CS3003D: Operating Systems
Assignment 2- Group 03

Ganesh G B190466CS

Pavithra Rajan B190632CS

Rose S Jose B190839CS

Shada Faisal B190180CS

Vishnu C B190402CS

Problem Statement

Creating a simple device driver (character device) for the compiled kernel (in Assignment-1) and test it with a sample application.

Methodology

A character device driver is one that transfers data directly to and from a user process and it can perform basic read-write operations

1. The drive driver code should be saved into a *C* program (*driver.c*)
2. All the details regarding the compilation of the driver module should be listed down into a *Makefile* which can then compile the module in a simple "*make*" command
3. The make command generates a module (*driver.ko*) that needs to be loaded onto the kernel using the "*insmod*" command
4. The device must be configured into the dev directory
5. The access mode of the file needs to be updated providing read and write permission for the device driver file for all user classes
6. A user-space application must be drafted to test out the device driver
7. Run and verify the output of the test application

Explanation

Character device Driver

A character device is one of the simplest ways to communicate with a module in the Linux kernel. These files are grouped into the /dev directory. System calls like open, read, write, close, etc. are redirected by the operating system to the device driver associated with the physical device. The device driver is a kernel component (usually a module) that interacts with a hardware device.

We have designed a character device driver that contains the functions to open and close the device driver. In this program, we have defined problems to write and read to the user-spaced program. In the driver file (i.e, driver.c) we don't have <**stdio.h**> (which is a user-space header), instead we use the analogous <**kernel.h**> (which is a kernel space header).

Headers

#include<linux/init.h> -- The macros __init and __exit is defined in this header.

#include<linux/module.h> -- MODULE _LICENSE, AUTHOR DESCRIPTION is all defined under this header

#include<linux/fs.h> --This is for the file structure.

#include<asm/uaccess.h> --To communicate between kernel and user we have used function copy to user and copy from user this header is for those.

#include<linux/kernel.h> -- kernel space header

Driver functions

static int OPEN_DEVICE(struct inode *inod, struct file *fil);   - This function is for opening a device file.

static ssize_t READ_DEVICE(struct file *filep,char *buf,size_t len,loff_t *off);   -This function is to read from the user space program

static ssize_t WRITE_DEVICE(struct file *flip,const char *buff,size_t len,loff_t *off);   -This function is to write to the user space program.

static int CLOSE_DEVICE(struct inode *inod,struct file *fil);   - This function is for closing a device file.

_init and _exit

```
module_init(entry);

module_exit(exiting);
```

In the "*driver.c*" module, entry and exit are the loading and unloading functions of the module. Both *_init* and *_exit* are macros. They encapsulate the compiler attributes used by the kernel. The functions modified by them are placed in specific segments by the compiler. For example, the functions modified by *_init* are placed in the. init.text segment. The functions in this segment are cleared after the initialization phase is completed to reduce unnecessary memory consumption. Both *_init* and *_exit* are optional, but *module_ini*t and *module_exit* must be added, and only they can ultimately lead to load and unload functions being called.

APIs Involved

In the read and write function, the following kernel APIs have been used

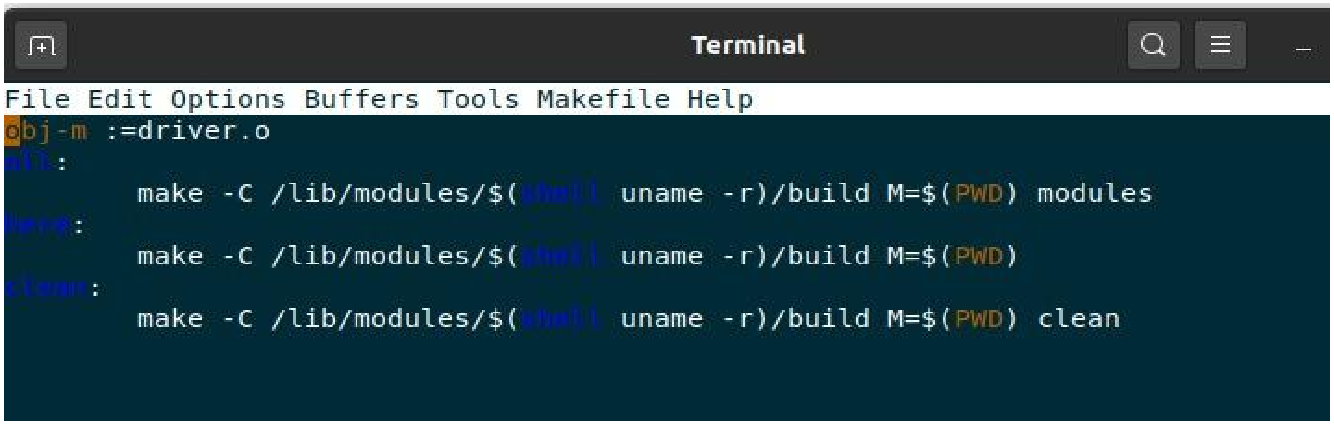
- copy_to_user - copies a block of data from the kernel to user-space.
- copy_from_user - copies a block of data from user space to kernel.

