Report

HuangPengxiang

12/7/2021

Contents

Overview

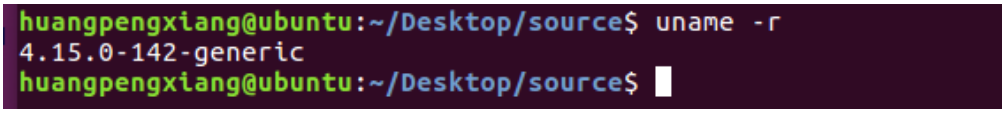
This is the Fifth Assignment of Operating System. In this assignment, The programming goal is to implement the simulation of prime calculation device. This simulation is tested via Linux system with kernel thread. This simulation is also provided file operation to control the device such as write and read. I also **implement the bonus** part, which is aimed to count the interrupt times of input device like keyboard. The kernel result message of this program could use command `dmesg` to check, and the terminal output will also be included.

Some Important Declarations

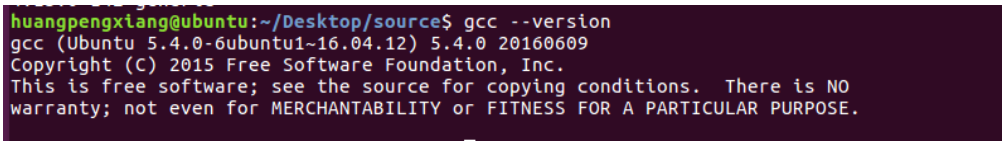
- There might be some Indentation unaccustomedness in your computer. I used Mac VSC to develop the program and test it on Ubuntu VM, and I found the Indentation is not comfortable to view when I open it on test computer. I am sorry for that and I hope it won't affect my grade.
- I finish the part of bonus part, which is to count the kerboard interupt time. You can check the result through kernel output.

Environment

- The Test Environment is showing below:

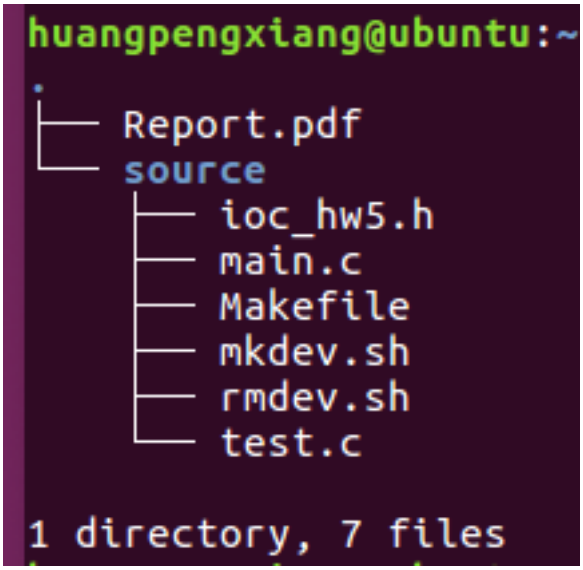


- The Gcc Version is showing below:



Running Guidance

The Tree of my program:



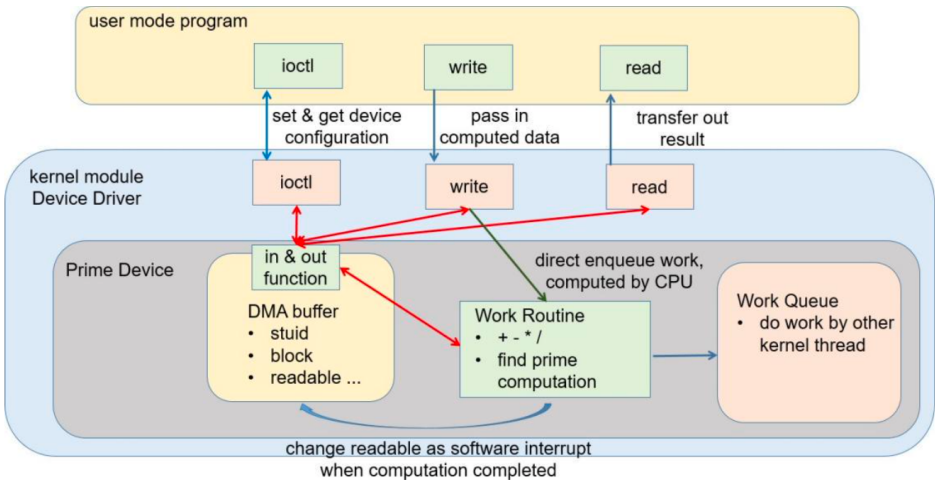
Detailed Steps to run my program

To run my program, basically you should follow the instruction below:

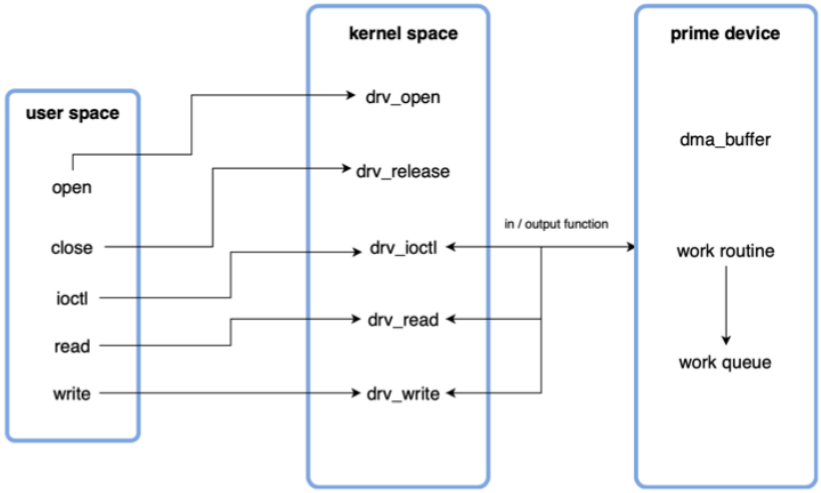
```
$ cd /* where the source located */
$ make // use this command to make
$ dmesg // use this command to found the device number
$ sudo sh mkdev.sh $1 $2  // build the file node, $1 represent the major while $2 represent minor
$ make clean // remove the module
$ sudo sh rmdev.sh // remove file node
```

Program Design

Basically, the global view of the program structure is showing below. There are user mode and kernel mode. In the user mode, the user program will firstly open the device using open operator, if the open failed, it will return an error. Then the user program will call the kernel program and pass the data to the kernel for further calculations. once the kernel finish the calculation, the user can use read operator to read the result. And the result will be shown in the terminal, also the kernel message will also be displayed by demesg command.



Source Design



The Design of this character device in kernel can be divided into 3 parts. The detailed information will be introduced below:

The first part is Initial the module. In this part, the kernel module will be initialized and the DMA buffer and work routine will be created at the same time. The module will bind the device and the file operations, and also create the major and minor number for the current device. you may use the command dmesg to check it. Moreover, the user mode can use the function `ioctl` to change the device configuration to implement different operations. There are 6 works of this function like Set the student ID, and check the RW, ioctl, IRQ, and set the blocking or non-blocking mode, confirm the readabel to make sure it can be read.

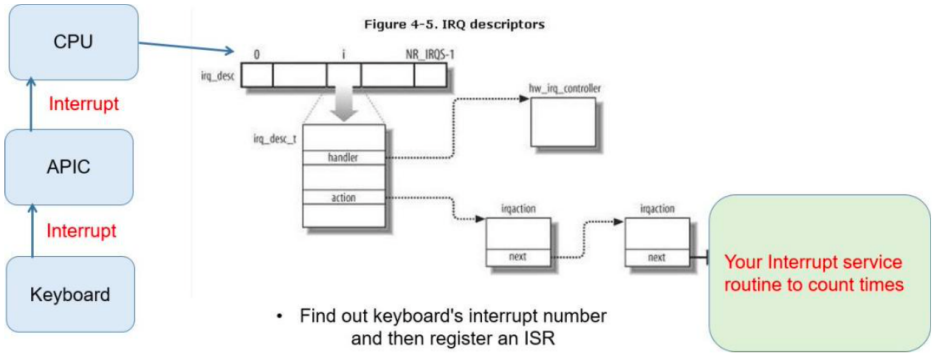
The second part is the calculation based on the input operator and operand. The kernel device will identify

the op code operand and pass it to kernel mode. Then the device will fetch those varible and to simulate the calculations based on the op code. And also, there is a workd routine, which is used to enqueue the task and processed those task by kernel thread. There are two difference way of process, blocking one and non-blocking one. the blocking one will simply wait the the computation is done then return the write operation. but for the non-blocking one, write will immediatly return once the queueing work. The programm will simply use a while loop to wait the kernel thread to finish the computation.

The last part is read the value and terminate the device. when read the value from the dma buffer, the result should be displayed in the terminal and also the original result and value should be clean, the readable variable should be set to false. when exit the module, the memory malloc in the initialize part should be clean.

Bouns Design

The basic flow for bonus is showing below. the bonus is ask us to implement the interrupt conter to count the interrupt like keyboard. To implement this, I design a request irq in module to add an ISR into an IRQ number's action list. when the interrupt occur, the number will plus one, which will implement the counter function. and when the kernel exit, the irq action will also be free to avoid cause kernel panic.



Progam Implementation

Source Implementation

when the initial function called, in the kernel mode, it will allocate the resource for the device and dma buffer.

```c
/* Register chrdev */
dev_t dev;
ret = alloc_chrdev_region(&dev, 0, 1, "mydev");
dev_major = MAJOR(dev);
dev_minor = MINOR(dev);

if(ret){
    printk(KERN_ALERT"Register chrdev failed!\n");
    return ret;
}else{
    printk("%s:%s():register chrdev(%d, %d)\n", PREFIX_TITLE, __func__, dev_major, dev_minor)
}

/* Init cdev and make it alive */
dev_cdev = cdev_alloc();
cdev_init(dev_cdev, &fops);
dev_cdev->ops = &fops;
dev_cdev->owner = THIS_MODULE;
ret = cdev_add(dev_cdev,dev,1);

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

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

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

return 0;
```

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 put the data into the DMA buffer.

```
switch (cmd) {
    case HW5_IOCSETSTUID:
        myouti(info, DMASTUIDADDR);
        printk("%s:%s(): My STUID is: %d\n", PREFIX_TITLE, __func__, info);
        break;
    case HW5_IOCSETRWOK:
        myouti(info, DMARWOKADDR);
        if(info == 1 ) printk("%s:%s(): RW OK\n", PREFIX_TITLE, __func__);
        break;
    case HW5_IOCSETIOCOK:
        myouti(info, DMAIOCOKADDR);
        if (info == 1) printk("%s:%s(): IOC OK\n", PREFIX_TITLE, __func__);
        break;
    case HW5_IOCSETIRQOK:
        myouti(info, DMAIRQOKADDR);
        if (info == 1) printk("%s:%s(): IRQ OK\n", PREFIX_TITLE, __func__);
        break;
    case 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__);
        break;
    case HW5_IOCWAITREADABLE:
        while (readable == 0){ // while not readable, wait till readable
            msleep(1000);
            readable = myini(DMAREADABLEADDR);
        }
        put_user(readable,(int *)arg);
        printk("%s:%s(): wait readable \n", PREFIX_TITLE, __func__);
        break;
    default:
        printk("no such operations \n");
        return -1;
```

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

```
/* get the input of operator and operand */
char op = myinc(DMAOPCODEADDR);
int oprand1 = myini(DMAOPERANDBADDR);
short oprand2 = myini(DMAOPERANDCADDR);

int ans;

/* calculate the answer */
switch (op) {
    case '+':
        ans= oprand1 + oprand2;
        break;
    case '-' :
        ans= oprand1 - oprand2;
        break;
    case '*':
        ans= oprand1 * oprand2;
        break;
    case '/':
        ans = oprand1 / oprand2;
        break;
    case 'p':
        ans = prime(oprand1, oprand2);
        break;
    default:
        ans = 0;
}

/* output the result */
myouti(ans,DMAANSADDR);
myouti(1, DMAREADABLEADDR); // set readable to true
```

