ 6465.659344] OS_AS5:exit_modules():..............End..............
```

The user mode for times

```
..............Start..............
100 * 10 = 1000

Blocking IO
ans=1000 ret=1000

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

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

The kernel mode for times

```
[ 9818.790141] OS_AS5:init_modules():..............Start..............
[ 9818.790143] OS_AS5:init_modules(): register chrdev(245,0)
[ 9818.790144] OS_AS5:init_modules(): allocate dma buffer
[ 9856.556781] OS_AS5:drv_open(): device open
[ 9856.556784] OS_AS5:drv_ioctl(): My student id: 117010032
[ 9856.556785] OS_AS5:drv_ioctl(): RW OK
[ 9856.556785] OS_AS5:drv_ioctl(): IOC OK
[ 9856.556785] OS_AS5:drv_ioctl(): IRQ OK
[ 9856.556869] OS_AS5:drv_ioctl(): Blocking IO
[ 9856.556870] OS_AS5:drv_write():queue work
[ 9856.556870] OS_AS5:drv_write():block
[ 9856.556874] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[ 9856.556876] OS_AS5:drv_read(): ans = 1000
[ 9856.556945] OS_AS5:drv_ioctl(): Non-blocking IO
[ 9856.556973] OS_AS5:drv_write():queue work
[ 9856.556973] OS_AS5,drv_write(): non-blocking
[ 9856.556975] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[ 9856.557003] OS_AS5:drv_ioctl(): wait readable 1
[ 9856.557469] OS_AS5:drv_read(): ans = 1000
[ 9856.557694] OS_AS5:drv_release(): device close
[ 9871.033060] OS_AS5:exit_modules(): free dma buffer
[ 9871.033062] OS_AS5:exit_modules():unregister chrdev
[ 9871.033062] OS_AS5:exit_modules():..............End..............
```

The user mode for minus

```
..............Start..............
100 - 10 = 90

Blocking IO
ans=90 ret=90

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

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

The kernel mode for minus

```
[ 9643.194965] OS_AS5:init_modules():..............Start..............
[ 9643.194966] OS_AS5:init_modules(): register chrdev(245,0)
[ 9643.194967] OS_AS5:init_modules(): allocate dma buffer
[ 9670.763816] OS_AS5:drv_open(): device open
[ 9670.763819] OS_AS5:drv_ioctl(): My student id: 117010032
[ 9670.763820] OS_AS5:drv_ioctl(): RW OK
[ 9670.763820] OS_AS5:drv_ioctl(): IOC OK
[ 9670.763821] OS_AS5:drv_ioctl(): IRQ OK
[ 9670.763977] OS_AS5:drv_ioctl(): Blocking IO
[ 9670.763979] OS_AS5:drv_write():queue work
[ 9670.763979] OS_AS5:drv_write():block
[ 9670.763982] OS_AS5:drv_arithmetic_routine(): 100 - 10 = 90
[ 9670.763985] OS_AS5:drv_read(): ans = 90
[ 9670.764055] OS_AS5:drv_ioctl(): Non-blocking IO
[ 9670.764083] OS_AS5:drv_write():queue work
[ 9670.764083] OS_AS5,drv_write(): non-blocking
[ 9670.764085] OS_AS5:drv_arithmetic_routine(): 100 - 10 = 90
[ 9670.766368] OS_AS5:drv_ioctl(): wait readable 1
[ 9670.766431] OS_AS5:drv_read(): ans = 90
[ 9670.766653] OS_AS5:drv_release(): device close
[ 9684.066789] OS_AS5:exit_modules(): free dma buffer
[ 9684.066792] OS_AS5:exit_modules():unregister chrdev
[ 9684.067014] OS_AS5:exit_modules():..............End..............
```

The user mode for divide

