Paleti Krishnasai
CED18I039

Device Drivers Lab 7

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

Driver Code:

```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include<linux/slab.h>                 //kmalloc()
#include<linux/uaccess.h>              //copy_to/from_user()
#include <linux/ioctl.h>
#define WR_VALUE _IOW('a','a',int32_t*)
#define RD_VALUE _IOR('a','b',int32_t*)
int32_t value = 0;
dev_t dev = 0;
static struct class *dev_class;
static struct cdev etx_cdev;

/*
** Function Prototypes
*/
static int      __init etx_driver_init(void);
static void     __exit etx_driver_exit(void);
static int      etx_open(struct inode *inode, struct file *file);
static int      etx_release(struct inode *inode, struct file *file);
static ssize_t  etx_read(struct file *filp, char __user *buf, size_t
len,loff_t * off);
static ssize_t  etx_write(struct file *filp, const char *buf, size_t len,
loff_t * off);
```

```
static long     etx_ioctl(struct file *file, unsigned int cmd, unsigned
long arg);

/*
** File operation sturcture
*/
static struct file_operations fops =
{
       .owner          = THIS_MODULE,
       .read           = etx_read,
       .write          = etx_write,
       .open           = etx_open,
       .unlocked_ioctl = etx_ioctl,
       .release        = etx_release,
};

/*
** This function will be called when we open the Device file
*/
static int etx_open(struct inode *inode, struct file *file)
{
       pr_info("Device File Opened...!!!\n");
       return 0;
}

/*
** This function will be called when we close the Device file
*/
static int etx_release(struct inode *inode, struct file *file)
{
       pr_info("Device File Closed...!!!\n");
       return 0;
}

/*
** This function will be called when we read the Device file
*/
static ssize_t etx_read(struct file *filp, char __user *buf, size_t len,
loff_t *off)
{
```

```
        pr_info("Read Function\n");
        return 0;
}

/*
** This function will be called when we write the Device file
*/
static ssize_t etx_write(struct file *filp, const char __user *buf, size_t
len, loff_t *off)
{
        pr_info("Write function\n");
        return len;
}

/*
** This function will be called when we write IOCTL on the Device file
*/
static long etx_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 Init function
*/
static int __init etx_driver_init(void)
{
	/*Allocating Major number*/
	if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) <0){
			pr_err("Cannot allocate major number\n");
			return -1;
	}
	pr_info("Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));
	/*Creating cdev structure*/
	cdev_init(&etx_cdev,&fops);
	/*Adding character device to the system*/
	if((cdev_add(&etx_cdev,dev,1)) < 0){
	    pr_err("Cannot add the device to the system\n");
	    goto r_class;
	}
	/*Creating struct class*/
	if((dev_class = class_create(THIS_MODULE,"etx_class")) == NULL){
	    pr_err("Cannot create the struct class\n");
	    goto r_class;
	}
	/*Creating device*/
	if((device_create(dev_class,NULL,dev,NULL,"etx_device")) == NULL){
	    pr_err("Cannot create the Device 1\n");
	    goto r_device;
	}
	pr_info("Device Driver Insert...Done!!!\n");
	return 0;
r_device:
	class_destroy(dev_class);
r_class:
	unregister_chrdev_region(dev,1);
	return -1;
}

/*
```

```
** Module exit function
*/
static void __exit etx_driver_exit(void)
{
        device_destroy(dev_class,dev);
        class_destroy(dev_class);
        cdev_del(&etx_cdev);
        unregister_chrdev_region(dev, 1);
        pr_info("Device Driver Remove...Done!!!\n");
}
module_init(etx_driver_init);
module_exit(etx_driver_exit);
MODULE_LICENSE("GPL");
```

Test File :

```
#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 fd;
        int32_t value, number;
        printf("*********************************\n");
        printf("*******WWW.EmbeTronicX.com*******\n");
        printf("\nOpening Driver\n");
        fd = open("/dev/etx_device", O_RDWR);
        if(fd < 0) {
                printf("Cannot open device file...\n");
                return 0;
        }
        printf("Enter the Value to send\n");
```

```
        scanf("%d",&number);
        printf("Writing Value to Driver\n");
        ioctl(fd, WR_VALUE, (int32_t*) &number);
        printf("Reading Value from Driver\n");
        ioctl(fd, RD_VALUE, (int32_t*) &value);
        printf("Value is %d\n", value);
        printf("Closing Driver\n");
        close(fd);
}
```

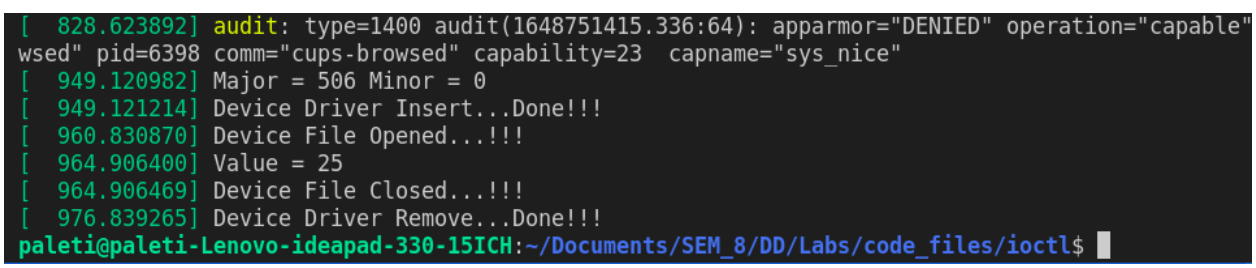
MakeFile:

```
obj-m += io.o
KDIR = /lib/modules/$(shell uname -r)/build
all:
   make -C $(KDIR)  M=$(shell pwd) modules
clean:
   make -C $(KDIR)  M=$(shell pwd) clean
```

Output :

```
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM_8/DD/Labs/code_files/ioctl$ gcc test.c -o test
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM_8/DD/Labs/code_files/ioctl$ sudo insmod io.ko
[sudo] password for paleti:
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM_8/DD/Labs/code_files/ioctl$ sudo ./test
*******************************
*******WWW.EmbeTronicX.com*******

Opening Driver
Enter the Value to send
25
Writing Value to Driver
Reading Value from Driver
Value is 25
Closing Driver
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM_8/DD/Labs/code_files/ioctl$ sudo rmmod io
paleti@paleti-Lenovo-ideapad-330-15ICH:~/Documents/SEM_8/DD/Labs/code_files/ioctl$
```



Reference : https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux