CS3003D: OPERATING SYSTEMS

ASSIGNMENT-2 : **CHARACTER DEVICE DRIVER**

Group No: 34

Group Members

1. B190605CS- K. Jogi Naidu
2. B190394CS- Keerthi N P
3. B190056CS- Yuvetha Ganesan
4. B191143CS- Panasa Teja
5. B190391CS- Bhavana Ketavath

Problem Statement:

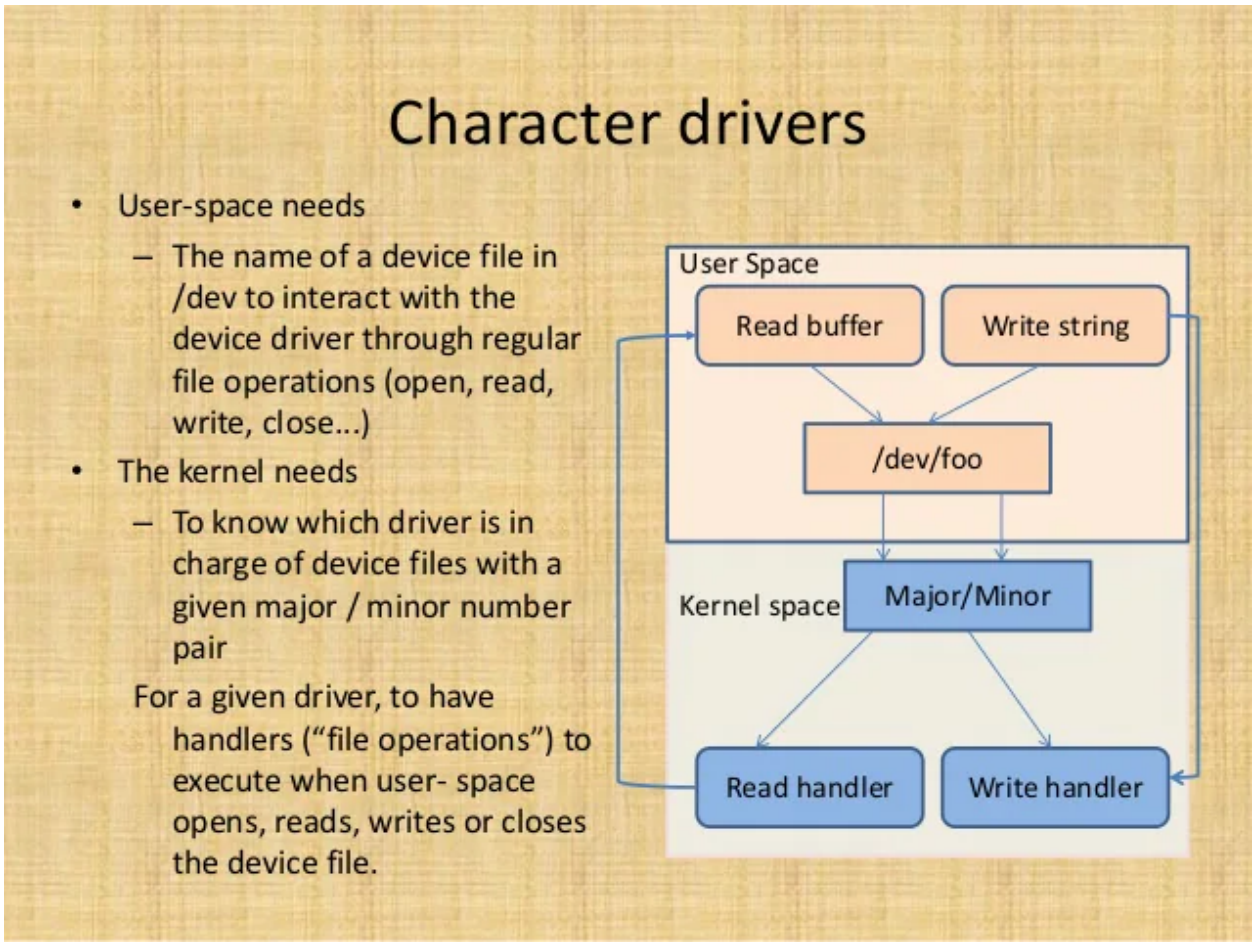
Create a simple device driver (for a character device) and test it with a sample application.

