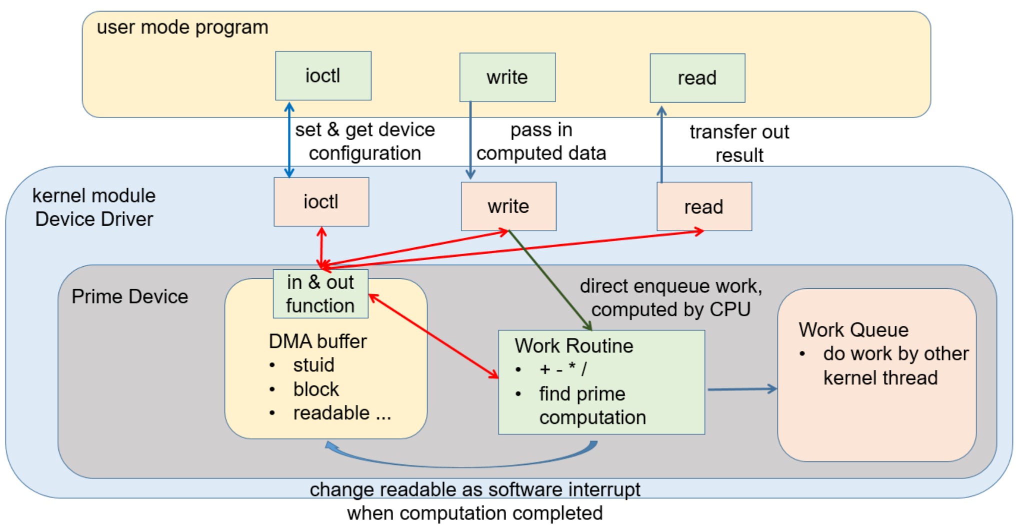
CSC3150 Assignment 5

In Assignment 5, you are required to make a prime device in Linux, and implement file operations in kernel module to control this device.

Outline:

- We will make a device under /dev by mknod command.
- This device can find n-th prime number.
- You will implement file operations in a kernel module to control this device.
- And implement ioctl function to change the device configuration.
- Simulate registers on device by allocating a memory region.

Global View:



Specification:

- **Register character device and make it live:**
  - You can use alloc_chrdev_region() to allocate a range of char device numbers.
  - And get available number by MAJOR() and MINOR() macro.
  - In your kernel module, you could allocate a cdev structure by cdev_alloc() and initialize it by cdev_init() to bind cdev file_operations.
  - Add a device by cdev_add() to make it live.

- **Test program and printk:**
  - Before write module, we need to know what this module do. So we provide a test program to test this device.
  - You can modify test program and test your module.
  - We will check with our test cases.
  - In kernel module, you need to write printk to help debug and use dmesg command to show message.
  - To help demo program, your printk must be started with "OS_AS5: function_name: message".

```
OS_AS5:init_modules():..............Start..............
OS_AS5:init_modules(): requset_irq 1 return 0
OS_AS5:init_modules(): register chrdev(245,0)
OS_AS5:init_modules(): allocate dma buffer
OS_AS5:drv_open(): device open
```

- **File operations:**
  - You should write a struct **file_ operations** to map the operations to functions in this module.
  - And use cdev_init() at module init to bind cdev and file_ operations.

```
static struct file_operations fops = {
        owner: THIS_MODULE,
        read: drv_read,
        write: drv_write,
        unlocked_ioctl: drv_ioctl,
        open: drv_open,
        release: drv_release,
};
```

- **ioctl:**
  - In Linux, device provide user mode program ioctl function to change the device configuration.
  - ioctl define many types of operation with switch case to do coordinated work.
  - And ioctl use mask to get value from these operation label.

