**LAB 4**

**LINUX KERNEL DEVELOPMENT**

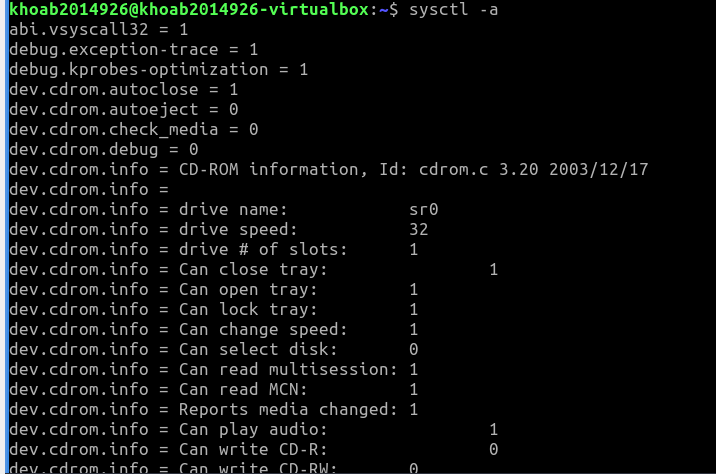
|  |
| --- |
| Fullname: Tran Dang Khoa  Student ID: B2014926 |

* Note: screenshots need to be clear and good-looking; submissions must be in PDF format.

1. **Modify kernel parameters and install new modules**

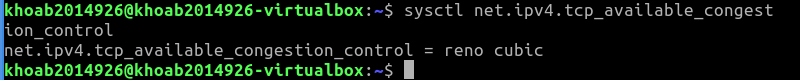
- List all linux kernel parameters on your OS:

sysctl –a



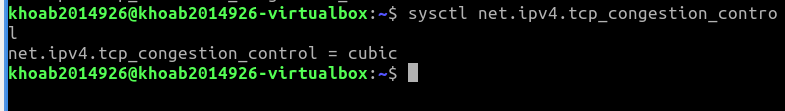
- List all available TCP congestion control algorithms:

sysctl net.ipv4.tcp\_available\_congestion\_control



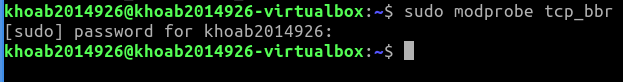
- Show which TCP congestion control algorithm is using:

sysctl net.ipv4.tcp\_congestion\_control



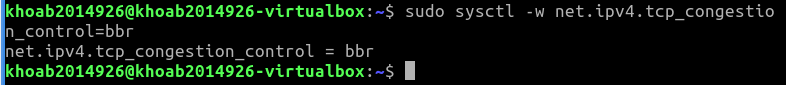
- Install bbr TCP congestion control algorithm module:

sudo modprobe tcp\_bbr

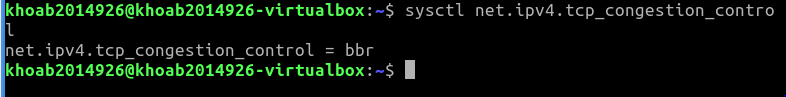


- Switch to the bbr TCP congestion control algorithm:

sudo sysctl -w net.ipv4.tcp\_congestion\_control=bbr



sysctl net.ipv4.tcp\_congestion\_control

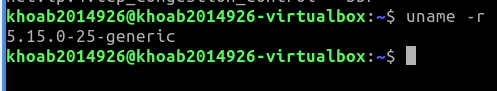


(take screenshots to show that you finish this exercise)

1. **Install new kernel version**

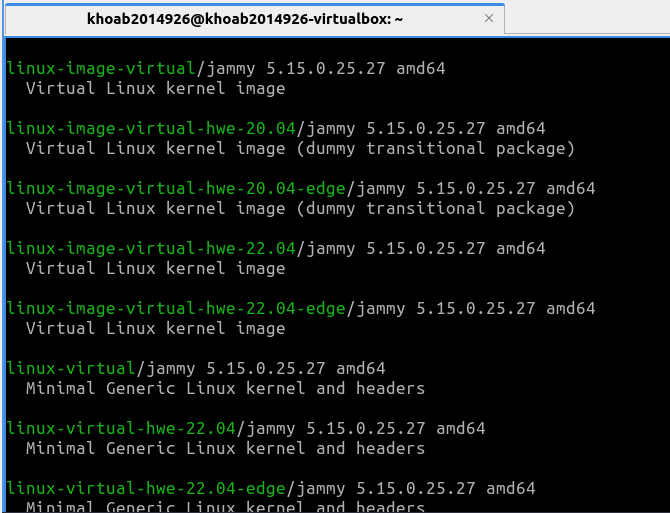
- Show your current kernel version:

uname –r



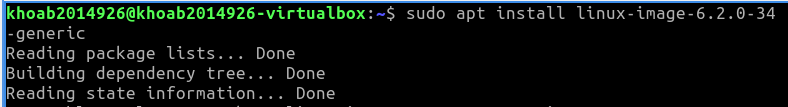
- Search for newer versions:

sudo apt search linux-image

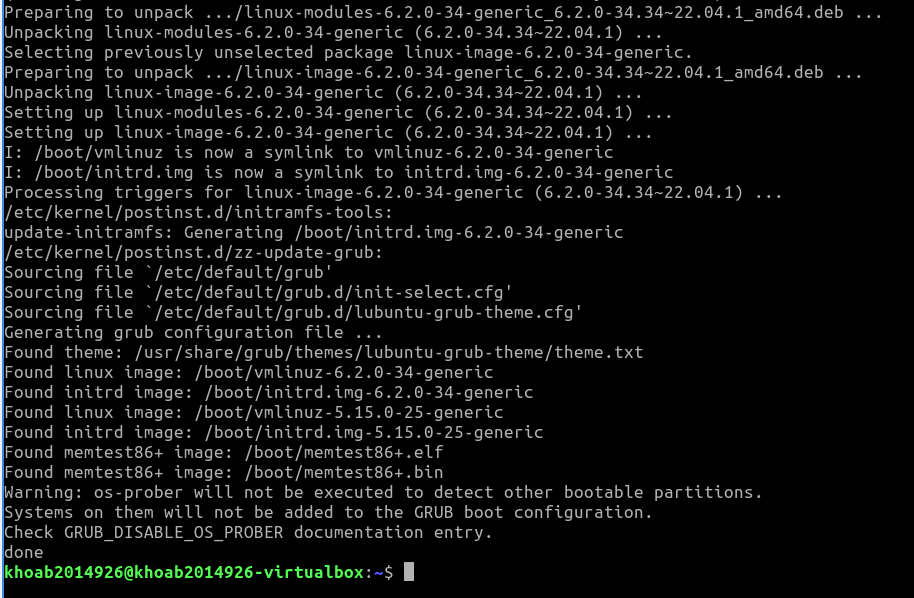


- Install the latest version you find:

sudo apt install linux-image-6.2.0-34-generic



=> Result: Accomplished

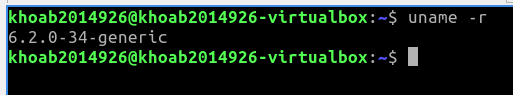


- After a kernel upgrade, you must reboot the system. Then, if the device driver you need is in the latest kernel, your hardware will work as expected:

sudo shutdown -r now

- Show your new current kernel version:

uname –r

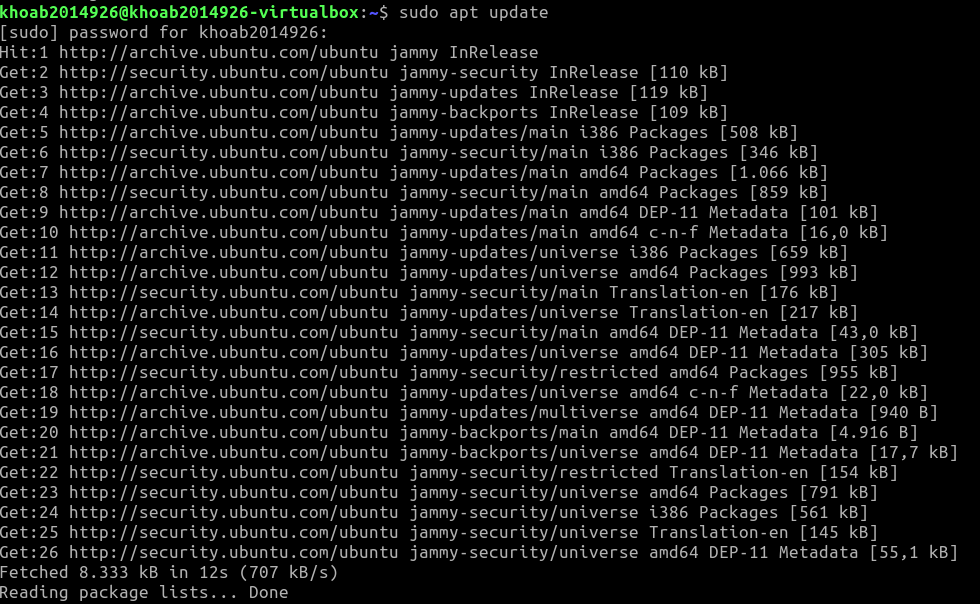


(take screenshots to show that you finish this exercise)

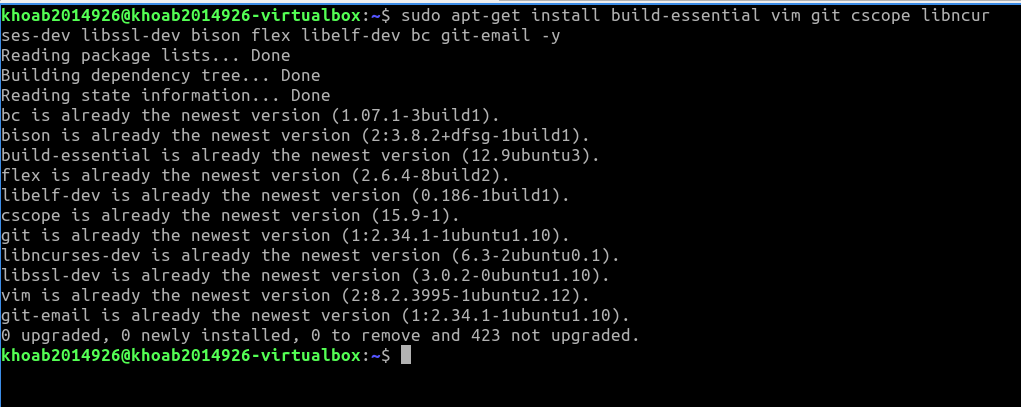
1. **Build and install a new kernel version**

- Get your system ready

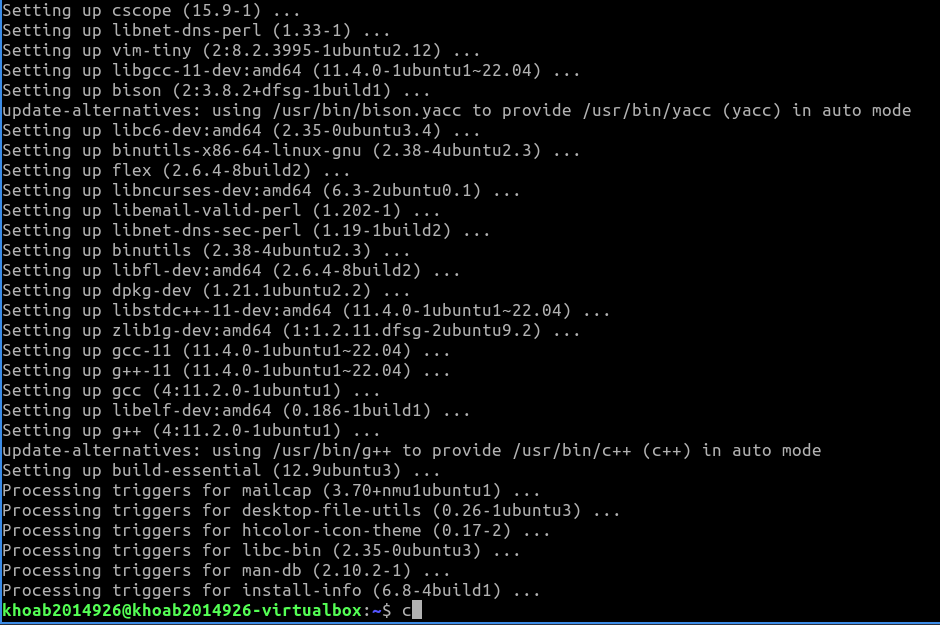
sudo apt update



sudo apt-get install build-essential vim git cscope libncurses-dev libssl-dev bison flex libelf-dev bc git-email –y



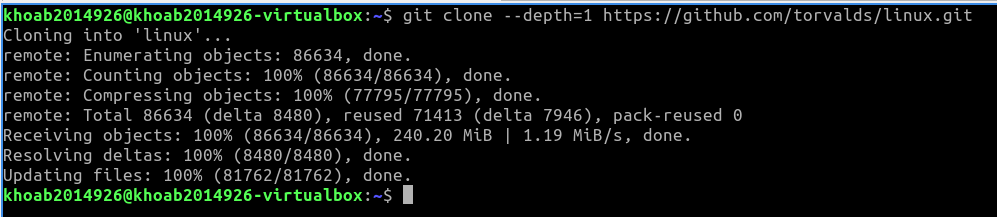
=> Result: Accomplished



- Clone a mainline kernel source code to your computer:

git clone --depth=1 \

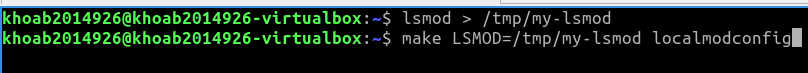
<https://github.com/torvalds/linux.git>



- To save time, just create a configuration file based on the list of modules currently loaded on your system (choose default values for other options).

lsmod > /tmp/my-lsmod

make LSMOD=/tmp/my-lsmod localmodconfig



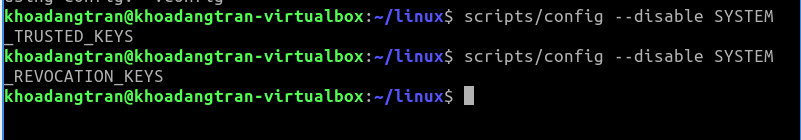
=> Result: Accomplished



- Disable certificate stuff:

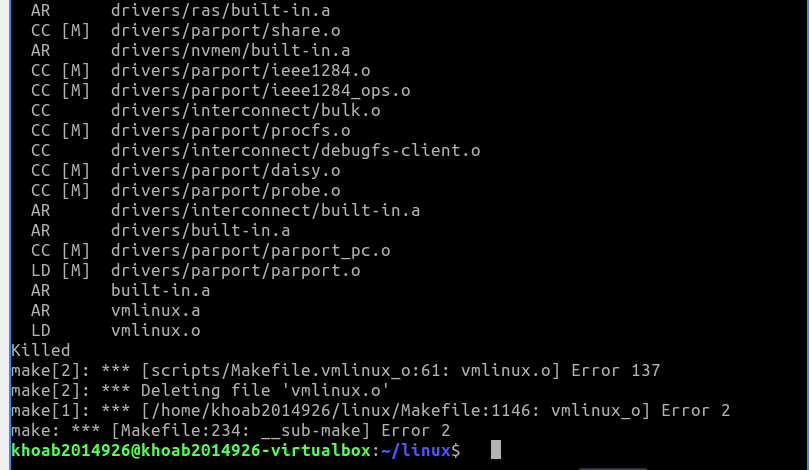
scripts/config --disable SYSTEM\_TRUSTED\_KEYS

scripts/config --disable SYSTEM\_REVOCATION\_KEYS



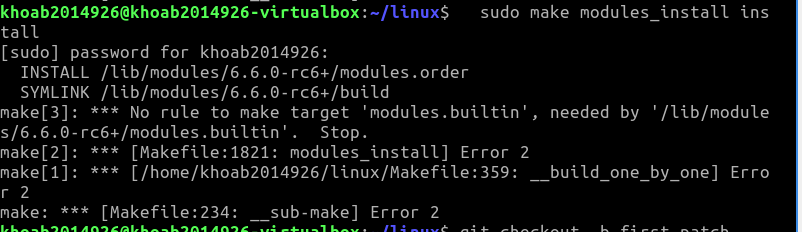
- Compile the kernel. The process takes about 1 hour, please be patient and enjoy a cup of coffee. It has been tested successfully on Lubuntu 20.04, if any errors occur, please try to fix them by yourself.

make -j3 all



- Install the new kernel:

sudo make modules\_install install



- Now it is time to reboot the system to boot the newly installed kernel:

sudo shutdown -r now

- Show your new current kernel version:

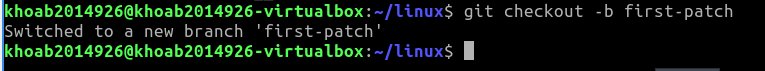
uname –r

(take screenshots to show that you finish this exercise)

1. **Writing Your First Kernel Patch**

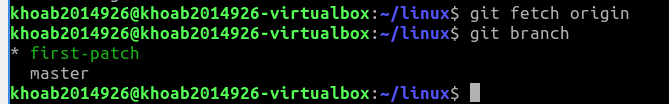
- Creating a new branch in the linux\_mainline repository (has been cloned in exercise 3)

git checkout -b first-patch

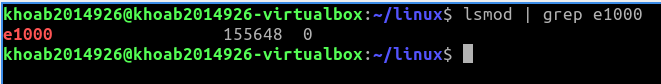


- Update the kernel

git fetch origin

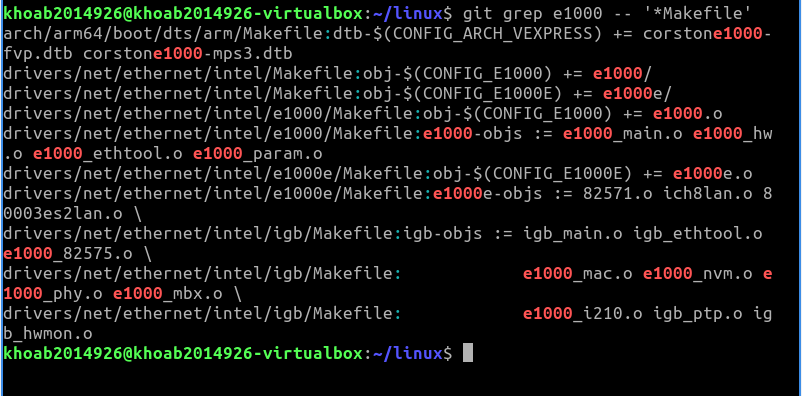


- Run lsmod to see the modules loaded on your system, and pick a driver to change. One driver that's included in all VM images is the e1000 driver, the Intel ethernet driver, or you can choose another driver depending on your working environment.



- Run git grep to look for e1000 files

git grep e1000 -- '\*Makefile'



- Make a small change to the probe function of the e1000 driver

nano drivers/net/ethernet/intel/e1000/e1000\_main.c

# Add a line of code as below

static int e1000\_probe(struct pci\_dev \*pdev, const struct pci\_device\_id \*ent) {

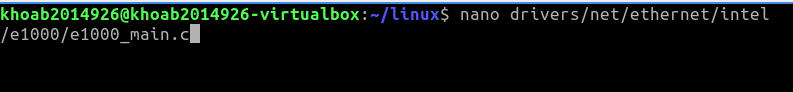
...

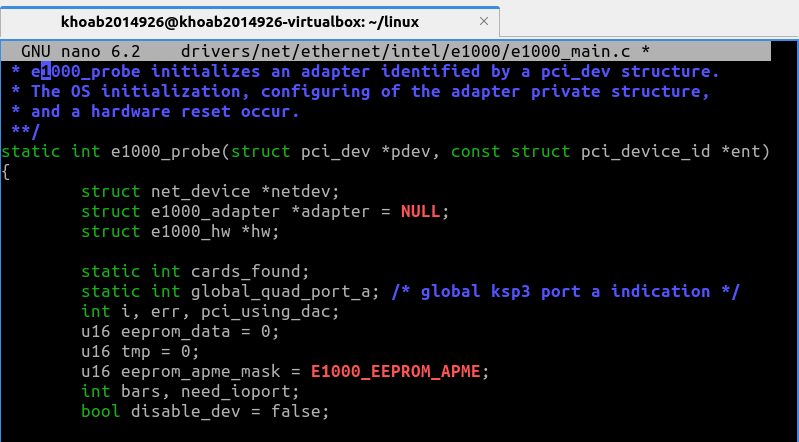
struct e1000\_hw \*hw;

printk(KERN\_DEBUG "I can modify the Linux kernel!\n");

static int cards\_found = 0;

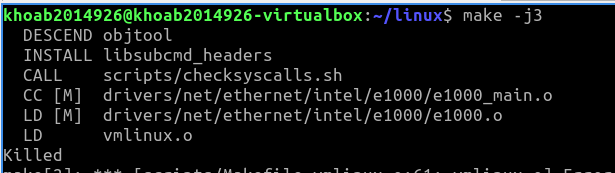
...



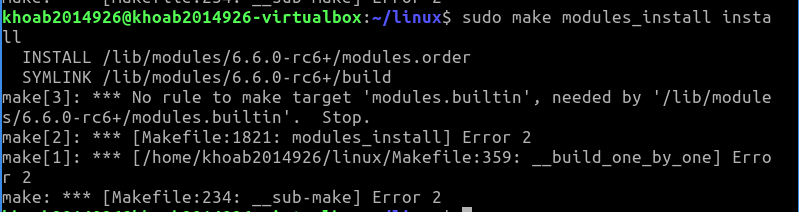


- Compile and install your changes:

make -j3



sudo make modules\_install install



- Reboot the system:

sudo shutdown -r now

- Show kernel buffer log:

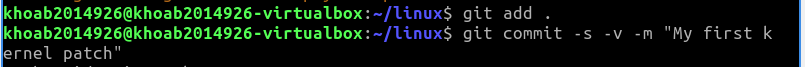
dmesg | less

# Search for your printk in the log file by typing "/I can modify"

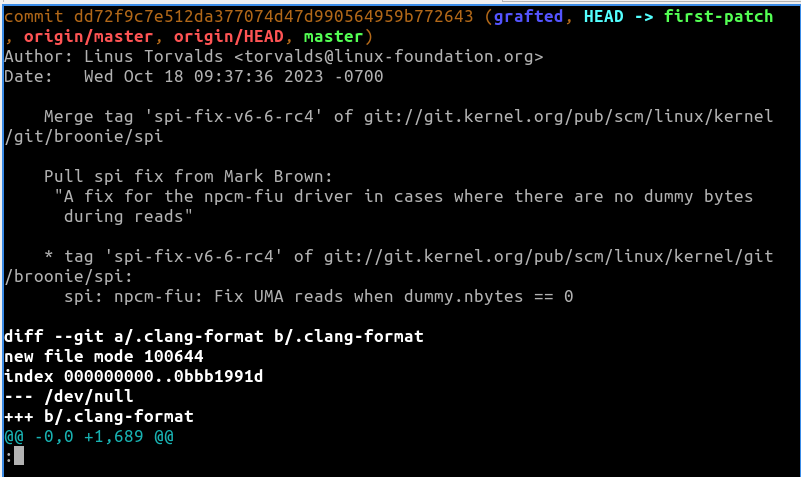
- Committing changes, and view your commit

git add .

git commit -s -v -m "My first kernel patch"



git show HEAD



- Find whom to send the patch to

git show HEAD | scripts/get\_maintainer.pl

- Create a patch

git format-patch -1 <commit ID> --to=<your email> Note: Please do not send your patch to a maintainer, send it to yourself instead.

- Modify ./git/config file to configure send-email

#.git/config

[sendemail]

smtpserver = smtp.googlemail.com

smtpencryption = tls

smtpserverport = 587

smtpuser = your gmail address (CTU student email is OK

- Send the patch

git send-email <patch\_file>

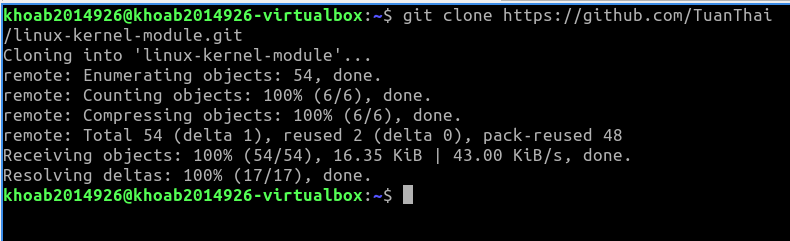
(take screenshots to show that you finish this exercise)

1. **Writing a simple Linux kernel module: Greeter sample**

This module simply takes a name as a parameter, and writes a greeting to the kernel log (/var/log/kern.log):

- Clone this repository to your computer:

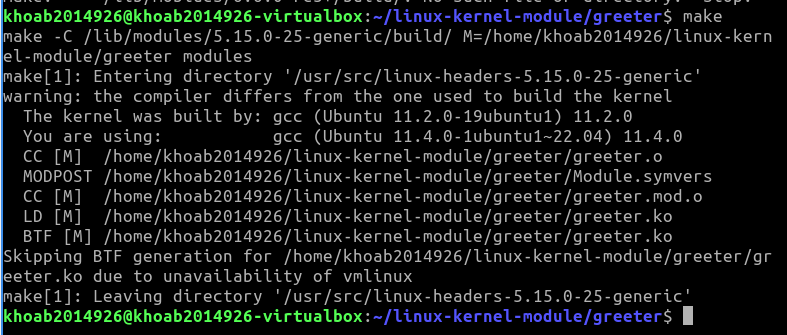
<https://github.com/TuanThai/linux-kernel-module.git>



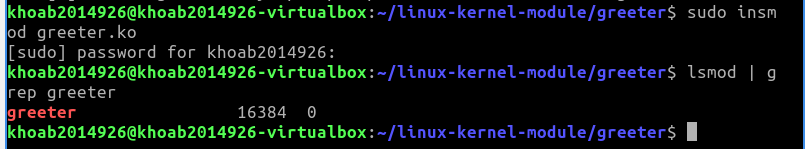
- Move into greeter/ directory.



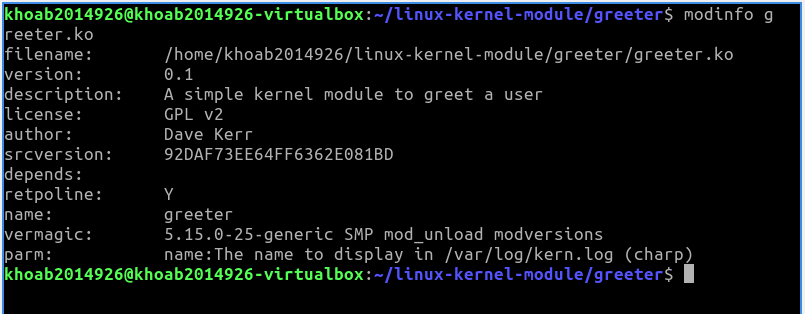
- Build the module using make command. The module is compiled to greeter.ko



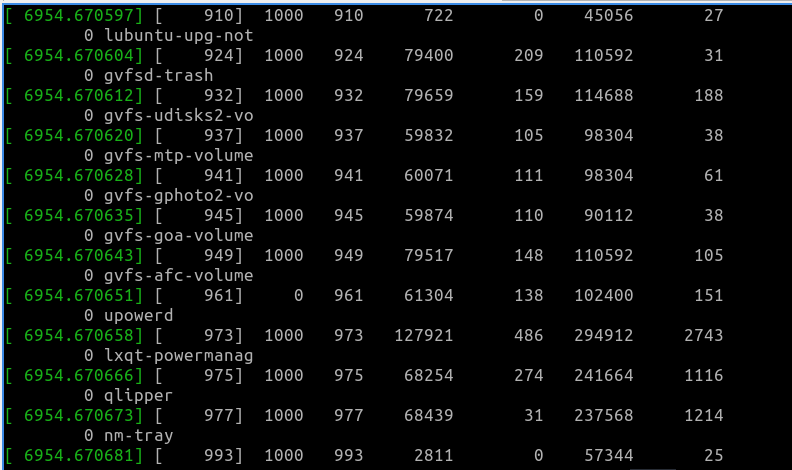
- Install the module using insmod greeter.ko command, then show that the module has been installed using lsmod | grep greeter command



