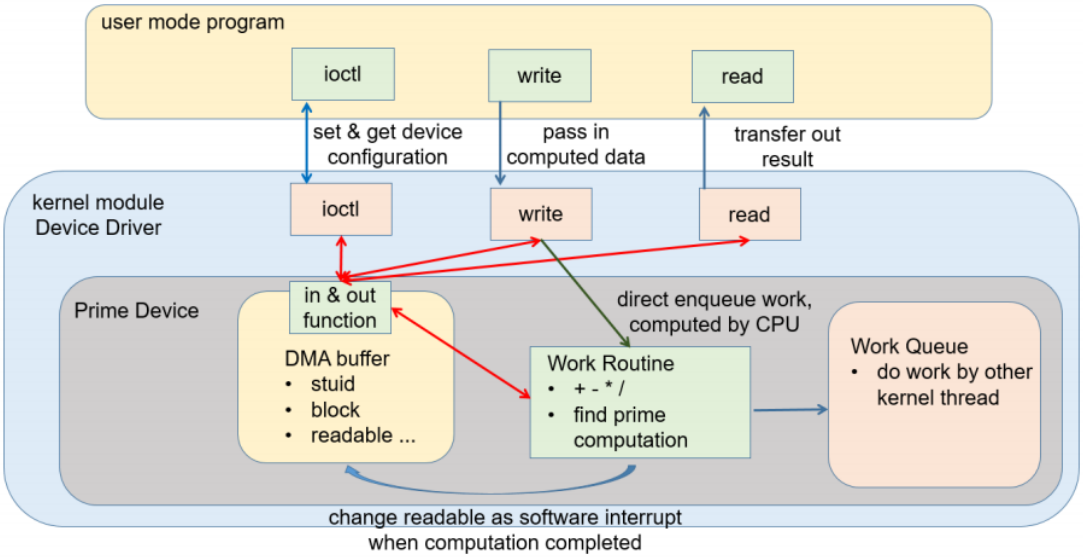
CSC3150 Assignment5 Report

Introduction

In this assignment, a prime device was made and related file operations in kernel module were implemented to control this device. The source code of this implementation is attached with this report, which is in the file *"main.c"*.

Program design



The graph above is the basic logical frame of the program. The following is the details of the implementation.

- Initialization of the module:
    - Register character device:

```
/* Register chrdev */
dev_t dev;
int ret = 0;

ret = alloc_chrdev_region(&dev, DEV_BASEMINOR, DEV_COUNT, DEV_NAME);
if (ret){
    printk("Cannot alloc chrdev");
    return ret;
}

dev_major = MAJOR(dev);
dev_minor = MINOR(dev);
printk("%s:%s(): register chrdev (%d, %d)\n", PREFIX_TITLE, __func__,
dev_major, dev_minor);
```

    - Initialized a cdev and add it to make it alive:

```
/* Init cdev and make it alive */
dev_cdev = cdev_alloc();

cdev_init(dev_cdev, &fops);
dev_cdev->owner = THIS_MODULE;
ret = cdev_add(dev_cdev, MKDEV(dev_major, dev_minor), 1);
if (ret < 0)
{
    printk("Add cdev failed\n");
    return ret;
}
```

- Allocate DMA buffer:

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

- Allocate work routine:

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

- Exit of the module:

```
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 */
    dev_t dev;

    dev = MKDEV(dev_major, dev_minor);
    cdev_del(dev_cdev);

    printk("%s:%s(): unregister chrdev\n", PREFIX_TITLE, __func__);
    unregister_chrdev_region(dev, 1);

    /* Free work routine */
    kfree(work_routine);

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

- API for I/O operations on DMA:

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

- Read operation:

```
int result;
result = myini(DMAANSADDR);         // read the result from DMA
printk("%s:%s(): ans = %d\n", PREFIX_TITLE, __func__, result);
put_user(result, (int*)buffer);     // deliver the result to user mode

myouti(0, DMAANSADDR);              // clean the result
myouti(0, DMAREADABLEADDR);         // set readable as false
```

- Write operation:
  - Check I/O configuration of the device and write to DMA:

```
if (myini(DMARWOKADDR) != 1) return -1;     // check if RW OK

struct DataIn data;
copy_from_user(&data, (struct DataIn*)buffer, sizeof(data));

if (myini(DMAIOCOKADDR) != 1) return -1;    // check if IOC OK

// output data to DMA
myoutc(data.a, DMAOPCODEADDR);
myouti(data.b, DMAOPERANDBADDR);
myouts(data.c, DMAOPERANDCADDR);
```

  - Specified the function (drv_arithmetic) that should be executed in work routine:

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

  - Check whether it is blocking or not, and place the work into work queue based on the I/O mode:

