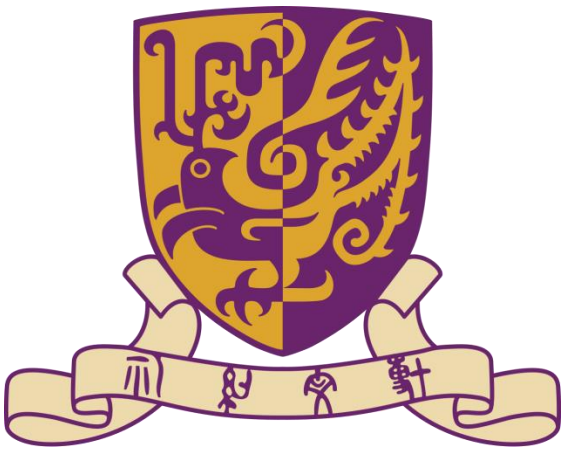

THE CNINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC3150

Operating Systems

Report for Assignment 5

Author:
Li Jingyu 李璟瑜

Student Number:
118010141

Dec 6, 2020

Contents

1 Design

Assignment 5 requires us to design a program that implements a device to simulate the mechanism of I/O system. An I/O device does not belong the bare computer, and it can input or output data to interact with the internal computing and storage resources and external world. The device in this assignment is a very simple one which can realize the basic four arithmetic operations and find the nth prime numbers bigger than a number. Several file operations should be implemented to connect the user mode and kernel mode.

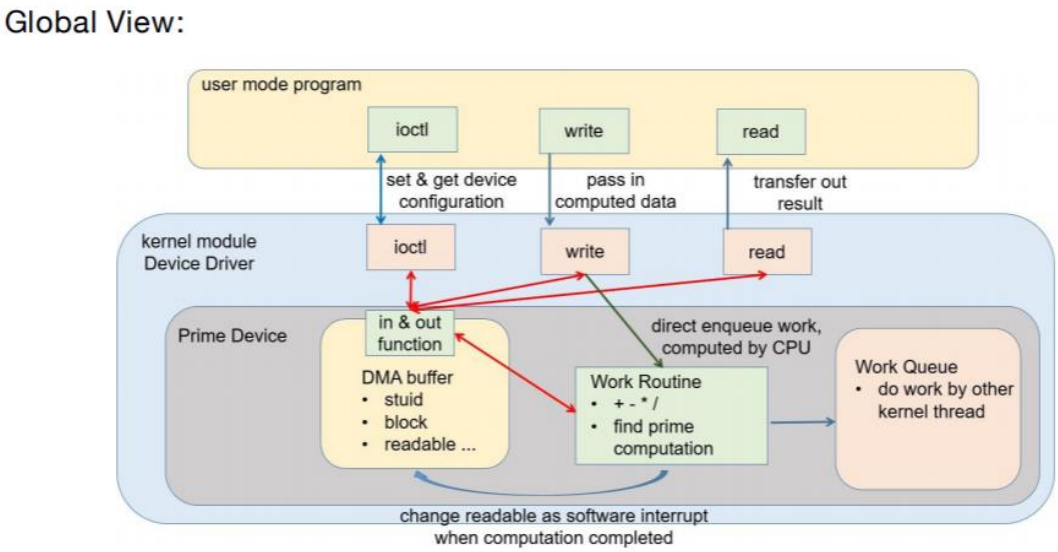

Figure 1: Global view of the program structure

The design will be shown in the order of the function requirement and the source code.

a. Module Initialization.