Screenshots

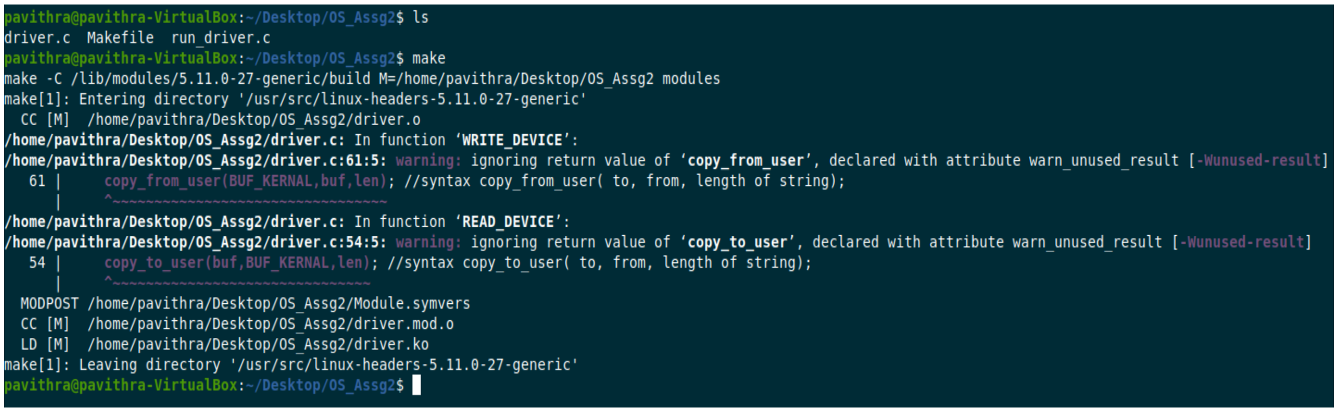
The following steps are implemented to run the character device driver file.

1. **$make**

   The make command invokes the execution of the Makefile. It is a special file that contains the shell commands that we create to maintain the project. The following Makefile invokes the kernel's build system from the kernel source, and the kernel's Makefile will in turn, invoke our Makefile to build our driver.



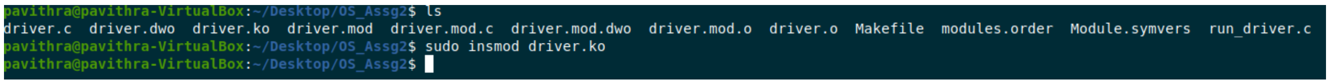
```
obj-m :=driver.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
here:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```



```
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ ls
driver.c  Makefile  run_driver.c
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ make
make -C /lib/modules/5.11.0-27-generic/build M=/home/pavithra/Desktop/OS_Assg2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.11.0-27-generic'
  CC [M]  /home/pavithra/Desktop/OS_Assg2/driver.o
/home/pavithra/Desktop/OS_Assg2/driver.c: In function 'WRITE_DEVICE':
/home/pavithra/Desktop/OS_Assg2/driver.c:61:5: warning: ignoring return value of 'copy_from_user', declared with attribute warn_unused_result [-Wunused-result]
   61 |     copy_from_user(BUF_KERNAL,buf,len); //syntax copy_from_user( to, from, length of string);
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/pavithra/Desktop/OS_Assg2/driver.c: In function 'READ_DEVICE':
/home/pavithra/Desktop/OS_Assg2/driver.c:54:5: warning: ignoring return value of 'copy_to_user', declared with attribute warn_unused_result [-Wunused-result]
   54 |     copy_to_user(buf,BUF_KERNAL,len); //syntax copy_to_user( to, from, length of string);
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  MODPOST /home/pavithra/Desktop/OS_Assg2/Module.symvers
  CC [M]  /home/pavithra/Desktop/OS_Assg2/driver.mod.o
  LD [M]  /home/pavithra/Desktop/OS_Assg2/driver.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.11.0-27-generic'
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ 
```

2. **$sudo insmod driver.ko**

   The insmod command loads the module driver.ko. Users can add or remove functionalities to the kernel while the system is running. These 'programs' that can be added to the kernel at runtime are called 'module' and built into individual files with .ko(Kernel Object) extension.

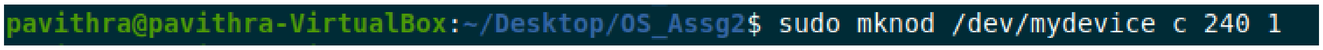


```
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ ls
driver.c  driver.dwo  driver.ko  driver.mod  driver.mod.c  driver.mod.dwo  driver.mod.o  driver.o  Makefile  modules.order  Module.symvers  run_driver.c
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ sudo insmod driver.ko
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ 
```

3. **$sudo mknod /dev/mydevice c 240 1**

   A device must be configured in the folder **/dev,** providing a class, a major and a minor number in order to identify it. This shell command will create a device called **mydevice** as a character device("c" is for a char device) with the major number 240 and minor number 1. The command is run as a superuser.

   The major number is used by the driver initialization for associating a module driver to a specific device. The minor number is used by the device driver programmer to access different functions in the same device.

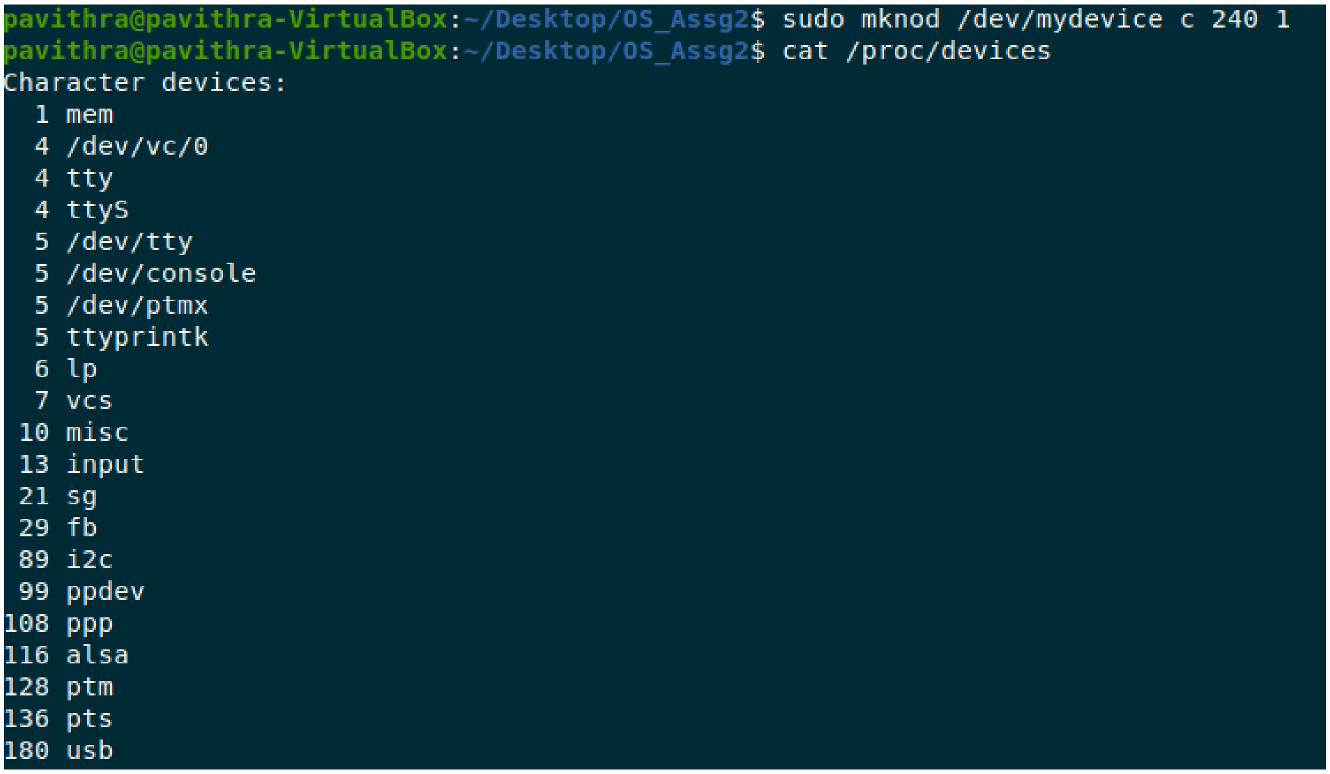