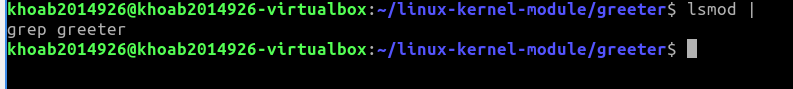
- Show the information of the module using modinfo greeter.ko



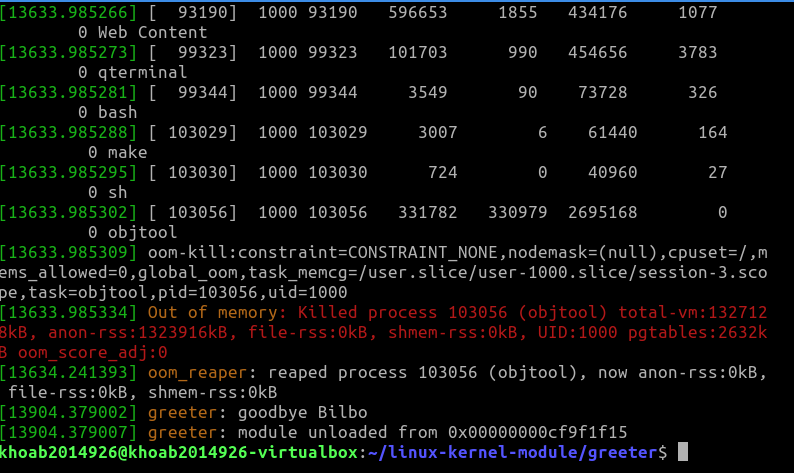
- Show kernel log with dmesg



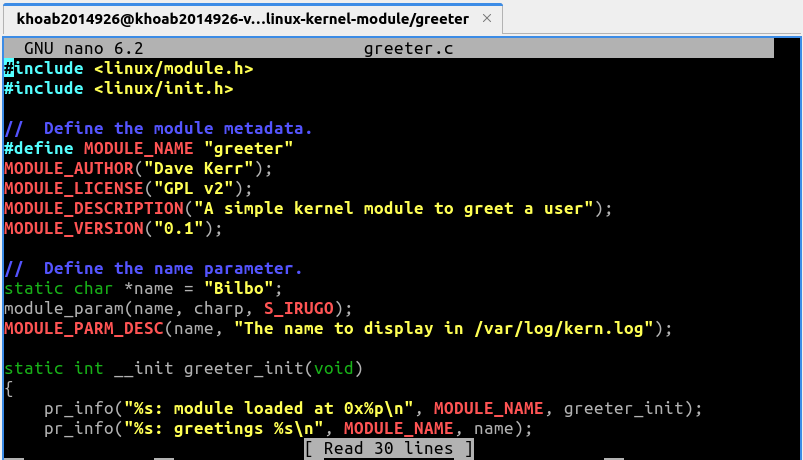
- Remove the module using rmmod greeter.ko command,then show that the module has been removed using lsmod | grep greeter command.



- Show kernel log with dmesg



- Move to greeter.c file, then briefly explain below functions:



**greeter\_init**: This function is called when the module is loaded into the kernel

with insmod. It logs the name of the module and its memory address, and then logs a

greeting message with the name parameter.

**greeter\_exit**: This function is called when the module is removed from the kernel

with rmmod. It logs a goodbye message with the name parameter, and then logs that the

module has been unloaded along with its memory address.

**module\_init(greeter\_init)**: The greeter\_init function will be called to initialize the

module and perform the necessary work.

**module\_exit(greeter\_exit**: The greeter\_exit function will be called to perform cleanup

and free up resources.

(take screenshots to show that you finish this exercise)

---END---