CSC3150 Project3 Report

118010045 Cui Yuncong

December 2020

Contents

1 Introduction

This program aims to make a prime device in Linux, and implement file operations in kernel module to control this device. The device is made under /dev by mknod command. File operations are implemented in a kernel module to control this device. ioctl() function is also implemented to change the device configuration. The device can Sulate registers by allocating a memory region. The program is implemented by C language on ubuntu. The user should input the test.c external file and the output will be the process of device control.

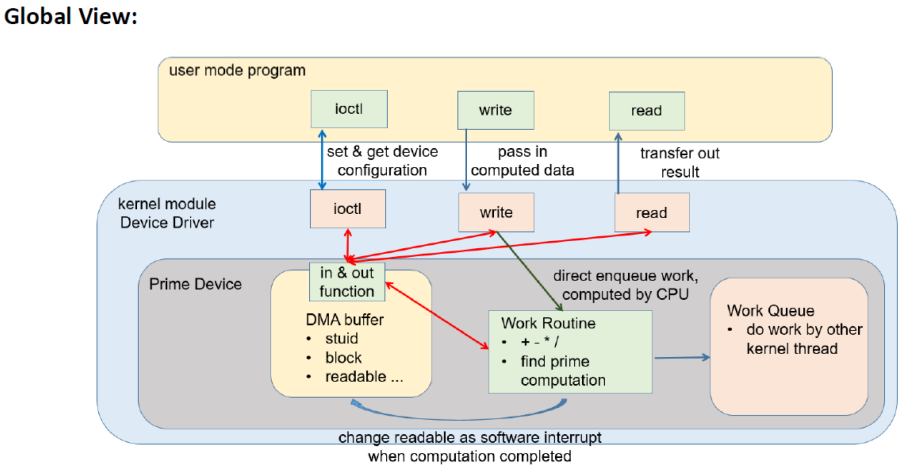

Figure 1: Structure

2 Design

2.1 Program Flow

The program consists of 6 parts:

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.

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

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.

4. The user program calls the arithmetic function, inside which the user program will firstly do the calculation itself and print the answer. 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 DMABLOCK-ADDR to the info number otherwise -1 and print the “set non-blocking/blocking failed”.

5. 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. 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.

6. 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. In the user mode, the program needs to wait until the computation complete by calling the ioctl() function, and then read the data.

2.2 Functions

This section describes the functions in the main.c.

1. static ssize_t drv_read()

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)
{
    if (myini(DMAREADABLEADDR) == 1)
    {
        printk("%s:%s(): ans = %i\n", PREFIX_TITLE, __func__, myini(DMAANSADDR));
        put_user(myini(DMAANSADDR), (int*)buffer);
        myouti(0, DMAREADABLEADDR);
        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(' ', DMAOPCODEADDR);
        myouti(0, DMAOPERANDCADDR);
        myouti(0, DMAOPERANDBADDR);
    }
    return 0;
}
```

Figure 2: static ssize_t drv_read() Function

2. static ssize_t drv_write()

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

Figure 3: static ssize_t drv_write() Function

3. static long drv_ioctl()

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.

```
static long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    int info, reads, ret;
    ret = get_user(info, (int*)arg);

    if (ret < 0)
        return ret;