```
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ sudo mknod /dev/mydevice c 240 1
```

4. **$cat /proc/devices**

   The cat command reads the data from the file and gives its content as output.

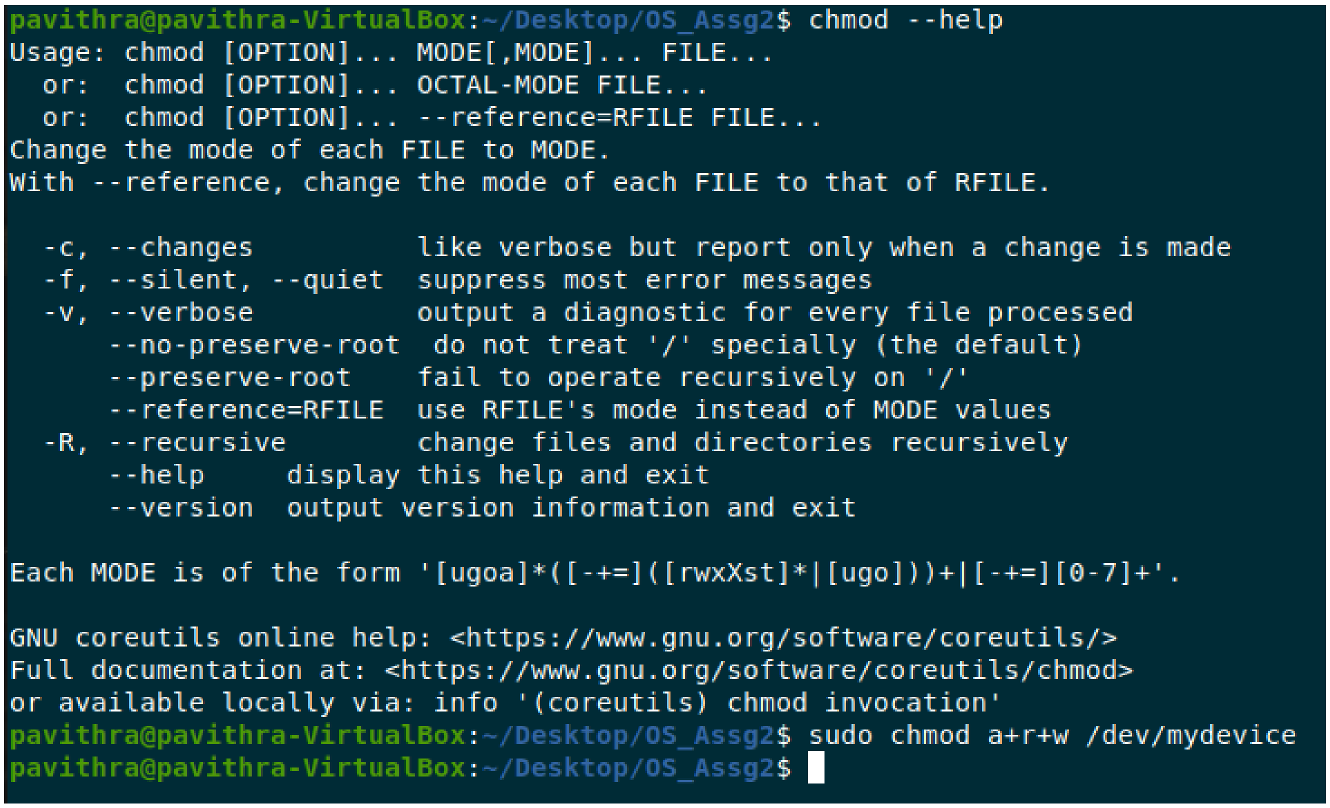
   This file displays the various character and block devices currently configured (not including devices whose modules are not loaded). It includes the major number and the name of the device.