METHODOLOGY:

1. Execution of character device driver
2. Makefile and Source Code
3. Commands that are followed

EXPLANATION:

Character Device Driver



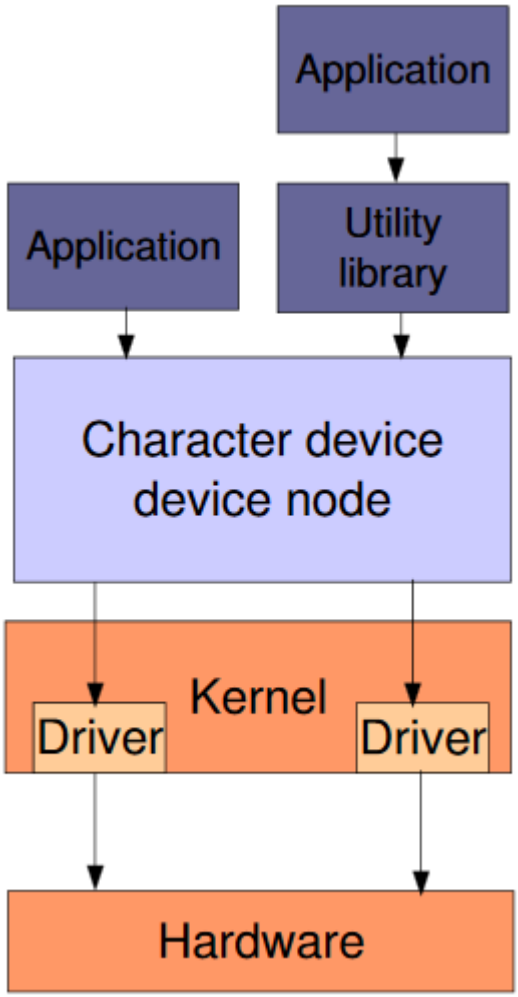
A "Character Device" typically transfers data to and from a user application. They behave like pipes or serial ports, instantly reading or writing the byte data in a character-by-character stream. They provide the framework for many typical drivers, such as those that are required for interfacing to serial communications, video capture, and audio devices. The main alternative to a character device is a "Block-Device". Block devices behave similarly to regular files, allowing a buffered array of cached data to be viewed or manipulated with operations such as reads, writes, and seeks. Both device types can be accessed through device files that are attached to the file system tree.

In this article we will write a character driver which will support various functions like open, close, read and write.

The Character Device Driver Code

The functionality of the device is to get an input string from the user (through the user program). The write function call writes it into a write buffer in the user space, which is then copied onto a kernel space buffer. This is then reversed and stored in a string in the kernel space.

When the read function is called, the reversed string is read from the string in kernel space and displayed as the output.



There are normal init() and exit() functions. However, there are additional file operations functions that are required for the character device and the basic functions are as follows:

module_init(void);

module_exit(void);

int assgn2_open (struct inode * ,struct file *);

ssize_t assgn2_read(struct file *, char __user *,size_t , loff_t *);

ssize_t assgn2_write(struct file *, const char __user * ,size_t , loff_t *);

int assgn2_close (struct inode *,struct file *);

Command Lines to Execute the Driver

1) write character driver code

Character driver code contains following components

Register driver

At the time of init , driver needs to register with the kernel. We can register driver by the following function.

register_chrdev (major_number,device,file_operation)

```
register_chrdev(240/* Major Number */,
                "Simple Char Drv"     /* Name of the driver */,
                &assgn2_file_operations    /* File Operations */ );
```

major_ number :- In our driver code major number is 240 .

device:- device is the name of the device for which you are writing the driver. In our character driver we are keeping the name of our device as "**Simple Char Drv**".

file_operation:- File operation is the structure which contains the functions supported by your device.

Unregister driver

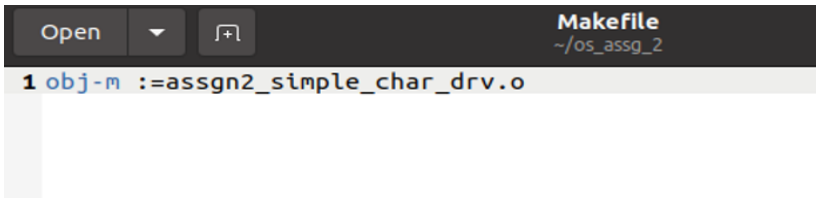
At the time of exit our driver needs to be unregister with the kernel. We can unregister it by using following function

unregister_chrdev(major_number, device)

```
unregister_chrdev(240,"Simple Char Dev");
```

2) Build Driver

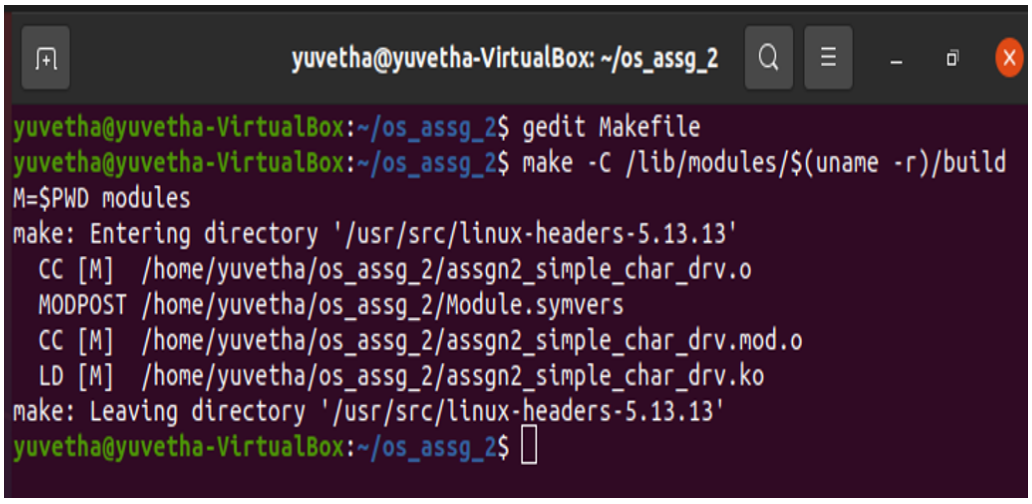
To build driver we need to write, make file and save it in the same directory where our driver code is save. Below is the example of make file.



save this file as Makefile

3) To compile the modules of driver run

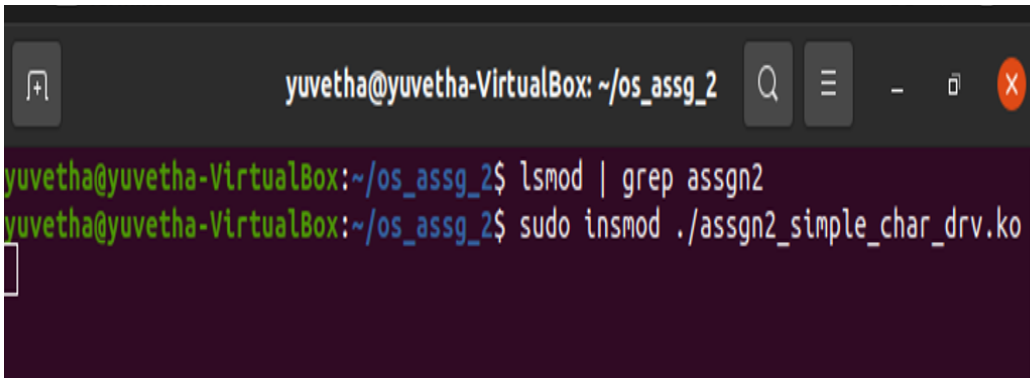
make -C /lib/modules/$(uname -r)/build M=$PWD modules



