LAB 4

LINUX KERNEL DEVELOPMENT

Fullname: Nguyen Minh Nhut

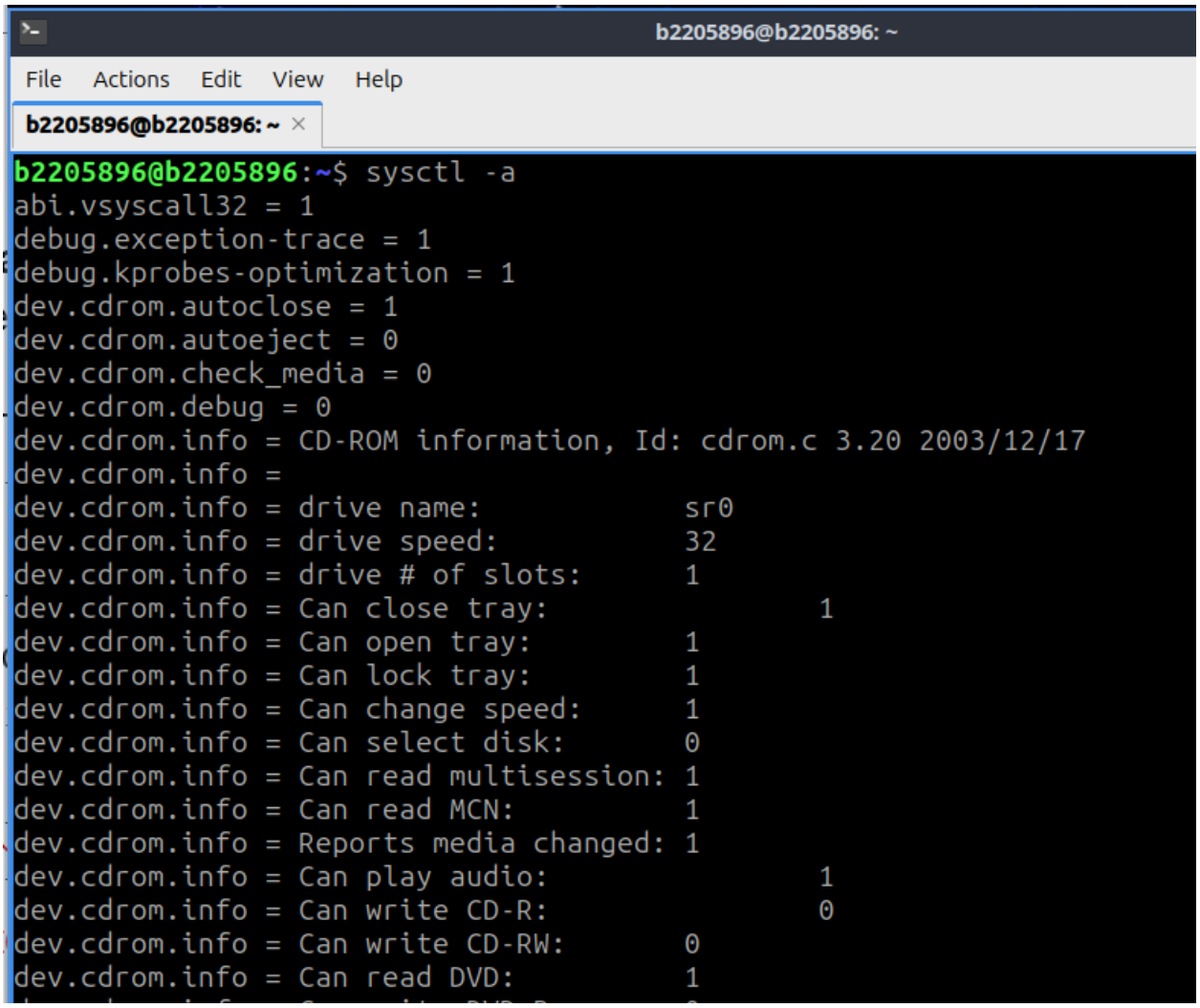
Student ID: B2205896

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

1. Modify kernel parameters and install new modules

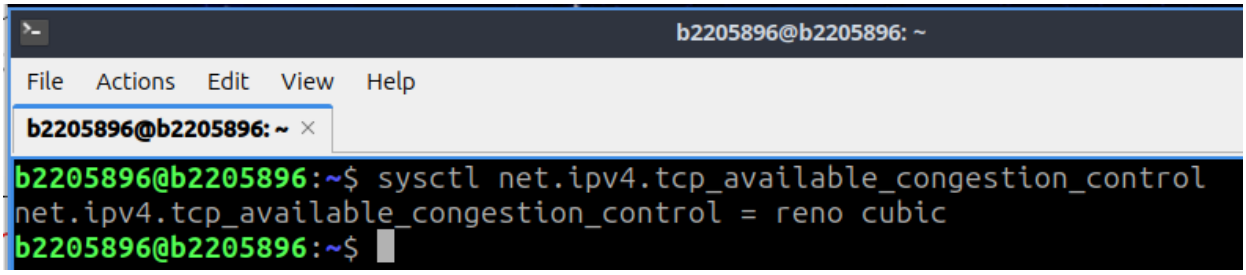
- List all linux kernel parameters on your OS:

```
sysctl -a
```



- List all available TCP congestion control algorithms:

```
sysctl net.ipv4.tcp_available_congestion_control
```



- Show which TCP congestion control algorithm is using:

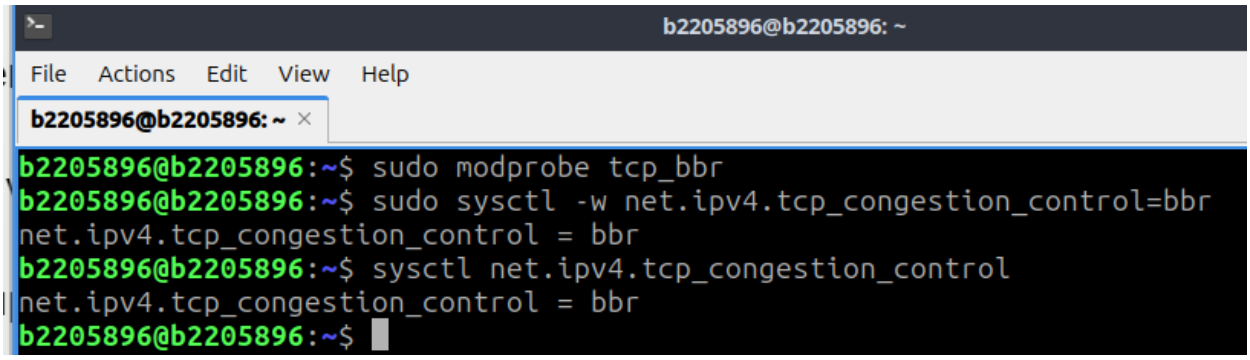
```
sysctl net.ipv4.tcp_congestion_control
```

- 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)

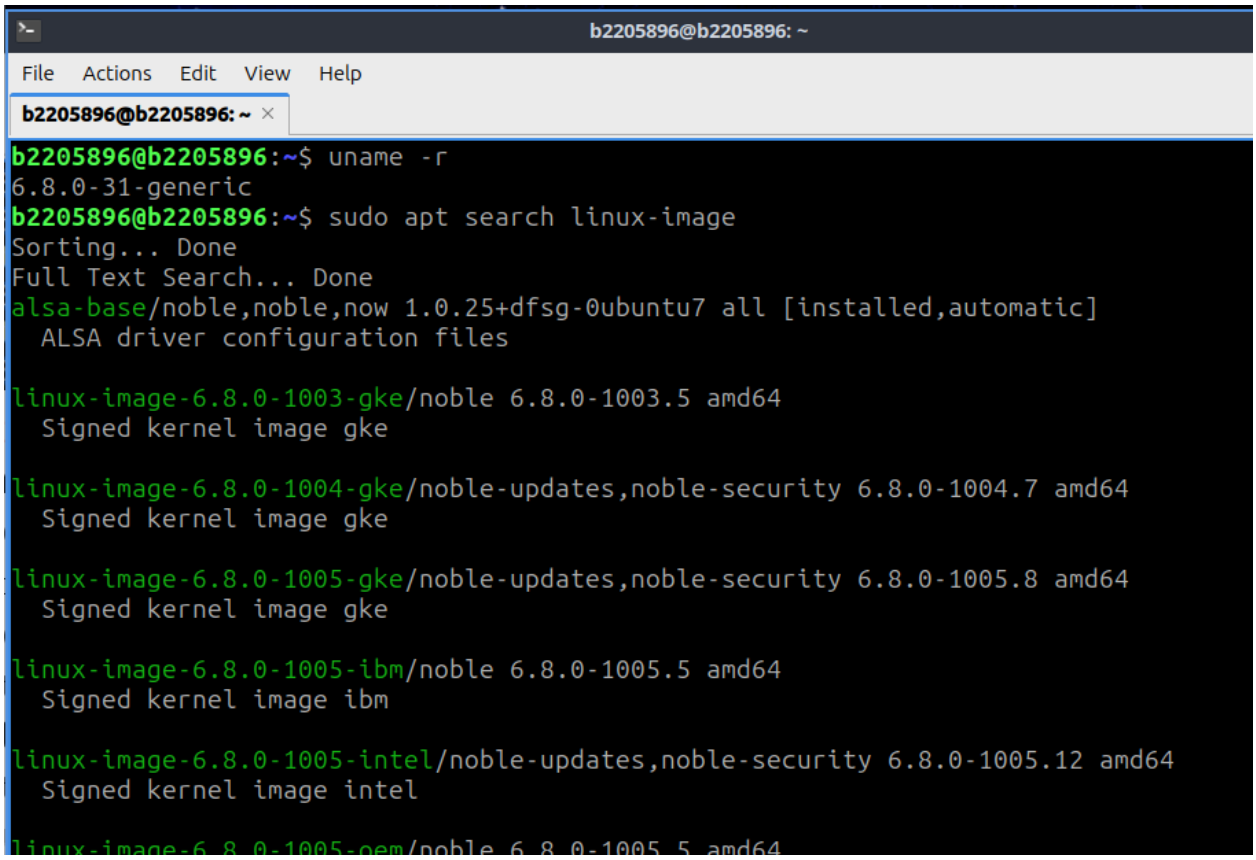
2. Install new kernel version

- Show your current kernel version:

```
uname -r
```

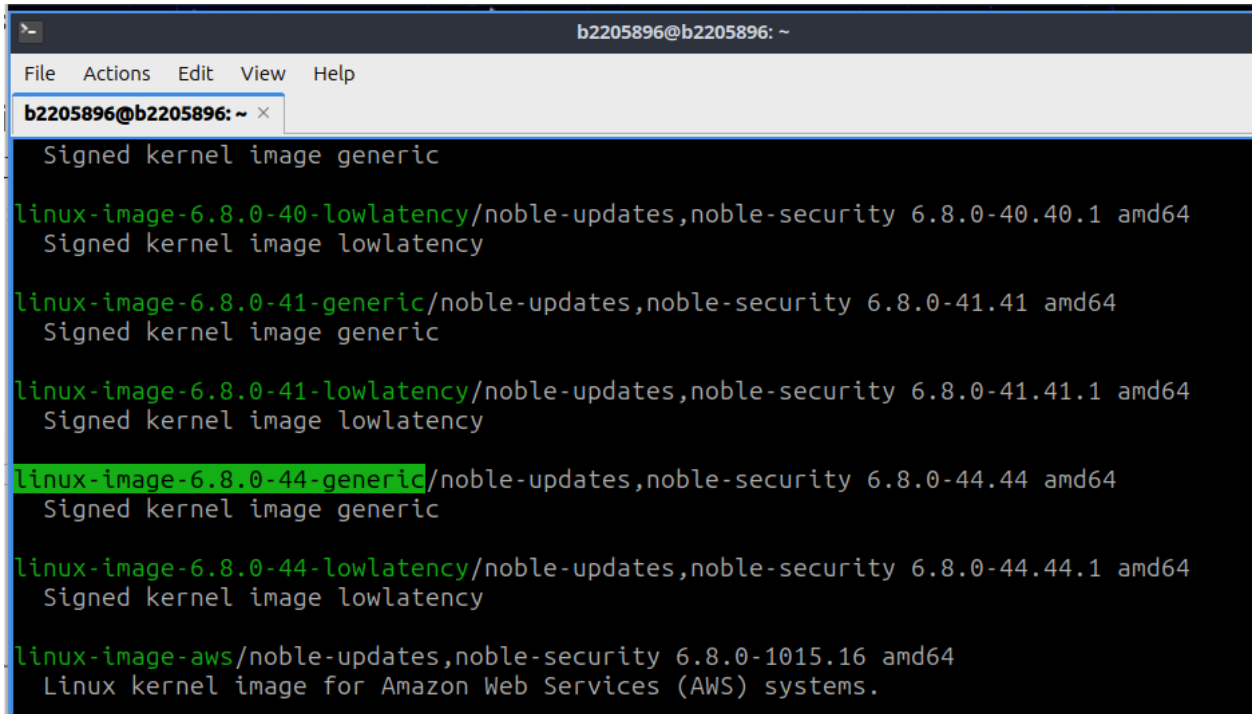
- Search for newer versions:

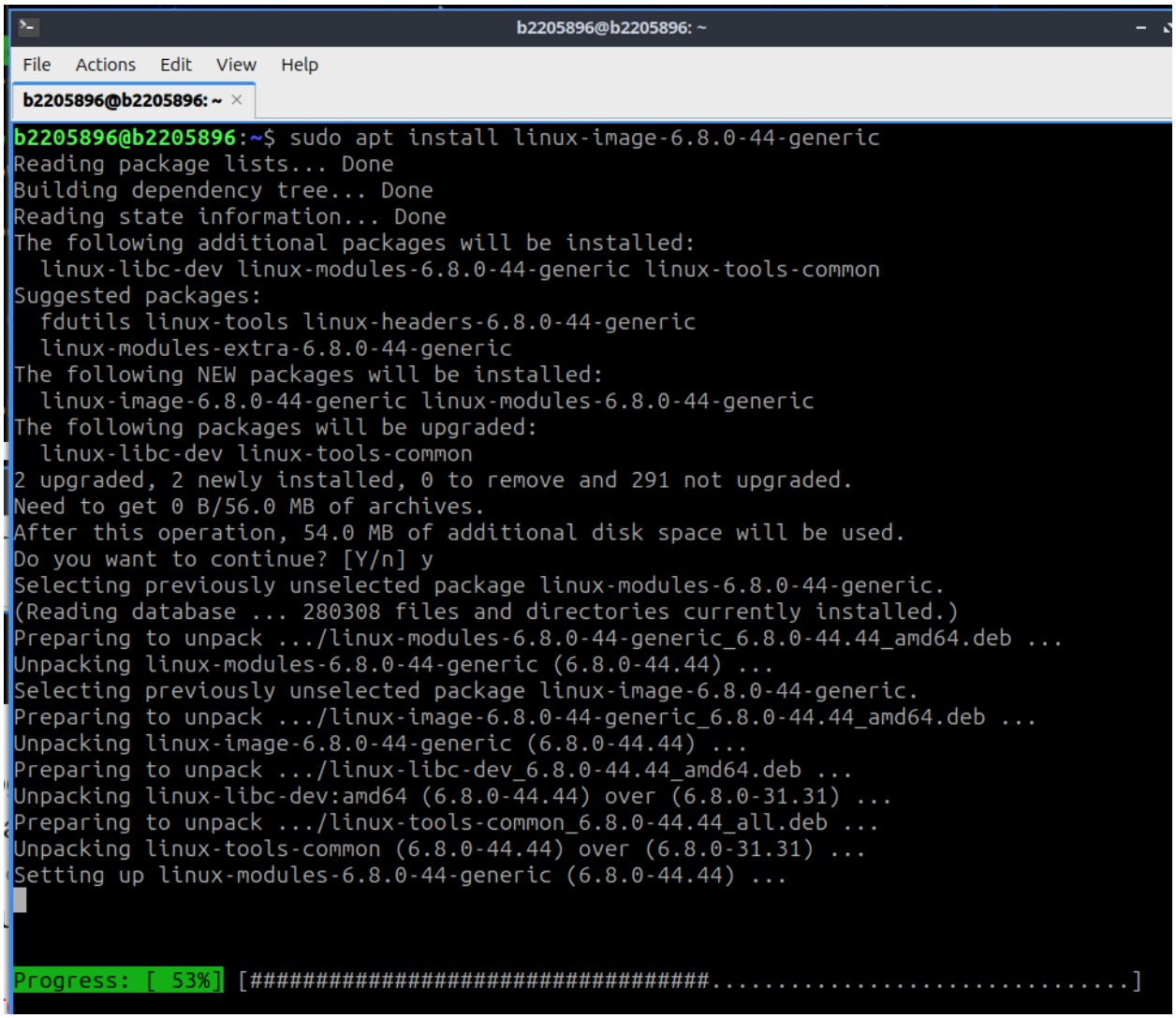
```
sudo apt search linux-image
```



- Install the latest version you find:

```
sudo apt install linux-image-x.x.x-x-generic
```



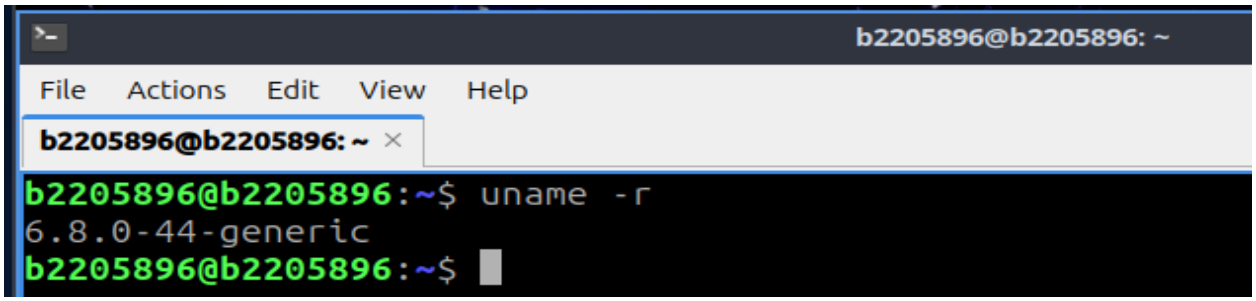


- After a kernel upgrade, you must reboot the system. Then, if the device driver you need is in the latest kernel, your hardware will work as expected:

```
sudo shutdown -r now
```

- Show your new current kernel version:

```
uname -r
```

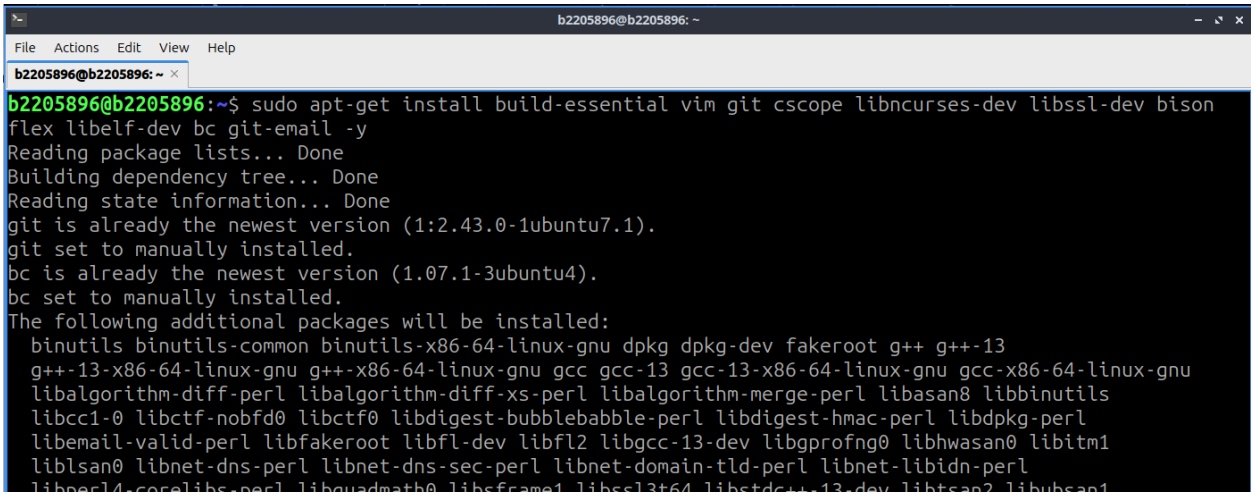


(take screenshots to show that you finish this exercise)

3. Build and install a new kernel version

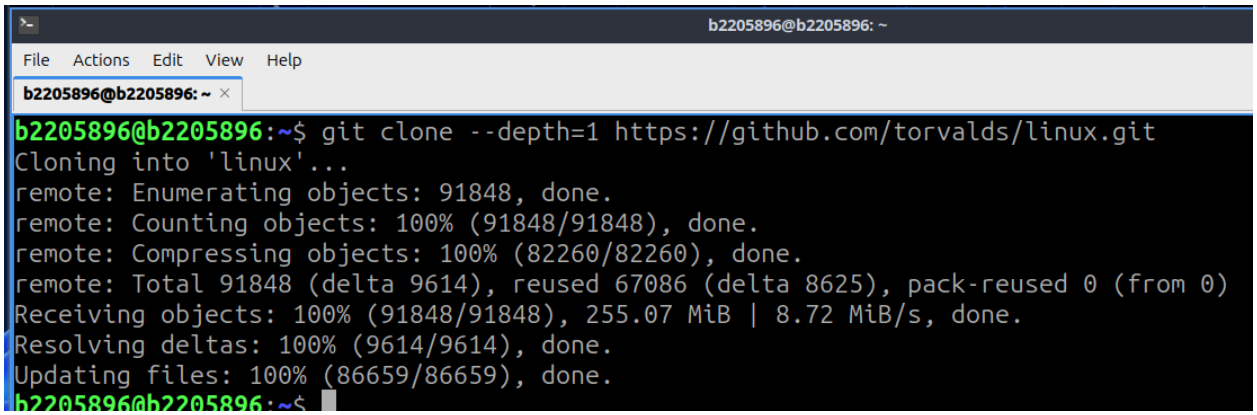
- Get your system ready

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



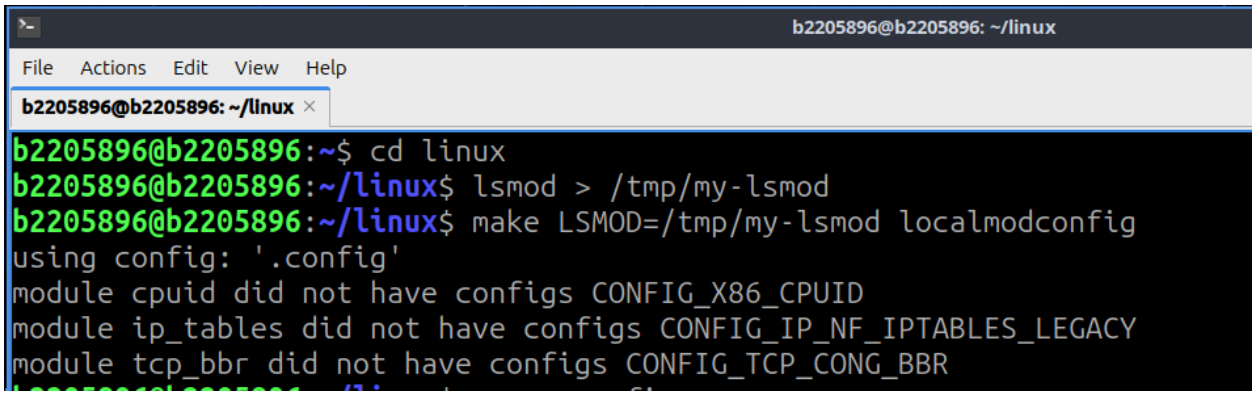
- Clone a mainline kernel source code to your computer:

```
git clone --depth=1 \
https://github.com/torvalds/linux.git
```



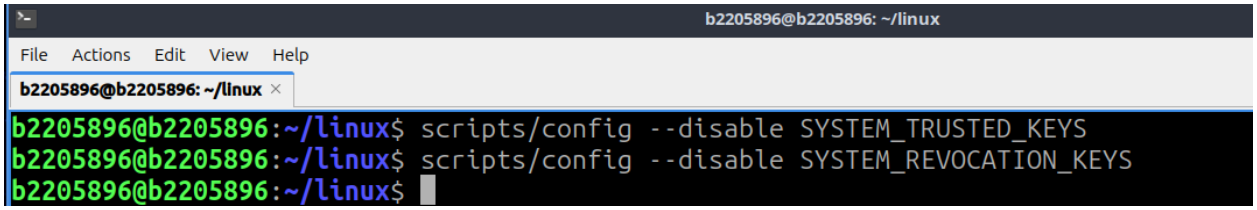
- To save time, just create a configuration file based on the list of modules currently loaded on your system (choose default values for other options).

```
lsmod > /tmp/my-lsmod
make LSMOD=/tmp/my-lsmod localmodconfig
```



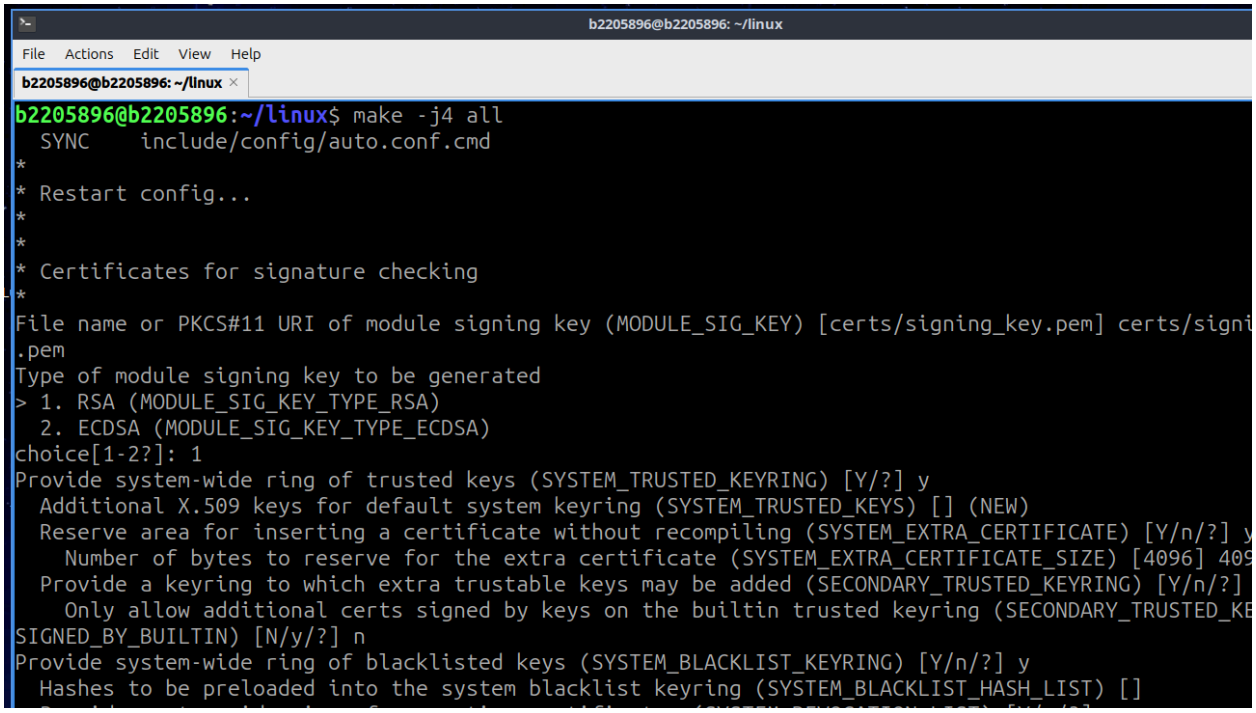
- Disable certificate stuff:

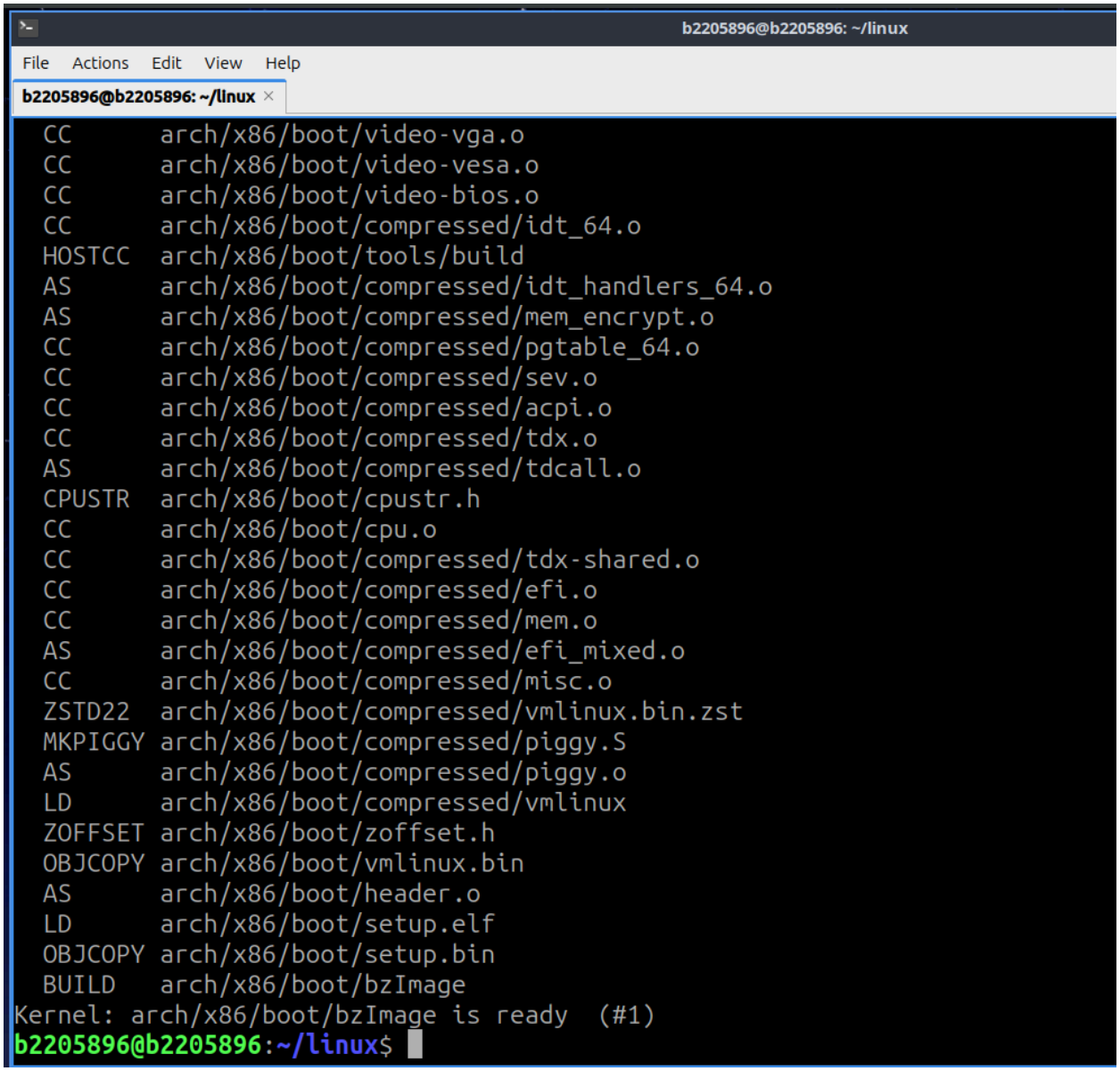
```
scripts/config --disable SYSTEM_TRUSTED_KEYS
scripts/config --disable SYSTEM_REVOCATION_KEYS
```



- 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.

