CS3003D: OPERATING SYSTEMS

(ASSIGNMENT-1)

A CHARACTER DEVICE DRIVER

Group No: 22

Group Members

1.	Sevakula Jyothi	B180359CS	Batch-A
2.	Billa Amulya	B180404CS	Batch-A
3.	Bhukya Vasanth Kumar	B180441CS	Batch-A
4.	Shaik Ashraf Mohiuddin	B180483CS	Batch-A
5.	Bangi Mohith Kumar	B180477CS	Batch-A

Date: 27th October 2020

Problem Statement

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

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 in a similar fashion 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.

The Character Device Driver Code

The source code for the **ebbchar** device driver is provided in Listing given below. 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:

```
static int __init ebbchar_init(void);

static void __exit ebbchar_exit(void);

static int     dev_open(struct inode *, struct file *);

static int     dev_release(struct inode *, struct file *);

static ssize_t dev_read(struct file *, char *, size_t, loff_t *);

static ssize_t dev_write(struct file *, const char *, size_t, loff_t *);
```

There are some additional points:

- This code has a fixed message size of 256 characters and this could also be improved using the dynamic memory allocation.
- This code is not multi-process safe that is addressed later in this report.
- The int _init ebbchar_init function is much longer than and that is because it is now automatically allocates a major number to the device, registering the device class, and registering the device driver. Importantly, you will notice that if anything goes wrong that the code carefully “backs out” of the successful operations. To achieve this we have repeated code but the alternative is to use goto statements, which is even less palatable.
- The functions sprintf() and strlen() are available in the kernel through the inclusion of linux/kernel.h and indirectly through linux/string.h respectively. The functions in **string.h** are architecture dependent.

➢ Finally, we need to create the Makefile for the device which would build the building objects for the directory.

➢ We should create a User-Space program for testing the Character Device Driver Module with the given input.

Conclusion

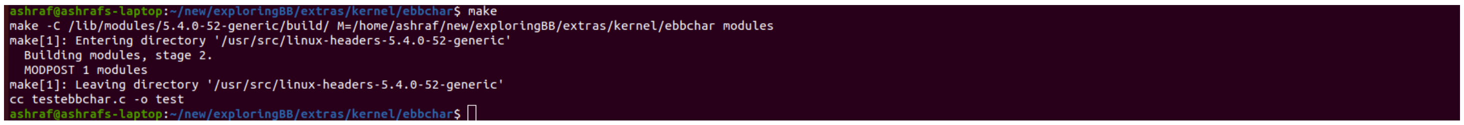
Important outcomes of this article are that:

- You can now create your own device such as **/dev/ebbchar**, which you can write information to and read information from. This is important, as it provides a bridge between the Linux user space and the Linux kernel space. It enables you to develop advanced drivers, such as communication devices, which can be controlled by C code that is running in user space.
- You can use command rules to alter the properties of a device as it is loaded.

Command Lines to Execute the Driver

To compile the modules of driver run:

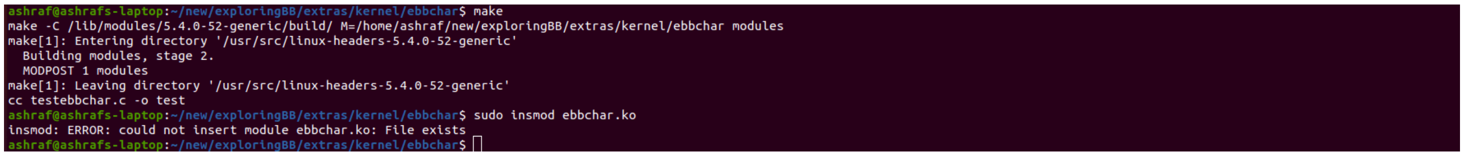
make



```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ make
make -C /lib/modules/5.4.0-52-generic/build/ M=/home/ashraf/new/exploringBB/extras/kernel/ebbchar modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-52-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-52-generic'
cc testebbchar.c -o test
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ 
```

To insert our compiled module to the kernel run

sudo insmod ebbchar.ko

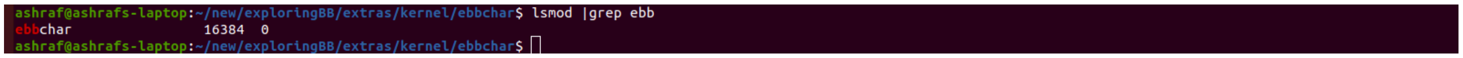


```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ make
make -C /lib/modules/5.4.0-52-generic/build/ M=/home/ashraf/new/exploringBB/extras/kernel/ebbchar modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-52-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-52-generic'
cc testebbchar.c -o test
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ sudo insmod ebbchar.ko
insmod: ERROR: could not insert module ebbchar.ko: File exists
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ 
```

To verify whether module inserted or not run

lsmod |grep ebb

<u>Explanation</u>: 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.

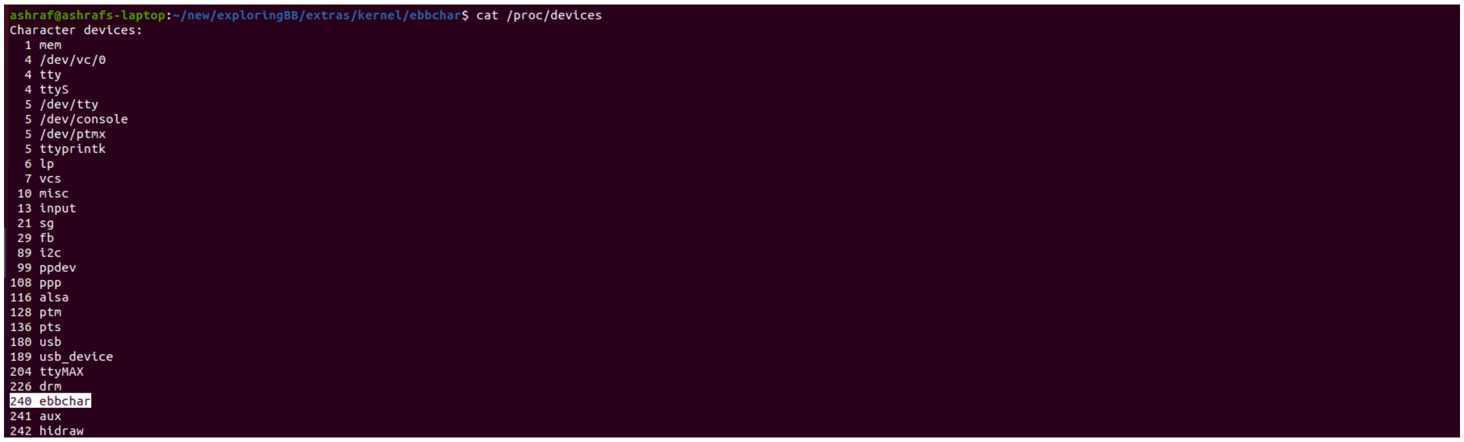


```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ lsmod |grep ebb
ebbchar                16384  0
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ 
```

To show list of character devices and block devices run

cat /proc/devices

Explanation: this outputs the major number and name of the device, and is broken into two major sections: Character devices and block devices. Our device named "ebbchar" should come under character devices.

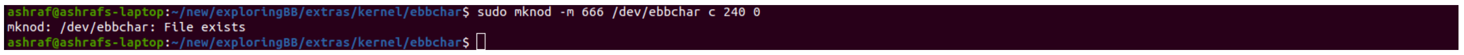


```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 89 i2c
 99 ppdev
108 ppp
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
204 ttyMAX
226 drm
240 ebbchar
241 aux
242 hidraw
```

To make the device accessible run

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

Explaination: 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 ebbchar had major no. 240 and minor no. 0.

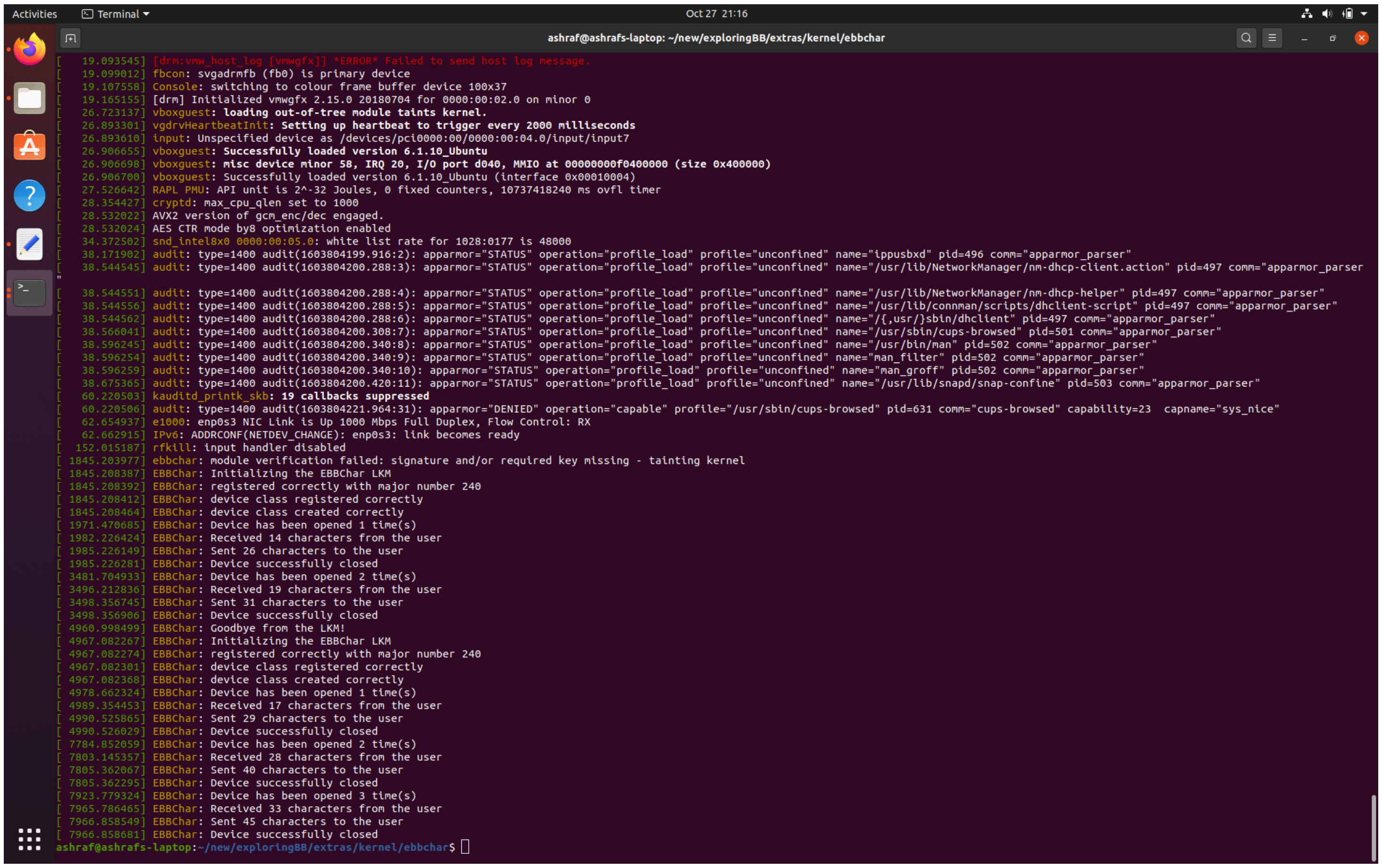


```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ sudo mknod -m 666 /dev/ebbchar c 240 0
mknod: /dev/ebbchar: File exists
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ 
```

To show the system log console run

dmesg

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


Activities
Terminal
Oct 27 21:16
ashraf@ashrafs-laptop: ~/new/exploringBB/extras/kernel/ebbchar

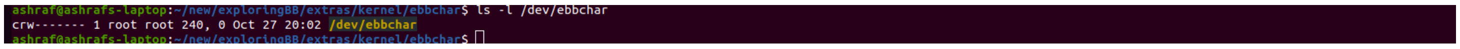
```
[   19.093545] [drm:vmw_host_log [vmwgfx]] *ERROR* Failed to send host log message.
[   19.099012] fbcon: svgadrmfb (fb0) is primary device
[   19.107558] Console: switching to colour frame buffer device 100x37
[   19.165155] [drm] Initialized vmwgfx 2.15.0 20180704 for 0000:00:02.0 on minor 0
[   26.723137] vboxguest: loading out-of-tree module taints kernel.
[   26.893301] vgdrvHeartbeatInit: Setting up heartbeat to trigger every 2000 milliseconds
[   26.893610] input: Unspecified device as /devices/pci0000:00/0000:00:04.0/input/input7
[   26.906655] vboxguest: Successfully loaded version 6.1.10_Ubuntu
[   26.906698] vboxguest: misc device minor 58, IRQ 20, I/O port d040, MMIO at 00000000f0400000 (size 0x400000)
[   26.906700] vboxguest: Successfully loaded version 6.1.10_Ubuntu (interface 0x00010004)
[   27.526642] RAPL PMU: API unit is 2^-32 Joules, 0 fixed counters, 10737418240 ms ovfl timer
[   28.354427] cryptd: max_cpu_qlen set to 1000
[   28.532022] AVX2 version of gcm_enc/dec engaged.
[   28.532024] AES CTR mode by8 optimization enabled
[   34.372502] snd_intel8x0 0000:00:05.0: white list rate for 1028:0177 is 48000
[   38.171902] audit: type=1400 audit(1603804199.916:2): apparmor="STATUS" operation="profile_load" profile="unconfined" name="ippusbxd" pid=496 comm="apparmor_parser"
[   38.544545] audit: type=1400 audit(1603804200.288:3): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-client.action" pid=497 comm="apparmor_parser
"

[   38.544551] audit: type=1400 audit(1603804200.288:4): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-helper" pid=497 comm="apparmor_parser"
[   38.544556] audit: type=1400 audit(1603804200.288:5): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/lib/connman/scripts/dhclient-script" pid=497 comm="apparmor_parser"
[   38.544562] audit: type=1400 audit(1603804200.288:6): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/{,usr/}sbin/dhclient" pid=497 comm="apparmor_parser"
[   38.566041] audit: type=1400 audit(1603804200.308:7): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/sbin/cups-browsed" pid=501 comm="apparmor_parser"
[   38.596245] audit: type=1400 audit(1603804200.340:8): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/bin/man" pid=502 comm="apparmor_parser"
[   38.596254] audit: type=1400 audit(1603804200.340:9): apparmor="STATUS" operation="profile_load" profile="unconfined" name="man_filter" pid=502 comm="apparmor_parser"
[   38.596259] audit: type=1400 audit(1603804200.340:10): apparmor="STATUS" operation="profile_load" profile="unconfined" name="man_groff" pid=502 comm="apparmor_parser"
[   38.675365] audit: type=1400 audit(1603804200.420:11): apparmor="STATUS" operation="profile_load" profile="unconfined" name="/usr/lib/snapd/snap-confine" pid=503 comm="apparmor_parser"
[   60.220503] kauditd_printk_skb: 19 callbacks suppressed
[   60.220506] audit: type=1400 audit(1603804221.964:31): apparmor="DENIED" operation="capable" profile="/usr/sbin/cups-browsed" pid=631 comm="cups-browsed" capability=23  capname="sys_nice"
[   62.654937] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[   62.662915] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[  152.015187] rfkill: input handler disabled
[ 1845.203977] ebbchar: module verification failed: signature and/or required key missing - tainting kernel
[ 1845.208387] EBBChar: Initializing the EBBChar LKM
[ 1845.208392] EBBChar: registered correctly with major number 240
[ 1845.208412] EBBChar: device class registered correctly
[ 1845.208464] EBBChar: device class created correctly
[ 1971.470685] EBBChar: Device has been opened 1 time(s)
[ 1982.226424] EBBChar: Received 14 characters from the user
[ 1985.226149] EBBChar: Sent 26 characters to the user
[ 1985.226281] EBBChar: Device successfully closed
[ 3481.704933] EBBChar: Device has been opened 2 time(s)
[ 3496.212836] EBBChar: Received 19 characters from the user
[ 3498.356745] EBBChar: Sent 31 characters to the user
[ 3498.356906] EBBChar: Device successfully closed
[ 4960.998499] EBBChar: Goodbye from the LKM!
[ 4967.082267] EBBChar: Initializing the EBBChar LKM
[ 4967.082274] EBBChar: registered correctly with major number 240
[ 4967.082301] EBBChar: device class registered correctly
[ 4967.082368] EBBChar: device class created correctly
[ 4978.662324] EBBChar: Device has been opened 1 time(s)
[ 4989.354453] EBBChar: Received 17 characters from the user
[ 4990.525865] EBBChar: Sent 29 characters to the user
[ 4990.526029] EBBChar: Device successfully closed
[ 7784.852059] EBBChar: Device has been opened 2 time(s)
[ 7803.145357] EBBChar: Received 28 characters from the user
[ 7805.362067] EBBChar: Sent 40 characters to the user
[ 7805.362295] EBBChar: Device successfully closed
[ 7923.779324] EBBChar: Device has been opened 3 time(s)
[ 7965.786465] EBBChar: Received 33 characters from the user
[ 7966.858549] EBBChar: Sent 45 characters to the user
[ 7966.858681] EBBChar: Device successfully closed
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$
```

