CS5250 – Advanced Operating Systems

AY2016/2017 Semester 2

Assignment 4

Deadline: Friday, 14 Apr 2017 • 11.59pm

1. Objectives

1. Extend the device driver

2. Rules

1. **Please submit your report in a PDF file and include your 8-character matric number. Only submit one PDF file in the folder. The report should be at most 10 pages (not including attached code). Failure to conform to this rule will result in 1 to 5 marks penalty.**
2. **Late assignments will lose 4 marks per day.**
3. For any question, contact the teaching assistant Ms Bao Ning, baoning@comp.nus.edu.sg.
4. It is fine to ask for "reasonable" amount of help from others, but ensure that you do all the tasks on your own and write the report on your own. The University's policy on plagiarism applies here and any breaches will be dealt with severely.
5. Read instructions and tips carefully. Lots of people make mistakes or waste time because they missed an explanation or a guideline.
6. Always read the error messages carefully, in particular the first one issued. For example, some of you stumbled on very simple errors just because they specified a wrong file path and didn't pay enough attention to the corresponding error message.

3. Tasks (25 marks)

I think this assignment is much more difficult than assignment 3. It may be the most difficult assignment in this module. You can definitely learn a lot from the assignment! Hope you can also have fun during the process. Good luck!

1. 4 marks, 5 minutes.

   Please give the screenshot of the github commit as assignment 3 after you finish all the tasks. I put this here just to remind you that you need to sync your code from the very beginning.

   You can give important codes when you feel necessary. Please attach your codes (only the device driver code, do not include the test program) at the end of the report. Do not submit your code separately or use a compressed file!
   Some hints for the assignment:

   You are encouraged to write multiple shell scripts to load, unload your drivers and run test programs so that you do not need to type in the commands manually. For a shell script file, it is really easy for the use of this assignment. You just need to input #!/bin/sh as the first line and write the commands the same as you input in the terminal. Here is an example of running the test program and print the driver context.
   test

   ```
   #!/bin/sh

   gcc test.c -o test
   ./test
   cat /dev/scull
   ```

2. 8 marks, about 2 hours

   Extend the previous one byte device to be a 4MB-device.

2 marks: Give the new kmalloc and kfree code in your init and exit function.

2 marks: Give the output of the following test case :

```
echo abc > /dev/<device>
cat /dev/<device>
echo defg > /dev/<device>
cat /dev/<device>
```

4 marks: Copy a 5MB file to your device. Print the return value of the write function which is the numbers of bytes written to the device. Also give the head and the tail context of your device and analyse which parts are not written into the device.

Hint : You can print the return value by writing a user mode program to open and write to the device. Remember that in Linux, everything is a file. So you can treat your device just as a file with **open/read/write** functions.

Another way is to print all the return values in your write function before return. However, this is not encouraged and only should be used during debugging. It is ok for an assignment.

3. 6 marks, about 30 minutes to 1 hour

The llseek function implements the lseek system call which is used to change the position of the file pointer. Please implement the llseek function in your device driver.

2 marks: Give a list your arguments of your llseek function and explain their meanings. What values can the final argument be and what are the meanings of the values?
4 marks: Run the following test program after you modify the device name to test whether the lseek function works. Print the context of the device after running the program. Give the screenshot of the results.

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
int lcd;
void test() {
	int k, i, sum;
	char s[3];
	memset(s, '2', sizeof(s));

	printf("test begin!\n");

	k = lseek(lcd, 4, SEEK_CUR);
	printf("lseek = %d\n", k);

	k = write(lcd, s, sizeof(s));
	printf("written = %d\n", k);

	k = lseek(lcd, 0, SEEK_END);
	printf("lseek = %d\n", k);

	k = lseek(lcd, -4, SEEK_END);
	printf("lseek = %d\n", k);