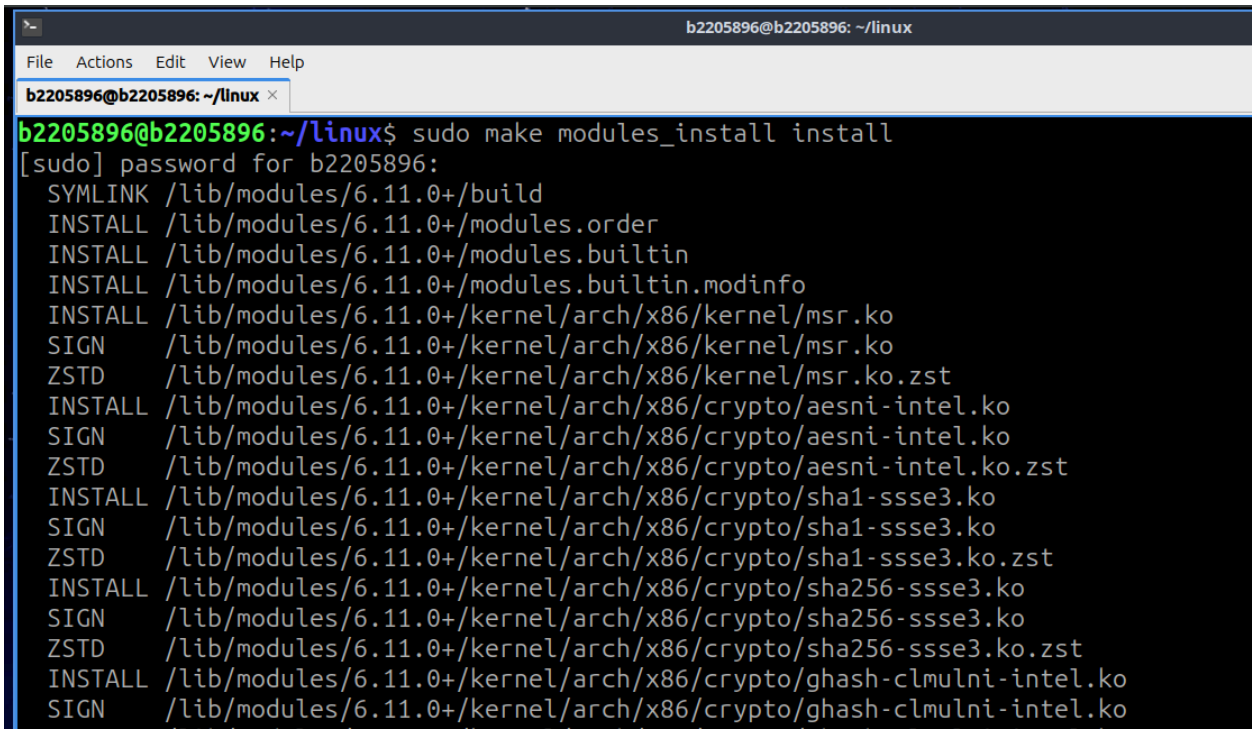
```
make -j3 all
```

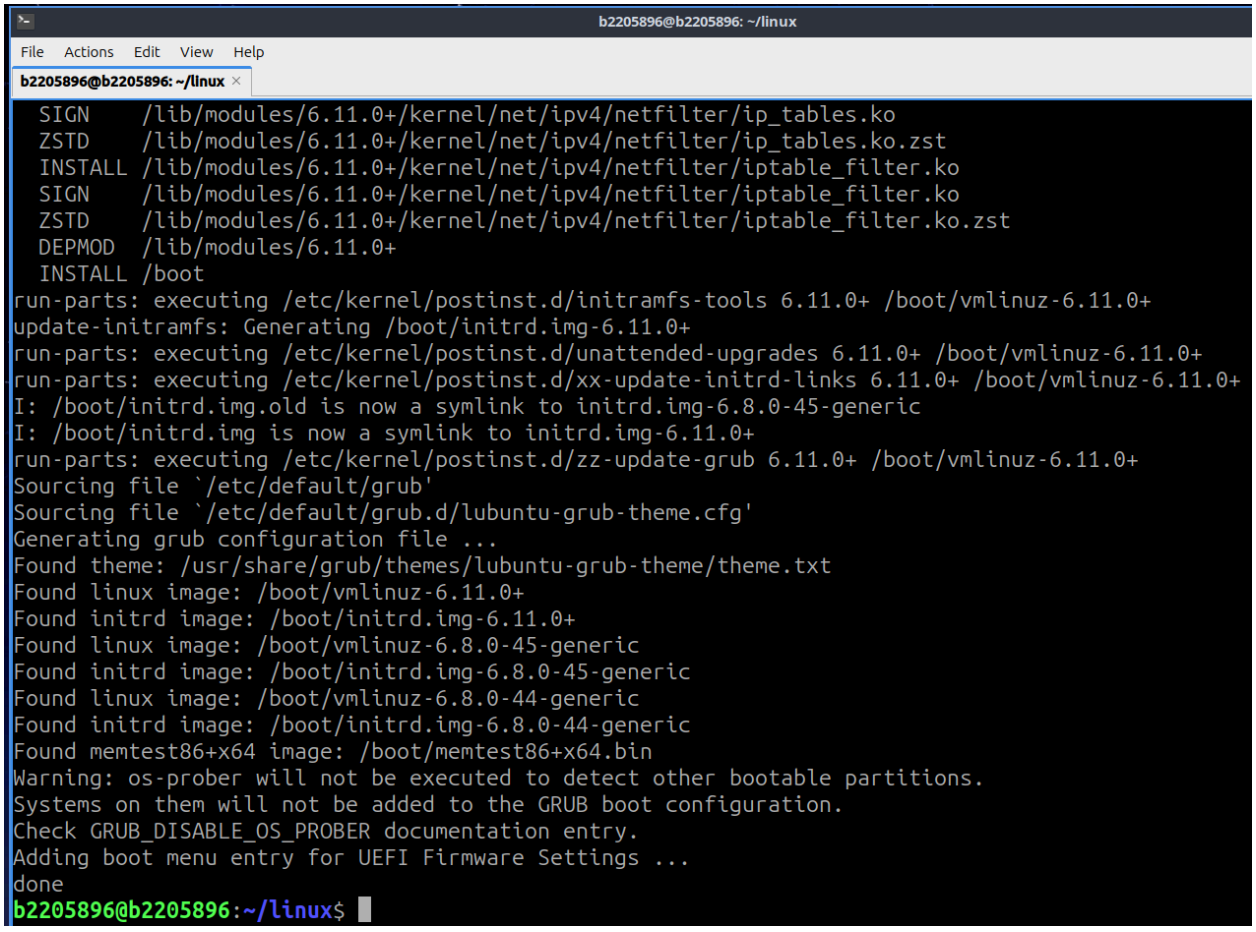




- Install the new kernel:

```
sudo make modules_install install
```


b2205896@b2205896: ~/linux
File Actions Edit View Help
b2205896@b2205896: ~/linux
b2205896@b2205896:~/linux$ sudo make modules_install install
[sudo] password for b2205896:
SYMLINK /lib/modules/6.11.0+/build
INSTALL /lib/modules/6.11.0+/modules.order
INSTALL /lib/modules/6.11.0+/modules.builtin
INSTALL /lib/modules/6.11.0+/modules.builtin.modinfo
INSTALL /lib/modules/6.11.0+/kernel/arch/x86/kernel/msr.ko
SIGN /lib/modules/6.11.0+/kernel/arch/x86/kernel/msr.ko
ZSTD /lib/modules/6.11.0+/kernel/arch/x86/kernel/msr.ko.zst
INSTALL /lib/modules/6.11.0+/kernel/arch/x86/crypto/aesni-intel.ko
SIGN /lib/modules/6.11.0+/kernel/arch/x86/crypto/aesni-intel.ko
ZSTD /lib/modules/6.11.0+/kernel/arch/x86/crypto/aesni-intel.ko.zst
INSTALL /lib/modules/6.11.0+/kernel/arch/x86/crypto/sha1-ssse3.ko
SIGN /lib/modules/6.11.0+/kernel/arch/x86/crypto/sha1-ssse3.ko
ZSTD /lib/modules/6.11.0+/kernel/arch/x86/crypto/sha1-ssse3.ko.zst
INSTALL /lib/modules/6.11.0+/kernel/arch/x86/crypto/sha256-ssse3.ko
SIGN /lib/modules/6.11.0+/kernel/arch/x86/crypto/sha256-ssse3.ko
ZSTD /lib/modules/6.11.0+/kernel/arch/x86/crypto/sha256-ssse3.ko.zst
INSTALL /lib/modules/6.11.0+/kernel/arch/x86/crypto/ghash-clmulni-intel.ko
SIGN /lib/modules/6.11.0+/kernel/arch/x86/crypto/ghash-clmulni-intel.ko

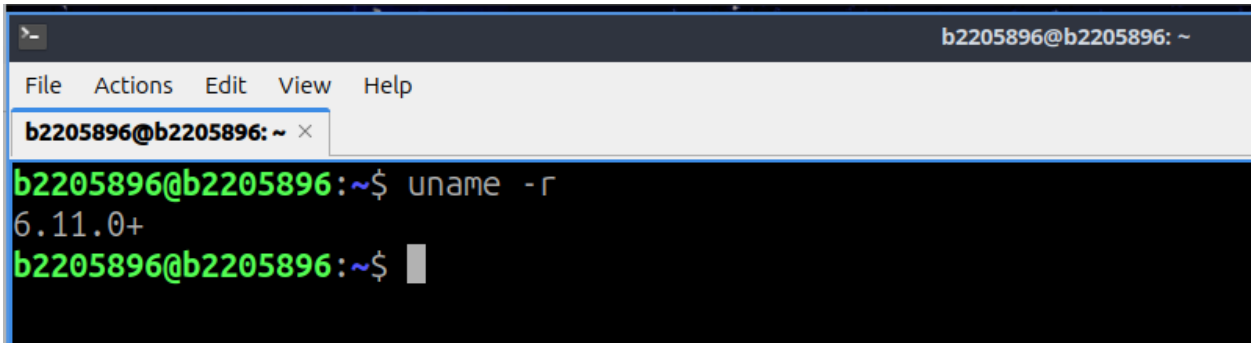

b2205896@b2205896: ~/linux
File Actions Edit View Help
b2205896@b2205896: ~/linux
SIGN /lib/modules/6.11.0+/kernel/net/ipv4/netfilter/ip_tables.ko
ZSTD /lib/modules/6.11.0+/kernel/net/ipv4/netfilter/ip_tables.ko.zst
INSTALL /lib/modules/6.11.0+/kernel/net/ipv4/netfilter/iptable_filter.ko
SIGN /lib/modules/6.11.0+/kernel/net/ipv4/netfilter/iptable_filter.ko
ZSTD /lib/modules/6.11.0+/kernel/net/ipv4/netfilter/iptable_filter.ko.zst
DEPMOD /lib/modules/6.11.0+
INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.11.0+ /boot/vmlinuz-6.11.0+
update-initramfs: Generating /boot/initrd.img-6.11.0+
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 6.11.0+ /boot/vmlinuz-6.11.0+
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 6.11.0+ /boot/vmlinuz-6.11.0+
I: /boot/initrd.img.old is now a symlink to initrd.img-6.8.0-45-generic
I: /boot/initrd.img is now a symlink to initrd.img-6.11.0+
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.11.0+ /boot/vmlinuz-6.11.0+
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/lubuntu-grub-theme.cfg'
Generating grub configuration file ...
Found theme: /usr/share/grub/themes/lubuntu-grub-theme/theme.txt
Found linux image: /boot/vmlinuz-6.11.0+
Found initrd image: /boot/initrd.img-6.11.0+
Found linux image: /boot/vmlinuz-6.8.0-45-generic
Found initrd image: /boot/initrd.img-6.8.0-45-generic
Found linux image: /boot/vmlinuz-6.8.0-44-generic
Found initrd image: /boot/initrd.img-6.8.0-44-generic
Found memtest86+x64 image: /boot/memtest86+x64.bin
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
b2205896@b2205896:~/linux$

- 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
```

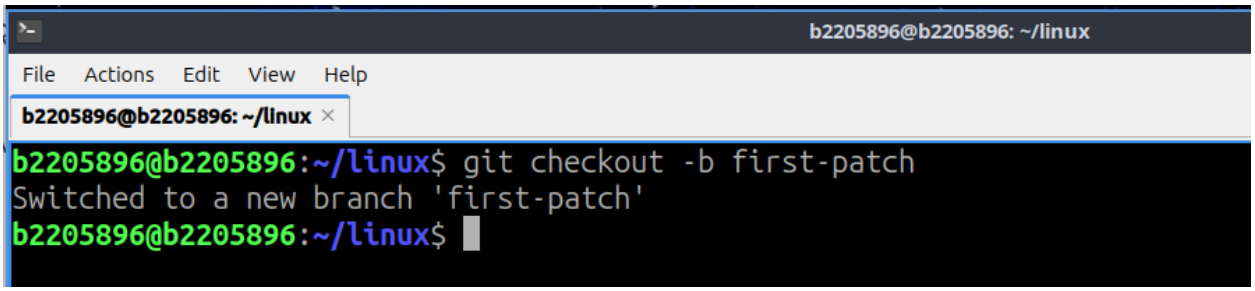


(take screenshots to show that you finish this exercise)

4. Writing Your First Kernel Patch

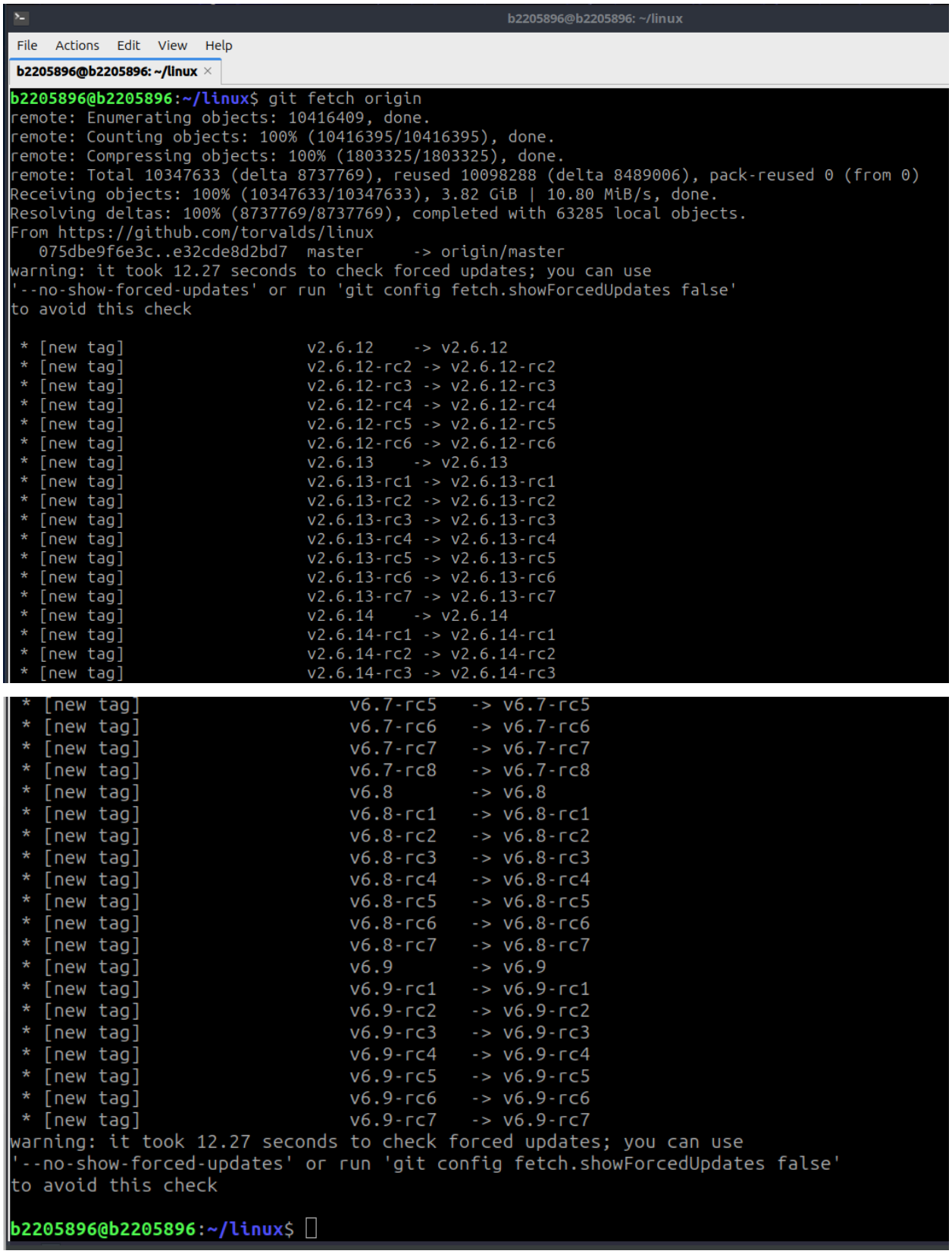
- Creating a new branch in the linux_mainline repository (has been cloned in exercise 3)

```
git checkout -b first-patch
```



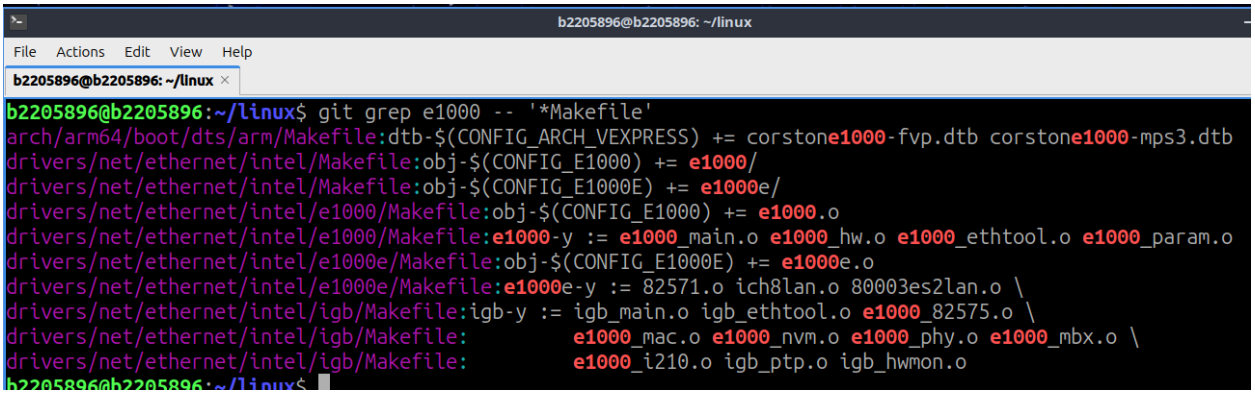
- Update the kernel

```
git fetch origin
```


b2205896@b2205896: ~/linux
File Actions Edit View Help
b2205896@b2205896: ~/linux
b2205896@b2205896:~/linux$ git fetch origin
remote: Enumerating objects: 10416409, done.
remote: Counting objects: 100% (10416395/10416395), done.
remote: Compressing objects: 100% (1803325/1803325), done.
remote: Total 10347633 (delta 8737769), reused 10098288 (delta 8489006), pack-reused 0 (from 0)
Receiving objects: 100% (10347633/10347633), 3.82 GiB | 10.80 MiB/s, done.
Resolving deltas: 100% (8737769/8737769), completed with 63285 local objects.
From https://github.com/torvalds/linux
075dbe9f6e3c..e32cde8d2bd7 master -> origin/master
warning: it took 12.27 seconds to check forced updates; you can use
'--no-show-forced-updates' or run 'git config fetch.showForcedUpdates false'
to avoid this check
* [new tag] v2.6.12 -> v2.6.12
* [new tag] v2.6.12-rc2 -> v2.6.12-rc2
* [new tag] v2.6.12-rc3 -> v2.6.12-rc3
* [new tag] v2.6.12-rc4 -> v2.6.12-rc4
* [new tag] v2.6.12-rc5 -> v2.6.12-rc5
* [new tag] v2.6.12-rc6 -> v2.6.12-rc6
* [new tag] v2.6.13 -> v2.6.13
* [new tag] v2.6.13-rc1 -> v2.6.13-rc1
* [new tag] v2.6.13-rc2 -> v2.6.13-rc2
* [new tag] v2.6.13-rc3 -> v2.6.13-rc3
* [new tag] v2.6.13-rc4 -> v2.6.13-rc4
* [new tag] v2.6.13-rc5 -> v2.6.13-rc5
* [new tag] v2.6.13-rc6 -> v2.6.13-rc6
* [new tag] v2.6.13-rc7 -> v2.6.13-rc7
* [new tag] v2.6.14 -> v2.6.14
* [new tag] v2.6.14-rc1 -> v2.6.14-rc1
* [new tag] v2.6.14-rc2 -> v2.6.14-rc2
* [new tag] v2.6.14-rc3 -> v2.6.14-rc3
* [new tag] v6.7-rc5 -> v6.7-rc5
* [new tag] v6.7-rc6 -> v6.7-rc6
* [new tag] v6.7-rc7 -> v6.7-rc7
* [new tag] v6.7-rc8 -> v6.7-rc8
* [new tag] v6.8 -> v6.8
* [new tag] v6.8-rc1 -> v6.8-rc1
* [new tag] v6.8-rc2 -> v6.8-rc2
* [new tag] v6.8-rc3 -> v6.8-rc3
* [new tag] v6.8-rc4 -> v6.8-rc4
* [new tag] v6.8-rc5 -> v6.8-rc5
* [new tag] v6.8-rc6 -> v6.8-rc6
* [new tag] v6.8-rc7 -> v6.8-rc7
* [new tag] v6.9 -> v6.9
* [new tag] v6.9-rc1 -> v6.9-rc1
* [new tag] v6.9-rc2 -> v6.9-rc2
* [new tag] v6.9-rc3 -> v6.9-rc3
* [new tag] v6.9-rc4 -> v6.9-rc4
* [new tag] v6.9-rc5 -> v6.9-rc5
* [new tag] v6.9-rc6 -> v6.9-rc6
* [new tag] v6.9-rc7 -> v6.9-rc7
warning: it took 12.27 seconds to check forced updates; you can use
'--no-show-forced-updates' or run 'git config fetch.showForcedUpdates false'
to avoid this check
b2205896@b2205896:~/linux$

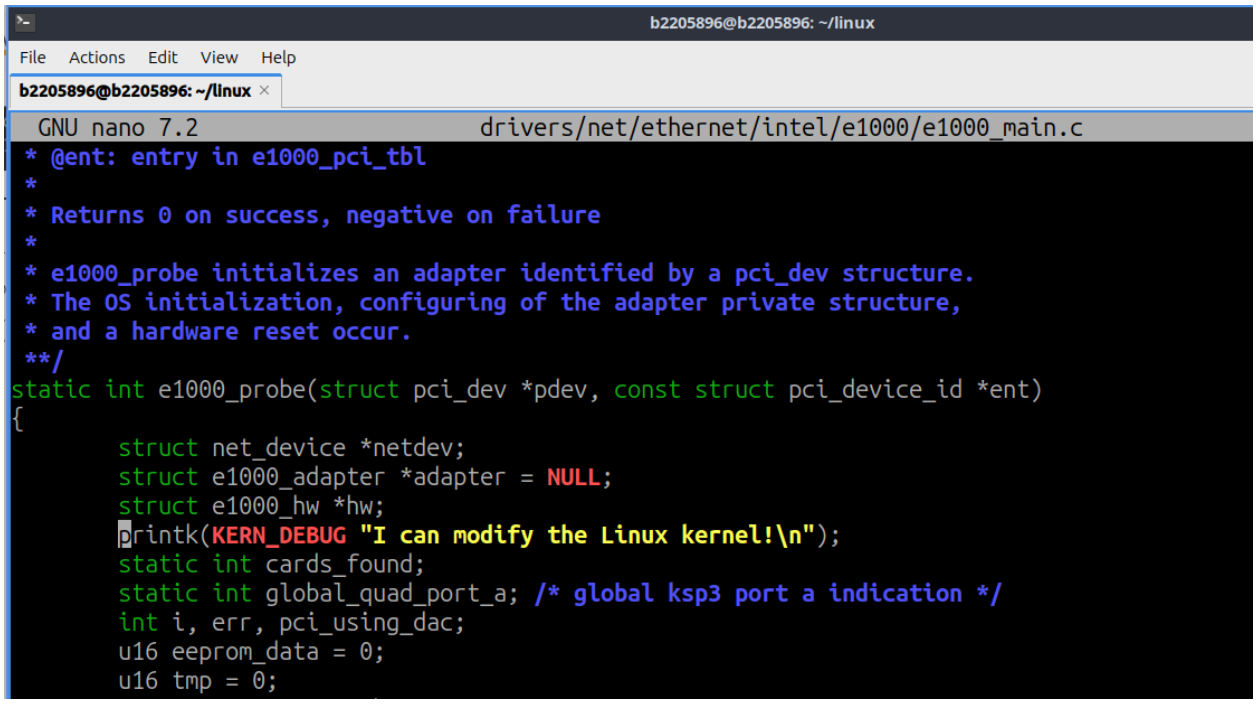
- Run `lsmod` to see the modules loaded on your system, and pick a driver to change. One driver that's included in all VM images is the `e1000` driver, the Intel ethernet driver, or you can choose another driver depending on your working environment.
- Run `git grep` to look for `e1000` files

```
git grep e1000 -- '*Makefile'
```



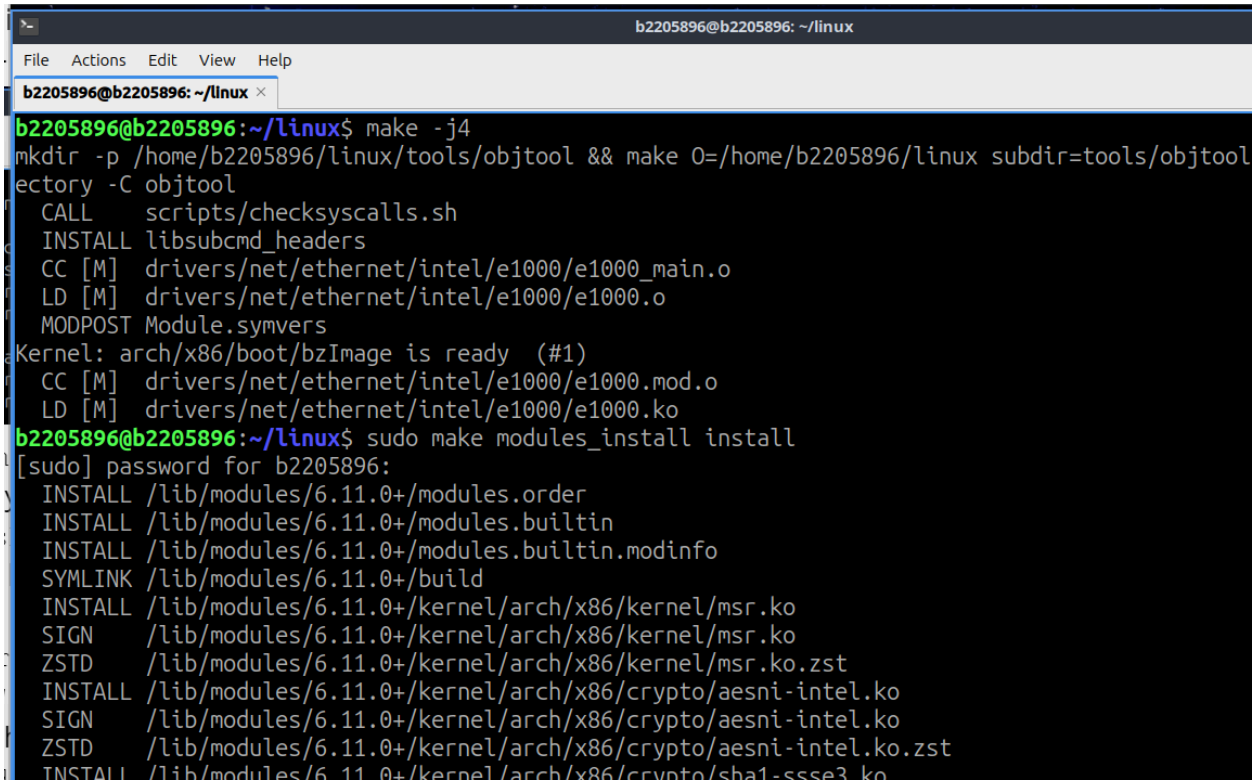
- Make a small change to the probe function of the e1000 driver

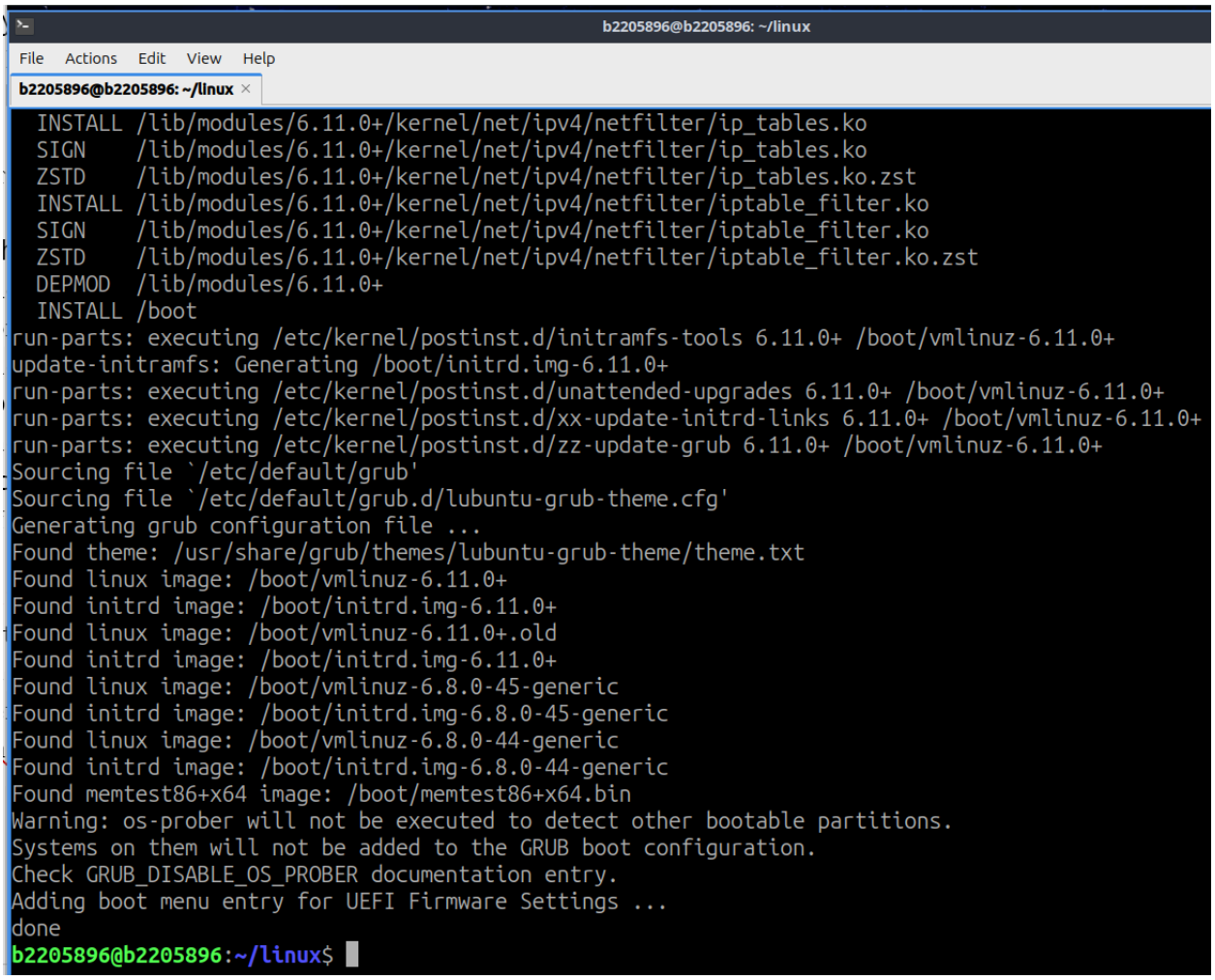
```
nano drivers/net/ethernet/intel/e1000/e1000_main.c
# Add a line of code as below
static int e1000_probe(struct pci_dev *pdev, const struct pci_device_id *ent) {
...
struct e1000_hw *hw;
printk(KERN_DEBUG "I can modify the Linux kernel!\n");
static int cards_found = 0;
...
```



- Compile and install your changes:

```
make -j3
sudo make modules_install install
```



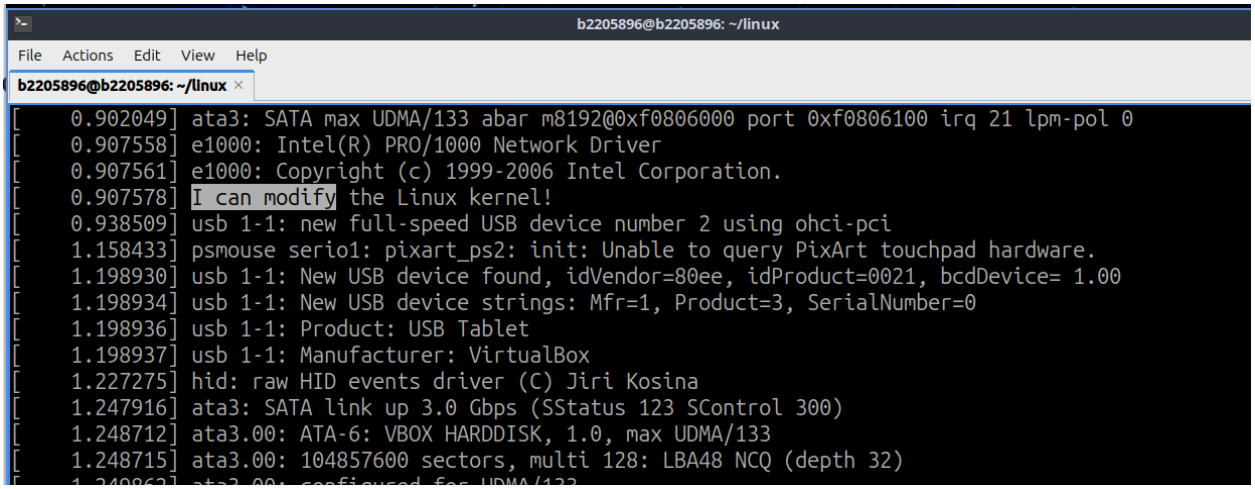


- Reboot the system:

```
sudo shutdown -r now
```

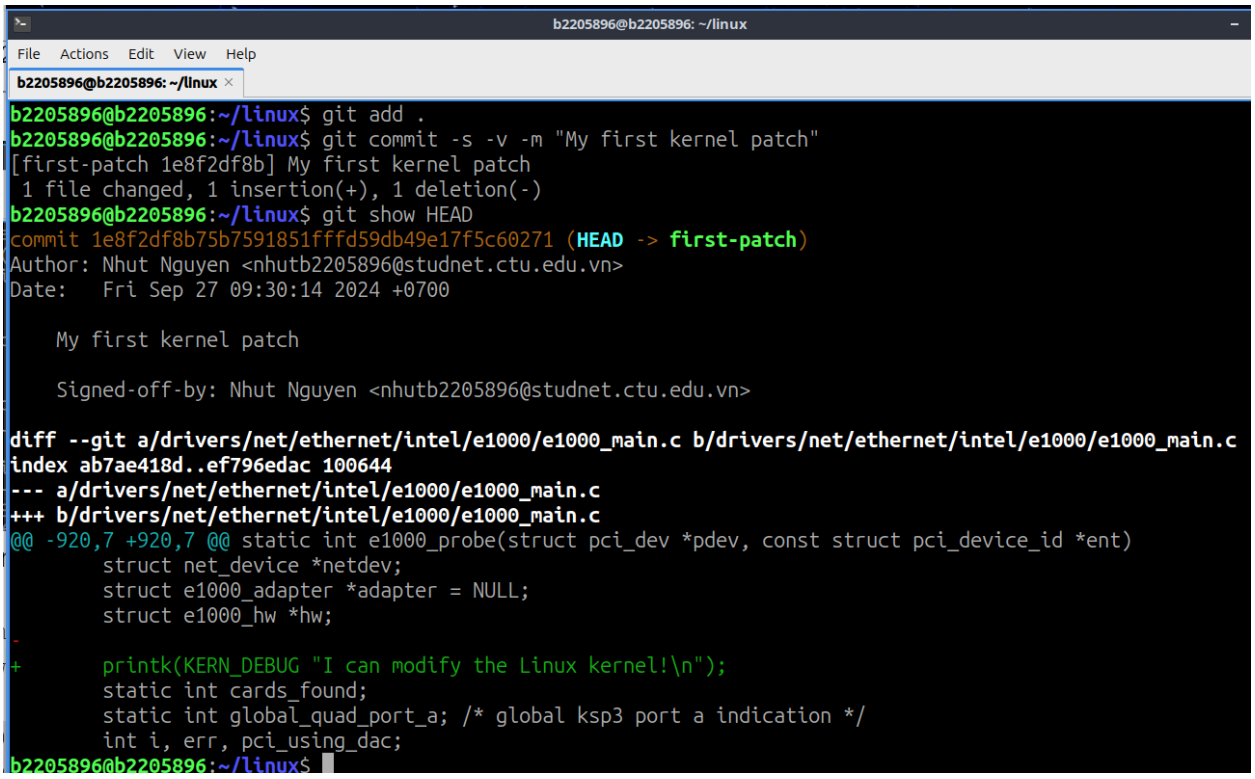
- Show kernel buffer log:

```
sudo dmesg | less
# Search for your printk in the log file by typing "/I can modify"
```



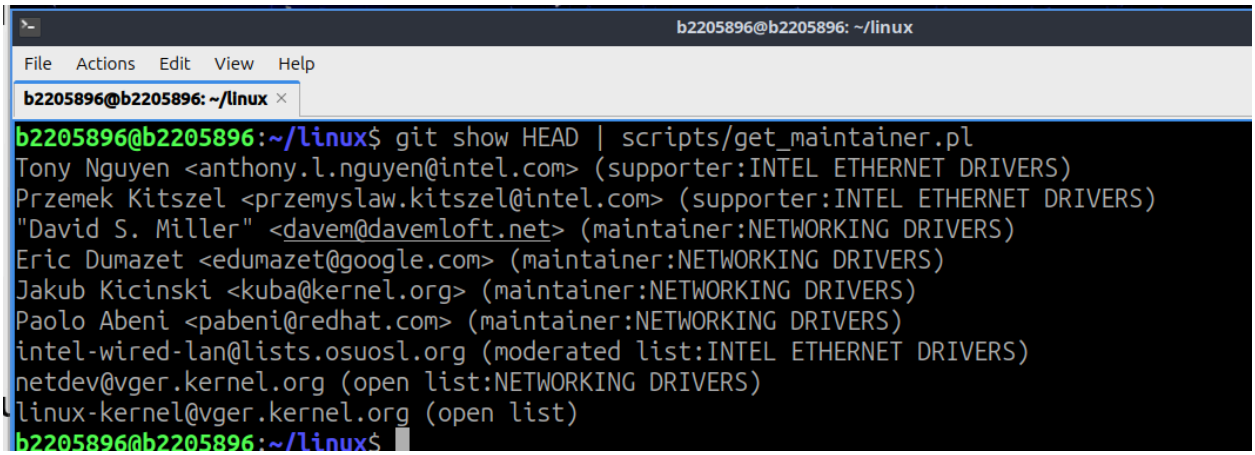
- Committing changes, and view your commit

