Character Device Driver

Files & media	05-Unix-Char-I(1).pdf 05-Unix-Char-II.pdf
Name	Demo 5
Review	☐

character device driver

```
cd arm-board
cd linux-3.0.8
cd drivers
cd char/

//editor
nano <code file name>
OR
vim <code file name>
```

code in driver file

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/string.h>
#include <asm/uaccess.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

#define DEVICE_NAME "comp3438_helloworld"
#define N_D 1 //number of devices
#define S_N 1 //start minor number
```

```
static char msg[] = "Hello World!";
static int major;
static dev_t devno; // device type
static struct cdev dev_chardemo1;

static struct file_operation file_ops = {
    owner: THIS_MODULE,
    open:  demo_char1_open,
    read:  demo_char1_read,
    release:  demo_char1_release,
}

static int demo_char1_open(struct inode *inode, struct file *fp
    printk("Device " DEVICE_NAME " open.\n");
    return 0;
}

static ssize_t demo_char1_read(struct file *fp, char *nuf, size_
    int num;
    int ret;
    if (count < strlen(msg)) num = count;
    else num = strlen(msg);
    ret = copy_to_usr(buf, msg, num);
    if (ret) {
        printk("Fail to copy data from the kernel space to the u
        return -1;
    }
    return num;
}

static int demo_char1_release(struct inode *inode, struct file *
    printk("Device " DEVICE_NAME " release.\n");
    return 0;
}
```

```
// __init: initializing code for linker
static int __init demo_char1_init(void){
    int ret;
    // register a major number
    ret = alloc_chrdev_region(&devno, S_N, N_D, DEVICE_NAME);
    if (ret < 0) {
        printk("Device " DEVICE_NAME "cannot get major number.\n
        return ret;
    }
    major = MAJOR(devno);
    printk("Device " DEVICE_NAME " initialized: major number = %

    // register a char device
    cdev_init(&dev_chardemo1, &file_ops);
    dev_chardemo1.owner = THIS_MODULE;
    dev_chardemo.ops = &file_ops;
    ret = cdev_add(&dev_chardemo1, devno, N_D);
    if (ret){
        printk("Device " DEVICE_NAME " register fail.\n");
        return ret;
    }
    return 0;
}

static void __exit demo_char1_exit(void){
    cdev_del(&dev_chardemo1);
    unregister_chardev_region(devno, N_D);
    printk("Device " DEVICE_NAME " unloaded.\n");
}

//provide function
module_init(demo_char1_init);
module_exit(demo1_char1_exit);

// management info
MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("Siyu Zhou");
MODULE_DESCRIPRION("Char Driver Development: Hello World. Course
```

search for info

```
man printf
elixir.bootlin.com
```

code for use file

COMP3438_chardemo1_app.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
    int fd;
    int countl
    char buf[100];
    fd =open("/dev/chardemo1", O_RDONLY);
    if (fd == -1){
        printf("Fail to open device chardemo1!\n");
        goto finish;
    }

    count = read(fd, buf, 99);
    printf("%d bytes are read from the kernel space.\n", count),
    buf[count] = '\0';
```

```
    print ("%s\n", buf);

finish:
    close(fd);
    return 0;
}
```

AGAIN code practicing

```
#include <linux/init.h> // module_init(); module_exit()
#include <linux/module.h> // MODULE_licence(); author(); descrip
#include <fs.h>

#define DRIVER_NAME "comp3438_hello_world_char_driver"
#define N_D 1 // max number of devices using this driver
#define S_N 1 // start of minor number

static dev_t devno; // dev_t is a device type
static int major;
static struct cdev dev_profile;
static strut file_operations fops{

}

// macro way to tell
// localized function, output: int, specialized as __init functi
static int __init helloworld_char_init(void){
    int ret;
    // register to kernel
    ret = alloc_chrdev_region(&devno, S_N, N_D, DRIVER_NAME);
    if(ret < 0){
        printk("Device " DEVICE_NAME " cannot get major number.\
        return ret;
    }
```

```
    major = MAJOR(devno);
    printk("Device " DRIVER_NAME " initialized (Major number %d
    // register a char driver
    cdev_init(&dev_profile, &fops);
    dev_profile.owner = THIS_MODULE;
    dev_profile = &fops;
    ret = cdev_add(&dev_profile, deno, N_D);
    if(ret){
        printk("Deriver " DERIVER_NAME "registerfail.\n");
    return 0;
    }
}
static void __exit helloworld_char_exit(void){
}

module_init(helloworld_char_init); // where to start
module_exit(helloworld_char_exit); // remove from kernel

MODULE_LICENCE("GPL");
MODULE_AUTHOR("Author");
MODULE_DESCRIPTION("Hello world character device driver");
```

on own computer

download linux-3.0.8.tar.gz

```
tar zxvf linux-3.0.8.tar.gz
```