```
// decide blocking or not
if (myini(DMABLOCKADDR)){
    // Blocking IO
    printk("%s:%s(): queue work\n", PREFIX_TITLE, __func__);
    schedule_work(work_routine);
    printk("%s:%s(): block\n", PREFIX_TITLE, __func__);
    flush_scheduled_work();
}else{
    // Non-blocking IO
    printk("%s:%s(): queue work\n", PREFIX_TITLE, __func__);
    myouti(0, DMAREADABLEADDR);
    schedule_work(work_routine);
}
```

- The ioctl setting:
  - Set device configuration:

```
int* ptr = (int*)arg;
switch (cmd)
{
    case HW5_IOCSETSTUID:
        printk("%s:%s(): My STUID is = %d\n", PREFIX_TITLE, __func__,
*ptr);
        myouti(*ptr, DMASTUIDADDR); // set on DMS
        break;
    case HW5_IOCSETRWOK:
        if (*ptr)
            printk("%s:%s(): RW OK\n", PREFIX_TITLE, __func__);
        myouti(*ptr, DMARWOKADDR);  // set on DMS
        break;
    case HW5_IOCSETIOCOK:
        if (*ptr)
            printk("%s:%s(): IOC OK\n", PREFIX_TITLE, __func__);
        myouti(*ptr, DMAIOCOKADDR); // set on DMS
        break;
    case HW5_IOCSETIRQOK:
        if (*ptr)
            printk("%s:%s(): IRQ OK\n", PREFIX_TITLE, __func__);
        myouti(*ptr, DMAIRQOKADDR); // set on DMS
        break;
    case HW5_IOCSETBLOCK:
        if (*ptr){
            printk("%s:%s(): Blocking IO\n", PREFIX_TITLE, __func__);
        }else{
            printk("%s:%s(): Non-Blocking IO\n", PREFIX_TITLE,
__func__);
        }
        myouti(*ptr, DMABLOCKADDR); // set on DMS
        break;
}
```

  - Synchronize the function to wait work completed:

```
case HW5_IOCWAITREADABLE:
    while (myini(DMAREADABLEADDR) == 0){
        msleep(10);     // sleep to do other work
    }
    printk("%s:%s(): wait readable 1\n", PREFIX_TITLE, __func__);
    *ptr = 1;
    break;
```

    Here it will wait and sleep until the work is completed and the readable flag is updated. This is designed to get the right result after the non-blocking write.

- Arithmetic routine:
  - Get the operator and operands:

```
char operator = myinc(DMAOPCODEADDR);
int operand1 = myini(DMAOPERANDBADDR);
short operand2 = myins(DMAOPERANDCADDR);
```

- Based on the operator, do the computation:

```
int ans;
int fnd=0;
int i, num, isPrime, base, nth;

switch(operator) {
    case '+':
        ans = operand1 + operand2;
        break;
    case '-':
        ans = operand1 - operand2;
        break;
    case '*':
        ans = operand1 * operand2;
        break;
    case '/':
        ans = operand1 / operand2;
        break;
    case 'p':
        base = operand1;
        nth = operand2;
        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 ++;
            }
        }   // end while
        ans = num;
        break;
    default:
        ans=0;
} // end switch
```

- Write the result to DMA and update the readable setting for non-blocking write:

```
printk("%s:%s(): %d %c %d = %d\n", PREFIX_TITLE, __func__, operand1,
operator, operand2, ans);
myouti(ans, DMAANSADDR);
if (myini(DMABLOCKADDR) == 0) myouti(1, DMAREADABLEADDR);
```

Steps to run the program

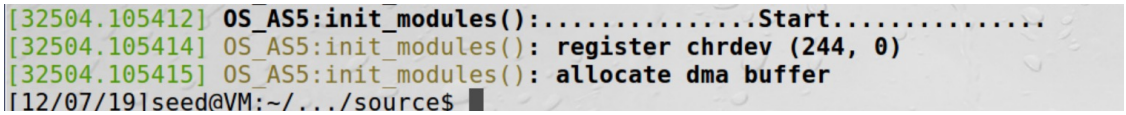
1. In the directory of source code, type the following command and enter to compile programs:

   ```
   $ make
   ```

2. Type the following command to check available device number:

   ```
   $ dmesg
   ```

   There would be some output displayed as follows:

   The last three lines were generated by step 1, check the second last line, the numbers in the brackets represent device number (MAJOR, MINOR). For example, according to the screen shot above, we could get the available device number: MAJOR = 244 and MINOR = 0. These two numbers would be used in the step 3.

3. Type the following command to build file node:

   ```
   $ sudo ./mkdev.sh MAJOR MINOR
   ```

   where the MAJOR and MINOR are the numbers we got in step 2.

4. Type the following command to start testing:

   ```
   $ ./test
   ```

   then there will be some output displayed under user mode. The sample output will be provided in the next part.

5. Type the following command to remove the module and check the message:

   ```
   $ make clean
   ```

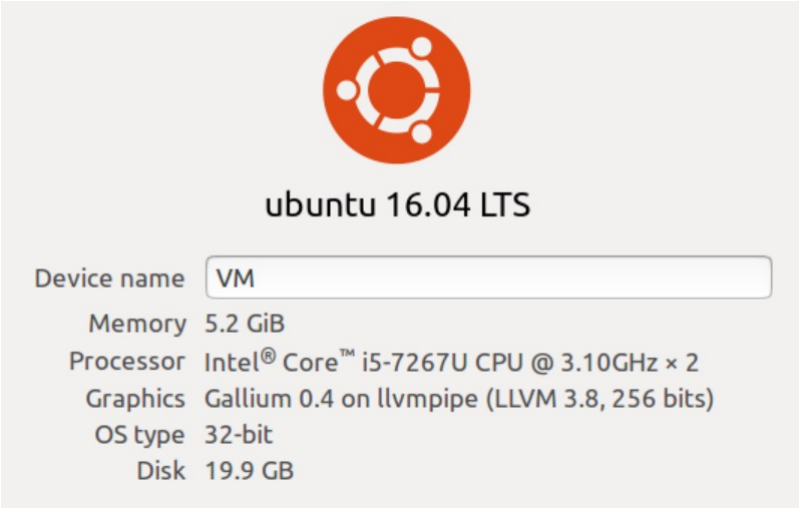
   then there will be some output displayed. The sample output will be provided in the next part.

6. Type the following command to remove the file node:

   ```
   sudo ./rmdev.sh
   ```

Sample output

Running environment



User mode output