    switch(cmd)
    {
        case HW5_IOCSETSTUID:
            myouti(info, DMASTUIDADDR);
            printk("%s:%s(): My STUID is = %i\n", PREFIX_TITLE, __func__, info);
            break;
        case HW5_IOCSETRWOK:
            if (info == 0 || info == 1)
            {
                printk("%s:%s(): RW OK\n", PREFIX_TITLE, __func__);
                myouti(info, DMARWOKADDR);
            }
            else
            {
                printk("%s:%s(): RW failed\n", PREFIX_TITLE, __func__);
                return -1;
            }
            break;
        case HW5_IOCSETIOCOK:
            if (info == 0 || info == 1)
            {
                myouti(info, DMAIOCOKADDR);
                printk("%s:%s(): IOC OK\n", PREFIX_TITLE, __func__);
            }
            else
            {
                printk("%s:%s(): IOC failed\n", PREFIX_TITLE, __func__);
                return -1;
            }
            break;
        case HW5_IOCSETIRQOK:
            if (info == 0 || info == 1)
            {
                myouti(info, DMAIRQOKADDR);
                printk("%s:%s(): IRQ OK\n", PREFIX_TITLE, __func__);
            }
            else
            {
                printk("%s:%s(): IRQ failed\n", PREFIX_TITLE, __func__);
                return -1;
            }
            break;
        case HW5_IOCSETBLOCK:
            if (info == 0 || info == 1)
                myouti(info, DMABLOCKADDR);
            else
                return -1;
            if (info)
                printk("%s:%s(): Blocking IO\n", PREFIX_TITLE, __func__);
            else
                printk("%s:%s(): Non-Blocking IO\n", PREFIX_TITLE, __func__);
            break;
        case HW5_IOCWAITREADABLE:
            reads = myini(DMAREADABLEADDR);
            while (!reads)
            {
                msleep(6000);
                reads = myini(DMAREADABLEADDR);
            }
            put_user(reads, (int *)arg);
            printk("%s:%s(): wait readable 1\n", PREFIX_TITLE, __func__);
            break;
        default:
            printk("%s:%s(): unknown error\n", PREFIX_TITLE, __func__);
    }
    return 0;
}
```

Figure 4: static long drv_ioctl() Function

4. prime()

The function is used to find nth prime number for the drv_arithmetic_routine function.

```
int prime(int base, short nth)
{
    int cnt=0;
    int i, n, tag;
    n = base;

    while(cnt < nth)
    {
        tag = 1;
        n++;
        if (n % 2)
        {
            for(i = 3; i * i <= n; i += 2)
                if(n % i == 0)
                {
                    tag = 0;
                    break;
                }
        }
        else
            tag = 0;
        cnt += tag;
    }
    return n;
}
```

Figure 5: prime() Function

5. static int __init init_modules()

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__);

    printk("%s:%s(): request_irq 1 return %i\n", PREFIX_TITLE, __func__,
        request_irq(1, irqcnt, IRQF_SHARED, "myinterrupts", irq_dev_id));

    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);

    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;
    }

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

    work = kmalloc(sizeof(typeof(*work)),GFP_KERNEL);
    return 0;
}
```

Figure 6: static int __init init_modules() Function

6. static void __exit exit_modules()

The function is used to free the DMA buffer and work routine. It will unregister and delete the character device.

```
static void __exit exit_modules(void)
{
    free_irq(1,irq_dev_id);
    printk("%s:%s(): interrupt count = %d\n", PREFIX_TITLE, __func__, interrupt_count);
    kfree(dma_buf);
    printk("%s:%s(): free dma buffer\n", PREFIX_TITLE, __func__);

    unregister_chrdev_region(MKDEV(dev_major,dev_minor), DEV_COUNT);
    cdev_del(dev_cdev);
    printk("%s:%s(): unregister chrdev\n", PREFIX_TITLE, __func__);
    kfree(work);
    printk("%s:%s():.............End.............\n", PREFIX_TITLE, __func__);
}
```

Figure 7: static void __exit exit_modules() Function

3 Problem and Solution

3.1 How User Mode and Kernel Mode Transfer Data with Each Other?

The user mode and the kernel mode will transfer the data with get_user() function

3.2 When Should the Readable Number Change?

The readable number should change 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.

3.3 Bonus

Add request_irq() at the beginning of the init_modules() function. In the exit_modules part, free the irq and print the count number.

4 Running Environment and Execution

Version of OS: Ubuntu 16.04.2

Kernel: Linux 4.10.14

```
[10/09/20]seed@VM:~$ sudo lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.2 LTS
Release:        16.04
Codename:       xenial
[10/09/20]seed@VM:~$ uname -srm
Linux 4.10.14 i686
```

Figure 8: Version of OS and Kernel

Enter into the source file folder.

Type "**`sudo su`**" to get the administration access.

Type "**`make`**" to build kernel module "mydev.ko" and insert kernel module.

Type "**`dmesg`**" to check the major and minor number.

Type "**`sudo ./mkdev.sh Major Minor`**" create a file node for "mydev" (the major and minor number should be given in the message part shown in the above step).

```
[12/04/20]seed@VM:~/.../source$ sudo su
root@VM:/home/seed/Desktop/source# make
make -C /lib/modules/`uname -r`/build M=`pwd` modules
make[1]: Entering directory '/home/seed/work/linux-4.10.14'
  CC [M]  /home/seed/Desktop/source/main.o
  LD [M]  /home/seed/Desktop/source/mydev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Desktop/source/mydev.mod.o
  LD [M]  /home/seed/Desktop/source/mydev.ko
make[1]: Leaving directory '/home/seed/work/linux-4.10.14'
sudo insmod mydev.ko
gcc -o test test.c
root@VM:/home/seed/Desktop/source# ./mkdev.sh 245 0
crw-rw-rw- 1 root root 245, 0 Dec  4 23:41 /dev/mydev
```

Figure 9: Build Kernel Module and Create a File Node

Type "**`./test`**" to run test.

Type "**`make clean`**" to remove kernel module, clean test executable file and display kernel message including "OS_AS5".

Type "**`sudo ./rmdev.sh`**" to remove this file node.

```
root@VM:/home/seed/Desktop/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@VM:/home/seed/Desktop/source# make clean
make -C /lib/modules/`uname -r`/build M=`pwd` clean
make[1]: Entering directory '/home/seed/work/linux-4.10.14'
  CLEAN   /home/seed/Desktop/source/.tmp_versions
  CLEAN   /home/seed/Desktop/source/Module.symvers
make[1]: Leaving directory '/home/seed/work/linux-4.10.14'
sudo rmmod mydev
rm test
dmesg | grep OS_AS5
[   89.342680] OS_AS5:init_modules():..............Start..............
[   89.342685] OS_AS5:init_modules(): request_irq 1 return 0
[   89.342686] OS_AS5:init_modules(): register chrdev(245,0)
[   89.342687] OS_AS5:init_modules(): allocate dma buffer
[  119.604706] OS_AS5:drv_open(): device open
[  119.604709] OS_AS5:drv_ioctl(): My STUID is = 118010045
[  119.604709] OS_AS5:drv_ioctl(): RW OK
[  119.604710] OS_AS5:drv_ioctl(): IOC OK
[  119.604710] OS_AS5:drv_ioctl(): IRQ OK
[  120.280831] OS_AS5:drv_ioctl(): Blocking IO
[  120.280833] OS_AS5:drv_write(): queue work
[  120.280833] OS_AS5:drv_write(): block
[  120.284437] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[  120.284443] OS_AS5:drv_read(): ans = 105019
[  120.284468] OS_AS5:drv_ioctl(): Non-Blocking IO
[  120.284473] OS_AS5:drv_write(): queue work
[  120.284473] OS_AS5:drv_write(): non-blocking
[  120.288279] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[  126.313535] OS_AS5:drv_ioctl(): wait readable 1
[  126.313601] OS_AS5:drv_read(): ans = 105019
[  126.313952] OS_AS5:drv_release(): device close
[  133.484890] OS_AS5:exit_modules(): interrupt count = 138
[  133.484891] OS_AS5:exit_modules(): free dma buffer
[  133.484893] OS_AS5:exit_modules(): unregister chrdev
[  133.484893] OS_AS5:exit_modules():..............End..............
root@VM:/home/seed/Desktop/source# ./rmdev.sh
ls: cannot access '/dev/mydev': No such file or directory
root@VM:/home/seed/Desktop/source#
```

Figure 10: Test and Display Kernel Message

5 Learning Remarks

5.1 User Mode and Kernel Mode

Use the get_user() function to transfer the data from user mode to the kernel mode.

5.2 Data and DMA Buffer

Use the myouti() and myoutc() functions to store the data in the correspongding DMA places. Use the myini() and myinc() functions get data from DMA buffer.

5.3 Blocking and Non-Blocking Modes

In the non-blocking mode, CPU will not wait for I/O but continue work. In the blocking modes, the user program will wait for the kernel to do the I/O instruction and then continue. 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.