```
................Start................
100 / 10 = 10

Blocking IO
ans=10 ret=10

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

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

The kernel mode for divide

```
[10182.294552] OS_AS5:init_modules():..............Start..............
[10182.294554] OS_AS5:init_modules(): register chrdev(245,0)
[10182.294556] OS_AS5:init_modules(): allocate dma buffer
[10191.131182] OS_AS5:drv_open(): device open
[10191.131189] OS_AS5:drv_ioctl(): My student id: 117010032
[10191.131190] OS_AS5:drv_ioctl(): RW OK
[10191.131191] OS_AS5:drv_ioctl(): IOC OK
[10191.131191] OS_AS5:drv_ioctl(): IRQ OK
[10191.131312] OS_AS5:drv_ioctl(): Blocking IO
[10191.131315] OS_AS5:drv_write():queue work
[10191.131316] OS_AS5:drv_write():block
[10191.131322] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[10191.131326] OS_AS5:drv_read(): ans = 10
[10191.131418] OS_AS5:drv_ioctl(): Non-blocking IO
[10191.131453] OS_AS5:drv_write():queue work
[10191.131453] OS_AS5,drv_write(): non-blocking
[10191.131456] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[10191.131494] OS_AS5:drv_ioctl(): wait readable 1
[10191.131530] OS_AS5:drv_read(): ans = 10
[10191.131834] OS_AS5:drv_release(): device close
[10203.195904] OS_AS5:exit_modules(): free dma buffer
[10203.195907] OS_AS5:exit_modules():unregister chrdev
[10203.195907] OS_AS5:exit_modules():..............End..............
```

The user mode another prime test with larger operands

```
..............Start..............
100 p 20000 = 225077

Blocking IO
ans=225077 ret=225077

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

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

The user mode another prime test with larger operands

```
[10302.588096] OS_AS5:init_modules():..............Start..............
[10302.588098] OS_AS5:init_modules(): register chrdev(245,0)
[10302.588099] OS_AS5:init_modules(): allocate dma buffer
[10313.043718] OS_AS5:drv_open(): device open
[10313.043722] OS_AS5:drv_ioctl(): My student id: 117010032
[10313.043722] OS_AS5:drv_ioctl(): RW OK
[10313.043723] OS_AS5:drv_ioctl(): IOC OK
[10313.043723] OS_AS5:drv_ioctl(): IRQ OK
[10316.420569] OS_AS5:drv_ioctl(): Blocking IO
[10316.420572] OS_AS5:drv_write():queue work
[10316.420572] OS_AS5:drv_write():block
[10318.915990] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
[10318.918885] OS_AS5:drv_read(): ans = 225077
[10318.919234] OS_AS5:drv_ioctl(): Non-blocking IO
[10318.919236] OS_AS5:drv_write():queue work
[10318.919237] OS_AS5,drv_write(): non-blocking
[10321.443759] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
[10324.118048] OS_AS5:drv_ioctl(): wait readable 1
[10324.118075] OS_AS5:drv_read(): ans = 225077
[10324.118396] OS_AS5:drv_release(): device close
[10334.245016] OS_AS5:exit_modules(): free dma buffer
[10334.245018] OS_AS5:exit_modules():unregister chrdev
[10334.245018] OS_AS5:exit_modules():..............End..............
```

Results: What Are the Problems I Met in this Assignment and My Solutions

1. The most difficult part in this assignment is to understand the working flow, how user mode and kernel mode transfer data with each other? When should we change the readable number? Where should we place the while loop in order to wait for computation complete?

   The solution is to read the text, read tutorial material and search on the Internet for some implement strategy. (1) the user mode and the kernel mode will transfer the data with get_user() function (2) I changed the readable number when the read function finished, when write with non-blocking mode, when finish the computation and when user set the readable number with ioctl function.

Conclusion: What I have learnt from this program

1. How to transfer data from user mode to the kernel mode: use the get_user() function to transfer the data.

2. How to store the data into the DMA buffer: use the myouti() and myoutc() functions to store the data in the correspongding DMA places.
3. How to get data from DMA buffer: use the myini() and myinc() functions
4. What is the difference between blocking and non-blocking modes: in the blocking modes, the user program will wait for the kernel to do the I/O instruction and then continue. In the non-blocking mode, CPU will not wait for I/O but continue work. However, when the user program what to read the file result, it has to continuously check if the file is readable or not. When the file is readable, it will start to read the result.

Bonus

In the bonus part we add request_irq() at the beginning of the init_modules() function.

