

# **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

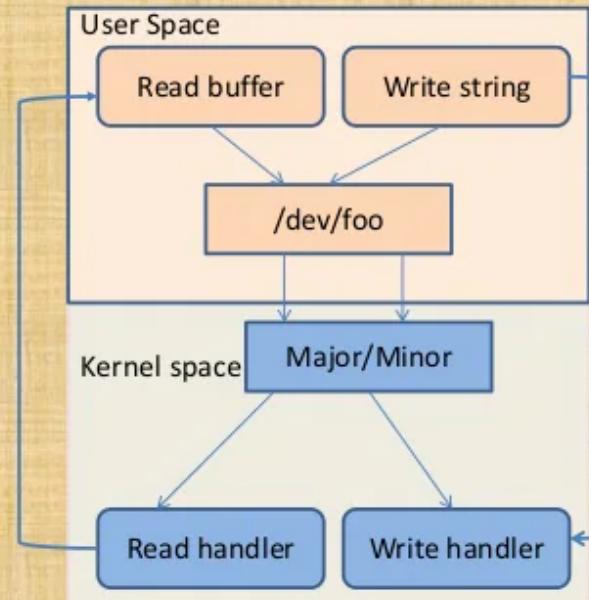
### **EXPLANATION:**

## Character Device Driver

# Character drivers

- User-space needs
  - The name of a device file in `/dev` to interact with the device driver through regular file operations (open, read, write, close...)
- The kernel needs
  - To know which driver is in charge of device files with a given major / minor number pair

For a given driver, to have handlers (“file operations”) to execute when user-space opens, reads, writes or closes the device file.



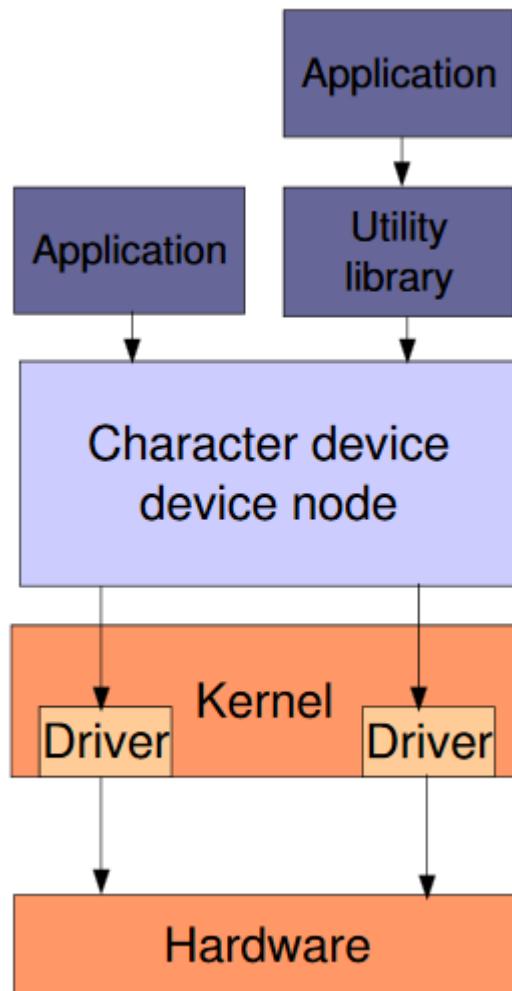
A “Character Device” typically transfers data to and from a user application. They behave like pipes or serial ports, instantly reading or writing the byte data in a character-by-character stream. They provide the framework for many typical drivers, such as those that are required for interfacing to 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 unregistered 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 saved. Below is the example of make file.



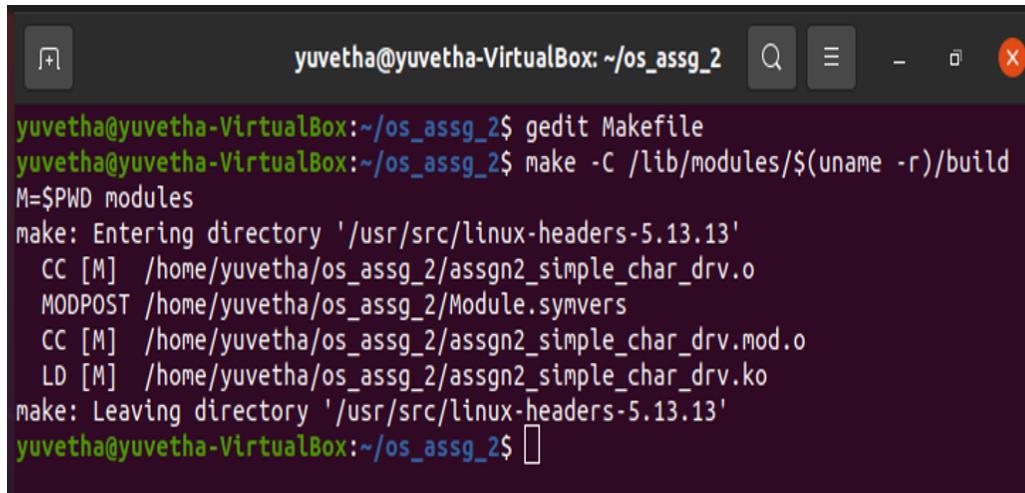
A screenshot of a terminal window titled "Makefile". The window has a dark header bar with "Open" and a dropdown arrow on the left, and a "Makefile" label with the path "os\_assg\_2" on the right. The main body of the terminal shows a single line of text:

```
1 obj-m := assgn2_simple_char_drv.o
```

Save this file as Makefile

### 3) To compile the modules of driver run

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



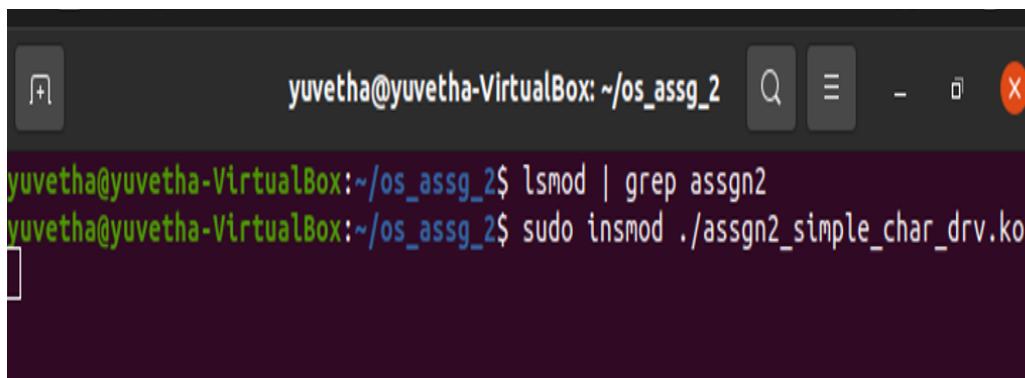
A screenshot of a terminal window titled "yuvetha@yuvetha-VirtualBox: ~/os\_assg\_2". The window shows the command "make -C /lib/modules/\$(uname -r)/build M=\$PWD modules" being run. The output of the command is displayed, showing the compilation process for a module named "assgn2\_simple\_char\_drv". The terminal interface includes standard Linux-style buttons at the top.

```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ gedit Makefile
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ make -C /lib/modules/$(uname -r)/build
M=$PWD modules
make: Entering directory '/usr/src/linux-headers-5.13.13'
CC [M] /home/yuvetha/os_assg_2/assgn2_simple_char_drv.o
MODPOST /home/yuvetha/os_assg_2/Module.symvers
CC [M] /home/yuvetha/os_assg_2/assgn2_simple_char_drv.mod.o
LD [M] /home/yuvetha/os_assg_2/assgn2_simple_char_drv.ko
make: Leaving directory '/usr/src/linux-headers-5.13.13'
yuvetha@yuvetha-VirtualBox:~/os_assg_2$
```

### 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.

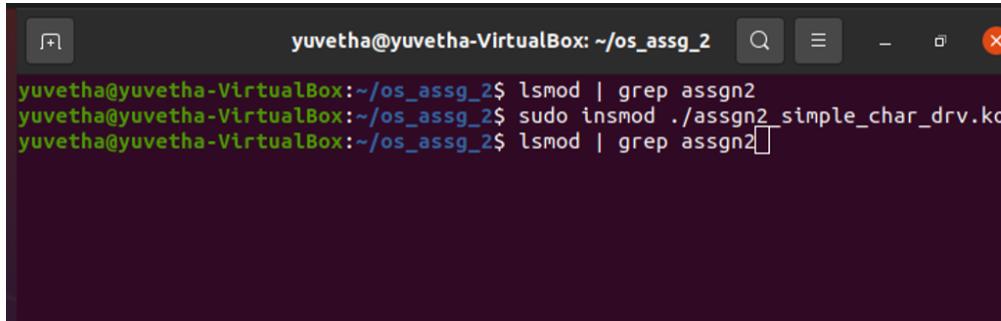


A screenshot of a terminal window titled "yuvetha@yuvetha-VirtualBox: ~/os\_assg\_2". The window shows the command "lsmod | grep assgn2" being run, followed by "sudo insmod ./assgn2\_simple\_char\_drv.ko". The terminal interface includes standard Linux-style buttons at the top.

```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ lsmod | grep assgn2
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo insmod ./assgn2_simple_char_drv.ko
```

### 5) To verify whether module inserted or not run

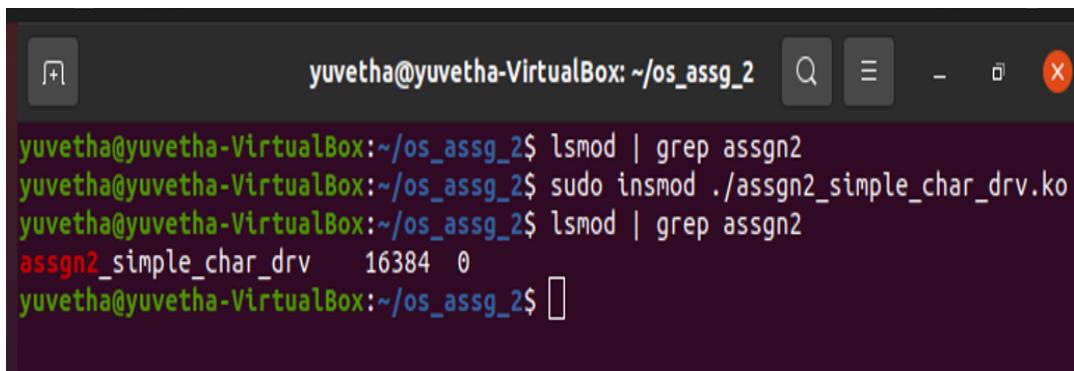
```
lsmod | grep assgn2
```



A terminal window titled "yuvetha@yuvetha-VirtualBox: ~/os\_assg\_2". The window shows three lines of command-line output:

```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ lsmod | grep assgn2
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo insmod ./assgn2_simple_char_drv.ko
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ lsmod | grep assgn2
```

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



A terminal window titled "yuvetha@yuvetha-VirtualBox: ~/os\_assg\_2". The window shows four lines of command-line output:

```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ lsmod | grep assgn2
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo insmod ./assgn2_simple_char_drv.ko
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ lsmod | grep assgn2
assgn2_simple_char_drv    16384  0
yuvetha@yuvetha-VirtualBox:~/os_assg_2$
```

## 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 ttvMAX
```

```
yuvetha@yuvetha-VirtualBox: ~/os_assg_2$ cat /proc/devices
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
```

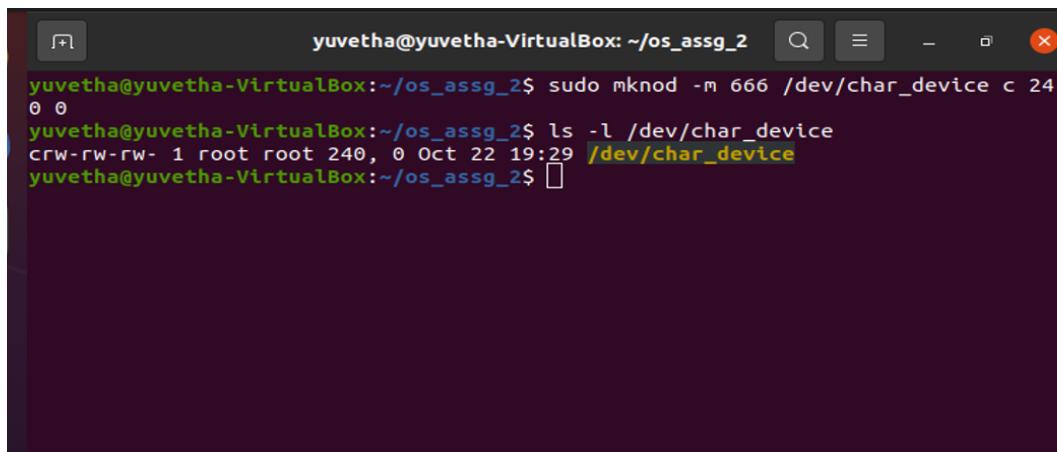
This enables everyone to access our device for read and write functionality. 240 indicate major number and 0 indicates the minor number and in our case device driver had major no. 240 and minor no. 0.



```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo mknod -m 666 /dev/char_device c 24 0 0
yuvetha@yuvetha-VirtualBox:~/os_assg_2$
```

**8) To verify whether char\_device is available for the given code to run**

```
ls -l /dev/char_device
```

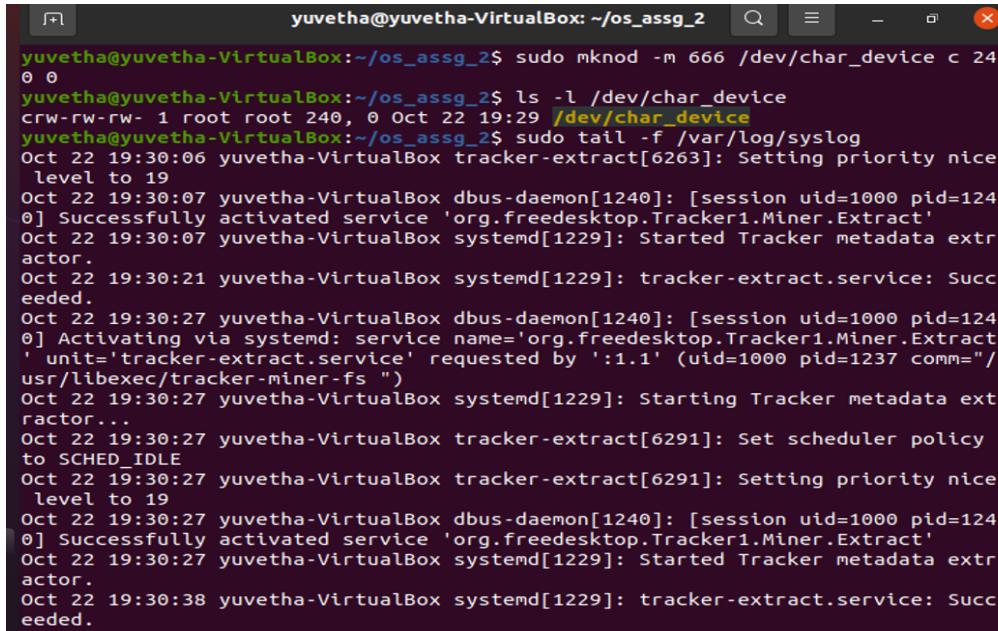


```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo mknod -m 666 /dev/char_device c 24 0 0
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ ls -l /dev/char_device
crw-rw-rw- 1 root root 240, 0 Oct 22 19:29 /dev/char_device
yuvetha@yuvetha-VirtualBox:~/os_assg_2$
```

**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.



```
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo mknod -m 666 /dev/char_device c 24 0 0
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ ls -l /dev/char_device
crw-rw-rw- 1 root root 240, 0 Oct 22 19:29 /dev/char_device
yuvetha@yuvetha-VirtualBox:~/os_assg_2$ sudo tail -f /var/log/syslog
Oct 22 19:30:06 yuvetha-VirtualBox tracker-extract[6263]: Setting priority nice
level to 19
Oct 22 19:30:07 yuvetha-VirtualBox dbus-daemon[1240]: [session uid=1000 pid=124
0] Successfully activated service 'org.freedesktop.Tracker1.Miner.Extract'
Oct 22 19:30:07 yuvetha-VirtualBox systemd[1229]: Started Tracker metadata extr
actor.
Oct 22 19:30:21 yuvetha-VirtualBox systemd[1229]: tracker-extract.service: Succ
eeded.
Oct 22 19:30:27 yuvetha-VirtualBox dbus-daemon[1240]: [session uid=1000 pid=124
0] Activating via systemd: service name='org.freedesktop.Tracker1.Miner.Extract
' unit='tracker-extract.service' requested by ':1.1' (uid=1000 pid=1237 comm="/
usr/libexec/tracker-miner-fs ")
Oct 22 19:30:27 yuvetha-VirtualBox systemd[1229]: Starting Tracker metadata ext
ractor...
Oct 22 19:30:27 yuvetha-VirtualBox tracker-extract[6291]: Set scheduler policy
to SCHED_IDLE
Oct 22 19:30:27 yuvetha-VirtualBox tracker-extract[6291]: Setting priority nice
level to 19
Oct 22 19:30:27 yuvetha-VirtualBox dbus-daemon[1240]: [session uid=1000 pid=124
0] Successfully activated service 'org.freedesktop.Tracker1.Miner.Extract'
Oct 22 19:30:27 yuvetha-VirtualBox systemd[1229]: Started Tracker metadata extr
actor.
Oct 22 19:30:38 yuvetha-VirtualBox systemd[1229]: tracker-extract.service: Succ
eeded.
```

## 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.

```
Oct 22 19:32:58 yuvetha@yuvetha-VirtualBox: ~/os_assg_2
Oct 22 19:32:58 yuvetha@yuvetha-VirtualBox: ~/...          =124
0] Activating vi [+] yuvetha@yuvetha-VirtualBox: ~/...
' unit='tracker-                         act
usr/libexec/trac[yuvetha@yuvetha-VirtualBox:~$ cd os_assg_2
Oct 22 19:32:58 yuvetha@yuvetha-VirtualBox:~/os_assg_2$ cat /dev/char_devi ext
ractor... ce
Oct 22 19:32:58 yuvetha@yuvetha-VirtualBox:~/os_assg_2$ echo "testing" > / .cy
to SCED_IDLE dev/char_device
Oct 22 19:32:58 yuvetha@yuvetha-VirtualBox:~/os_assg_2$ █ nice
level to 19
Oct 22 19:32:59 yuvetha-VirtualBox dbus-daemon[1240]: [session uid=1000 pid=124
0] Successfully activated service 'org.freedesktop.Tracker1.Miner.Extract'
Oct 22 19:32:59 yuvetha-VirtualBox systemd[1229]: Started Tracker metadata extr
actor.
Oct 22 19:33:02 yuvetha-VirtualBox kernel: [ 7635.466111] Inside the assgn2_ope
n function
Oct 22 19:33:02 yuvetha-VirtualBox kernel: [ 7635.466139] Inside the assgn2_rea
d function
Oct 22 19:33:02 yuvetha-VirtualBox kernel: [ 7635.466161] Inside the assgn2_clo
se function
Oct 22 19:33:18 yuvetha-VirtualBox systemd[1229]: tracker-extract.service: Succ
eeded.
Oct 22 19:33:29 yuvetha-VirtualBox kernel: [ 7661.769572] Inside the assgn2_ope
n function
Oct 22 19:33:29 yuvetha-VirtualBox kernel: [ 7661.769612] Inside the assgn2_wri
te function
Oct 22 19:33:29 yuvetha-VirtualBox kernel: [ 7661.769620] Inside the assgn2_clo
se function
█
```