6 Other Tests

6.1 Add Numbers Test

```
root@VM:/home/seed/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

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

Figure 11: Add Numbers Test

```
[ 3880.610934] OS_AS5:init_modules():...............Start...............
[ 3880.610938] OS_AS5:init_modules(): request_irq 1 return 0
[ 3880.610940] OS_AS5:init_modules(): register chrdev(245,0)
[ 3880.610941] OS_AS5:init_modules(): allocate dma buffer
[ 3897.957672] OS_AS5:drv_open(): device open
[ 3897.957675] OS_AS5:drv_ioctl(): My STUID is = 118010045
[ 3897.957676] OS_AS5:drv_ioctl(): RW OK
[ 3897.957676] OS_AS5:drv_ioctl(): IOC OK
[ 3897.957677] OS_AS5:drv_ioctl(): IRQ OK
[ 3897.957686] OS_AS5:drv_ioctl(): Blocking IO
[ 3897.957688] OS_AS5:drv_write(): queue work
[ 3897.957688] OS_AS5:drv_write(): block
[ 3897.957695] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[ 3897.957704] OS_AS5:drv_read(): ans = 110
[ 3897.957724] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 3897.957726] OS_AS5:drv_write(): queue work
[ 3897.957727] OS_AS5:drv_write(): non-blocking
[ 3897.957732] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[ 3904.008955] OS_AS5:drv_ioctl(): wait readable 1
[ 3904.008981] OS_AS5:drv_read(): ans = 110
[ 3904.009140] OS_AS5:drv_release(): device close
[ 3950.657721] OS_AS5:exit_modules(): interrupt count = 104
[ 3950.657722] OS_AS5:exit_modules(): free dma buffer
[ 3950.657724] OS_AS5:exit_modules(): unregister chrdev
[ 3950.657724] OS_AS5:exit_modules():...............End.............
```

Figure 12: Add Numbers Test

6.2 Minus Numbers Test

```
root@VM:/home/seed/Desktop/source# ./test
..............Start..............
100 - 10 = 90

Blocking IO
ans=90 ret=90

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

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

Figure 13: Minus Numbers Test

```
[ 3984.264605] OS_AS5:init_modules():..............Start..............
[ 3984.264609] OS_AS5:init_modules(): request_irq 1 return 0
[ 3984.264610] OS_AS5:init_modules(): register chrdev(245,0)
[ 3984.264611] OS_AS5:init_modules(): allocate dma buffer
[ 3988.456705] OS_AS5:drv_open(): device open
[ 3988.456710] OS_AS5:drv_ioctl(): My STUID is = 118010045
[ 3988.456711] OS_AS5:drv_ioctl(): RW OK
[ 3988.456712] OS_AS5:drv_ioctl(): IOC OK
[ 3988.456713] OS_AS5:drv_ioctl(): IRQ OK
[ 3988.456727] OS_AS5:drv_ioctl(): Blocking IO
[ 3988.456730] OS_AS5:drv_write(): queue work
[ 3988.456730] OS_AS5:drv_write(): block
[ 3988.456741] OS_AS5:drv_arithmetic_routine(): 100 - 10 = 90
[ 3988.456748] OS_AS5:drv_read(): ans = 90
[ 3988.456754] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 3988.456757] OS_AS5:drv_write(): queue work
[ 3988.456758] OS_AS5:drv_write(): non-blocking
[ 3988.456762] OS_AS5:drv_arithmetic_routine(): 100 - 10 = 90
[ 3988.456767] OS_AS5:drv_ioctl(): wait readable 1
[ 3988.456770] OS_AS5:drv_read(): ans = 90
[ 3988.456957] OS_AS5:drv_release(): device close
[ 4328.230491] OS_AS5:exit_modules(): interrupt count = 112
[ 4328.230492] OS_AS5:exit_modules(): free dma buffer
[ 4328.230493] OS_AS5:exit_modules(): unregister chrdev
[ 4328.230494] OS_AS5:exit_modules():..............End..............
```