This part is for initialization and preparation of the device. First define and register a character device. Then obtain the major and minor number of the

device (to identify the device driver). Allocate a space for the device and initialize it and add it to make it alive. The preparation work code has been given in the tutorial. Then allocate spaces for a DMA buffer and the work routine.

b. Read operation

Read function is used to read the computed result in DMA buffer. Use a variable to store the answer and use put_user() function to transfer the result to user space. After reading the answer, the result should be cleaned (set to zero) and readable status will be set to false.

c. Write operation

The data to write in have a specific data structure defined. The structure (DataIn) has three variables of opcode (char) and two operands (integer and short). In order to successfully write the data in DMA buffer, I allocate (kmalloc) a space to store dataIn and then use copy_from_user() as well as three myout() functions to do the data transfer. Then the work routine will be initialized. The IOMode of write should be determined. If it is blocking write, the program should wait until the computation finishes, just schedule the work and then flush all the scheduled work. If it is non-blocking write, it means the program will return immediately and CPU can continue to do other jobs. Schedule the work (see the following function description). The write function will just return 0.

d. Ioctl operation

Ioctl is used to change the configuration of the device. It receives the command (specific setting) and an address of the argument to pass. There are in total 6 commands. They are used to:

(1) set student id and print it out. `HW5_IOCSETSTUID`

(2) set read/write status and print ok if it functions well. `HW5_IOCSETRWOK`

(3) set ioctl status and print ok if it functions well. `HW5_IOCSETIOCOK`

(4) set interrupt service routine status and print ok. `HW5_IOCSETIRQOK`

(5) set blocking/non-blocking setting and print out the current mode.

(6) Wait for the read permission. This function is used to synchronize the non-blocking write. It will regularly check the readable status (5000ms). If it is not readable, sleep and wait for the completion. If it becomes readable, exit and pass the readable parameter to user space.

`HW5_IOCSETBLOCK` `HW5_IOCWAITREADABLE` (mask)

e. Arithmetic routine

Arithmetic routine function is used to perform the arithmetic operation (scheduled work). First, the operator and two operands will be read from dma buffer. According to the operator, corresponding operations will be performed (addition, subtraction, multiplication, division, find prime numbers). The computed answer will be written to dma buffer. Once completed, the readable status will be set to true (computation finished, can read the result).

f. Module exit.

This part is to exit the device and finish execution. First free the space allocated for dma buffer (and work routine later). Then unregister the device, deleting it and print the ending information.

g. Test program.

The test program lies in the user space to test the device functioning. The execution logic is pretty similar with the device. When running the user mode function, kernel mode functions will be invoked. Open the device, do several dma buffer ioctl (change settings) operations. Then perform the arithmetic operations, calculating the nth prime number. Then it comes to two writing mode – blocking and non-blocking. For non-blocking one, it will use ioctl function to continuously check whether the program is readable or not. If not, keep waiting. Then print out the answer computed by user program and kernel program, which should be the same.

h. DMA buffer

DMA buffer is to map the device to the main memory directly. It is used to record information and store the data in this project. The address and relative information recorded has been defined at the head of the main.c. Doing the subtraction of the address can get the size of each information. Here list the contents:

```
// 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
```

i. File operation relationships.

The function prototype and implementation of the user program and kernel program is different. Some internal rule is used to map them. Here a structure is used to realize the correspondence.

```
// cdev 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,
};
```

j. Data transfer.

Data transfer between outside data and DMA buffer is realized by myin and myout functions. There are 3 different functions for in/out with different data types. The argument port is to specify the address of in/out. Sample:

```
void myoutc(unsigned char data,unsigned short int port);
```

k. Open & release.

This two kernel functions will be invoked at the beginning of the program (run open() in user program) and at the end (automatically call). They mean the open and close of the device.