```
git add .
git commit -s -v -m "My first kernel patch"
git show HEAD
```



- Find whom to send the patch to

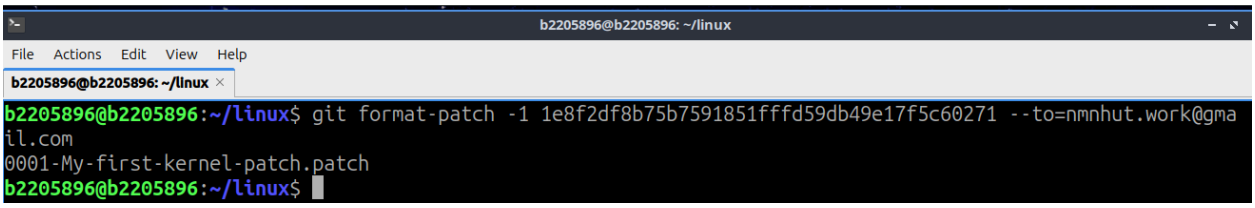
```
git show HEAD | scripts/get_maintainer.pl
```



- Create a patch

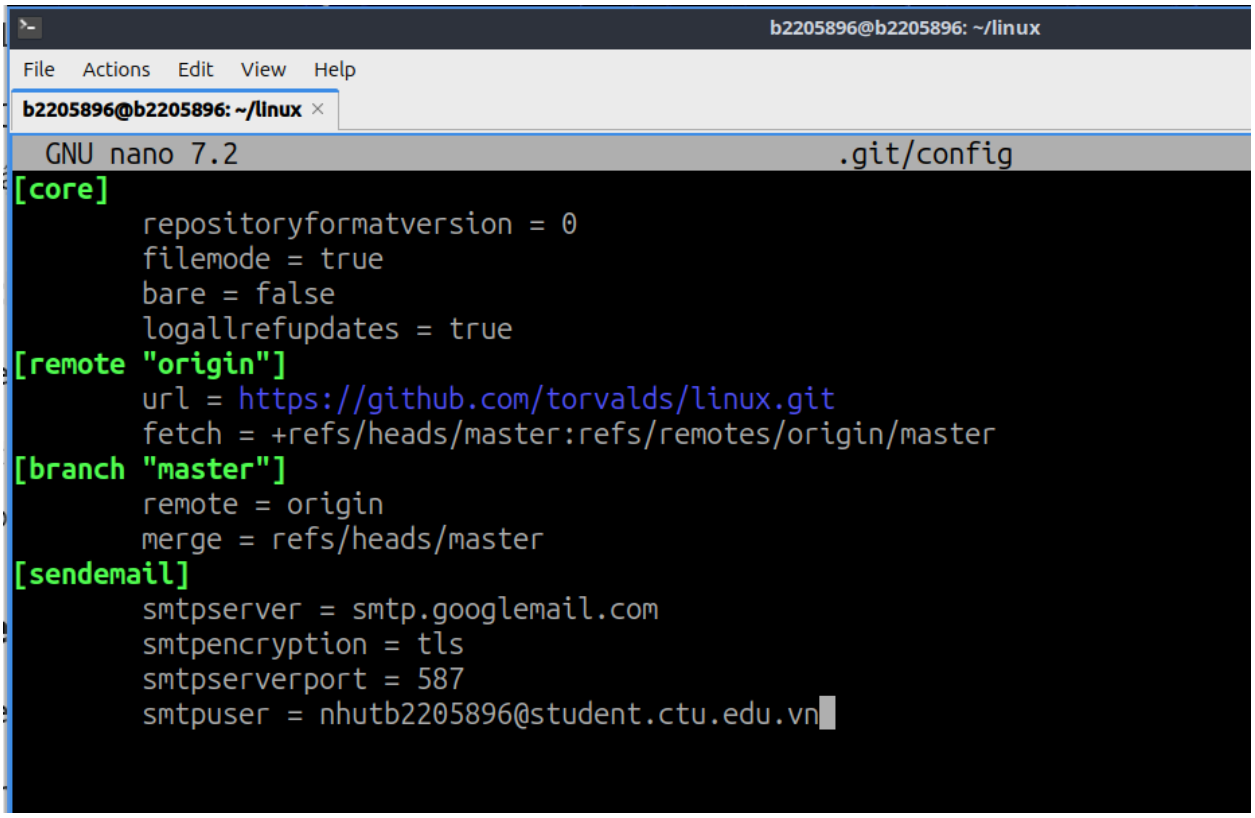
```
git format-patch -1 <commit ID> --to=<your email>
```

Note: Please do not send your patch to a maintainer, send it to yourself instead.



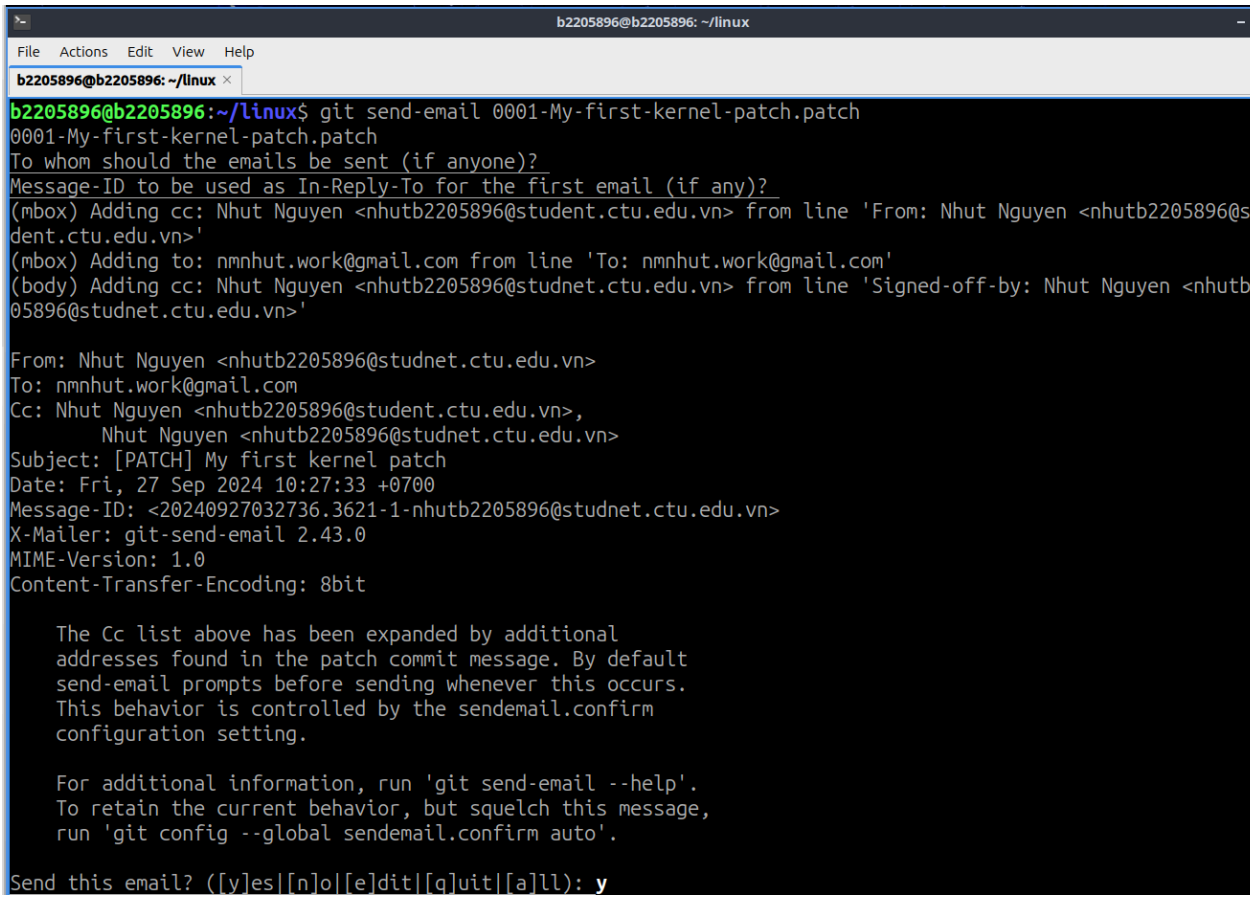
- Modify `./git/config` file to configure send-email

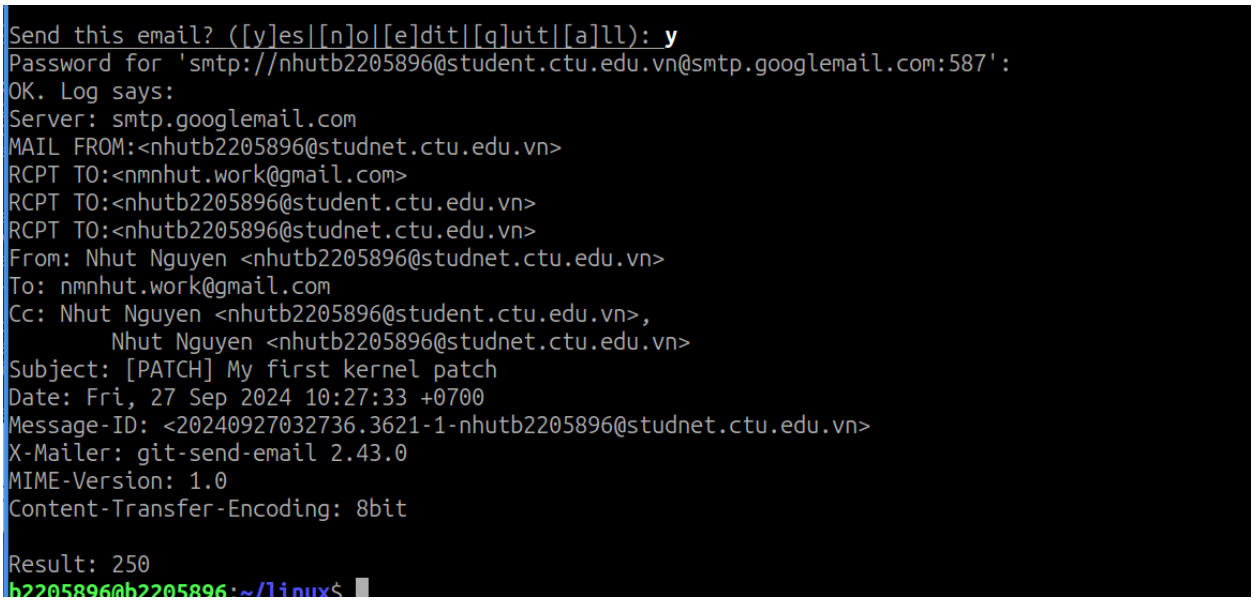
```
 #.git/config
[sendemail]
    smtpserver = smtp.googlemail.com
    smtpencryption = tls
    smtpserverport = 587
   smtpuser = your gmail address (CTU student email is OK
```



- Send the patch