	k = lseek(lcd, -4, -1);
	printf("lseek = %d\n", k);
}
void initial(char i) {
	char s[10];
	memset(s, i, sizeof(s));
	write(lcd, s, sizeof(s));
	char c[20] = "";
	int k = lseek(lcd, 0, SEEK_SET);
	printf("lseek = %d\n", k);
}
int main(int argc, char **argv) {
	lcd = open("/dev/lcd", O_RDWR);
	if (lcd == -1) {
		printk("unable to open lcd");
		exit(EXIT_FAILURE);
	}
	initial('1');
	test();
	close(lcd);
	return 0;
}
```

Hint : You need a size value in your driver device for the end version of llseek. The value of size means the current number of bytes in the device driver, rather than the total capacity of the device.

For marking, there are 6 printk codes with arguments and 1 command of printing the device. Each worth 1 mark.

4. 7 marks, about 2 hours

Most drivers need the ability to perform various types of hardware control via the device driver. These operations are usually supported via the ioctl method, which implements the system call by the same name. This task is designed to enable you to write ioctl.

Please check Documentation/ioctl/ioctl-number.txt for details about ioctl. I will give you a simple example of ioctl. It gives the basic structure for ioctl.

device_driver.c (Only a small portion of the device driver file in kernel mode)

```
#include <linux/ioctl.h> /* needed for the _IOW etc stuff used later */
#define SCULL_IOC_MAGIC  'k'
#define SCULL_HELLO _IO(SCULL_IOC_MAGIC,   1)
long ioctl_example(struct file *filp, unsigned int cmd, unsigned long arg)
                        {

              int err = 0, tmp;
               int retval = 0;
             /*
                * extract the type and number bitfields, and don't decode
                * wrong cmds: return ENOTTY (inappropriate ioctl) before access_ok()
                */
               if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC)  return -ENOTTY;
               if (_IOC_NR(cmd) > SCULL_IOC_MAXNR) return -ENOTTY;
                /*
                * the direction is a bitmask, and VERIFY_WRITE catches R/W
                * transfers. `Type' is user-oriented, while
                * access_ok is kernel-oriented, so the concept of "read" and
                * "write" is reversed
                */
               if (_IOC_DIR(cmd) & _IOC_READ)
                          err = !access_ok(VERIFY_WRITE, (void __user *)arg, _IOC_SIZE(cmd));
               else if (_IOC_DIR(cmd) & _IOC_WRITE)
                          err =  !access_ok(VERIFY_READ, (void __user *)arg, _IOC_SIZE(cmd));
               if (err) return -EFAULT;
               switch(cmd) {
                        case SCULL_HELLO:
                          printk(KERN_WARNING "hello\n");
                          break;
```

```
    default:  /* redundant, as cmd was checked against MAXNR */
            return -ENOTTY;
  }
  return retval;

}
…
```

In file_operations, add **.unlocked_ioctl = ioctl_example.**

test.c (The file you write to test the device driver in user mode)

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
//needed for IO things. Attention that this is different from kernel mode
int lcd;
#define SCULL_IOC_MAGIC  'k'
#define SCULL_HELLO _IO(SCULL_IOC_MAGIC,   1)

void test()
{
        int k, i, sum;
        char s[3];

        memset(s, '2', sizeof(s));
        printf("test begin!\n");

        k = write(lcd, s, sizeof(s));
        printf("written = %d\n", k);
```

```
	k = ioctl(lcd, SCULL_HELLO);
	printf("result = %d\n", k);

}
int main(int argc, char **argv)
{
	lcd = open("/dev/scull", O_RDWR);
	if (lcd == -1) {
	perror("unable to open lcd");
	exit(EXIT_FAILURE);
	}

	test();

	close(lcd);
	return 0;
}
```

<u>2 marks</u>: Add the basic hello function to your device driver and show the hello message printed by the ioctl in kernel messages.

<u>4 marks</u>: Implement a char array to store message in your device driver called ***dev_msg***. Implement a ioctl function to set ***dev_msg*** using _IOW macro. Call the ioctl function to set the value. Implement another value to copy ***dev_msg*** to a variable called ***user_msg*** in your test program using _IOR macro. Call the new ioctl function and print ***user_msg*** to see whether the two functions work well.

<u>1 mark</u>: Extend 3c so that you can change ***dev_msg*** and also store the original value of ***dev_msg*** in ***user_msg*** using _IOWR. Print the new ***dev_msg*** using printk. Call the ioctl function and print ***user_msg***. Also use dmesg to see the value of ***dev_msg.***

END OF ASSIGNMENT