To verify whether ebbchar is available for the given code to run

ls -l /dev/ebbchar

Explaination: This should show our ebbchar



```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ ls -l /dev/ebbchar
crw------- 1 root root 240, 0 Oct 27 20:02 /dev/ebbchar
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$
```

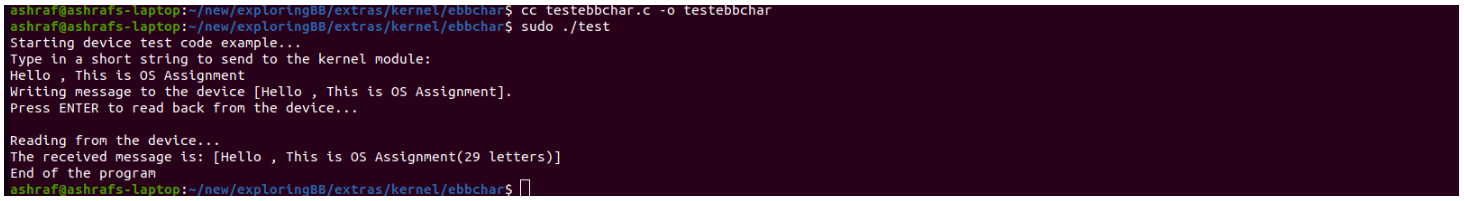
To compile the test.c file

cc testbbchar.c -o testbbchar

To run the character drive

sudo ./test

__Explanation__: We can check the functionality (which is accepting a string and displaying the same) of our device driver.



```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ cc testebbchar.c -o testebbchar
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ sudo ./test
Starting device test code example...
Type in a short string to send to the kernel module:
Hello , This is OS Assignment
Writing message to the device [Hello , This is OS Assignment].
Press ENTER to read back from the device...

Reading from the device...
The received message is: [Hello , This is OS Assignment(29 letters)]
End of the program
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ 
```