4) To insert our compiled module to the kernel run

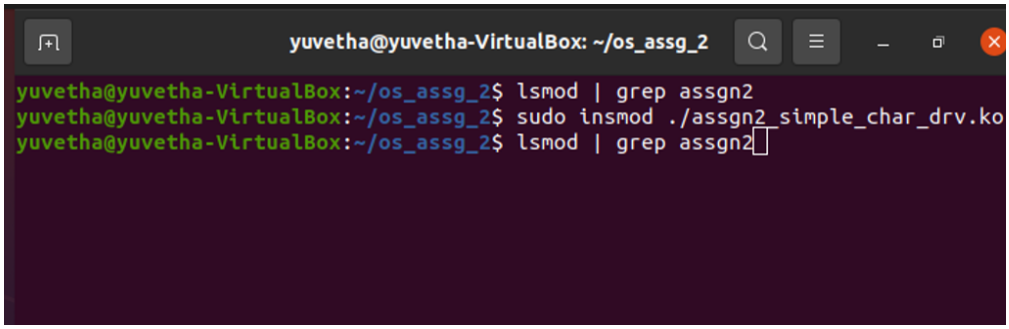
sudo insmod ./assgn2_simple_char_drv.ko

We can insert our driver module in kernel by using insmod command.

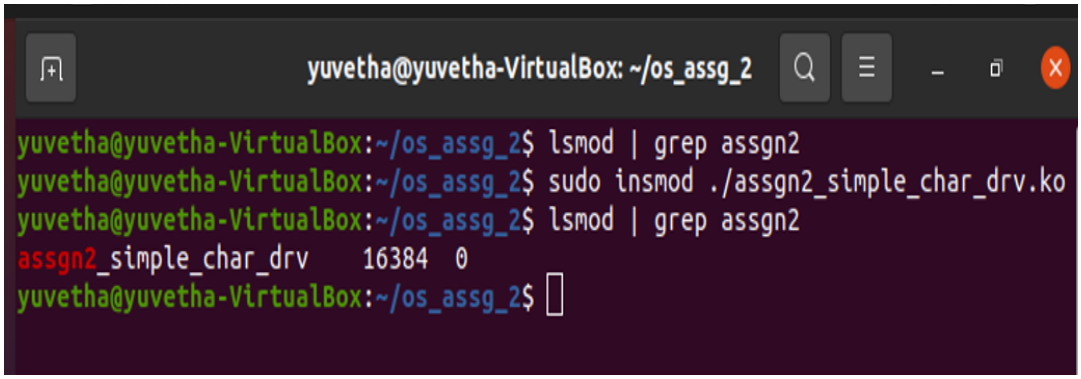


5) To verify whether module inserted or not run

lsmod | grep assgn2



This shows a list of inserted modules in the kernal having name char driver in the beginning, our driver module should display the indicating amount of memory taken by our driver in bytes.



6) To show list of character devices and block devices run

cat /proc/devices

This outputs the major number and name of the device, and is broken into two major sections: Character devices and block devices. Our device come under character devices.

```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ 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
189 usb_device
204 ttyMAX
```

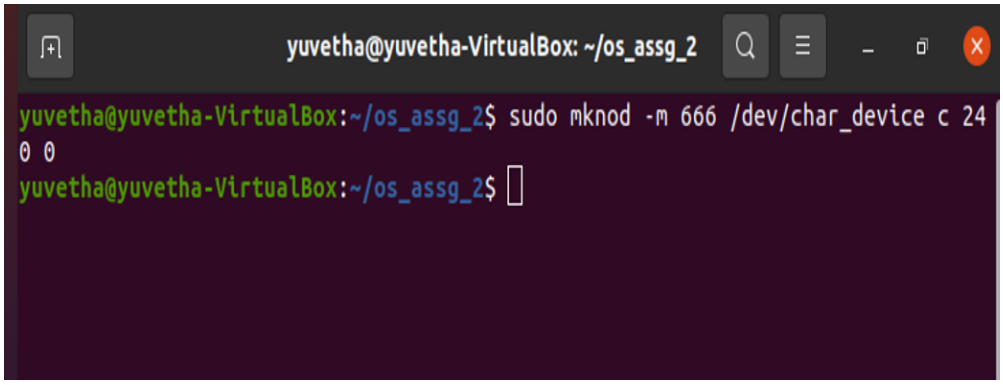
```
yuvetha@yuvetha-VirtualBox: ~/os_assg_2
251 dimmctl
252 ndctl
253 tpm
254 gpiochip

Block devices:
  7 loop
  8 sd
  9 md
 11 sr
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
253 device-mapper
254 mdp
259 blkext
yuvetha@yuvetha-VirtualBox:~/os_assg_2$
```

