DEVICE DRIVERS – LAB EXERCISE 7

Submitted By:

Kiran Thomas Cherian

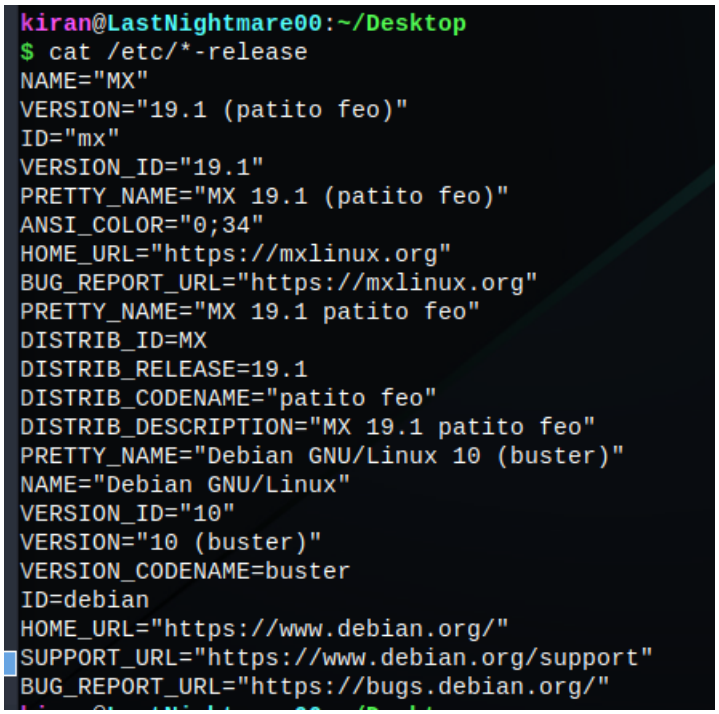
CED18I028

OBJECTIVE:

Write a C program which contains ioctl() and execute.

Linux Distribution Used:

MX Linux 19.1 (Running on Virtual machine)



IOCTL is referred to as Input and Output Control, which is used to talking to device drivers. This system call, available in most driver categories. The major use of this is in case of handling some specific operations of a device for which the kernel does not have a system call by default.

There are some steps involved to use IOCTL.

- Create IOCTL command in driver
- Write IOCTL function in the driver
- Create IOCTL command in a Userspace application
- Use the IOCTL system call in a Userspace

Code:

We will add the IOCTL command and function to the device driver program used in previous assignment (assignment 6)

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include<linux/slab.h>                 //kmalloc()
#include <linux/ioctl.h>
```

```
#define MAX_DEV 1
#define WR_VALUE _IOW('a','a',int32_t*)
#define RD_VALUE _IOR('a','b',int32_t*)

int32_t value = 0;


static int mychardev_open(struct inode *inode, struct file *file);
static int mychardev_release(struct inode *inode, struct file *file);
static ssize_t mychardev_read(struct file *file, char __user *buf, size_t count, loff_t *offset);
static ssize_t mychardev_write(struct file *file, const char __user *buf, size_t count, loff_t *offset);
static long     my_ioctl(struct file *file, unsigned int cmd, unsigned long arg);

static const struct file_operations mychardev_fops = {
   .owner      = THIS_MODULE,
   .open       = mychardev_open,
   .release    = mychardev_release,
   .read       = mychardev_read,
   .write      = mychardev_write,
         .unlocked_ioctl = my_ioctl
};

struct mychar_device_data {
   struct cdev cdev;
};
```

```
static int dev_major = 0;
static struct class *mychardev_class = NULL;
static struct mychar_device_data mychardev_data[MAX_DEV];

static int mychardev_uevent(struct device *dev, struct kobj_uevent_env *env)
{
   add_uevent_var(env, "DEVMODE=%#o", 0666);
   return 0;
}

static int __init mychardev_init(void)
{
   int err, i;
   dev_t dev;

   err = alloc_chrdev_region(&dev, 0, MAX_DEV, "kiranchardev");

   dev_major = MAJOR(dev);

   mychardev_class = class_create(THIS_MODULE, "kiranchardev");
   mychardev_class->dev_uevent = mychardev_uevent;

   for (i = 0; i < MAX_DEV; i++) {
      cdev_init(&mychardev_data[i].cdev, &mychardev_fops);
      mychardev_data[i].cdev.owner = THIS_MODULE;
```