Bonus:

Bonus question asks to deal with the keyboard interrupt. Whenever there is a keyboard hit, an interrupt happens. First define a global variable representing the total interrupt number. Then define a handler which adds one to the count whenever there is an interrupt.

```
irq_handler_t irq_handler(int irq, void *dev_id, struct pt_regs *regs){
    interrupt_count++;
    return (irq_handler_t) IRQ_HANDLED;
}
```

In initialization and exit module functions, register an IRQ handler using request_irq() function and use free_irq() function to free the interrupt service routine.

```
/* Initialize interrupt module and register IRQ handler */
interrupt_count = 0;
ret = request_irq(IRQ_NUM, (irq_handler_t) irq_handler, IRQF_SHARED, DEV_NAME, (void *)(irq_handler));
```

```
free_irq(IRQ_NUM, (void *)(irq_handler));
```

Some more things to add:

1. To differentiate blocking and non-blocking write, the prime number function is computed and there will be obvious delay for non-blocking since it needs the

readable to be set as true (checked every 5000ms) for synchronization. Blocking needs to wait for the computation to finish while non-blocking write just returns immediately.

2. Specific student id is set in test program in dma buffer. My student id is 118010141.
3. To debug the program, different print function should be used. In user mode, use printf and printk is for kernel mode.
4. Use get_user() and put_user() and copy_from_user() to do the data transfer.

For more details, please refer to the codes.

2 Problems and Solutions

In this part I will discuss the problems (bugs) I met and how I solved them when writing the assignment.

(a) **Problem**: Understanding of the I/O device and project requirement.

Solution: At first it is hard for me to make the mechanism clear. I am confused about the many terminology and many objects in the project (dma buffer, file operations, etc). Go to the lecture and the tutorial. Have peer discussion. Use Internet to learn more knowledge about the I/O device and kernel.

(b) **Problem**: Program execution.

Solution: Fail to run sudo ./mkdev.sh 245 0 at first. There are two solutions. One is to run the command lines in the shell scripts one by one directly. The other is to add sh after sudo.

(c) **Problem**: Lethal Typo.

Solution: I wrongly use the myin() and myout() functions firstly. Do not notice the data type for dataIn is char, int and short respectively. Some other typos.

(d) **Problem**: Fail to make.

Solution: Do not unzip the file in the share folder. Some unexpected errors will happen.

(e) **Problem**: Data transfer.

Solution: Fail to transfer the dataIn structure to dma buffer. Cannot use get_user() to pass a whole structure (for simple variable). I make use of copy_from_user() and kmalloc() to finish the task.

(f) **Problem**: Confused parenthesis.

Solution: When using switch-case sentence. If I add {} at each case and there is initialization of variable, then the program will fail to run. The storage place will get wrong. Delete the parenthesis.

(g) **Problem**: Pointer operation for data transfer.

Solution: The argument of ioctl is a pointer instead of the real value. Should use (int*) and get_user() to access the data.

3 Execution

OS : **Linux Ubuntu 16.04**

Kernel version: 4.10.14

Steps and commands to execute the program:

1. **make**

   Makefile has already encapsulated the shell commands to build the file node and compile the source codes (build the main.c as kernel module and insert "mydev.ko", and then build "test.c" as executable file "test").

2. **dmesg**

   Check the major and minor numbers of the device.

3. **sudo sh ./mkdev.sh 245 0**

   Install the file node with specified major and minor numbers.

4. **./test**

   Run the executable test file. Show the user program outputs here.

5. **make clean**

Remove the module and automatically use dmesg to list the kernel logs for checking the program running status. Here kernel log will be shown.

6. **Sudo sh ./rmdev.sh**

   Remove the file node. This step can be eliminated for repeatedly running the program. If not do this step, when making the device, it will output “file node exists”.

Relative program outputs will show in the command line window. There is user program output as well as the kernel log output. They should have the same results. The difference between blocking and non-blocking write can be shown by the output delay.