Figure 14: Minus Numbers Test

6.3 Multiple Numbers Test

```
root@VM:/home/seed/Desktop/source# ./test
..............Start..............
100 * 10 = 1000

Blocking IO
ans=1000 ret=1000

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

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

Figure 15: Multiple Numbers Test

```
[ 4380.394394] OS_AS5:init_modules():..............Start..............
[ 4380.394399] OS_AS5:init_modules(): request_irq 1 return 0
[ 4380.394400] OS_AS5:init_modules(): register chrdev(245,0)
[ 4380.394401] OS_AS5:init_modules(): allocate dma buffer
[ 4382.071051] OS_AS5:drv_open(): device open
[ 4382.071057] OS_AS5:drv_ioctl(): My STUID is = 118010045
[ 4382.071058] OS_AS5:drv_ioctl(): RW OK
[ 4382.071059] OS_AS5:drv_ioctl(): IOC OK
[ 4382.071060] OS_AS5:drv_ioctl(): IRQ OK
[ 4382.071080] OS_AS5:drv_ioctl(): Blocking IO
[ 4382.071082] OS_AS5:drv_write(): queue work
[ 4382.071083] OS_AS5:drv_write(): block
[ 4382.071094] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[ 4382.071101] OS_AS5:drv_read(): ans = 1000
[ 4382.071108] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 4382.071111] OS_AS5:drv_write(): queue work
[ 4382.071112] OS_AS5:drv_write(): non-blocking
[ 4382.071119] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[ 4388.119125] OS_AS5:drv_ioctl(): wait readable 1
[ 4388.119201] OS_AS5:drv_read(): ans = 1000
[ 4388.119519] OS_AS5:drv_release(): device close
[ 4393.224818] OS_AS5:exit_modules(): interrupt count = 58
[ 4393.224819] OS_AS5:exit_modules(): free dma buffer
[ 4393.224821] OS_AS5:exit_modules(): unregister chrdev
[ 4393.224822] OS_AS5:exit_modules():..............End..............
```

Figure 16: Multiple Numbers Test

6.4 Division Numbers Test

```
root@VM:/home/seed/Desktop/source# ./test
...............Start...............
100 / 10 = 10

Blocking IO
ans=10 ret=10

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

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

Figure 17: Division Numbers Test

```
[ 4396.900671] OS_AS5:init_modules():...............Start...............
[ 4396.900675] OS_AS5:init_modules(): request_irq 1 return 0
[ 4396.900676] OS_AS5:init_modules(): register chrdev(245,0)
[ 4396.900679] OS_AS5:init_modules(): allocate dma buffer
[ 4399.622403] OS_AS5:drv_open(): device open
[ 4399.622405] OS_AS5:drv_ioctl(): My STUID is = 118010045
[ 4399.622406] OS_AS5:drv_ioctl(): RW OK
[ 4399.622406] OS_AS5:drv_ioctl(): IOC OK
[ 4399.622407] OS_AS5:drv_ioctl(): IRQ OK
[ 4399.622418] OS_AS5:drv_ioctl(): Blocking IO
[ 4399.622419] OS_AS5:drv_write(): queue work
[ 4399.622419] OS_AS5:drv_write(): block
[ 4399.622425] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[ 4399.622428] OS_AS5:drv_read(): ans = 10
[ 4399.622432] OS_AS5:drv_ioctl(): Non-Blocking IO
[ 4399.622433] OS_AS5:drv_write(): queue work
[ 4399.622433] OS_AS5:drv_write(): non-blocking
[ 4399.622437] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[ 4405.774018] OS_AS5:drv_ioctl(): wait readable 1
[ 4405.774052] OS_AS5:drv_read(): ans = 10
[ 4405.774252] OS_AS5:drv_release(): device close
[ 4445.122732] OS_AS5:exit_modules(): interrupt count = 62
[ 4445.122734] OS_AS5:exit_modules(): free dma buffer
[ 4445.122735] OS_AS5:exit_modules(): unregister chrdev
[ 4445.122736] OS_AS5:exit_modules():...............End.............
```

Figure 18: Division Numbers Test