```
static int __init init_modules(void) {

    dev_t dev;
    request_irq(1,handler,IRQF_SHARED,"myinterrupts",irq_dev_id);

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

    /* Register chrdev */
    if (alloc_chrdev_region(&dev, DEV_BASEMINOR, DEV_COUNT, DEV_NAME) < 0){
        printk(KERN_ALERT"Register chrdev failed!\n");
        return -1;
    }else {
        printk("%s:%s(): register chrdev(%i,%i)\n",PREFIX_TITLE,__func__,MAJOR(dev),MINOR(dev));
    }
```

(1) The first parameter is the interrupt line to allocate, here is 1 since we want our device to share the interrupt number with the keyboard whose index is 1

```
[12/01/19]seed@VM:.../CSC3150_A5$ cat /proc/interrupts
           CPU0
  0:         33    XT-PIC  timer
  1:       4028    XT-PIC  i8042
```

(2) The second parameter is the handler which is a function to be called the IRQ occurs
The function is defined as below

```
static irqreturn_t handler(int irq, void* dev_id){
    if (irq == IRQ_NUM){
        interrupt_count++;
    }
    return IRQ_NONE;
}
```

The handler will count the interrupt number by adding interrupt count number by 1 every time when interrupt occurs

(3) The third parameter in the function is IRQF_SHARED which is the interrupt type flag for shared interrupt.

(4) The forth parameter is the device name, here we claim it as “myinterrupts”.

(5) The fifth parameter is the device id which is a cookie and is defined as

```
static int interrupt_count = 0;
static int IRQ_NUM = 1;
void *irq_dev_id = (void *)&IRQ_NUM;
struct cdev *dev_cdev;
```

In the exit_modules part, we also have to free the irq and print the count number

```
static void __exit exit_modules(void) {

    free_irq(1,irq_dev_id);
    printk("%s:%s(): interrupt count = %d\n", PREFIX_TITLE, __func__, interrupt_count);
    /* Free DMA buffer when exit modules */
```

The output should be like
Before insert the mydev.ko

```
[12/01/19]seed@VM:.../CSC3150_A5$ cat /proc/interrupts
           CPU0
  0:         33    XT-PIC  timer
  1:       3912    XT-PIC  i8042
```

After insert the mydev.ko

```
[12/01/19]seed@VM:.../CSC3150_A5$ cat /proc/interrupts
           CPU0
  0:         33    XT-PIC  timer
  1:       3950    XT-PIC  i8042, myinterrupts
```

After running the test and remove the mydev.ko

```
[12/01/19]seed@VM:.../CSC3150_A5$ cat /proc/interrupts
           CPU0
  0:         33    XT-PIC  timer
  1:       4028    XT-PIC  i8042
```

Interrupt counting

```
[11925.312833] OS_AS5:init_modules():..............Start..............
[11925.312835] OS_AS5:init_modules(): register chrdev(245,0)
[11925.312836] OS_AS5:init_modules(): allocate dma buffer
[12016.725127] OS_AS5:drv_open(): device open
[12016.725130] OS_AS5:drv_ioctl(): My student id: 117010032
[12016.725130] OS_AS5:drv_ioctl(): RW OK
[12016.725131] OS_AS5:drv_ioctl(): IOC OK
[12016.725131] OS_AS5:drv_ioctl(): IRQ OK
[12017.593851] OS_AS5:drv_ioctl(): Blocking IO
[12017.593854] OS_AS5:drv_write():queue work
[12017.593854] OS_AS5:drv_write():block
[12018.185043] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[12018.188638] OS_AS5:drv_read(): ans = 105019
[12018.188649] OS_AS5:drv_ioctl(): Non-blocking IO
[12018.188650] OS_AS5:drv_write():queue work
[12018.188650] OS_AS5,drv_write(): non-blocking
[12018.780207] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[12023.445618] OS_AS5:drv_ioctl(): wait readable 1
[12023.445640] OS_AS5:drv_read(): ans = 105019
[12023.445880] OS_AS5:drv_release(): device close
[12028.426330] OS_AS5:exit_modules(): interrupt count = 52
[12028.426331] OS_AS5:exit_modules(): free dma buffer
[12028.426333] OS_AS5:exit_modules():unregister chrdev
[12028.426333] OS_AS5:exit_modules():..............End..............
```