4 Output

In this part the relevant program outputs will be shown.

a. make

```
[12/06/20]seed@VM:~/.../source$ make
make -C /lib/modules/`uname -r`/build M=`pwd` modules
make[1]: Entering directory '/home/seed/whale/linux-4.10.14'
  CC [M]  /home/seed/Downloads/5/source/main.o
  LD [M]  /home/seed/Downloads/5/source/mydev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Downloads/5/source/mydev.mod.o
  LD [M]  /home/seed/Downloads/5/source/mydev.ko
make[1]: Leaving directory '/home/seed/whale/linux-4.10.14'
sudo insmod mydev.ko
gcc -o test test.c
[12/06/20]seed@VM:~/.../source$ dmesg
```

b. dmesg (major and minor number)

```
[23322.429936] OS_AS5:init_modules():..............Start..............
[23322.429940] OS_AS5:init_modules(): register chrdev(245,0)
[23322.429942] OS_AS5:init_modules(): allocate dma buffer
```

c. Install the file node

```
[12/06/20]seed@VM:~/.../source$ sudo sh ./mkdev.sh 245 0
crw-rw-rw- 1 root root 245, 0 Dec  6 02:52 /dev/mydev
```

d. Run test executable – find nth prime number (100^{th} > 10000).

(For non-blocking write, it will wait for 3-5 seconds)

```
[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..............
```

e. Make clean

```
[12/06/20]seed@VM:~/.../source$ make clean
```

```
[23743.290953] OS_AS5:init_modules():..............Start.............
[23743.290957] OS_AS5:init_modules(): register chrdev(245,0)
[23743.290959] OS_AS5:init_modules(): allocate dma buffer
[23745.575039] OS_AS5:drv_open(): device open
[23745.575044] OS_AS5:drv_ioctl(): My STUID is = 118010141
[23745.575046] OS_AS5:drv_ioctl(): RW (read/write operation) OK!
[23745.575047] OS_AS5:drv_ioctl(): IOC (ioctl function) OK!
[23745.575048] OS_AS5:drv_ioctl(): IRC (interrupt service routine) OK!
[23746.346937] OS_AS5:drv_ioctl(): Blocking IO
[23746.346942] OS_AS5:drv_write: queue work
[23746.346943] OS_AS5:drv_write(): block
[23746.956378] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[23746.956500] OS_AS5:drv_read():ans = 105019
[23746.956539] OS_AS5:drv_ioctl(): Non-Blocking IO
[23746.956542] OS_AS5:drv_write: queue work
[23747.566925] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[23752.188319] OS_AS5:drv_ioctl(): wait readable 1
[23752.188358] OS_AS5:drv_read():ans = 105019
[23752.188522] OS_AS5:drv_release(): device close
[23868.696868] OS_AS5:exit_modules():free dma buffer
[23868.696875] OS_AS5:exit_modules(): unregister chrdev
[23868.696877] OS_AS5:exit_modules():..............End.............
```

f. Uninstall the file node

```
[12/06/20]seed@VM:~/.../source$ sudo sh ./rmdev.sh
ls: cannot access '/dev/mydev': No such file or directory
```

g. Run another arithmetic function (addition)

```
[12/06/20]seed@VM:~/.../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..............
```

```
[24191.714298] OS_AS5:init_modules():..............Start..............
[24191.714301] OS_AS5:init_modules(): register chrdev(245,0)
[24191.714303] OS_AS5:init_modules(): allocate dma buffer
[24199.853061] OS_AS5:drv_open(): device open
[24199.853068] OS_AS5:drv_ioctl(): My STUID is = 118010141
[24199.853070] OS_AS5:drv_ioctl(): RW (read/write operation) OK!
[24199.853071] OS_AS5:drv_ioctl(): IOC (ioctl function) OK!
[24199.853072] OS_AS5:drv_ioctl(): IRC (interrupt service routine) OK!
[24199.853088] OS_AS5:drv_ioctl(): Blocking IO
[24199.853091] OS_AS5:drv_write: queue work
[24199.853091] OS_AS5:drv_write(): block
[24199.853100] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[24199.853132] OS_AS5:drv_read():ans = 110
[24199.853147] OS_AS5:drv_ioctl(): Non-Blocking IO
[24199.853153] OS_AS5:drv_write: queue work
[24199.853156] OS_AS5:drv_arithmetic_routine(): 100 + 10 = 110
[24199.853162] OS_AS5:drv_ioctl(): wait readable 1
[24199.853169] OS_AS5:drv_read():ans = 110
[24199.853695] OS_AS5:drv_release(): device close
[24218.798273] OS_AS5:exit_modules():free dma buffer
[24218.798277] OS_AS5:exit_modules(): unregister chrdev
[24218.798278] OS_AS5:exit_modules():..............End..............
```