```
[12/07/19]seed@VM:~/.../source$ sudo ./mkdev.sh 244 0
mknod: /dev/mydev: File exists
crw-rw-rw- 1 root root 244, 0 Dec  7 04:25 /dev/mydev
[12/07/19]seed@VM:~/.../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..............
[12/07/19]seed@VM:~/.../source$
```

Kernel mode output

```
[32504.105412] OS_AS5:init_modules():..............Start..............
[32504.105414] OS_AS5:init_modules(): register chrdev (244, 0)
[32504.105415] OS_AS5:init_modules(): allocate dma buffer
[33290.426057] OS_AS5:drv_open(): device open
[33290.426060] OS_AS5:drv_ioctl(): My STUID is = 117020343
[33290.426061] OS_AS5:drv_ioctl(): RW OK
[33290.426061] OS_AS5:drv_ioctl(): IOC OK
[33290.426062] OS_AS5:drv_ioctl(): IRQ OK
[33291.238931] OS_AS5:drv_ioctl(): Blocking IO
[33291.238935] OS_AS5:drv_write(): queue work
[33291.238937] OS_AS5:drv_write(): block
[33291.877436] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[33291.877506] OS_AS5:drv_read(): ans = 105019
[33291.877522] OS_AS5:drv_ioctl(): Non-Blocking IO
[33291.877524] OS_AS5:drv_write(): queue work
[33292.537071] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[33292.537393] OS_AS5:drv_ioctl(): wait readable 1
[33292.537406] OS_AS5:drv_read(): ans = 105019
[33292.537664] OS_AS5:drv_release(): device close
[33365.631196] OS_AS5:exit_modules(): free dma buffer
[33365.631198] OS_AS5:exit_modules(): unregister chrdev
[33365.631198] OS_AS5:exit_modules():..............End..............
[12/07/19]seed@VM:~/.../source$
```

NOTE: Here the student ID **(117020343)** was updated in test case and was print in kernel mode.

Encountered problems

Problem

In the implementation of readable setting, how to use *drv_ioctl* to block and check whether the computation has finished?

Solution

In the function *drv_ioctl*, using while loop to keep checking the readable flag until the readable setting is updated by arithmetic routine.

```
while (myini(DMAREADABLEADDR) == 0){
    msleep(10);
}
```

Problem

When non-blocking write was adopted, the result could not be rightly generated.

Solution

According to the wrong result displayed, its value is 0, which is the value set after the last read operation clear the result in DMA. Thus, the right result might not be updated to the DMA in the non-blocking operations.

The code was checked and the reason was found: the readable setting was updated before the answer was written into DMA. Since right after the readable setting was updated, the function *drv_ioctl* returned and the function *drv_read* was called, but the right answer had not be updated to DMA, then the wrong result was read. Therefore, the bug was fixed by simply changing the order of two lines of code.

The following is the right order of the two lines of code:

```
myouti(ans, DMAANSADDR);
if (myini(DMABLOCKADDR) == 0) myouti(1, DMAREADABLEADDR);
```

What I learned

- How to design a driver program
    - Register a device and make it live
    - Map operations to functions in the designed module
    - Use ioctl function to change the device configuration
    - The data transmission between user mode and kernel mode
    - Use work routine to schedule the work
    - ...
- More understanding on I/O system
- ...