```
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ sudo mknod /dev/mydevice c 240 1
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 89 i2c
 99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
180 usb
```

5. **$sudo chmod a+r+w /dev/mydevice**

   The **chmod** command is used to change the access mode of a file. The above command gives the permission to read and write our device driver file to all user classes, including owner, group and others.



```
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ chmod --help
Usage: chmod [OPTION]... MODE[,MODE]... FILE...
  or:  chmod [OPTION]... OCTAL-MODE FILE...
  or:  chmod [OPTION]... --reference=RFILE FILE...
Change the mode of each FILE to MODE.
With --reference, change the mode of each FILE to that of RFILE.

  -c, --changes          like verbose but report only when a change is made
  -f, --silent, --quiet  suppress most error messages
  -v, --verbose          output a diagnostic for every file processed
      --no-preserve-root  do not treat '/' specially (the default)
      --preserve-root    fail to operate recursively on '/'
      --reference=RFILE  use RFILE's mode instead of MODE values
  -R, --recursive        change files and directories recursively
      --help     display this help and exit
      --version  output version information and exit

Each MODE is of the form '[ugoa]*([-+=]([rwxXst]*|[ugo]))+|[-+=][0-7]+'.

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/chmod>
or available locally via: info '(coreutils) chmod invocation'
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ sudo chmod a+r+w /dev/mydevice
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ 
```

User Space Application

To test our character device driver, a sample program called run_diver.c is created.

int fd=open("/dev/mydevice",O_RDWR)

The device file is opened for reading and writing. The user inputs a string which is then written to the device.

write(fd,buf,strlen(buf))

Then the string is read from the device, and the content is displayed. Hence the working of the device driver is verified.

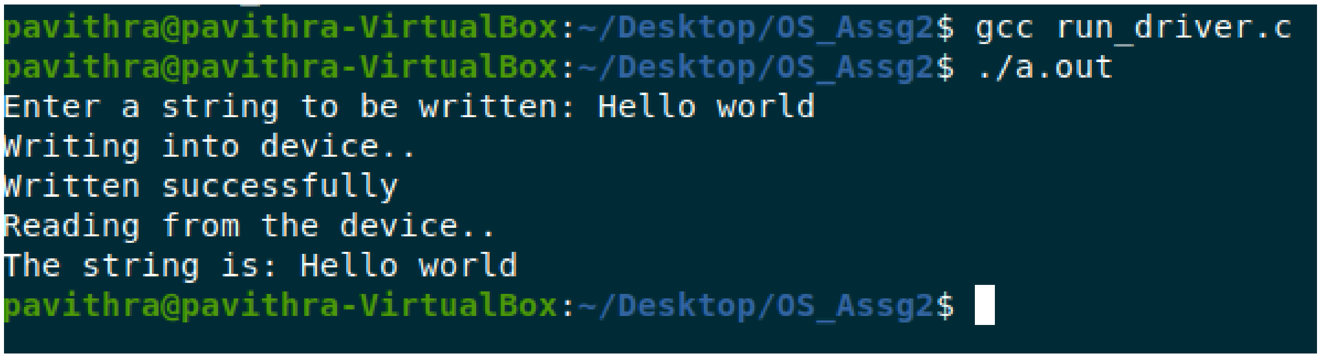
read(fd,rbuf,100);

printf("%s",rbuf);

6. **$gcc run_driver.c**

$ ./a.out

The program to test the device and driver, run_driver.c is compiled and executed.



```
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ gcc run_driver.c
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$ ./a.out
Enter a string to be written: Hello world
Writing into device..
Written successfully
Reading from the device..
The string is: Hello world
pavithra@pavithra-VirtualBox:~/Desktop/OS_Assg2$
```