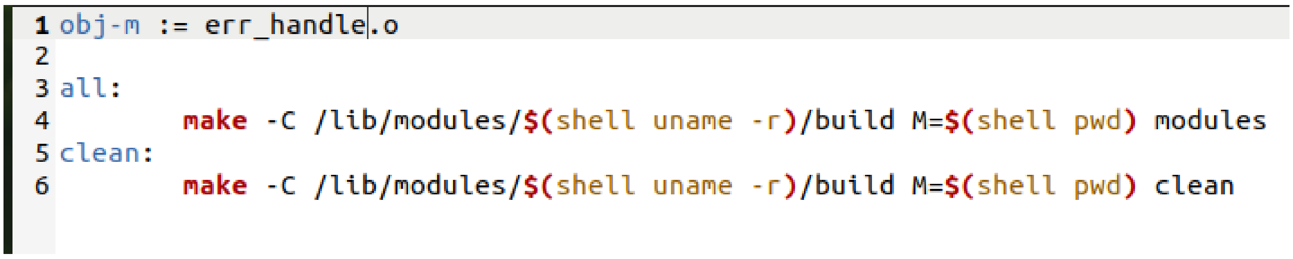
DEVICE DRIVERS LAB 7

DONE BY:
A S V DHANUSH
CS20B1057

Error handling

Makefile



```
obj-m := err_handle.o

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

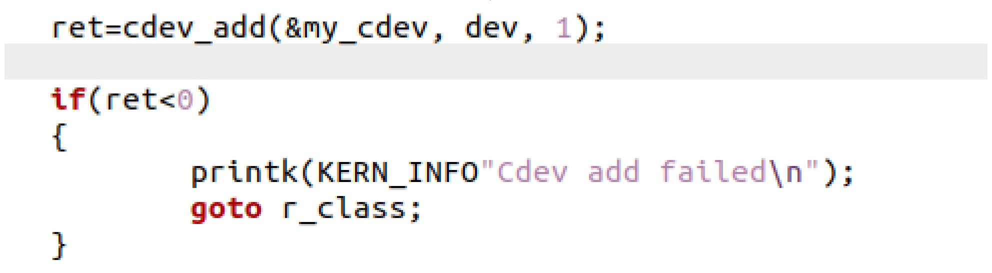
err_handle.c

```
#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>
#include<linux/uaccess.h>
#include<linux/ioctl.h>
#include<linux/err.h>

#define mem_size 0  // Macro for memory size


int32_t val=0;

dev_t dev = 0;
struct device *dev_my;
static struct class *dev_class;
static struct cdev my_cdev;

uint8_t *kernel_buffer;

static int __init chr_driver_init(void);
static void __exit chr_driver_exit(void);

static int my_open(struct inode *inode, struct file *file);
static int my_release(struct inode *inode, struct file *file);
static ssize_t my_read(struct file *filp, char __user *buf, size_t len, loff_t *off);
static ssize_t my_write(struct file *filp, const char *buf, size_t len, loff_t *off);
static struct file_operations fops=
{
        .owner          =       THIS_MODULE,
        .read           =       my_read,
        .write          =       my_write,
        .open           =       my_open,
        .release        =       my_release,
};

static int my_open(struct inode *inode, struct file *file)
{
        // Creating physical Memory
        if((kernel_buffer = kmalloc(mem_size, GFP_KERNEL))==0)
        {
                printk(KERN_INFO"Can NOT allocate the memory to kernel ...\n");
                return -1;
        }
        printk(KERN_INFO "Device File Opened...\n");
        return 0;
}

static int my_release(struct inode *inode, struct file *file)
{
        kfree(kernel_buffer);
        printk(KERN_INFO"Device File Closed...\n");
        return 0;
}

static ssize_t my_read(struct file *filp, char __user *buf, size_t  len, loff_t *loff)
{
        copy_to_user(buf, kernel_buffer,mem_size);
        printk(KERN_INFO "Data Read: DONE....\n");
        return mem_size;
}

static ssize_t my_write(struct file *filp, const char __user *buf, size_t len, loff_t  *loff)
{
        copy_from_user(kernel_buffer, buf, len);
        printk(KERN_INFO "Data is written Successfully...\n");
        return len;
}


static int __init chr_driver_init(void)
{
      int ret;
        // Allocating Major Number Dynamically
        ret=alloc_chrdev_region(&dev, 0, 1, "my_Dev");
        if(ret<0){

                goto out;
        }
        printk(KERN_INFO"Major = %d and Minor = %d..\n", MAJOR(dev),MINOR(dev));

        // Creating  cdev structure

        cdev_init(&my_cdev, &fops);

        // Addding Character device to the system
        my_cdev.owner= THIS_MODULE;
        ret=cdev_add(&my_cdev, dev, 1);