7) To make the device accessible run

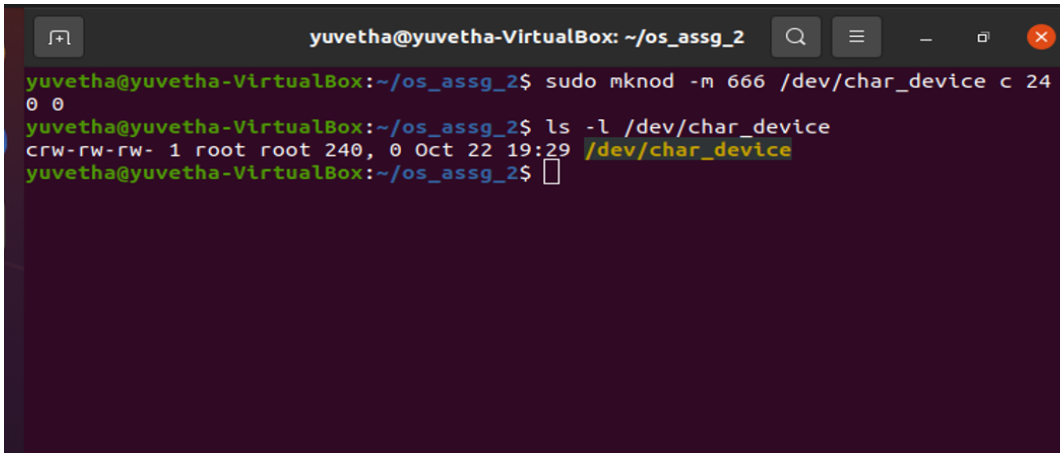
sudo mknod -m 666 /dev/char_device c 240 0

This enables everyone to access our device for read and write functionality. 240 indicate major number and 0 indicates the minor number and in our case device driver had major no. 240 and minor no. 0.