h. Run multiplication and division.

```
[12/06/20]seed@VM:~/.../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

100 / 10 = 10

Blocking IO
ans=10 ret=10

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

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

```
[24337.578080] OS_AS5:init_modules():..............Start..............
[24337.578083] OS_AS5:init_modules(): register chrdev(245,0)
[24337.578085] OS_AS5:init_modules(): allocate dma buffer
[24339.697052] OS_AS5:drv_open(): device open
[24339.697058] OS_AS5:drv_ioctl(): My STUID is = 118010141
[24339.697059] OS_AS5:drv_ioctl(): RW (read/write operation) OK!
[24339.697060] OS_AS5:drv_ioctl(): IOC (ioctl function) OK!
[24339.697062] OS_AS5:drv_ioctl(): IRC (interrupt service routine) OK!
[24339.697071] OS_AS5:drv_ioctl(): Blocking IO
[24339.697073] OS_AS5:drv_write: queue work
[24339.697074] OS_AS5:drv_write(): block
[24339.697103] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[24339.697107] OS_AS5:drv_read():ans = 1000
[24339.697112] OS_AS5:drv_ioctl(): Non-Blocking IO
[24339.697115] OS_AS5:drv_write: queue work
[24339.697120] OS_AS5:drv_arithmetic_routine(): 100 * 10 = 1000
[24344.831609] OS_AS5:drv_ioctl(): wait readable 1
[24344.831650] OS_AS5:drv_read():ans = 1000
[24344.831674] OS_AS5:drv_ioctl(): Blocking IO
[24344.831677] OS_AS5:drv_write: queue work
[24344.831678] OS_AS5:drv_write(): block
[24344.831700] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[24344.834213] OS_AS5:drv_read():ans = 10
[24344.834225] OS_AS5:drv_ioctl(): Non-Blocking IO
[24344.834229] OS_AS5:drv_write: queue work
[24344.834379] OS_AS5:drv_arithmetic_routine(): 100 / 10 = 10
[24349.945357] OS_AS5:drv_ioctl(): wait readable 1
[24349.945380] OS_AS5:drv_read():ans = 10
[24349.945579] OS_AS5:drv_release(): device close
[24380.105175] OS_AS5:exit_modules():free dma buffer
[24380.105183] OS_AS5:exit_modules(): unregister chrdev
[24380.105185] OS_AS5:exit_modules():..............End..............
```

i. Run the second prime number operation (100^{th} > 20000).

```
[12/06/20]seed@VM:~/.../source$ ./test
..............Start..............
100 p 20000 = 225077

