CS3003D: OPERATING SYSTEMS

(ASSIGNMENT-1)

A CHARACTER DEVICE DRIVER

BHUKYA VASANTH KUMAR B180441CS 27th October 2020

Group No: 22 Batch: A

Date: 27th October 2020

Problem Statement

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

Individual Contribution: Makefile to build LKM

I have made the Makefile as a part of the project in the character device driver. This part is used to build the kernel module that we want. Most Makefiles within the kernel are kbuild Makefiles that use the kbuild infrastructure. Even I have also written a kbuild Makefiles to build the kernel module for this project.

```
obj-m+=ebbchar.o
```

The 1st line defines the files to be built, any special compilation options, and any subdirectories to be entered recursively.

Here the obj-m defines a loadable module goal.

This describes to the kbuild that there is one object in that directory, named ebbchar.o and since I have used obj-m it will be built as a loadable kernel module. The ebbchar.o will be built from ebbchar.c that we write.

Similarly there is obj-y which indicates a built-in object goal. If it is neither y nor m, then the file will not be compiled nor linked. In this particular case we wanted to build a kernel module from only one object.

```
all:

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

	$(CC) testebbchar.c -o test
```

These lines of code are run when the user runs make or make all in the shell. The $(shell uname -r) returns the current kernel build. The -C option switches the directory to the kernel directory before performing any make tasks. The M = $(PWD) variable assignment tells the make command where the actual project files exist. The modules target is the default target for external kernel modules.

After the execution, we will be able to see a kernel module, that is ebbchar.ko in the build directory. Kernel modules have a .ko extension which easily distinguishes them from conventional object files. The reason for this is that they contain an additional .modinfo section where additional information about the module is kept.

```
clean:

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

	rm test
```

The first line removes the a.out if it exists or else it echos “removed”. The second line removes the compiled modules.

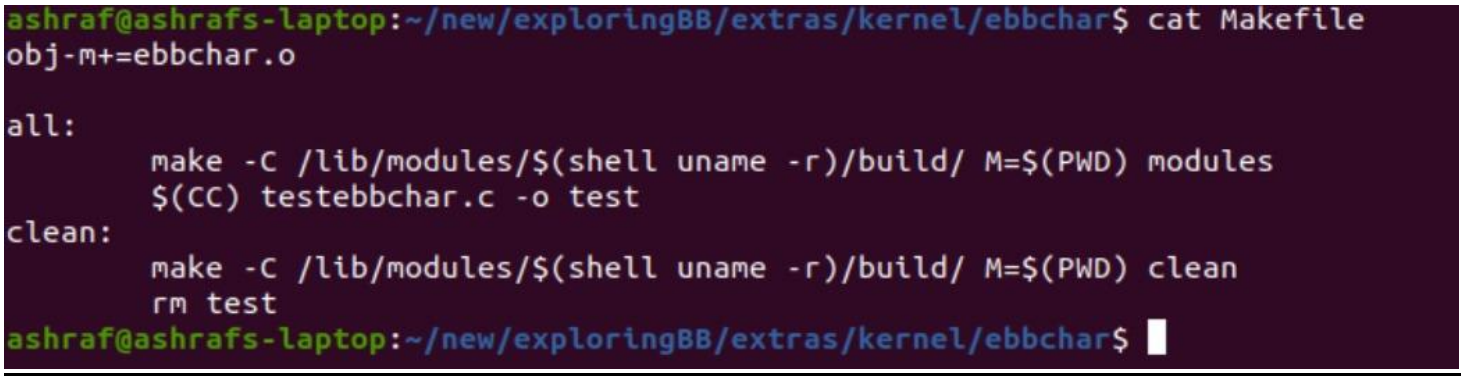
```
sudo insmod ebbchar.ko
```

This checks if ebbchar.ko exists in the directory or not, if not, it includes it in the directory.

As I said, ebbchar.ko is created by Makefile.

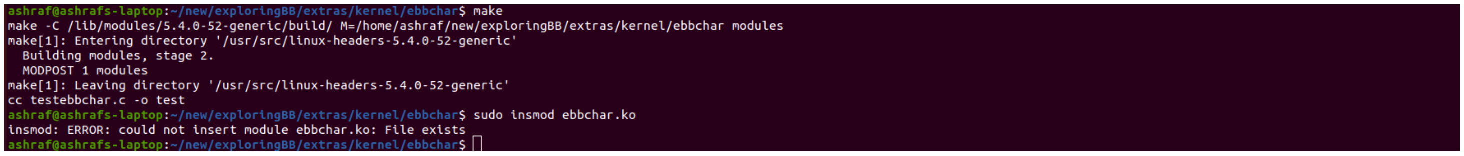
Screenshots

The code



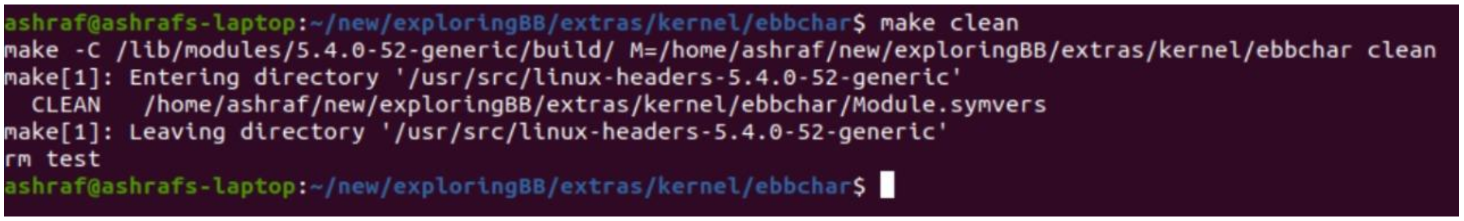
```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ cat Makefile
obj-m+=ebbchar.o

all:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
	$(CC) testebbchar.c -o test
clean:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
	rm test
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$
```

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$ sudo insmod ebbchar.ko
insmod: ERROR: could not insert module ebbchar.ko: File exists
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$
```

make clean



```
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$ make clean
make -C /lib/modules/5.4.0-52-generic/build/ M=/home/ashraf/new/exploringBB/extras/kernel/ebbchar clean
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-52-generic'
  CLEAN   /home/ashraf/new/exploringBB/extras/kernel/ebbchar/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-52-generic'
rm test
ashraf@ashrafs-laptop:~/new/exploringBB/extras/kernel/ebbchar$
```