```
git send-email <patch_file>
```





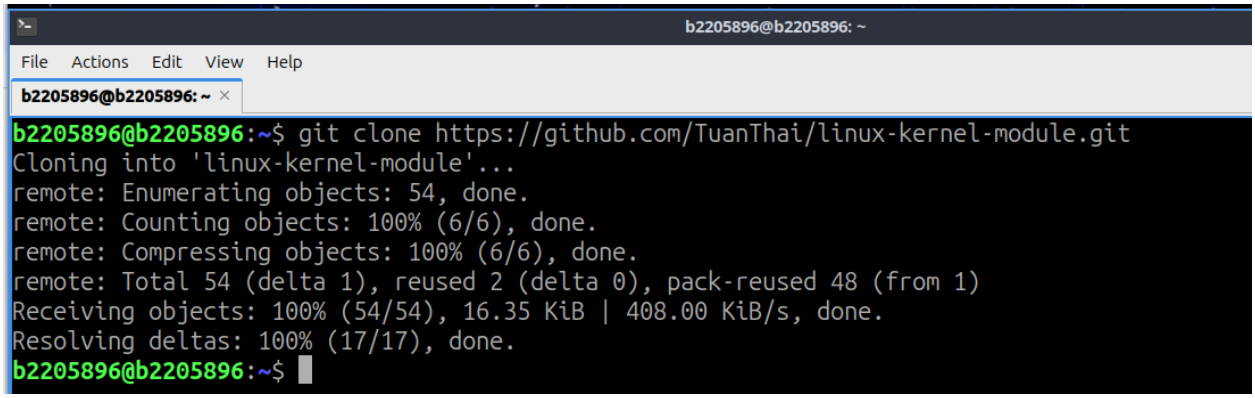
(take screenshots to show that you finish this exercise)

5. Writing a simple Linux kernel module: Greeter sample

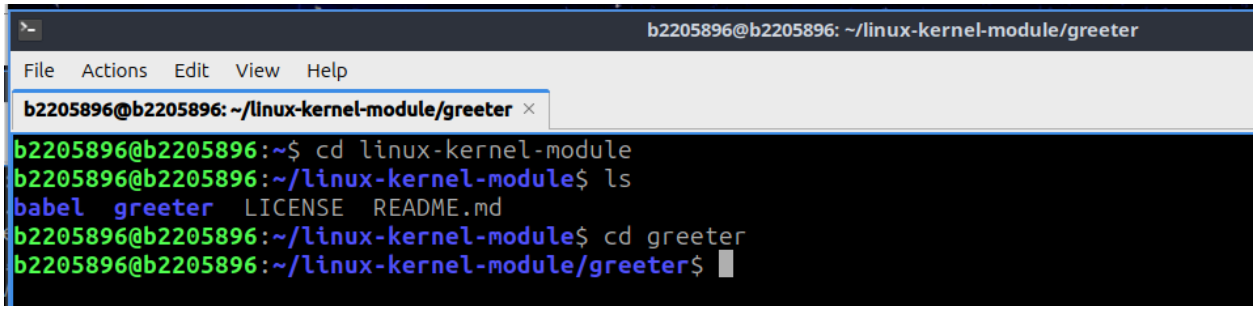
This module simply takes a name as a parameter, and writes a greeting to the kernel log (/var/log/kern.log):

- Clone this repository to your computer:

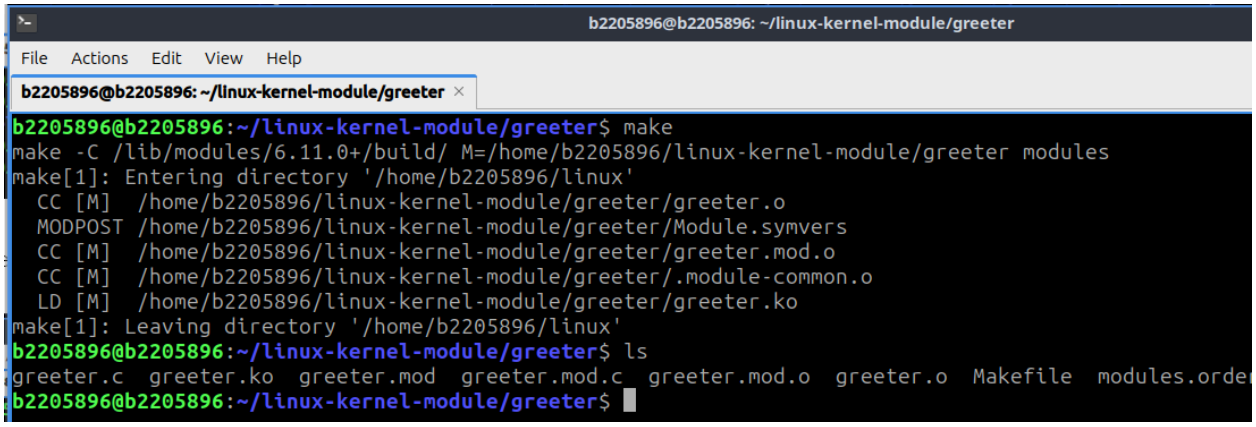
https://github.com/TuanThai/linux-kernel-module.git



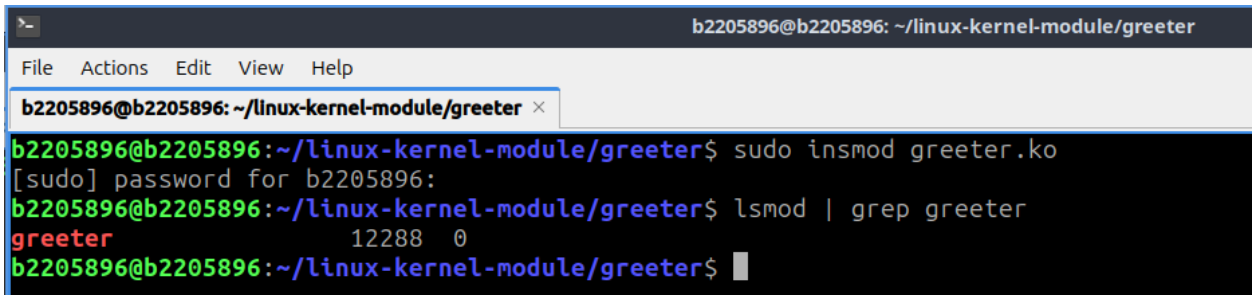
- Move into `greeter/` directory.



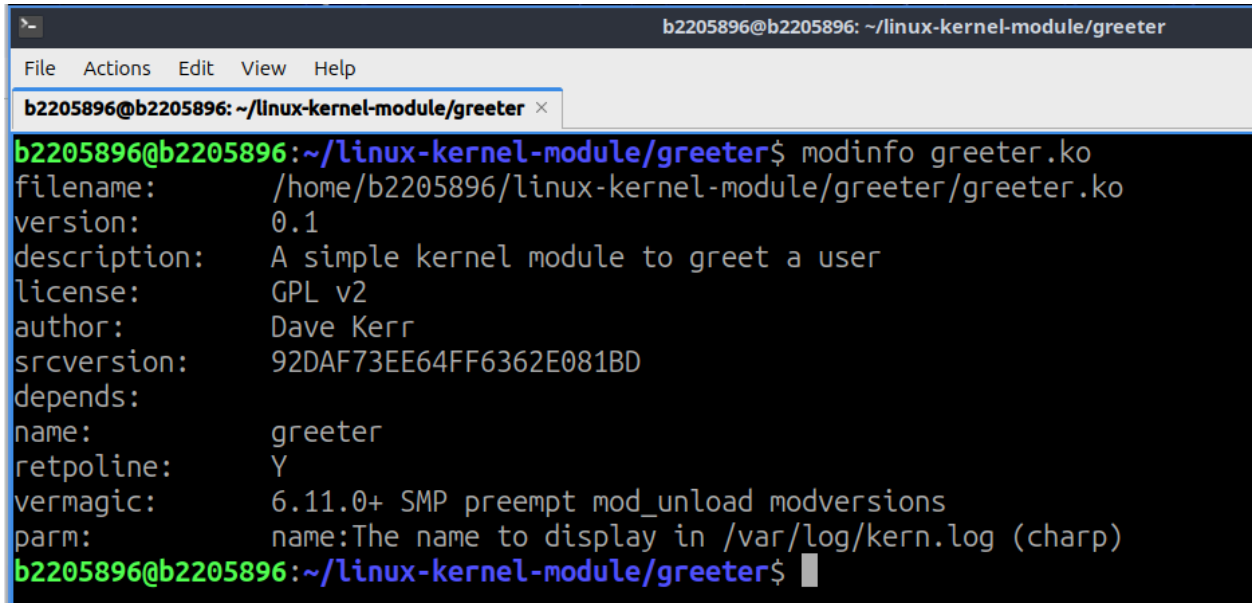
- Build the module using `make` command. The module is compiled to `greeter.ko`



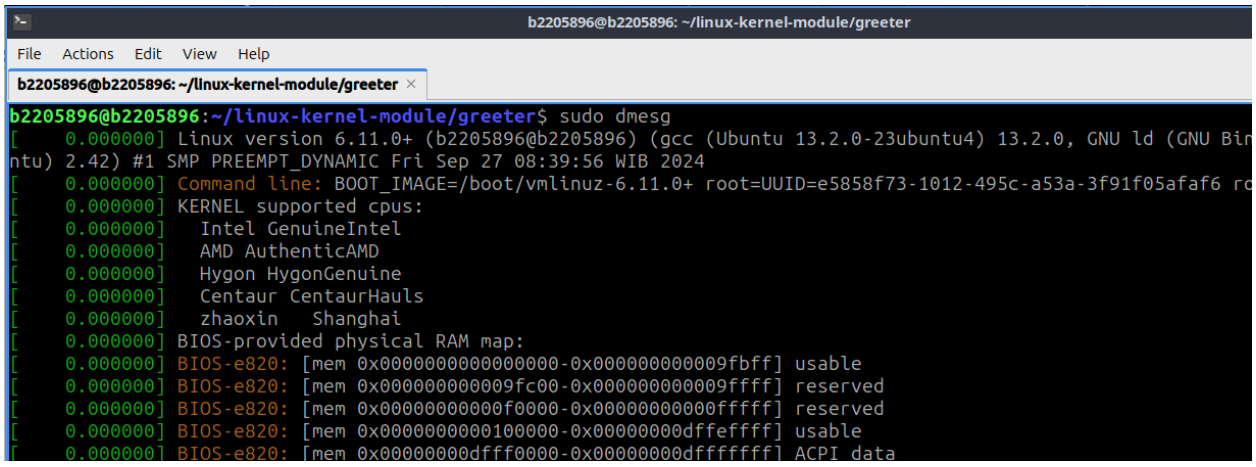
- Install the module using `insmod greeter.ko` command, then show that the module has been installed using `lsmod | grep greeter` command



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

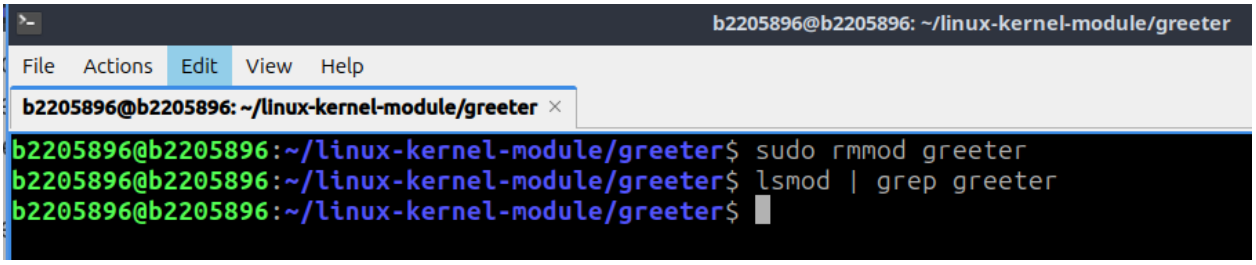


- Show kernel log with `dmesg`

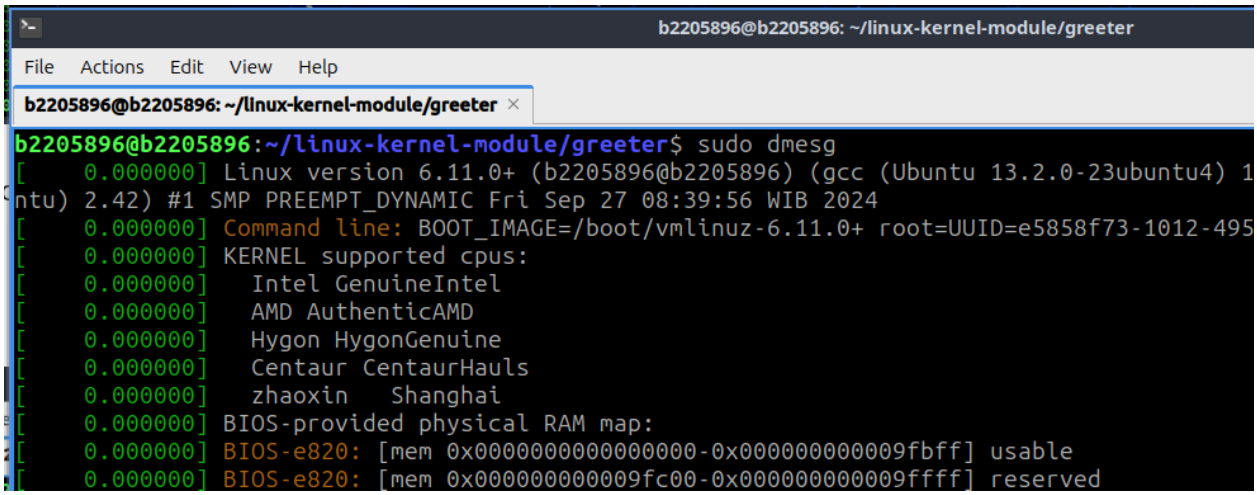


```
[  543.230072] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 7 times, consider switching to WQ_
[  681.979457] workqueue: blk_mq_run_work_fn hogged CPU for >10000us 11 times, consider switching to WQ_UNB
[  776.937850] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 11 times, consider switching to WQ
[ 1733.232671] workqueue: vmstat_shepherd hogged CPU for >10000us 4 times, consider switching to WQ_UNBOUND
[ 1832.381868] workqueue: vmstat_shepherd hogged CPU for >10000us 5 times, consider switching to WQ_UNBOUND
[ 2029.042063] greeter: loading out-of-tree module taints kernel.
[ 2029.042075] greeter: module verification failed: signature and/or required key missing - tainting kernel
[ 2029.044973] greeter: module loaded at 0x00000000d969e2b8
[ 2029.044979] greeter: greetings Bilbo
b2205896@b2205896:~/linux-kernel-module/greeter$
```

- 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`




```
[  681.979457] workqueue: blk_mq_run_work_fn hogged CPU for >10000us 11 times, consider su
[  776.937850] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 11 times, consid
[ 1733.232671] workqueue: vmstat_shepherd hogged CPU for >10000us 4 times, consider switc
[ 1832.381868] workqueue: vmstat_shepherd hogged CPU for >10000us 5 times, consider switc
[ 2029.042063] greeter: loading out-of-tree module taints kernel.
[ 2029.042075] greeter: module verification failed: signature and/or required key missing
[ 2029.044973] greeter: module loaded at 0x00000000d969e2b8
[ 2029.044979] greeter: greetings Bilbo
[ 2186.924088] workqueue: e1000_watchdog [e1000] hogged CPU for >10000us 19 times, consid
[ 2234.939024] greeter: goodbye Bilbo
[ 2234.939034] greeter: module unloaded from 0x00000000a4dc832e
b2205896@b2205896:~/linux-kernel-module/greeter$
```

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

```
greeter_init
greeter_exit
module_init(greeter_init)
module_exit(greeter_exit)
```

The brief explanation of the functions in `greeter.c`:

- **greeter_init**: It's invoked when the module is loaded into the kernel. Within this function:
    - pr_info logs a message indicating the module's loading address.
    - It displays a greeting to the user, and the **name** parameter.
    - Returns 0 to signify successful initialization.

- **greeter_exit**: This function is called when the module is unloaded from the kernel. Its responsibilities include:
    - Logging a message to the user.
    - Indicating the module's unloading address.

- **module_init(greeter_init)**: This macro registers the greeter_init function as the entry point for the module. It tells the kernel to call greeter_init when the module is loaded.
.
- **module_exit(greeter_exit)**: This macro registers the greeter_exit function as the exit point for the module. It tells the kernel to call greeter_exit when the module is unloaded.

(take screenshots to show that you finish this exercise)

---END---