8) To verify whether char_device is available for the given code to run

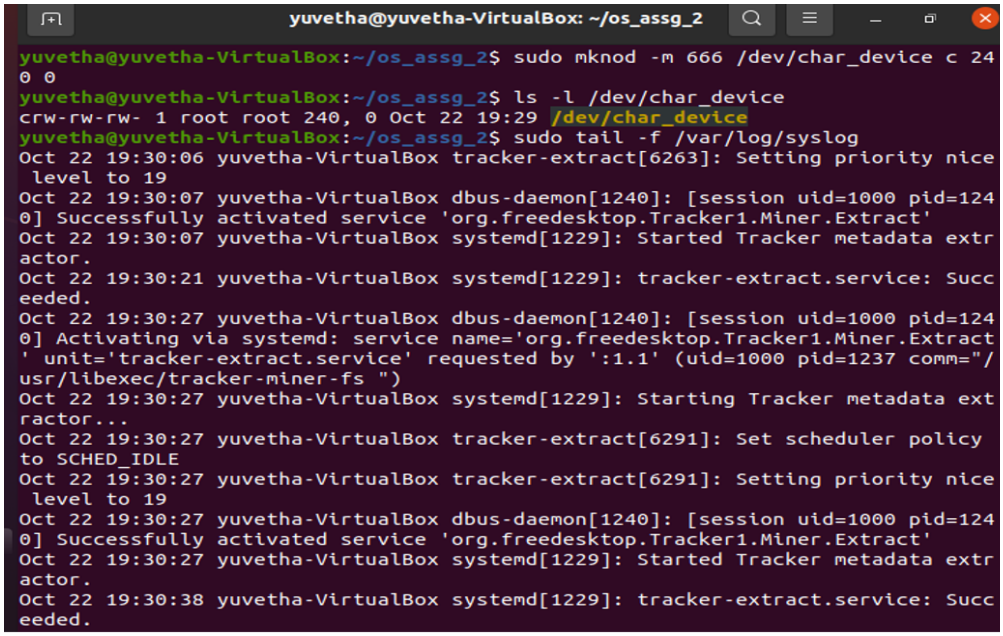
ls -l /dev/char_device



9) To show the system log console run

sudo tail -f /var/log/syslog

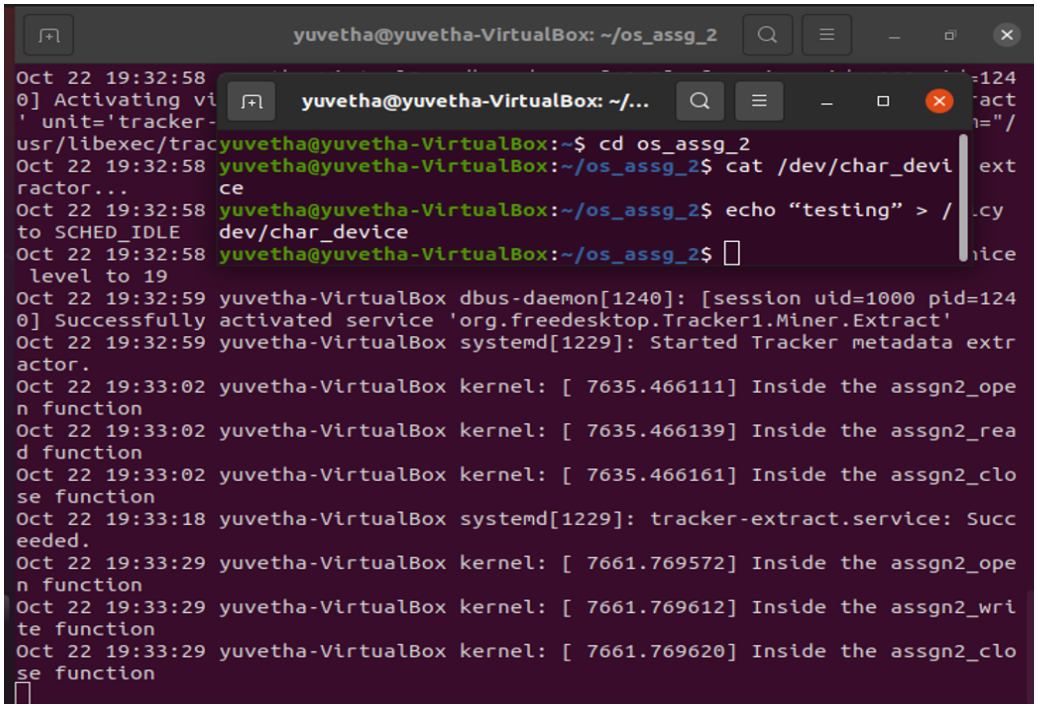
To show the current execution of driver and the past records.



10) check the functionality of our device driver

Cat opens /dev/char_device , read it and on completion of read close the device.

echo opens /dev/char_device ,writes “testing” into that and closes it.


yuvetha@yuvetha-VirtualBox: ~/os_assg_2
yuvetha@yuvetha-VirtualBox: ~/...
Oct 22 19:32:58
0] Activating vi
' unit='tracker-
usr/libexec/trac
Oct 22 19:32:58
ractor...
Oct 22 19:32:58
to SCHED_IDLE
Oct 22 19:32:58
level to 19
yuvetha@yuvetha-VirtualBox:~$ cd os_assg_2
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ cat /dev/char_devi
ce
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ echo “testing” > /
dev/char_device
yuvetha@yuvetha-VirtualBox:~/os_assg_2$
Oct 22 19:32:59 yuvetha-VirtualBox dbus-daemon[1240]: [session uid=1000 pid=1240] Successfully activated service 'org.freedesktop.Tracker1.Miner.Extract'
Oct 22 19:32:59 yuvetha-VirtualBox systemd[1229]: Started Tracker metadata extractor.
Oct 22 19:33:02 yuvetha-VirtualBox kernel: [7635.466111] Inside the assgn2_open function
Oct 22 19:33:02 yuvetha-VirtualBox kernel: [7635.466139] Inside the assgn2_read function
Oct 22 19:33:02 yuvetha-VirtualBox kernel: [7635.466161] Inside the assgn2_close function
Oct 22 19:33:18 yuvetha-VirtualBox systemd[1229]: tracker-extract.service: Succeeded.
Oct 22 19:33:29 yuvetha-VirtualBox kernel: [7661.769572] Inside the assgn2_open function
Oct 22 19:33:29 yuvetha-VirtualBox kernel: [7661.769612] Inside the assgn2_write function
Oct 22 19:33:29 yuvetha-VirtualBox kernel: [7661.769620] Inside the assgn2_close function