Running Environment:

Software

1. Ubuntu 16.04 LTS

Hardware

2. CPU: Intel Core i5-7200U
3. RAM: 4GB*2 DDR4 1033MHz

Progress Diary

Stage	Step	Task Description	Comments	Time
1	1	Download ubuntu-16.04.3-desktop-amd64.iso from Linux operating system Ubuntu (64 bit) from ubuntu.com		Start date: 8/10/2017 End date: 8/10/2017 Demo date: 11/10/2017
1	2	Write a character device driver	Not familiar to Linux Kernel function, so encountered some reading and write problem. Not familiar to device so spend a lot of time on device register.	Start date: 8/10/2017 End date: 8/10/2017 Demo date: 11/10/2017
	3	Write a make file	Makefile is quite different from normal user application	Start date: 8/10/2017 End date: 8/10/2017 Demo date: 11/10/2017

	4	Write a user application	Encounter problem when call driver from system module.	Start date: 8/10/2017 End date: 8/10/2017 Demo date: 11/10/2017

Stage1

1. File s3560808.c (The driver)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#define  DEVICE_NAME "S3560808Device"
#define  CLASS_NAME  "S3560808"



static int    majorNumber;
static short  size_of_message;
static struct class*  s3560808  = NULL;
static struct device* s3560808Device = NULL;

struct virtual_device{
```

```
    char data[100];
    struct semaphore sem;
}myDevice;

static int     dev_open(struct inode *, struct file *);
static int     dev_release(struct inode *, struct file *);
static ssize_t dev_read(struct file *, char *, size_t, loff_t *);
static ssize_t dev_write(struct file *, const char *, size_t, loff_t *);

static struct file_operations fops =
{
   .open = dev_open,
   .read = dev_read,
   .write = dev_write,
   .release = dev_release,
};

static int __init s3560808_init(void){
   printk(KERN_INFO "S3560808 Device info: Initializing the S3560808 Device info LKM\n");

   // Try to dynamically allocate a major number for the device
   majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
   if (majorNumber<0){
      printk(KERN_ALERT "S3560808 Device info failed to register a major number\n");
      return majorNumber;
   }
```

```
   printk(KERN_INFO "S3560808 Device info: registered correctly with major number %d\n", majorNumber);

   // Register the device class
   s3560808 = class_create(THIS_MODULE, CLASS_NAME);
   if (IS_ERR(s3560808)){               // Check for error and clean up if there is
      unregister_chrdev(majorNumber, DEVICE_NAME);
      printk(KERN_ALERT "Failed to register device class\n");
      return PTR_ERR(s3560808);         // Correct way to return an error on a pointer
   }
   printk(KERN_INFO "S3560808 Device info: device class registered correctly\n");

   // Register the device driver
   s3560808Device = device_create(s3560808, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
   if (IS_ERR(s3560808Device)){              // Clean up if there is an error
      class_destroy(s3560808);          // Repeated code but the alternative is goto statements
      unregister_chrdev(majorNumber, DEVICE_NAME);
      printk(KERN_ALERT "Failed to create the device\n");
      return PTR_ERR(s3560808Device);
   }
   printk(KERN_INFO "S3560808 Device info: device class created correctly\n"); // Made it! device was initialized
   return 0;
}

static void __exit s3560808_exit(void){
   device_destroy(s3560808, MKDEV(majorNumber, 0));     // remove the device
   class_unregister(s3560808);                          // unregister the device class
```

```
   class_destroy(s3560808);                              // remove the device class
   unregister_chrdev(majorNumber, DEVICE_NAME);             // unregister the major number
   printk(KERN_INFO "S3560808 Device info: Goodbye from the LKM!\n");
}

static int dev_open(struct inode *inodep, struct file *filep){
   printk(KERN_INFO "S3560808 Device info: Device is opened successfully!\n");
   return 0;
}

static ssize_t dev_read(struct file *filep, char *buffer, size_t len, loff_t *offset){
   int error_count = 0;
   // copy_to_user has the format ( * to, *from, size) and returns 0 on success
   error_count = copy_to_user(buffer, myDevice.data, strlen(myDevice.data));

   if (error_count==0){            // if true then have success
      printk(KERN_INFO "S3560808 Device info: Sent %d characters to the user\n", size_of_message);
      return (size_of_message=0);  // clear the position to the start and return 0
   }
   else {
      printk(KERN_INFO "S3560808 Device info: Failed to send %d characters to the user\n", error_count);
      return -EFAULT;              // Failed -- return a bad address message (i.e. -14)
   }
}

static ssize_t dev_write(struct file *filep, const char *buffer, size_t len, loff_t *offset){
```

```
   copy_from_user(myDevice.data,buffer,len);
   size_of_message = strlen(myDevice.data);
   printk(KERN_INFO "S3560808 Device info: Received %zu characters from the user\n", len);
   return len;
}

static int dev_release(struct inode *inodep, struct file *filep){
   printk(KERN_INFO "S3560808 Device info: Device successfully closed\n");
   return 0;
}

module_init(s3560808_init);
module_exit(s3560808_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kai Zhang s3560808");
MODULE_DESCRIPTION("A simple Linux char driver for assignment 2");
MODULE_VERSION("1.0");
```

2. S3560808UserApplication.c

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<fcntl.h>
#include<string.h>
```

```
#include<unistd.h>

#define BUFFER_LENGTH 100
static char receive[BUFFER_LENGTH];

int main(){
   int ret, fd;
   char stringToSend[BUFFER_LENGTH];
   printf("Starting device test code example...\n");
   fd = open("/dev/S3560808Device", O_RDWR);            // Open the device with read/write access
   if (fd < 0){
      perror("Failed to open the device...");
      return errno;
   }
   printf("Type in a short string to send to the kernel module:\n");
   scanf("%[^\n]%*c", stringToSend);               // Read in a string (with spaces)
   printf("Writing message to the device [%s].\n", stringToSend);
   ret = write(fd, stringToSend, strlen(stringToSend)); // Send the string to the LKM
   if (ret < 0){
      perror("Failed to write the message to the device.");
      return errno;
   }

   printf("Press ENTER to read back from the device...\n");
   getchar();
```

```
   printf("Reading from the device...\n");
   ret = read(fd, receive, BUFFER_LENGTH);        // Read the response from the LKM
   if (ret < 0){
      perror("Failed to read the message from the device.");
      return errno;
   }
   printf("The received message is: [%s]\n", receive);
   printf("End of the program\n");
   return 0;
}
```

3. Makefile

```
obj-m+=s3560808.o

all:
  make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
  make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

4. Test and result

```
kai@kai-Inspiron-7460:~/1$ sudo insmod s3560808.ko
kai@kai-Inspiron-7460:~/1$ sudo ./test
Starting device test code example...
Type in a short string to send to the kernel module:
This is assignemnt 2 message to my deivce
Writing message to the device [This is assignemnt 2 message to my deivce].
Press ENTER to read back from the device...

Reading from the device...
The received message is: [This is assignemnt 2 message to my deivce]
End of the program
kai@kai-Inspiron-7460:~/1$
```