        //TAKING A FLAG TO SO THAT WE GO TO THE ERROR (IF) CONDITION
        bool flag = =true;

        if(ret<0 || flag == true )
        {       printk(KERN_INFO"I MADE THIS ERROR!");
                printk(KERN_INFO"Cdev add failed\n");
                goto r_class;
        }

        // Creating Struct Class
        dev_class = class_create(THIS_MODULE,"my_class");
        if(IS_ERR(dev_class))
        {
                printk(KERN_INFO"class creation failed\n");
                ret= PTR_ERR(dev_class);
                goto r_fail;    // Unrecognize the character device
        }

        // Creating Device
        dev_my = device_create(dev_class, NULL, dev, NULL,"my_device");
        if(IS_ERR(dev_my))
        {
                printk(KERN_INFO"Can NOT create the device...\n");
                ret= PTR_ERR(dev_my);
                goto r_device;
        }

        printk(KERN_INFO"Device Driver is inserted properly DONE...\n");
        return 0;


r_device:
        class_destroy(dev_class);
r_fail:
        cdev_del(&my_cdev);

r_class:
        unregister_chrdev_region(dev,1);
out:
    return ret;
}
```

```
void __exit chr_driver_exit(void)
{
        device_destroy(dev_class, dev);
        class_destroy(dev_class);
        cdev_del(&my_cdev);
        unregister_chrdev_region(dev,1);
        printk(KERN_INFO"Device Driver is Removed Successfully...\n");
}

module_init(chr_driver_init);
module_exit(chr_driver_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("IIITDM KANCHEEPURAM");
MODULE_DESCRIPTION("Error Handling in Character Device Driver");
```

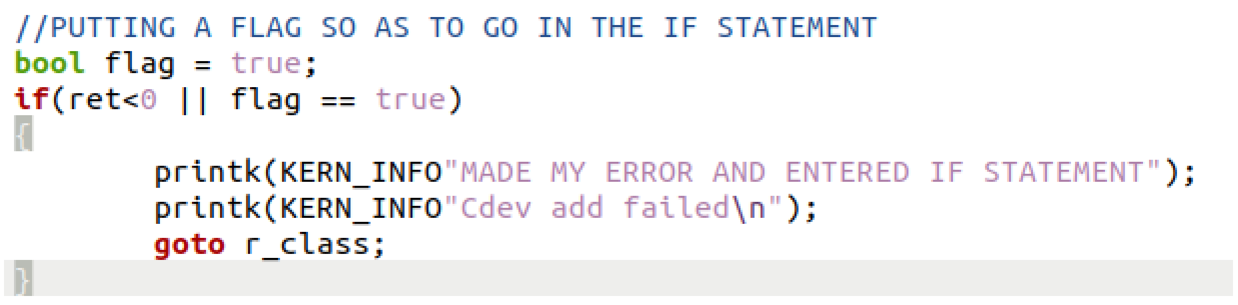
<u>Part of code where I made the error</u>

Before modification



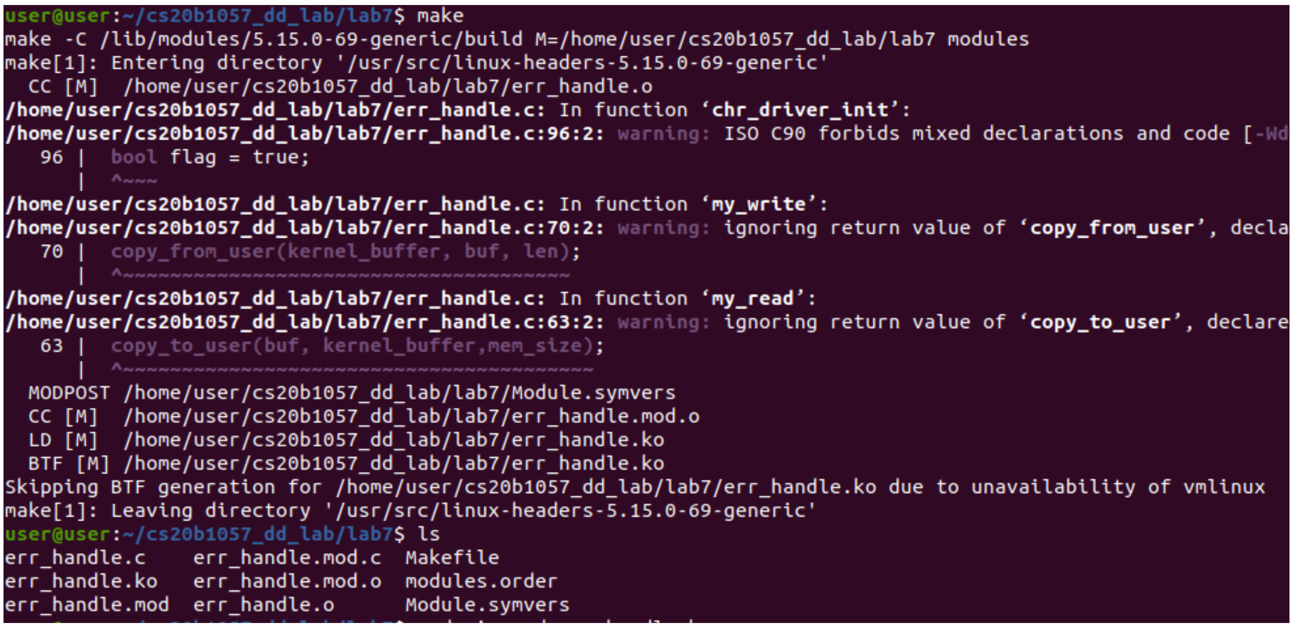
```
ret=cdev_add(&my_cdev, dev, 1);

if(ret<0)
{
        printk(KERN_INFO"Cdev add failed\n");
        goto r_class;
}
```

After modification



```
//PUTTING A FLAG SO AS TO GO IN THE IF STATEMENT
bool flag = true;
if(ret<0 || flag == true)
{
        printk(KERN_INFO"MADE MY ERROR AND ENTERED IF STATEMENT");
        printk(KERN_INFO"Cdev add failed\n");
        goto r_class;
}
```

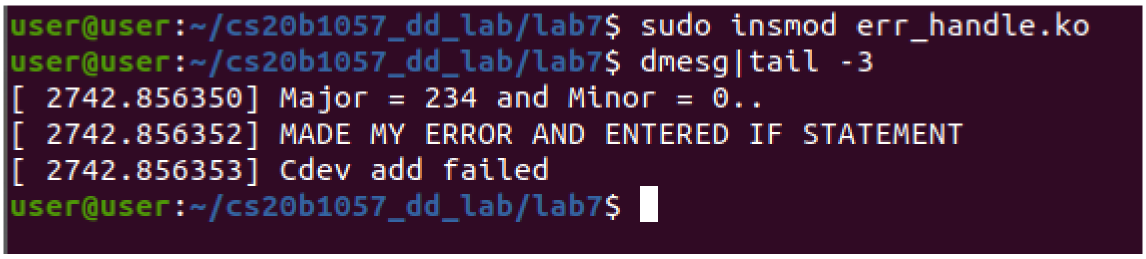
Using make to create err_handle.ko



```
user@user:~/cs20b1057_dd_lab/lab7$ make
make -C /lib/modules/5.15.0-69-generic/build M=/home/user/cs20b1057_dd_lab/lab7 modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-69-generic'
  CC [M]  /home/user/cs20b1057_dd_lab/lab7/err_handle.o
/home/user/cs20b1057_dd_lab/lab7/err_handle.c: In function ‘chr_driver_init’:
/home/user/cs20b1057_dd_lab/lab7/err_handle.c:96:2: warning: ISO C90 forbids mixed declarations and code [-Wd
   96 |  bool flag = true;
      |  ^~~~
/home/user/cs20b1057_dd_lab/lab7/err_handle.c: In function ‘my_write’:
/home/user/cs20b1057_dd_lab/lab7/err_handle.c:70:2: warning: ignoring return value of ‘copy_from_user’, decla
   70 |  copy_from_user(kernel_buffer, buf, len);
      |  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/user/cs20b1057_dd_lab/lab7/err_handle.c: In function ‘my_read’:
/home/user/cs20b1057_dd_lab/lab7/err_handle.c:63:2: warning: ignoring return value of ‘copy_to_user’, declare
   63 |  copy_to_user(buf, kernel_buffer,mem_size);
      |  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  MODPOST /home/user/cs20b1057_dd_lab/lab7/Module.symvers
  CC [M]  /home/user/cs20b1057_dd_lab/lab7/err_handle.mod.o
  LD [M]  /home/user/cs20b1057_dd_lab/lab7/err_handle.ko
  BTF [M] /home/user/cs20b1057_dd_lab/lab7/err_handle.ko
Skipping BTF generation for /home/user/cs20b1057_dd_lab/lab7/err_handle.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-69-generic'
user@user:~/cs20b1057_dd_lab/lab7$ ls
err_handle.c    err_handle.mod.c  Makefile
err_handle.ko   err_handle.mod.o  modules.order
err_handle.mod  err_handle.o      Module.symvers
```

trying to insert the module using sudo insmod
and using dmesg to check if module is inserted properly or not

(NOTE : Since we made an error the device is not inserted and hence the custom printk statements are printed)



```
user@user:~/cs20b1057_dd_lab/lab7$ sudo insmod err_handle.ko
user@user:~/cs20b1057_dd_lab/lab7$ dmesg|tail -3
[ 2742.856350] Major = 234 and Minor = 0..
[ 2742.856352] MADE MY ERROR AND ENTERED IF STATEMENT
[ 2742.856353] Cdev add failed
user@user:~/cs20b1057_dd_lab/lab7$
```

we can observe that our custom printk statements have been printed and the error has been handled successfully.