```
        cdev_add(&mychardev_data[i].cdev, MKDEV(dev_major, i), 1);


        device_create(mychardev_class, NULL, MKDEV(dev_major, i), NULL,
"kiranchardev-%d", i);
    }
          printk(KERN_INFO "Inserting my module..\n");
    return 0;
}

static void __exit mychardev_exit(void)
{
    int i;

    for (i = 0; i < MAX_DEV; i++) {
        device_destroy(mychardev_class, MKDEV(dev_major, i));
    }

    class_unregister(mychardev_class);
    class_destroy(mychardev_class);

    unregister_chrdev_region(MKDEV(dev_major, 0), MINORMASK);
          printk(KERN_INFO "Removing my module.bye..\n");
}

static int mychardev_open(struct inode *inode, struct file *file)
{
```

```
    printk("KIRANCHARDEV: Device opened\n");
    return 0;
}

static int mychardev_release(struct inode *inode, struct file *file)
{
    printk("KIRANCHARDEV: Device closed\n");
    return 0;
}



static ssize_t mychardev_read(struct file *file, char __user *buf, size_t count,
loff_t *offset)
{
    uint8_t *data = "Greetings from the kernel side\n";
    size_t datalen = strlen(data);

    printk("Reading from device: %d\n", MINOR(file->f_path.dentry->d_inode-
>i_rdev));

    if (count > datalen) {
        count = datalen;
    }

    if (copy_to_user(buf, data, count)) {
        return -EFAULT;
    }
```

```
    return count;
}

static ssize_t mychardev_write(struct file *file, const char __user *buf, size_t
count, loff_t *offset)
{
    size_t maxdatalen = 30, ncopied;
    uint8_t databuf[maxdatalen];

    printk("Writing to device: %d\n", MINOR(file->f_path.dentry->d_inode-
>i_rdev));

    if (count < maxdatalen) {
        maxdatalen = count;
    }

    ncopied = copy_from_user(databuf, buf, maxdatalen);

    if (ncopied == 0) {
        printk("Copied %zd bytes from the user\n", maxdatalen);
    } else {
        printk("Could't copy %zd bytes from the user\n", ncopied);
    }

    databuf[maxdatalen] = 0;
```

```
    printk("Data from user side: %s\n", databuf);

    return count;
}

static long my_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
        switch(cmd) {
                case WR_VALUE:
                        if( copy_from_user(&value ,(int32_t*) arg, sizeof(value)) )
                        {
                                pr_err("Data Write : Err!\n");
                        }
                        pr_info("Value = %d\n", value);
                        break;
                case RD_VALUE:
                        if( copy_to_user((int32_t*) arg, &value, sizeof(value)) )
                        {
                                pr_err("Data Read : Err!\n");
                        }
                        break;
                default:
                        pr_info("Default\n");
                        break;
        }
        return 0;
```

```
}


MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kiran Thomas Cherian");