- Here we provide "ioc_hw5.h" to define 6 works.
    1) (HW5_IOC_SETSTUID) Set student ID: printk your student ID
    2) (HW5_IOCSETRWOK) Set if RW OK: printk OK if you complete R/W function
    3) (HW5_IOCSETIOCOK) Set if ioctl OK: printk OK if you complete ioctl function
    4) (HW5_IOCSETIRQOK) Set if IRQ OK: printk OK if you complete bonus
    5) (HW5_IOCSETBLOCK) Set blocking or non-blocking: set write function mode
    6) (HW5_IOCWAITREADABLE) Wait if readable now (synchronize function): used before read to confirm it can read answer now when use non-blocking write mode.
- ioctl lables defined in "ioc_hw5.h"
  "_IOW(type, nr, size)" is used for an ioctl to write data to the driver. It is to generate command numbers.

```
#ifndef IOC_HW5_H
#define IOC_HW5_H

#define HW5_IOC_MAGIC          'k'
#define HW5_IOCSETSTUID        _IOW(HW5_IOC_MAGIC, 1, int)
#define HW5_IOCSETRWOK         _IOW(HW5_IOC_MAGIC, 2, int)
#define HW5_IOCSETIOCOK        _IOW(HW5_IOC_MAGIC, 3, int)
#define HW5_IOCSETIRQOK        _IOW(HW5_IOC_MAGIC, 4, int)
#define HW5_IOCSETBLOCK        _IOW(HW5_IOC_MAGIC, 5, int)
#define HW5_IOCWAITREADABLE    _IOR(HW5_IOC_MAGIC, 6, int)
#define HW5_IOC_MAXNR          6

#endif
```

- Demo for ioctl call in user mode:

  ioctl(fd, HW5_IOCSETBLOCK, &ret)

    - fd: an open file descriptor
    - HW5_IOCSETBLOCK: device-dependent request code.
    - &ret: an untyped pointer to memory

- **write:**
    - Define a data struct that is passed in write function.

```
struct dataIn {
    char a;
    int b;
    short c;
};
```

    - a is operator: '+', '-', '*', '/', or 'p' ('p' means find prime number)
    - b is operand 1
    - c is operand 2
    - Use INIT_WORK() and schedule_work() to queue work to system queue.

- **Find Prime operation:**
  - It finds c-th prime number bigger than b.
    (e.g, “1 p 3” means to find 3rd prime number which is bigger than 1, then it should be 5.)
  - And you will feel the I/O latency when execute test program for “100 p 10000” and “100 p 20000”.
  - We will check your blocking and non-blocking IO by observing the delay of the message printed by test program.
  - R/W function packaged in arithmetic function in user mode program.

    arithmetic(fd, 'p', 100, 10000);

    - fd: an open file descriptor
    - p: operator
    - 100: operand1
    - 10000: operand2

- **Work Routine:**
  - The work you enqueued should be written in a work routine function in module.
  - These work will be processed by another kernel thread.
  - computation is written in a work routine in module

```
// Arithmetic funciton
static void drv_arithmetic_routine(struct work_struct* ws);
```

- **Blocking and Non-Blocking IO:**
  - The test program can use ioctl to set blocking or non-blocking.
  - Your write function in module can be blocking or non-blocking.
  - Blocking write need to wait computation completed.
  - Non-blocking write just return after queueing work.
  - Read function only has blocking, because not queueing work.

- **Blocking Write:**
  - In test program, we just need a write function.
  - Do not need another synchronize function.
  - But block when writing.

- Blocking write in test program:

```
/*****************Blocking IO*****************/
printf("Blocking IO\n");
ret = 1;
if (ioctl(fd, HW5_IOCSETBLOCK, &ret) < 0) {
    printf("set blocking failed\n");
    return -1;
}

write(fd, &data, sizeof(data));

//Do not need to synchronize
//But need to wait computation completed

read(fd, &ret, sizeof(int));

printf("ans=%d ret=%d\n\n", ans, ret);
/***********************************************/
```

- **Non- Blocking Write:**
  - In test program, we can do something after write function.
  - Write function return after queueing work, it is non-blocking.
  - But need another synchronize function to wait work completed.
  - Non-blocking write in test program:

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

- **Interrupt driven IO:**
  - When implementing blocking write and synchronize function, they use a while loop busy waiting the interrupt.
  - You can use a variable to simulate the interrupt.
  - At the final of the work routine function, change this variable as triggering the interrupt.
  - And then, blocking write and synchronize function can exit the while loop.

- **DMA Buffer:**

  - To simulate register and memory on device, you need to kmalloc a dma buffer.
  - This buffer is as I/O port mapping in main memory.
  - What device do is written in work routine function. This function get data from this buffer.
  - Defined value written into dma buffer:

```
// DMA
#define DMA_BUFSIZE 64
#define DMASTUIDADDR 0x0          // Student ID
#define DMARWOKADDR 0x4           // RW function complete
#define DMAIOCOKADDR 0x8          // ioctl function complete
#define DMAIRQOKADDR 0xc          // ISR function complete
#define DMACOUNTADDR 0x10         // interrupt count function complete
#define DMAANSADDR 0x14           // Computation answer
#define DMAREADABLEADDR 0x18      // READABLE variable for synchronize
#define DMABLOCKADDR 0x1c         // Blocking or non-blocking IO
#define DMAOPCODEADDR 0x20        // data.a opcode
#define DMAOPERANDBADDR 0x21      // data.b operand1
#define DMAOPERANDCADDR 0x25      // data.c operand2
void *dma_buf;
```

- **In and out functions:**

  - You need to implement in & out function to access dma buffer just like physical device.
  - out function is used to output data to dma buffer.
  - in function is used to input data from dma buffer.
  - The 6 in & out functions are definded in module to operate dma_buf:
    ('c', 's' and 'i' maps with data type ‘char’, ‘short’ and ‘int’)

```
// 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);
```

  - Demo usage of in and out functions:

    myouti(value, DMAIOCOKADDR)

    - value: data you want to write into dma_buffer
    - DMAIOCOKADDR: port in dma_buffer

- **Data transfer between kernel and user space:**
  - get_user(x, ptr)
    - Get a simple variable from user space.
    - x: Variable to store result.
    - ptr: Source address, in user space.
  - put_user(x, ptr)
    - Write a simple value into user space.
    - x: Value to copy to user space.
    - ptr: Destination address, in user space.

Template structure:

- Makefile is provided:
  - Command: **make**
    (It will firstly build your main.c as kernel module "mydev.ko", insert "mydev.ko", and then build "test.c" as executable file "test".)

```
mname := mydev
$(mname)-objs := main.o
obj-m := $(mname).o

KERNELDIR := /lib/modules/`uname -r`/build

all:
	$(MAKE) -C $(KERNELDIR) M=`pwd` modules
	sudo insmod mydev.ko
	gcc -o test test.c
```

  - Command: **make clean**
    (It will remove "mydev.ko" and use "dmesg" to list kernel logs that includes keyword "OS_AS5")

```
clean:
	$(MAKE) -C $(KERNELDIR) M=`pwd` clean
	sudo rmmod mydev
	rm test
	dmesg | grep OS_AS5
```

- "mknod" script is provided:
  - Script

```
#!/bin/bash
mknod /dev/mydev c $1 $2
chmod 666 /dev/mydev
ls -l /dev/mydev
```

  - In mknod command: c means character device. Followed two number are Major and Minor number to specify device.
  - You can get available number by MAJOR() and MINOR() macro after alloc_chrdev_region() in module_init() function.
  - Demo of how to use mknod script: (Refer to Tutorial_11 Slide 3 to 6)

```
[10182.366961] Tutorial_11:init_modules():.............Start.............
[10182.366962] Tutorial_11:init_modules():register chrdev(241,0)
[10191.485302] Tutorial_11:exit_modules():.............End.............
[11/27/18]seed@VM:~/.../Check_Device_Num$ sudo ./mkdev.sh 241 0
crw-rw-rw- 1 root root 241, 0 Nov 27 00:24 /dev/mydev
```

- Steps you need to run the template:
  - Run “make”
  - Run “dmesg” to check available device number
  - Run “sudo ./mkdev.sh MAJOR MINOR” to build file node (MAJOR and MINOR are the available device number checked from previous step)
  - Run “./test” to start testing
  - Run “make clean” to remove the module and check the messages
  - Run “sudo ./rmdev.sh” to remove the file node
  - You will get output:

```
[11/28/18]seed@VM:~/.../source$ ./test
...............Start...............
can't open device!
```

```
[24715.172801] OS_AS5:init_modules():...............Start..............
[24762.366586] OS_AS5:exit_modules():.............End.............
[11/28/18]seed@VM:~/.../source$
```

- You should complete module init and exit functions:

```c
static int __init init_modules(void) {

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

	/* Register chrdev */

	/* Init cdev and make it alive */

	/* Allocate DMA buffer */

	/* Allocate work routine */

	return 0;
}

static void __exit exit_modules(void) {

	/* Free DMA buffer when exit modules */

	/* Delete character device */

	/* Free work routine */


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

- Implement read/write/ioctl operations and arithmetic routine:

```c
static ssize_t drv_read(struct file *filp, char __user *buffer, size_t ss, loff_t* lo) {
	/* Implement read operation for your device */
	return 0;
}
static ssize_t drv_write(struct file *filp, const char __user *buffer, size_t ss, loff_t* lo) {
	/* Implement write operation for your device */
	return 0;
}
static long drv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg) {
	/* Implement ioctl setting for your device */
	return 0;
}

static void drv_arithmetic_routine(struct work_struct* ws) {
	/* Implement arthemetic routine */
}
```

Function Requirements (90 points):

- Register a character device when module initialized. (5 points)
- Initialized a cdev and add it to make it alive. (5 points)
- Allocate DMA buffer. (5 points)
- Allocate work routine. (5 points)
- Implement read operation for your device. (10 points)
- Implement write operation for your device. (20 points)
- Implement ioctl setting for your device. (15 points)
- Implement arithmetic routine for your device. (10 points)
- Complete module exit functions. (5 points)
- Update your student ID in test case and make it be print in kernel ioctl. (5 points)
- Run test cases to check write and read operations. (5 points)

Demo Output:

- Test case: (: '+', '-', '*', '/' is for your testing, we will mainly test 'p' operation)

```
//arithmetic(fd, '+', 100, 10);
//arithmetic(fd, '-', 100, 10);
//arithmetic(fd, '*', 100, 10);
//arithmetic(fd, '/', 100, 10);
arithmetic(fd, 'p', 100, 10000);
//arithmetic(fd, 'p', 100, 20000);
```

- User mode output:

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

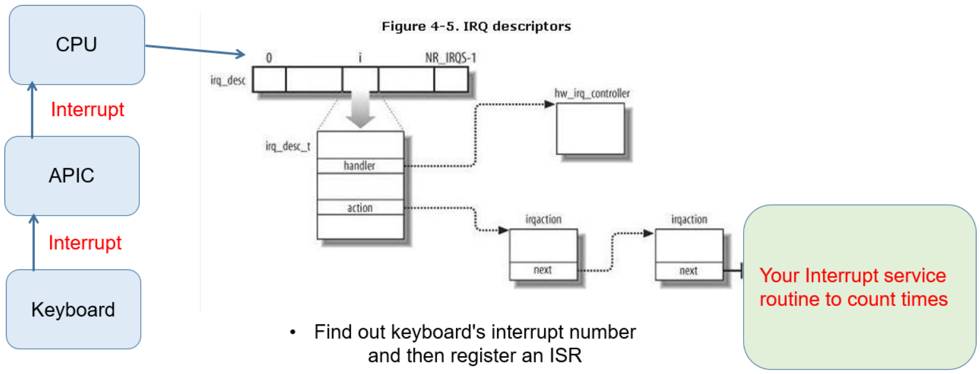
- Kernel mode output:

```
[15311.490581] OS_AS5:init_modules():..............Start..............
[15311.490586] OS_AS5:init_modules(): register chrdev(245,0)
[15311.490587] OS_AS5:init_modules(): allocate dma buffer
[15315.133727] OS_AS5:drv_open(): device open
[15315.133730] OS_AS5:drv_ioctl(): My STUID is = 123456789
[15315.133730] OS_AS5:drv_ioctl(): RW OK
[15315.133731] OS_AS5:drv_ioctl(): IOC OK
[15316.140030] OS_AS5:drv_ioctl(): Blocking IO
[15316.140032] OS_AS5:drv_write(): queue work
[15316.140033] OS_AS5:drv_write(): block
[15316.773221] OS_AS5:drv_arithmetic_routine():100 p 10000 = 105019
[15316.775273] OS_AS5:drv_read(): ans = 105019
[15316.775284] OS_AS5:drv_ioctl(): Non-Blocking IO
[15316.775285] OS_AS5:drv_write(): queue work
[15317.411242] OS_AS5:drv_arithmetic_routine():100 p 10000 = 105019
[15317.779631] OS_AS5:drv_ioctl(): wait readable 1
[15317.779669] OS_AS5:drv_read(): ans = 105019
[15317.780042] OS_AS5:drv_release(): device close
[15323.564774] OS_AS5:exit_modules(): free dma buffer
[15323.564775] OS_AS5:exit_modules(): unregister chrdev
[15323.564776] OS_AS5:exit_modules():.............End.............
```

- Steps for internal executions:
  - find major and minor number
  - allocate DMA buffer
  - ioctl print set and get value
  - write to queue work
  - arithmetic routine to compute answer
  - read to get answer
  - free DMA buffer
  - unregister device

Bonus (10 points)

- Global View (Bonus)



- Count the interrupt times of input device like keyboard.
- Hint: watch -n 1 cat /proc/interrupts
- Use request_irq() in module_init to add an ISR into an IRQ number's action list.
- And free_irq() when module_ exit, otherwise kernel panic.
- Please define IRQ_NUM at head of code.
- Demo output:

```
[23839.601441] OS_AS5:init_modules():...............Start...............
[23839.601447] OS_AS5:init_modules(): requset_irq 1 return 0
[23839.601449] OS_AS5:init_modules(): register chrdev(245,0)
[23839.601450] OS_AS5:init_modules(): allocate dma buffer
[23841.578267] OS_AS5:drv_open(): device open
[23841.578270] OS_AS5:drv_ioctl(): My STUID is = 123456789
[23841.578270] OS_AS5:drv_ioctl(): RW OK
[23841.578271] OS_AS5:drv_ioctl(): IOC OK
[23841.578271] OS_AS5:drv_ioctl(): IRC OK
[23842.600228] OS_AS5:drv_ioctl(): Blocking IO
[23842.600231] OS_AS5:drv_write(): queue work
[23842.600231] OS_AS5:drv_write(): block
[23843.247482] OS_AS5:drv_arithmetic_routine():100 p 10000 = 105019
[23843.249952] OS_AS5:drv_read(): ans = 105019
[23843.249964] OS_AS5:drv_ioctl(): Non-Blocking IO
[23843.249965] OS_AS5:drv_write(): queue work
[23843.895819] OS_AS5:drv_arithmetic_routine():100 p 10000 = 105019
[23844.265982] OS_AS5:drv_ioctl(): wait readable 1
[23844.265998] OS_AS5:drv_read(): ans = 105019
[23844.266243] OS_AS5:drv_release(): device close
[23847.810533] OS_AS5:exit_modules(): interrupt count=36
[23847.810534] OS_AS5:exit_modules(): free dma buffer
[23847.810535] OS_AS5:exit_modules(): unregister chrdev
[23847.810536] OS_AS5:exit_modules():..............End..............
```

Report (10 points)

Write a report for your assignment, which should include main information as below:

- How did you design your program?
- What problems you met in this assignment and what is your solution?
- The steps to execute your program.
- Screenshot of your program output.
- What did you learn from this assignment?

Submission

- Please submit the file as package with directory structure as below:
  - **CSC3150_Assignment_5_Student ID**
    - Source
      - main.c (if you complete bonus, submit only one "main.c")
      - test.c
      - ioc_hw5.h
      - makefile
      - mkdev.sh
      - rmdev.sh
    - Report
- Due date: End (23:59) of 03 Dec, 2019

Grading rules

Completion	Marks
Report	10 points
Bonus	10 points
Completed with good quality	80 ~ 90
Completed accurately	80 +
Fully Submitted (compile successfully)	60 +
Partial submitted	0 ~ 60
No submission	0
Late submission	**Not allowed**