Running Environment:

**Software**

1. Ubuntu 16.04 LTS

**Hardware**

1. CPU: Intel Core i5-7200U
2. 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");

1. **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;

}

1. **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

1. **Test and result**