module_init(mychardev_init);
module_exit(mychardev_exit);
```

User Application Source Code (ioctl_.c):

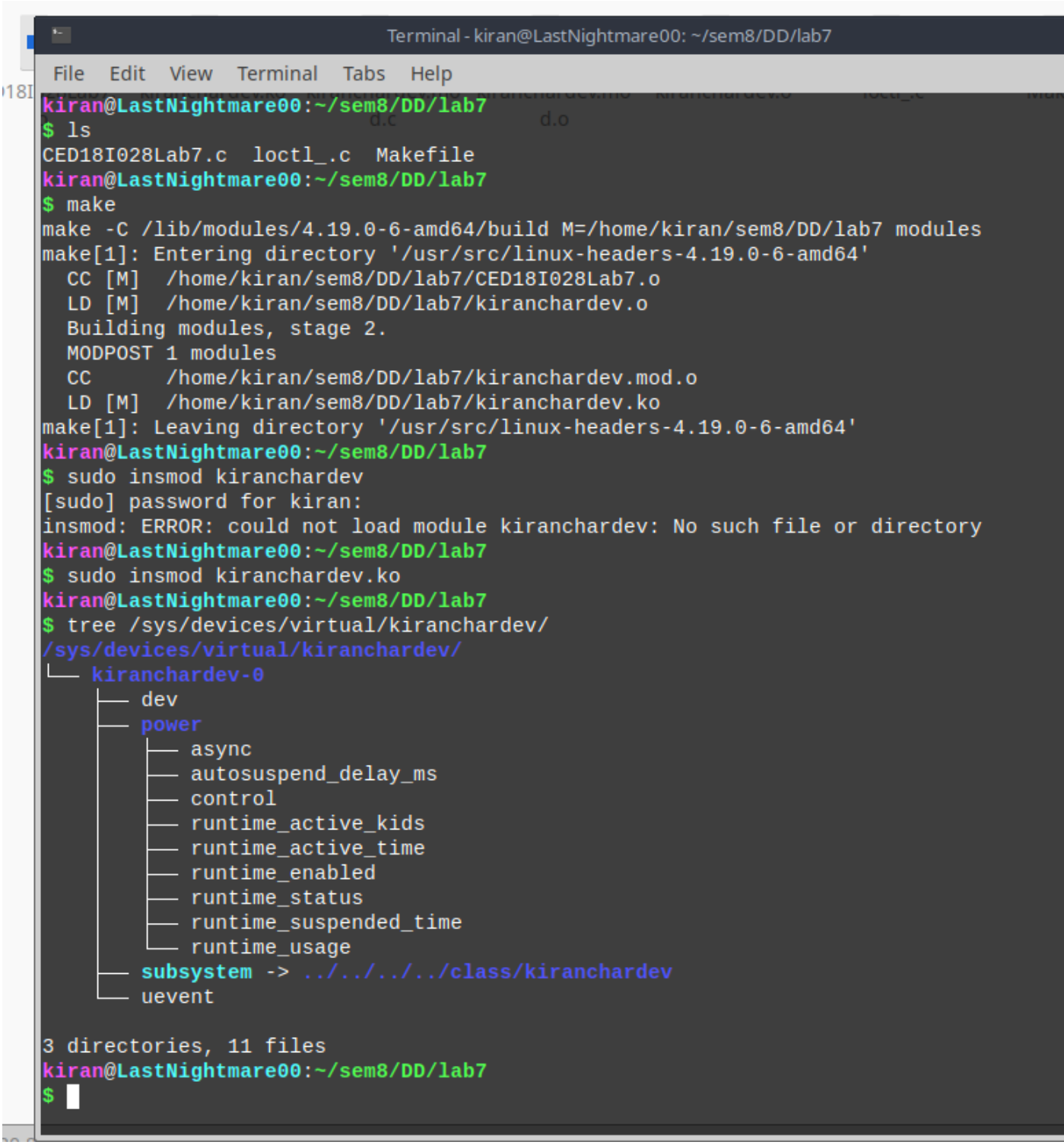
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include<sys/ioctl.h>

#define WR_VALUE _IOW('a','a',int32_t*)
#define RD_VALUE _IOR('a','b',int32_t*)

int main()
{
```

```
	int des;
	int32_t val,num;

	printf("\nOpening kiranchardev Driver\n");
	des = open("/dev/kiranchardev-0", O_RDWR);
	if(des < 0) {
		printf("Cannot open device file!!!\n");
		return 0;
	}

	printf("Enter the number to send\n");
	scanf("%d",&num);
	printf("Writing input to Driver\n");
	ioctl(des, WR_VALUE, (int32_t*) &num);

	printf("Reading number from Driver\n");
	ioctl(des, RD_VALUE, (int32_t*) &val);
	printf("Number is %d\n", val);

	printf("Closing Driver\n");
	close(des);
}
```

