1. Environment:
   a) version of OS: Ubuntu 16.04.2
   b) version of kernel: 4.10.14

2. Design of the program:
   a) Basic ideas

      This project will create a device which can do simple arithmetic operations and find n-th prime number. The device will be created in the "/dev" folder using "mknod" command. Also, the project will implement some basic operations to the device in the kernel which can responds to the function calls in the user space.

   b) init_modules(void)

      The function will execute the following operations step by step to set up the device:

      i. add an interrupt service routine(ISR) to the interrupt request(IRQ). For the ISR, a function called "record_interrrupt" is used to count the interrupt times of the keyboard.
      ii. allocate device numbers for the device using "alloc_chrdev_region" function.
      iii. create a character device using "cdev_alloc" function. Initialize it with its file operations using "cdev_init". Then, the device will be bound to the device number generated in the second step using "cdev_add".
      iv. allocate memory to simulate the dma_buffer using "kmalloc".
      v. allocate memory for work_routine using "kmalloc".

   c) drv_read(struct file *filp, char __user *buffer, size_t ss, loff_t* lo):

      First the program will check out whether the result is readable or not.

      1. If it is readable, the program would read the data to the user buffer. After that, it will clean the result and set the result to be non-readable.

2. If it is not readable, the program would print out the error information in the logs.

d) drv_write(struct file *filp, const char __user *buffer, size_t ss, loff_t* lo) :

   i. First, it will read the data from the user using "get_user" and put in the dma buffer.

   ii. Then it will initialize a work_routine and schedule it according to the IO mode:

      a) If the IO mode is blocking, then it will first schedule the work_routine and then flush it out. After that, it will set the result to be readable.

      b) If the IO mode is non_blocking, then it will first schedule the work_routine. Then it will set the answer to be non-readable.

e) drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg):

   The function will check the input command from user and do the corresponding operations in the kernel space.

   i. If "cmd" is HW5_IOCSETSTUID:
   Store the data passed from the user space in "arg" to the DMA buffer indicated by DMASTUIDADDR and print out the operation.

   ii. If "cmd" is HW5_IOCSETRWOK:
   Store the data passed from the user space in "arg" to the DMA buffer indicated by DMARWOKADDR and print out the operation.

   iii. If "cmd" is HW5_ IOCSETIOCOK:
   Store the data passed from the user space in "arg" to the DMA buffer indicated by DMAIOCOKADDR and print out the operation.

   iv. If "cmd" is HW5_IOCSETIRQOK:
   Store the data passed from the user space in "arg" to the DMA buffer

indicated by DMAIRQOKADDR and print out the operation.

v. If “cmd” is HW5_IOCSETBLOCK:

   Store the data passed from the user space in “arg” to the DMA buffer indicated by DMABLOCKADDR and print out the blocking type according to the user input.

vi. If “cmd” is HW5_IOCWAITREADABLE:

   First, it will check out the readable signal in the DMA which is indicated by DMAREADABLEADDR. If the readable signal 0 which means the answer is not ready, then it will wait until the work routine is finished. After that, it will return the readable signal 1 to the user.

vii. If “cmd” is not one of the previous signals:

   The function will print out the error.

f) drv_arithmetic_routine(struct work_struct* ws):

   This function is used to do the arithmetic calculation in the work routine.

   i. First, it will set the readable signal to 0 indicating false.
   ii. Second, it will retrieve the data stored in the DMA memory.
   iii. Third, it will do the corresponding operations of the data according to the operator character. And the calculated result will be store in the DMA buffer indicated by DMAANSADDR.
   iv. Then it will check the I/O mode in the DMA and set the readable to 1 if it is in non-blocking mode.

g) prime(int base, short nth):

   This function is just the one used in the test.c. It is used to find out the n^{th} prime number.

h) record_interrupt(int irq, void* dev_id):

   This function is used to record the interrupt count. If the keyboard interrupt occurs, it will increment the count by 1.

i) exit_modules(void):

   This function will be called when the module is removed from the kernel. And it will do the following steps by order:

   i. free the interrupt service routine using “free_irq”
   ii. free the DMA memory using “kfree”
   iii. delete the character device using “cdev_del”. Then it will unregister the device numbers allocated for the device.
   iv. free the work routine.

3. Execution steps of the program：

   a) open the terminal in the “Source” folder.
   b) enter the command “sudo make” in the terminal
   c) type the “dmesg” to get the allocated device number
   d) use “sudo ./mkdev.sh (major) (minor)” to create the device to create a file system node. The (major) and (minor) in the command are the device numbers shown in the last step.
   e) if the node already exists, you can type “sudo ./rmdev.sh” to remove the existed node and created it again with last step.
   f) type “./test” in the terminal to run the test program.
   g) type “sudo make clean” to remove the module.
   h) type “sudo ./rmdev.sh” to remove the created system node.

4. Output:

   a) user program:

```
[12/06/20]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...............
```

b) kernel output:

```
[  344.586140] OS_AS5:init_modules():..............Start..............
[  344.586148] OS_AS5:init_modules(): request_irq 1 return 0
[  344.586150] OS_AS5:init_modules(): register chrdev(245,0)
[  344.586152] OS_AS5:init_modules(): allocate dma buffer
[  377.327567] OS_AS5:drv_open(): device open
[  377.327571] OS_AS5:drv_ioctl(): My STUID is = 118010224
[  377.327572] OS_AS5:drv_ioctl(): RW OK
[  377.327573] OS_AS5:drv_ioctl(): IOC OK
[  377.327574] OS_AS5:drv_ioctl(): IRQ OK
[  378.410424] OS_AS5:drv_ioctl(): Blocking IO
[  378.410427] OS_AS5:drv_write(): queue work
[  378.410428] OS_AS5:drv_write(): block
[  379.104313] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[  379.106820] OS_AS5:drv_read(): ans = 105019
[  379.106830] OS_AS5:drv_ioctl(): Non-Blocking IO
[  379.106832] OS_AS5:drv_write(): queue work
[  379.797744] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[  384.231228] OS_AS5:drv_ioctl(): wait readable 1
[  384.231249] OS_AS5:drv_read(): ans = 105019
[  384.231530] OS_AS5:drv_release(): device close
[  396.259526] OS_AS5:exit_modules(): interrupt count=128
[  396.259530] OS_AS5:exit_modules(): free dma buffer
[  396.259531] OS_AS5:exit_modules(): unregister chrdev
[  396.259532] OS_AS5:exit_modules():.............End.............
[12/06/20]seed@VM:~/.../Source$
```

5. Problems I met in this project:

Since it is the first time for me to program with the I/O operation, I didn't know where to start this project at first. Then after reading and studying the tutorials' materials and lecture notes, I got to know the basic knowledge of this program and understand the basic framework of this project. Finally, I managed to finish the project.

6. What I learnt from this project:

   First, I learnt some basic knowledge of the I/O system. By implementing a simple prime number device, I learnt how to allocate device numbers and set up and initialize devices. Also, I also learnt the basic knowledge of the interrupt routine and how to set the handler when the interrupt occurs.