Blocking IO
ans=225077 ret=225077

Non-Blocking IO
Queueing work
Waiting
Can read now.
ans=225077 ret=225077

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

```
[24439.186387] OS_AS5:init_modules():.............Start.............
[24439.186391] OS_AS5:init_modules(): register chrdev(245,0)
[24439.186393] OS_AS5:init_modules(): allocate dma buffer
[24445.117561] OS_AS5:drv_open(): device open
[24445.117566] OS_AS5:drv_ioctl(): My STUID is = 118010141
[24445.117567] OS_AS5:drv_ioctl(): RW (read/write operation) OK!
[24445.117569] OS_AS5:drv_ioctl(): IOC (ioctl function) OK!
[24445.117570] OS_AS5:drv_ioctl(): IRC (interrupt service routine) OK!
[24448.405735] OS_AS5:drv_ioctl(): Blocking IO
[24448.405741] OS_AS5:drv_write: queue work
[24448.405742] OS_AS5:drv_write(): block
[24451.019778] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
[24451.019955] OS_AS5:drv_read():ans = 225077
[24451.019987] OS_AS5:drv_ioctl(): Non-Blocking IO
[24451.019990] OS_AS5:drv_write: queue work
[24453.736094] OS_AS5:drv_arithmetic_routine(): 100 p 20000 = 225077
[24456.197606] OS_AS5:drv_ioctl(): wait readable 1
[24456.197647] OS_AS5:drv_read():ans = 225077
[24456.197851] OS_AS5:drv_release(): device close
[24478.061054] OS_AS5:exit_modules():free dma buffer
[24478.061058] OS_AS5:exit_modules(): unregister chrdev
[24478.061060] OS_AS5:exit_modules():.............End.............
```

j. Bonus

```
[  708.436408] OS_AS5:init_modules():.............Start.............
[  708.436412] OS_AS5:init_modules(): request_irq 1 return 0
[  708.436414] OS_AS5:init_modules(): register chrdev(245,0)
[  708.436415] OS_AS5:init_modules(): allocate dma buffer
[  769.356227] OS_AS5:drv_open(): device open
[  769.356230] OS_AS5:drv_ioctl(): My STUID is = 118010141
[  769.356231] OS_AS5:drv_ioctl(): RW (read/write operation) OK!
[  769.356232] OS_AS5:drv_ioctl(): IOC (ioctl function) OK!
[  769.356232] OS_AS5:drv_ioctl(): IRC (interrupt service routine) OK!
[  770.136557] OS_AS5:drv_ioctl(): Blocking IO
[  770.136561] OS_AS5:drv_write: queue work
[  770.136561] OS_AS5:drv_write(): block
[  770.761370] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[  770.761472] OS_AS5:drv_read():ans = 105019
[  770.761504] OS_AS5:drv_ioctl(): Non-Blocking IO
[  770.761506] OS_AS5:drv_write: queue work
[  771.368553] OS_AS5:drv_arithmetic_routine(): 100 p 10000 = 105019
[  775.869195] OS_AS5:drv_ioctl(): wait readable 1
[  775.869233] OS_AS5:drv_read():ans = 105019
[  775.869458] OS_AS5:drv_release(): device close
[  784.209035] OS_AS5:exit_modules(): interrupt count=84
[  784.209036] OS_AS5:exit_modules(): free dma buffer
[  784.209038] OS_AS5:exit_modules(): unregister chrdev
[  784.209041] OS_AS5:exit_modules():.............End.............
```

5 Feeling

Here I will share several feelings I have when writing assignment 5.

a. Assignment 5 is pretty similar with assignment 1. I need to spend a lot of time viewing the tutorial notes, understanding the basic concept, discussing

with classmates to make the logic clear. However, I still not figure out some basic file operations and device configuration clearly and it brings me some troubles when debugging.

b. It takes me a lot of time seeing the material and not to start to write the code. A simple task becomes time-consuming if not start earlier. I think I should not think too much next time and try to write some codes directly!!

c. The project is pretty helpful and meaningful. I learn a lot about the device definition and I/O system and get more familiar with kernel related items. More hard-core computer science and engineering knowledge obtained!! Also I improve my coding and debugging ability a little bit.

d. I do not expect that I will spend a bit much time in debugging this time. Though the program logic is not difficult, there are still some subtle mistakes. Bugs always exist.

e. I finish the project pretty late this time. I even write the report when listening to the CUHKSZ concert, traversing the limit.

f. Since this is the last project, I originally think it will be very hard. However, it doesn't and it is highly similar with project 1. This is called "A good beginning and a good ending (shou wei hu ying)". Wonderful design. Goodbye CSC3150 project. It really benefits me a lot.

g. By the way, thanks for your grading! It must be a hard work checking more than one hundred students' homework. Respect!

That's all.