Then the read function will be called to get the final answer, the variable should be cleaned after use.

```
static ssize_t drv_read(struct file *filp, char __user *buffer, size_t ss, loff_t* lo) {
    /* Implement read operation for your device */
    int result;
    result = myini(DMAANSADDR);
    put_user(result, (int*) buffer);
    printk("%s:%s(): ans = %d\n", PREFIX_TITLE, __func__, result);

    // reset the readable to false
    myouti(0, DMAREADABLEADDR);

    // clean the value
    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;
}
```

Bonus Implementation

Initialize the counter device.

```
    /* add the ISR to keyboard IRQ */
    interruptcount = 0;
    ret = request_irq(IRQ_NUM, (irq_handler_t) handler, IRQF_SHARED, "InterruptCount", keyBoardISR_devID);
    if (ret) {
        printk(KERN_ALERT"IRQ initialize failed\n");
        return ret;
    }
```

exit this module to avoid kernel panic.

```
static void __exit exit_modules(void) {
    /* Free IRQ */
    free_irq(IRQ_NUM, keyBoardISR_devID);
    printk("%s:%s():Interrupt count = %d\n", PREFIX_TITLE, __FUNCTION__, interruptcount);
```

The problem I have met

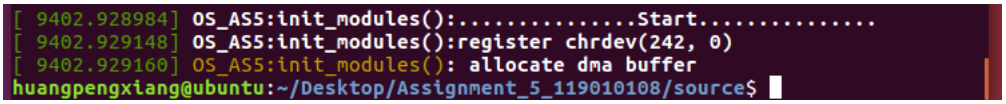
The most difficult part in this assignment is to understand the working flow, how user mode and kernel mode transfer data with each other? When should we change the readable number? and Where should we place the while loop in order to wait for computation complete? The solution is the user mode and the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

What I learned from this program

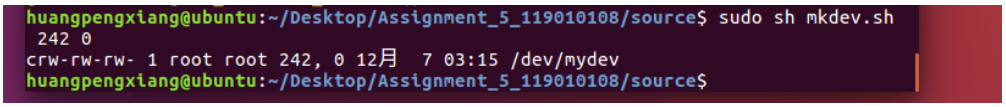
In this assignment, I clarified the interaction between the IO device and the operating system. They actually communicate through a interface called device driver. Without the device driver, the operating system cannot utilize the IO device. Besides, my understanding on the layer design of the operating system is deepened. I used to conceptually know that the operating system is divided into user and kernel spaces. The usage of functions get user and file operations helped me understand these to layers more.

Demo Output

The device major and minor:



command make mkmod



terminal output of testing the device

```
huangpengxiang@ubuntu:~/Desktop/Assignment_5_119010108/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..............
huangpengxiang@ubuntu:~/Desktop/Assignment_5_119010108/source$
```

Kernel message output:

```
2216.421401] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
2216.421493] OS_AS5:drv_read(): ans = 105019
2216.421870] OS_AS5:drv_ioctl(): Non-Blocking IO
2216.422026] OS_AS5:drv_write():queue work
2216.422026] OS_AS5,drv_write(): non-blocking
2216.977171] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
2217.537632] OS_AS5:drv_ioctl(): wait readable
2217.537671] OS_AS5:drv_read(): ans = 105019
2220.839359] OS_AS5:drv_ioctl(): Blocking IO
2220.839361] OS_AS5:drv_write():queue work
2220.839361] OS_AS5:drv_write():block
2223.248169] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
2223.248322] OS_AS5:drv_read(): ans = 225077
2223.248339] OS_AS5:drv_ioctl(): Non-Blocking IO
2223.248341] OS_AS5:drv_write():queue work
2223.248341] OS_AS5,drv_write(): non-blocking
2225.675563] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
2226.368961] OS_AS5:drv_ioctl(): wait readable
2226.369004] OS_AS5:drv_read(): ans = 225077
2226.369166] OS_AS5:drv_release(): device close
2259.036507] OS_AS5:exit_modules():Interrupt count = 90
2259.036509] OS_AS5:exit_modules(): free dma buffer
2259.036511] OS_AS5:exit_modules():unregister chrdev
2259.036511] OS_AS5:exit_modules():.............End.............
```