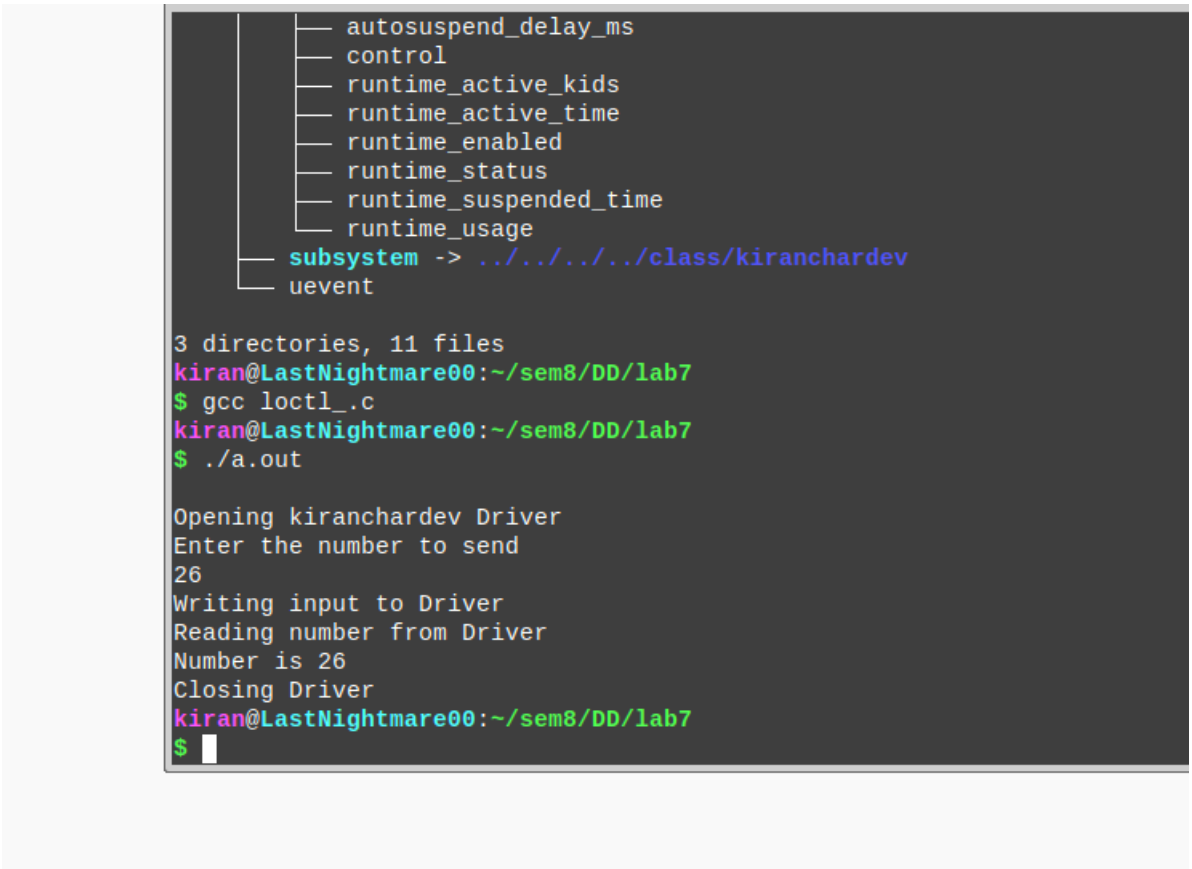
Makefile Contents:

```
BINARY      := kiranchardev

KERNEL      := /lib/modules/$(shell uname -r)/build

ARCH        := x86

C_FLAGS     := -Wall

KMOD_DIR    := $(shell pwd)

TARGET_PATH := /lib/modules/$(shell uname -r)/kernel/drivers/char


OBJECTS := CED18I028Lab7.o


ccflags-y += $(C_FLAGS)


obj-m += $(BINARY).o


$(BINARY)-y := $(OBJECTS)


$(BINARY).ko:

        make -C $(KERNEL) M=$(KMOD_DIR) modules


install:

        cp $(BINARY).ko $(TARGET_PATH)

        depmod -a


uninstall:

        rm $(TARGET_PATH)/$(BINARY).ko

        depmod -a


clean:

        make -C $(KERNEL) M=$(KMOD_DIR) clean
```

Run the make command to compile the source code. Then load the module and verify it is loaded correctly.



Now run the user application(ioctl_.c), which makes use of ioctl() to write and read a number from the device.



Checking log messages and removing module after usage: