Introduction to Operative Systems

Project 1: Linux kernel download, patch, compilation, debugging and profiling

Project 1A:

Linux kernel download, patch, compilation and debugging (To be lectured on 2020-09-30)

Project 1B:

Linux kernel profiling (To be lectured on 2020-10-16)

Deadline:

- **Project 1A:** 2020-10-16 (Fri) 23:59:59.
- **Project 1B:** 2020-10-30 (Fri) 23:59:59.

Q&A:

If you have any questions, please post it on the E3 discussion board, and it will be answered in two days.

Deliverables:

1. **Demo video** (3-5 minutes – upload to YouTube – add link in the first page of the report).
2. **Report** (pdf file with file name of the form **OS_Project1A_StudentID.pdf**)
    - Screenshots + one explanation paragraph per screenshot.
    - Answers to questions below.
3. In the demo video and report, for each screenshot, explain:
    a. What has been done, and
    b. The reasoning behind the steps.

Objective: The objective of this project is to help the student to get familiar with basic concepts and tools related to the Operative Systems class. The student will learn how to install and use the tools needed for compiling, debugging and profiling the Linux Kernel.

Scope: Identify the basic components of a Linux-based operative system, and build a working environment capable of performing kernel compilation, debug and profiling.

Tools to use:

1. OS: Ubuntu 16.04.7 (AMD64)
    - https://releases.ubuntu.com/16.04/
2. Vmware player 15:
    - https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html
3. Gcc.
4. Visual studio code.

5. DDD:
   - https://www.gnu.org/software/ddd/manual/html_mono/ddd.html#Sample%20Session
   - https://www.gnu.org/software/ddd/manual/html_mono/ddd.html#Sample%20Program
6. Linux Kernel:
   - https://www.kernel.org/
7. KDB – KGDB:
   - http://landley.net/kdocs/Documentation/DocBook/xhtml-nochunks/kgdb.html#CompileKGDB

Table of contents:

- Section 1: Operative system installation and additional configurations
- Section 2: Linux Kernel download
- Section 3: Linux Kernel patch and build
- Section 4: Linux Kernel debug [KGDB]

Questions to be answered in the report:

1. What is a Kernel? What are the differences between *mainline, stable* and *longterm*? What is a Kernel panic?
2. What are the differences between *building*, *debugging* and *profiling*?
3. What are GCC, GDB, and KGDB, and what they are used for?
4. What are the /usr/, /boot/, /home/, /boot/grub folders for?
5. What are the general steps to debug a Linux Kernel? **(Extra points if you add at least two figures explaining the needed steps).**
6. For this project, why do we need two virtual machines?
7. In Section 3.3, what are the differences between **make**, **make modules_install** and **make install**?
8. In Section 3.4, what are the commands **kgdbwait** and **kgdboc=ttyS1,115200** for?
9. What is **grub**? What is **grub.cfg**?
10. List at least 10 commands you can use with GDB **(Extra points if you add at least two figures).**

SECTION 1:

OPERATIVE SYSTEM INSTALLATION AND ADDITIONAL CONFIGURATIONS

## Section 1: Operative system installation

In this section, we will install the operative system required for this project, and perform some initial configurations prior proceeding with the Linux Kernel debugging.

Section 1.1: Installation

1. Install VMWare Player 15
   https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html
2. Create a folder called "VHD-VMWarePlayer" in a central location (example C or D), and create two subfolders:
   a. **Ubuntu1604Target** (This will be the virtual machine that will contain our Linux Kernel code).
   b. **Ubuntu1604Debugger** (This will be the virtual machine that will debug the target).
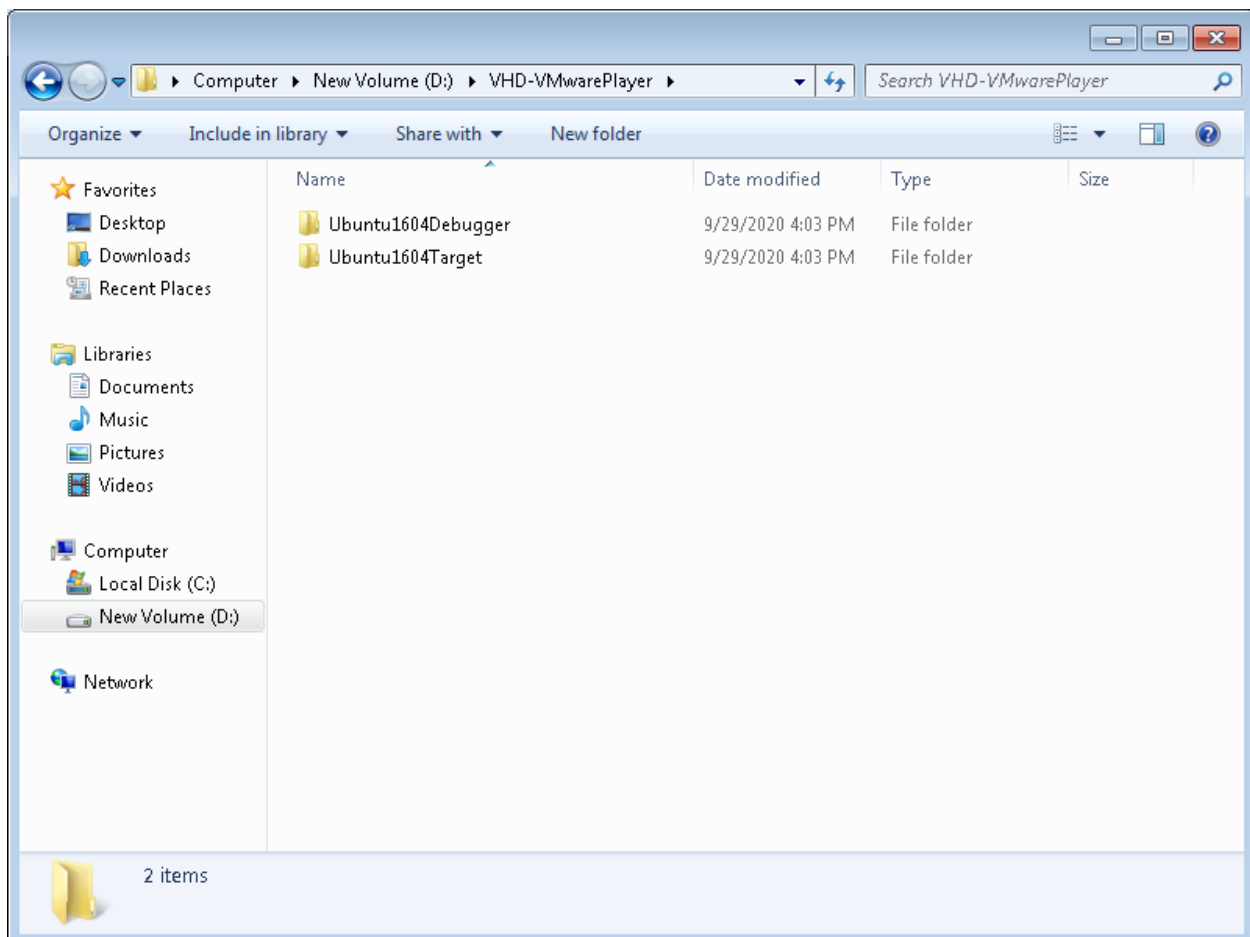
3. Create two virtual machines (named Ubuntu1604Target and Ubuntu1604Host) using Ubuntu 16.04.7 and save them in the folders mentioned above. For these virtual machines, use the following specs:
    a. 2Gb RAM (2048 Mb)
    b. 100 Gb Hard drive (Select **Split virtual disk into multiple files)**
    c. 1 Processor core.
    d. Use the following usernames
        i.   Ubuntu1604Target username: **usertest+StudentID**
        ii. Ubuntu1604Host username: **userhost+StudentID**
             Example: If your student ID is 12345, the users should be **usertest12345** and **userhost12345.**
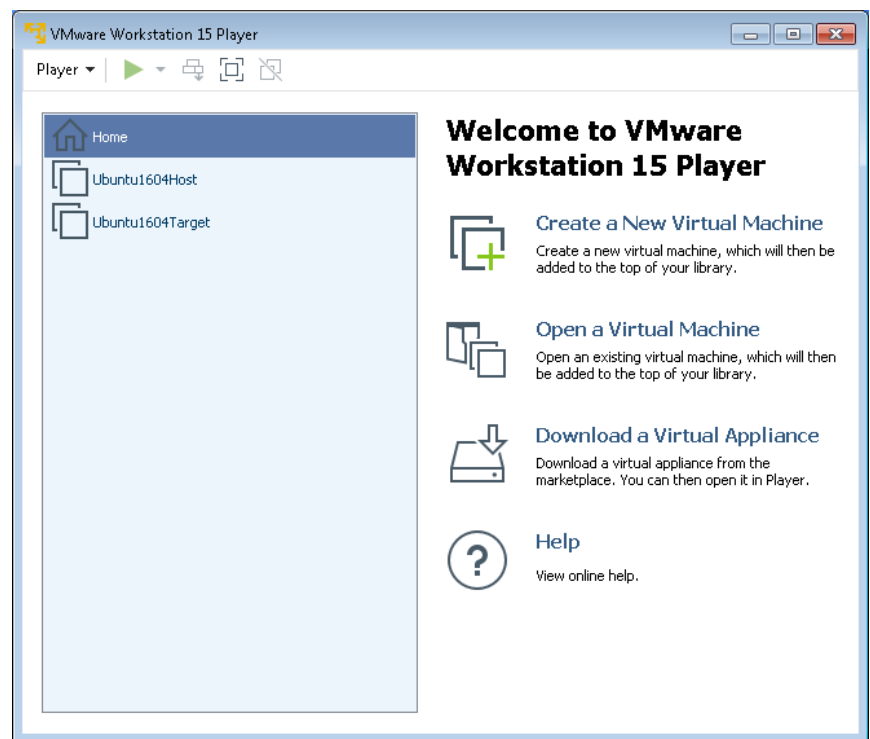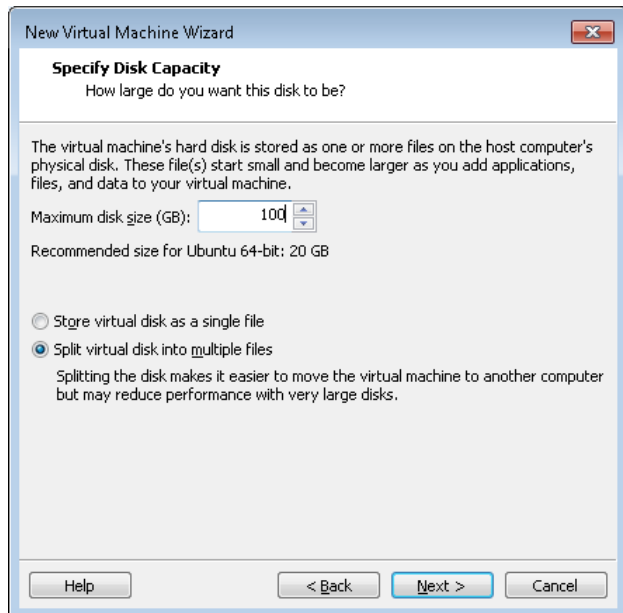
AT THIS POINT, YOU SHOULD HAVE TWO WORKING VIRTUAL MACHINES WITH THE SAME SPECS.

[Screenshot # 1: Create a screenshot with both virtual machines folder and add it to your video and report]

Section 1.2: Serial port communication

Now that the two virtual machines have been created, we need to enable two communication channels between them:

- **Serial port communication:** This will be the communication channel used when debugging (We cover how to activate it in this section).
- **SSH:** This will be the communication channel for when moving the compiled kernel's symbols from the target to the host machine (We cover how to activate it in the next section).

1. With both virtual machines **off**, we will modify the settings of each one of them as follows:
    a. **In the Target machine:**
        i. Add a Serial Port.
        ii. Select **Use named pipe**, and use **\\.\pipe\testpipe** as name.
        iii. Select **This end is the server** and **The other end is a virtual machine** options.
        iv. Select **Yield CPU on poll**.
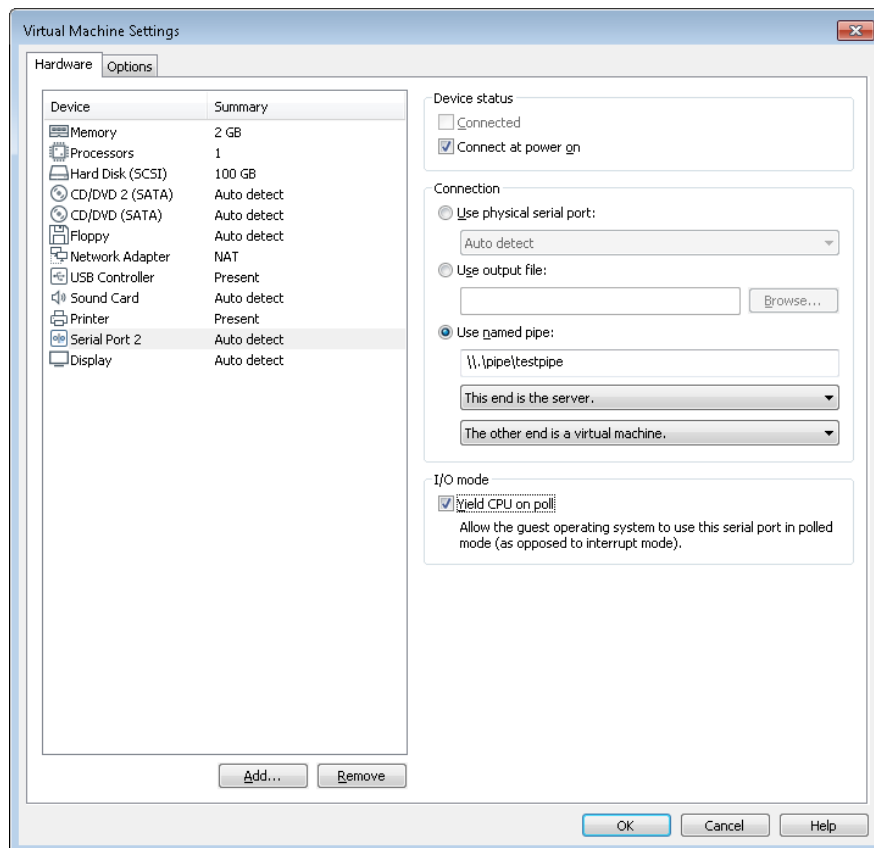
b. In the Host machine:
   - i. Add a Serial Port.
   - ii. Select **Use named pipe**, and use **\\.\pipe\testpipe** as name.
   - iii. Select **This end is the client** and **The other end is a virtual machine** options.
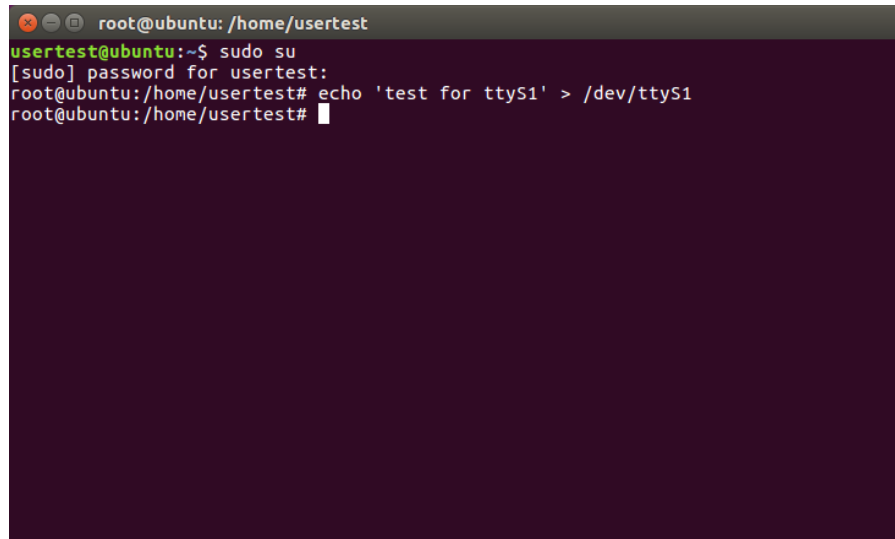   - iv. Select **Yield CPU on poll**.
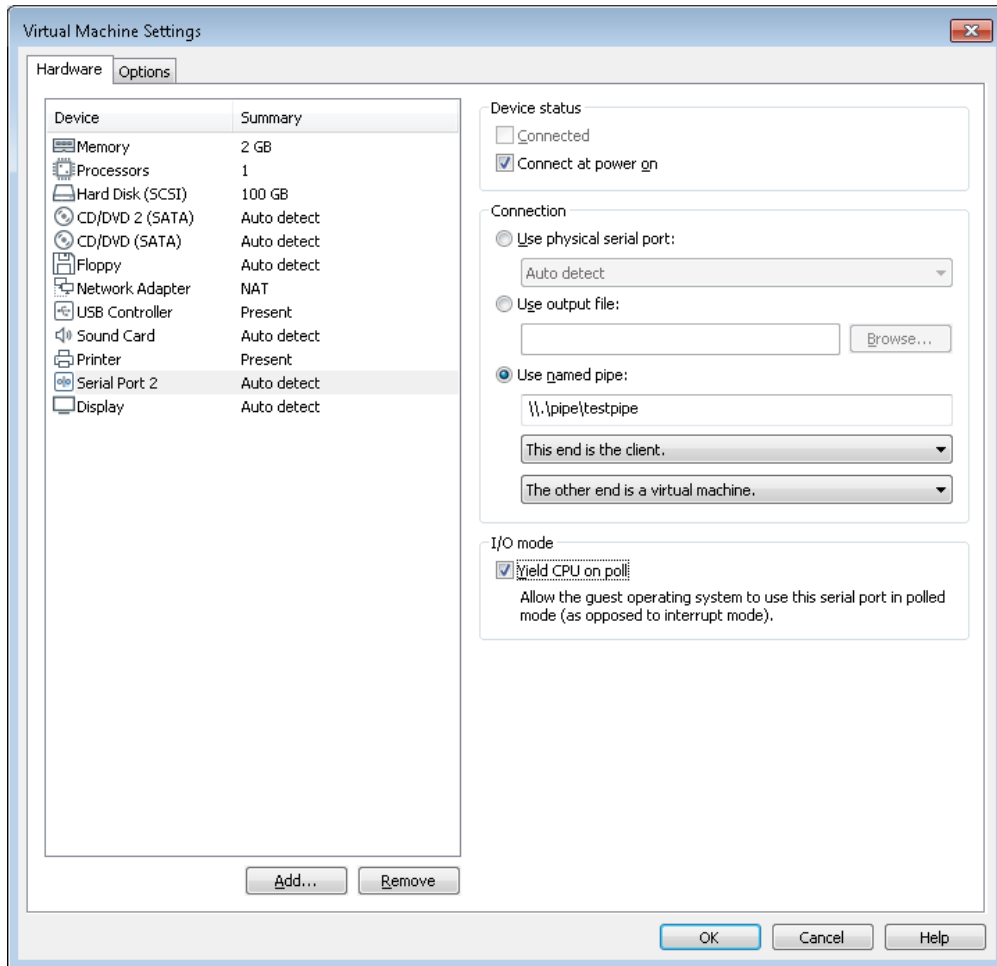
Important notes:

a. At this point, any time the serial port is connected in the Host machine, the Target must be on in order for it to boot.

b. If you need to turn them on individually, just disconnect the Serial Port prior to booting the machine as shown in the figure below.
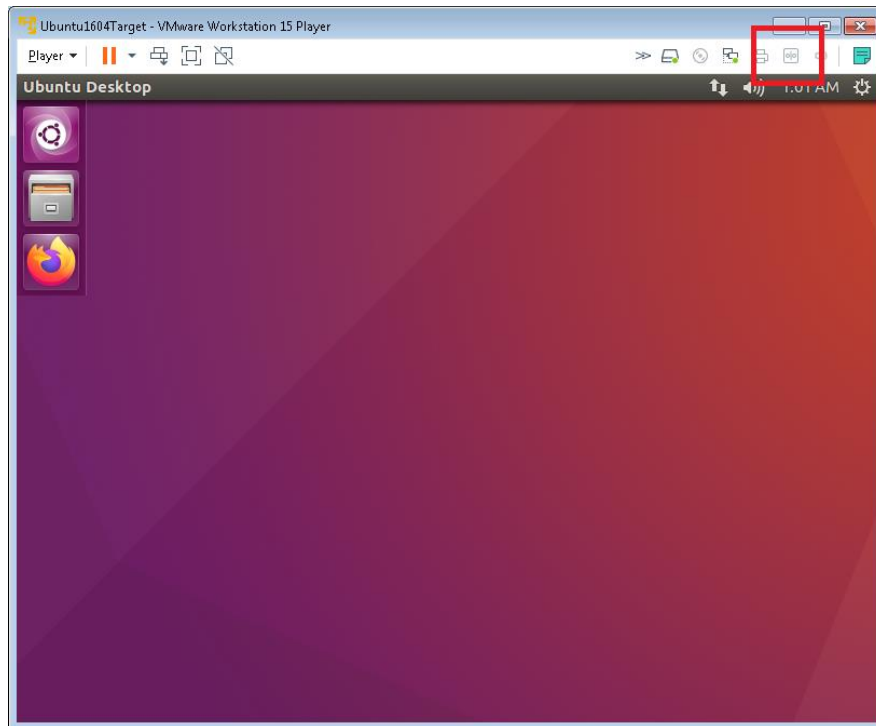
2. With the serial port connected, turn on both virtual machines (first turn on the Target machine, then the Host machine).
3. In the Host machine, open a terminal and type
   $ sudo su (followed by root's password)
   # cat /dev/ttyS1
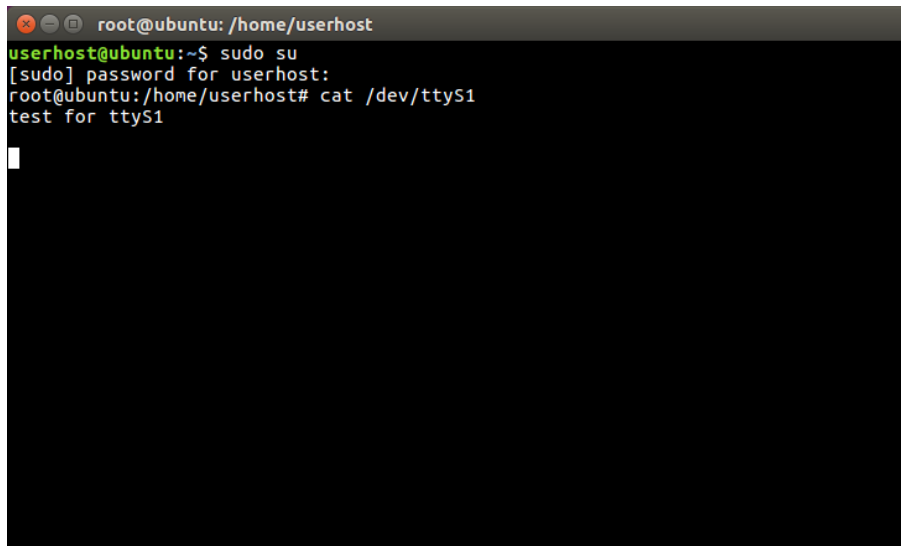
4. In the Terminal machine, open a terminal and type
   $sudo su (followed by root's password)
   #echo 'test for ttyS1 + studentID' > /dev/ttyS1

5. In the Host machine, in the terminal of step 3, the message **test for ttyS1** + studentID should appear.

AT THIS POINT, IF YOU SEE THE MESSAGE APPEARING IN THE HOST'S TERMINAL, THEN THE SERIAL PORT COMMUNICATION IS CORRECT.

[Screenshot # 2: Create a screenshot with both virtual machines messages and add it to your video and report]

6. Turn off both virtual machines and disconnect the serial port of each one of them.
   (DISCONNECT, DO NOT REMOVE)

Section 1.3: SSH Connection

In this section we will install the ssh server, so the virtual machines can communicate using ssh.

1. In the Target machine, install the openssh server using the command:
   $ sudo apt install openssh-server

```
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo apt install openssh-server
[sudo] password for usertest:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ncurses-term openssh-sftp-server ssh-import-id
Suggested packages:
  ssh-askpass rssh molly-guard monkeysphere
The following NEW packages will be installed:
  ncurses-term openssh-server openssh-sftp-server ssh-import-id
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 633 kB of archives.
After this operation, 5,136 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

2. Enable the ssh server through the firewall by using the command
   $sudo ufw allow ssh

```
Preparing to unpack .../ssh-import-id_5.5-0ubuntu1_all.deb ...
Unpacking ssh-import-id (5.5-0ubuntu1) ...
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for systemd (229-4ubuntu21.29) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for ufw (0.35-0ubuntu2) ...
Setting up ncurses-term (6.0+20160213-1ubuntu1) ...
Setting up openssh-sftp-server (1:7.2p2-4ubuntu2.10) ...
Setting up openssh-server (1:7.2p2-4ubuntu2.10) ...
Creating SSH2 RSA key; this may take some time ...
2048 SHA256:7PoEzf6iwaj7a65gc6VWe6RDNKKRlUfipnmRpVS+puQ root@ubuntu (RSA)
Creating SSH2 DSA key; this may take some time ...
1024 SHA256:BE9OxM5CwsIRu3Zl/toun9V1367N3OU2QEEzF7hFZ4o root@ubuntu (DSA)
Creating SSH2 ECDSA key; this may take some time ...
256 SHA256:QiBMABnOAUHJgxwvy9dvFdrBMHfJ+Vk7VOpNq5HPbGQ root@ubuntu (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 SHA256:JZ0dEjmoZzMcSqW9XSJEJiXlxvmtVqfOsGCN0TfdU9k root@ubuntu (ED25519)
Setting up ssh-import-id (5.5-0ubuntu1) ...
Processing triggers for systemd (229-4ubuntu21.29) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for ufw (0.35-0ubuntu2) ...
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo ufw allow ssh
```

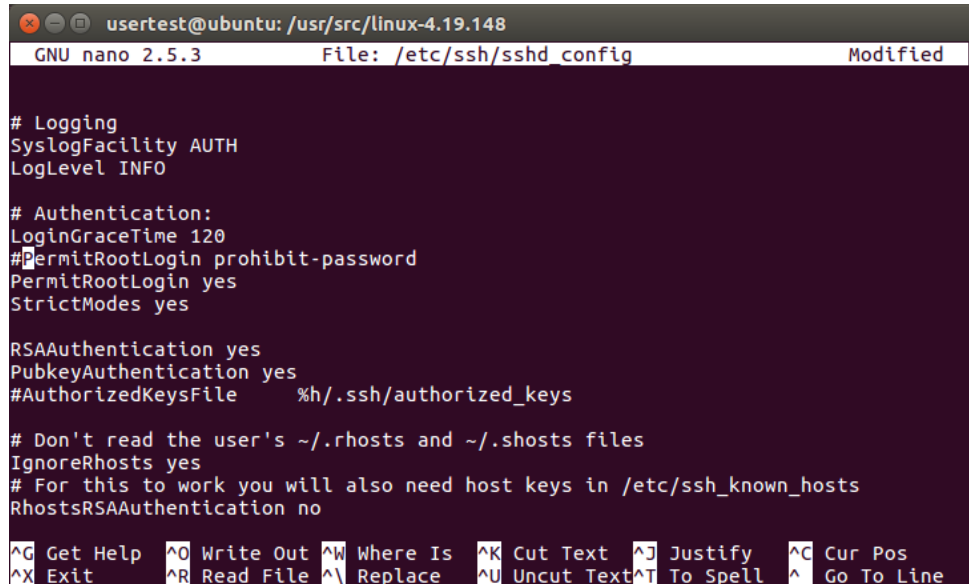3. Enable remote connection in the /etc/ssh/sshd_config file, by using the command
$sudo nano /etc/ssh/sshd_config
And modifying the line
PermitRootLogin prohibit-password
To
PermitRootLogin yes

```
usertest@ubuntu: /usr/src/linux-4.19.148
  GNU nano 2.5.3          File: /etc/ssh/sshd_config              Modified


# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 120
#PermitRootLogin prohibit-password
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile      %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no

^G Get Help   ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit       ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^  Go To Line
```
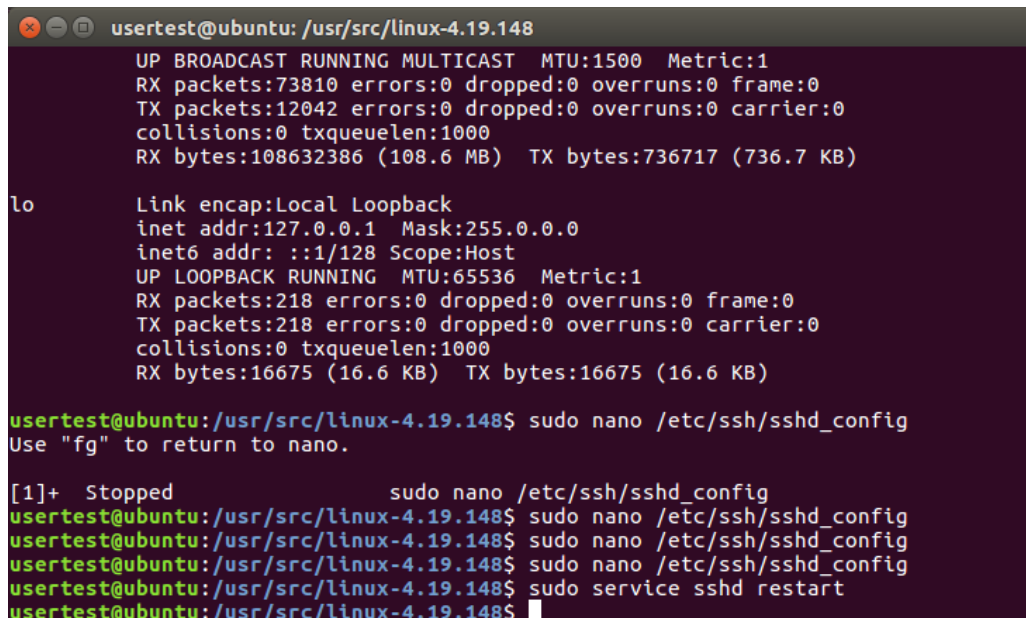
4. Restart the ssh service by using the command
$sudo service sshd restart

```
usertest@ubuntu: /usr/src/linux-4.19.148
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:73810 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12042 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:108632386 (108.6 MB)  TX bytes:736717 (736.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:218 errors:0 dropped:0 overruns:0 frame:0
          TX packets:218 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16675 (16.6 KB)  TX bytes:16675 (16.6 KB)

usertest@ubuntu:/usr/src/linux-4.19.148$ sudo nano /etc/ssh/sshd_config
Use "fg" to return to nano.

[1]+  Stopped                 sudo nano /etc/ssh/sshd_config
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo nano /etc/ssh/sshd_config
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo nano /etc/ssh/sshd_config
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo nano /etc/ssh/sshd_config
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo service sshd restart
usertest@ubuntu:/usr/src/linux-4.19.148$
```

5. In Target's Desktop, create a folder named **Test**, and a document called **TestDoc+studentID.txt.** Save any message you desire in the text file.
6. Get Target's IP address by using the command
   $ifconfig
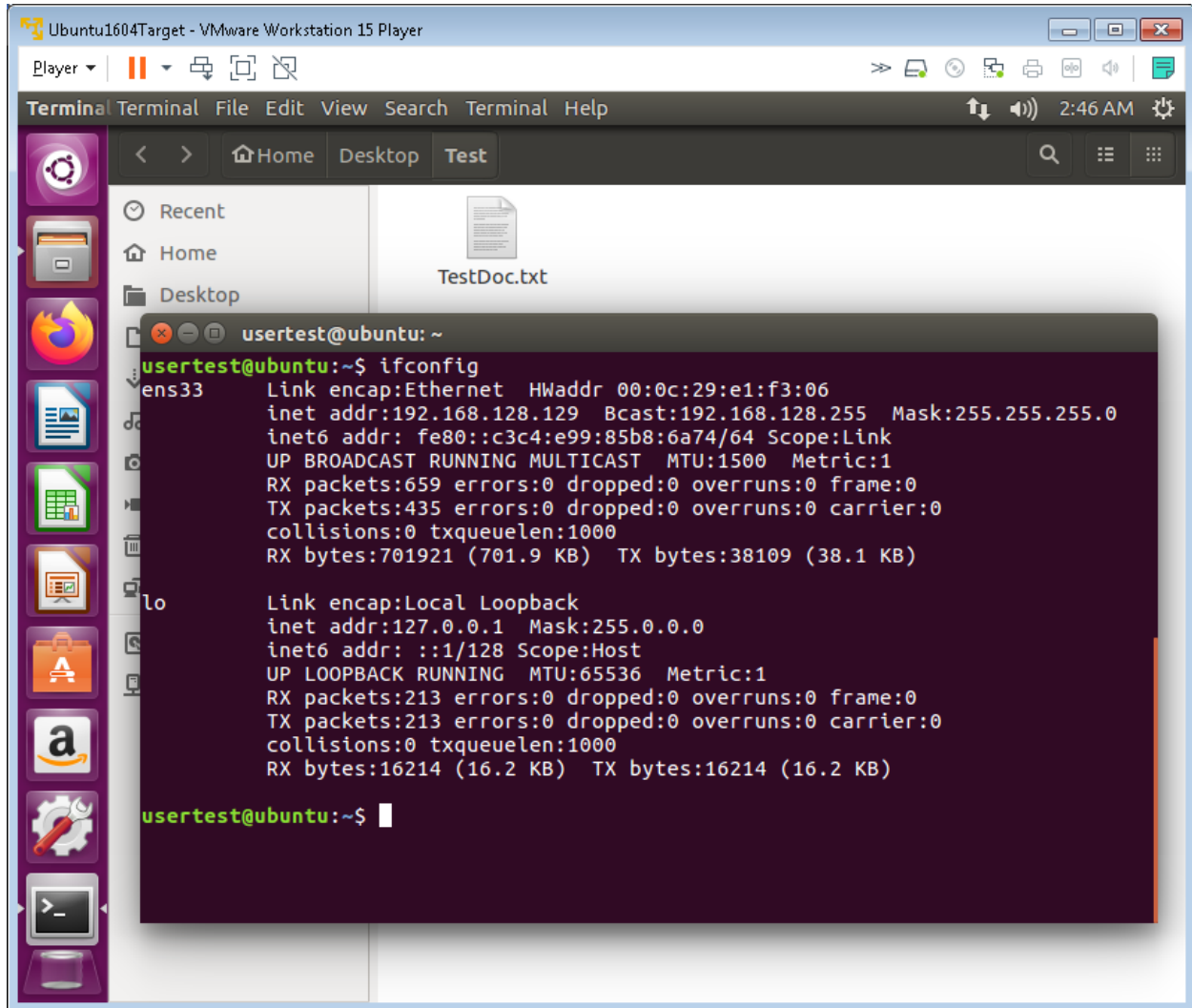
NOTE: From this point on, any time you see **brackets []** it indicates that you need to replace the content for its value.
Example: If your student ID is 12345, then **[usertest+StudentID] = usertest12345**

7. In the Host machine, run the command
   $sudo scp [usertest+StudentID]@[ip address of target]:/home/[usertest+StudentID]/Desktop/Test/[TestDoc+studentID.txt] .
   (Notice that there is a dot at the end of the command. This means to save the file in the current folder.)
   Example: *$sudo scp usertest12345@192.168.128.129:/home/usertest12345/Desktop/Test/TestDoc12345.txt .*
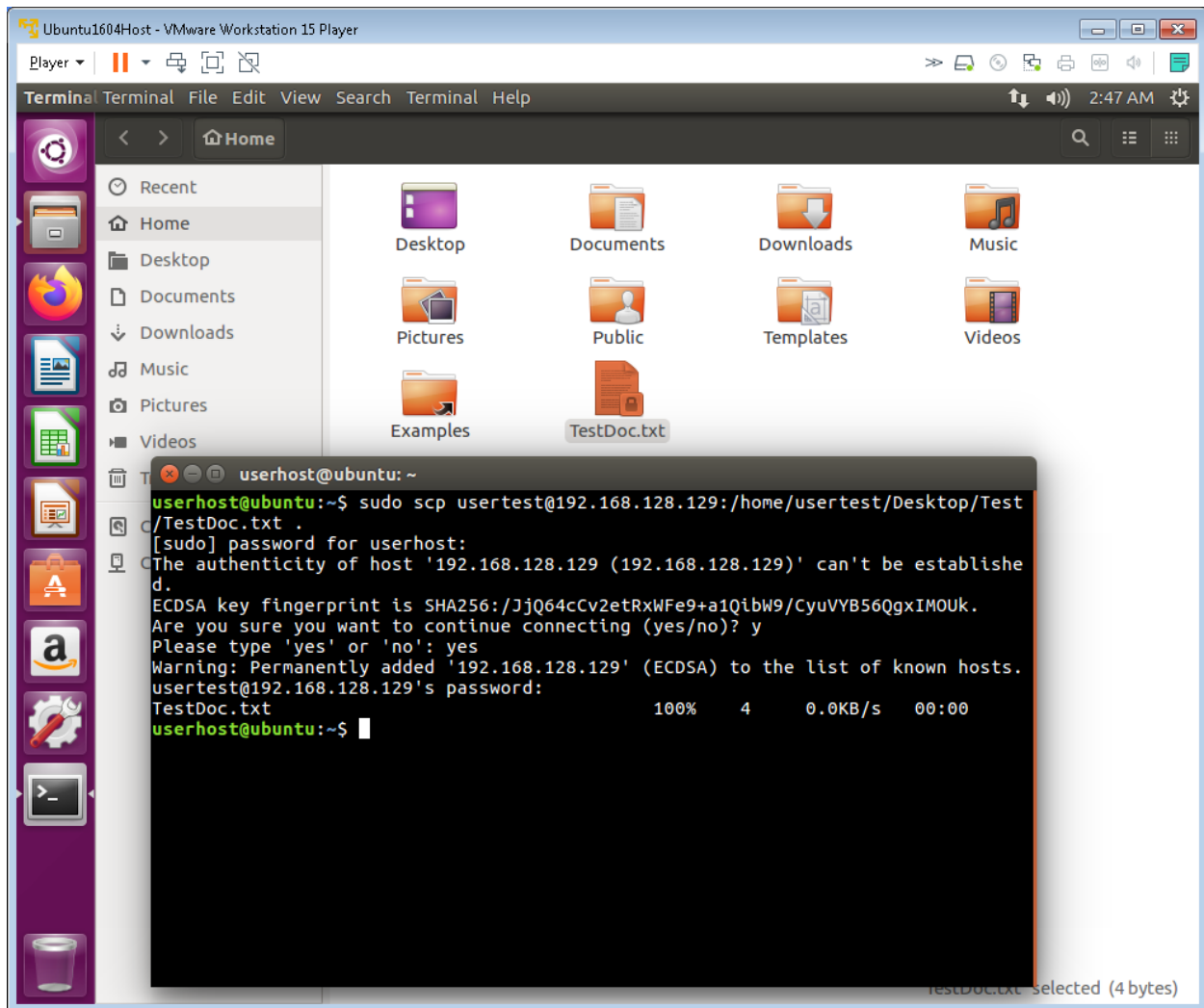
8. Accept the key fingerprint and save the target's password when prompted.

AT THIS POINT, YOU SHOULD HAVE A COPY OF THE TESTDOC FILE IN THE HOST'S HOME DIRECTORY. OPEN IT TO CHECK THAT THE CONTENTS ARE PRESERVED.

Additional links:

If you have questions about some of the steps mentioned above, please refer to the following links:

- https://animal0day.blogspot.com/2017/12/linux-kernel-debugging-with-vmware.html
- https://www.vmware.com/support/ws45/doc/devices_serial_ws.html
- https://linuxize.com/post/how-to-enable-ssh-on-ubuntu-18-04/
- https://monovm.com/post/35/how-to-search-in-nano

SECTION 2:

LINUX KERNEL DOWNLOAD

Section 2: Linux Kernel Download
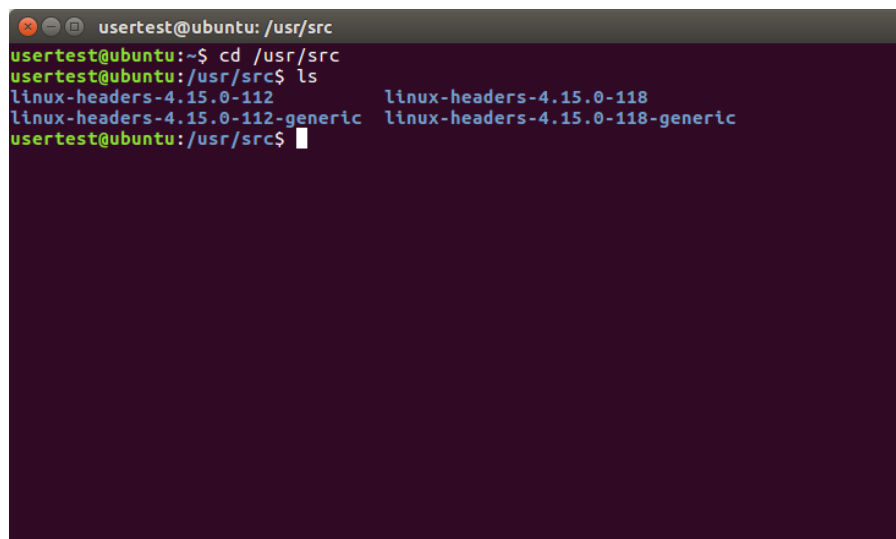
In this section, we proceed to <mark>download the code source</mark> for the Linux Kernel we will use in our virtual machines.

Important notes about this section:

1. You will need Internet connection to proceed.
2. Some of the steps, depending on your computer, will take several hours to finish.

Section 2.1: Linux Kernel Download

1. In the Target Machine, go to /usr/src and check that you have the generic Linux kernel files as shown below.
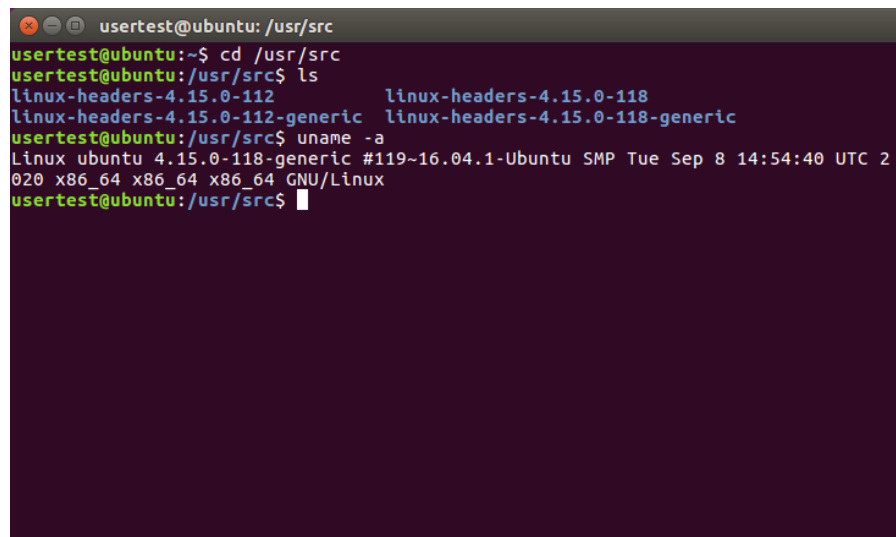
```
usertest@ubuntu: /usr/src
usertest@ubuntu:~$ cd /usr/src
usertest@ubuntu:/usr/src$ ls
linux-headers-4.15.0-112         linux-headers-4.15.0-118
linux-headers-4.15.0-112-generic  linux-headers-4.15.0-118-generic
usertest@ubuntu:/usr/src$
```

2. Get the current kernel version by running the command
   $uname -a
   Example: Linux ubuntu 4.15.0-118-generic

```
usertest@ubuntu: /usr/src
usertest@ubuntu:~$ cd /usr/src
usertest@ubuntu:/usr/src$ ls
linux-headers-4.15.0-112         linux-headers-4.15.0-118
linux-headers-4.15.0-112-generic  linux-headers-4.15.0-118-generic
usertest@ubuntu:/usr/src$ uname -a
Linux ubuntu 4.15.0-118-generic #119~16.04.1-Ubuntu SMP Tue Sep 8 14:54:40 UTC 2
020 x86_64 x86_64 x86_64 GNU/Linux
usertest@ubuntu:/usr/src$
```

3. Go to https://www.kernel.org/ and check which is the closes kernel version to the one displayed. (For this example it is 4.19.148).
4. Right click on the tarball link and copy the link address.

5. Download the tarball in terminal, by using the command
   $sudo wget [copied link address]
   (you can right click + paste the link in terminal)

6. Decompress the compressed file by using the command
   $sudo unxz -v [compressed tar name *.tar.xz]
   <u>Example:</u> $sudo unxz -v linux-4.19.148.tar.xz

```
😣 ⊖ ⊡   usertest@ubuntu: /usr/src
Resolving cdn.kernel.org (cdn.kernel.org)... 151.101.229.176, 2a04:4e42:36::432
Connecting to cdn.kernel.org (cdn.kernel.org)|151.101.229.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 103486520 (99M) [application/x-xz]
linux-4.19.148.tar.xz: Permission denied

Cannot write to 'linux-4.19.148.tar.xz' (Success).
usertest@ubuntu:/usr/src$ sudo wget https://cdn.kernel.org/pub/linux/kernel/v4.x
/linux-4.19.148.tar.xz
[sudo] password for usertest:
--2020-09-28 02:47:49--  https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19
.148.tar.xz
Resolving cdn.kernel.org (cdn.kernel.org)... 151.101.229.176, 2a04:4e42:36::432
Connecting to cdn.kernel.org (cdn.kernel.org)|151.101.229.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 103486520 (99M) [application/x-xz]
Saving to: 'linux-4.19.148.tar.xz'

linux-4.19.148.tar. 100%[===================>]  98.69M  8.72MB/s    in 12s

2020-09-28 02:48:02 (8.20 MB/s) - 'linux-4.19.148.tar.xz' saved [103486520/10348
6520]

usertest@ubuntu:/usr/src$ sudo unxz -v linux-4.19.148.tar.xz ▉
```

7. Untar the tar file by using the command
   $sudo tar xvf [generated tar file *.tar]
   <u>Example:</u> $sudo tar xvf linux-4.19.148.tar

```
😣 ⊖ ⊡   usertest@ubuntu: /usr/src
Length: 103486520 (99M) [application/x-xz]
linux-4.19.148.tar.xz: Permission denied

Cannot write to 'linux-4.19.148.tar.xz' (Success).
usertest@ubuntu:/usr/src$ sudo wget https://cdn.kernel.org/pub/linux/kernel/v4.x
/linux-4.19.148.tar.xz
[sudo] password for usertest:
--2020-09-28 02:47:49--  https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19
.148.tar.xz
Resolving cdn.kernel.org (cdn.kernel.org)... 151.101.229.176, 2a04:4e42:36::432
Connecting to cdn.kernel.org (cdn.kernel.org)|151.101.229.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 103486520 (99M) [application/x-xz]
Saving to: 'linux-4.19.148.tar.xz'

linux-4.19.148.tar. 100%[===================>]  98.69M  8.72MB/s    in 12s

2020-09-28 02:48:02 (8.20 MB/s) - 'linux-4.19.148.tar.xz' saved [103486520/10348
6520]

usertest@ubuntu:/usr/src$ sudo unxz -v linux-4.19.148.tar.xz
linux-4.19.148.tar.xz (1/1)
  100 %        98.7 MiB / 802.6 MiB = 0.123    24 MiB/s      0:32
usertest@ubuntu:/usr/src$ sudo tar xvf linux-4.19.148.tar ▉
```

AT THIS POINT, A NEW FOLDER (linux-4.19.148) SHOULD EXIST UNDER /usr/src/.

Section 2.2: Pre-build Additional Configurations

In this section, we will perform two additional configurations prior to proceed with the kernel build.

1. Get into the new kernel's folder (/usr/src/linux-4.19.148)
2. Copy the current .config file into this folder by using the command
   $sudo cp -v /boot/config-$(uname -r) .config
   Note: If you don't run sudo, you will get a **Permission denied** exception.

   If this step is successful, you should get an output of the form
   '/boot/config-4.15.0-118-generic' -> '.config'

```
usertest@ubuntu: /usr/src/linux-4.19.148
linux-4.19.148/virt/lib/
linux-4.19.148/virt/lib/Kconfig
linux-4.19.148/virt/lib/Makefile
linux-4.19.148/virt/lib/irqbypass.c
usertest@ubuntu:/usr/src$ cd linux-4.19.148/
arch/          drivers/       ipc/           net/          tools/
block/         firmware/      kernel/        samples/      usr/
certs/         fs/            lib/           scripts/      virt/
crypto/        include/       LICENSES/      security/
Documentation/ init/          mm/            sound/
usertest@ubuntu:/usr/src$ cd linux-4.19.148/
arch/          drivers/       ipc/           net/          tools/
block/         firmware/      kernel/        samples/      usr/
certs/         fs/            lib/           scripts/      virt/
crypto/        include/       LICENSES/      security/
Documentation/ init/          mm/            sound/
usertest@ubuntu:/usr/src$ cd linux-4.19.148
usertest@ubuntu:/usr/src/linux-4.19.148$ cp -v /boot/config-$(uname -r) .config
'/boot/config-4.15.0-118-generic' -> '.config'
cp: cannot create regular file '.config': Permission denied
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo cp -v /boot/config-$(uname -r) .co
nfig
'/boot/config-4.15.0-118-generic' -> '.config'
usertest@ubuntu:/usr/src/linux-4.19.148$
```

3. Install the needed build software by running the command
   $ sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev
   (This step will take from 5 to 15 minutes to complete)

```
usertest@ubuntu: /usr/src/linux-4.19.148
linux-4.19.148/virt/lib/Kconfig
linux-4.19.148/virt/lib/Makefile
linux-4.19.148/virt/lib/irqbypass.c
usertest@ubuntu:/usr/src$ cd linux-4.19.148/
arch/          drivers/       ipc/           net/          tools/
block/         firmware/      kernel/        samples/      usr/
certs/         fs/            lib/           scripts/      virt/
crypto/        include/       LICENSES/      security/
Documentation/ init/          mm/            sound/
usertest@ubuntu:/usr/src$ cd linux-4.19.148/
arch/          drivers/       ipc/           net/          tools/
block/         firmware/      kernel/        samples/      usr/
certs/         fs/            lib/           scripts/      virt/
crypto/        include/       LICENSES/      security/
Documentation/ init/          mm/            sound/
usertest@ubuntu:/usr/src$ cd linux-4.19.148
usertest@ubuntu:/usr/src/linux-4.19.148$ cp -v /boot/config-$(uname -r) .config
'/boot/config-4.15.0-118-generic' -> '.config'
cp: cannot create regular file '.config': Permission denied
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo cp -v /boot/config-$(uname -r) .co
nfig
'/boot/config-4.15.0-118-generic' -> '.config'
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo apt-get install build-essential li
bncurses-dev bison flex libssl-dev libelf-dev
```

AT THIS POINT, YOU SHOULD HAVE A .config FILE UNDER THE linux-4.19.148 FOLDER.
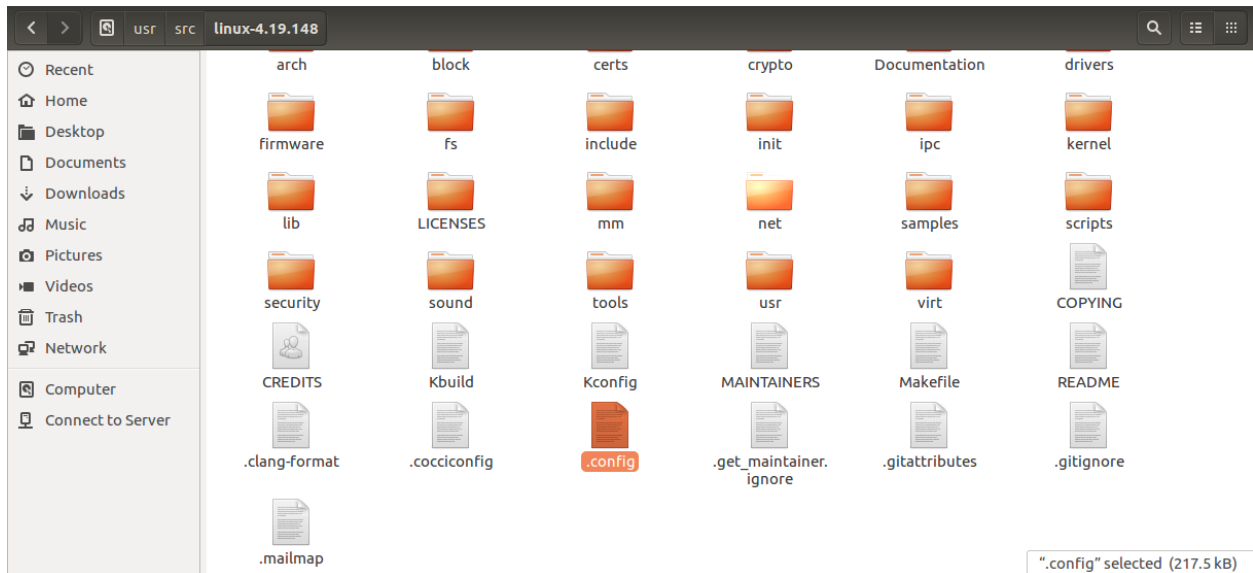
IN CASE YOU CANNOT SEE IT, ENABLE THE OPTION TO DISPLAY HIDDEN FILES.
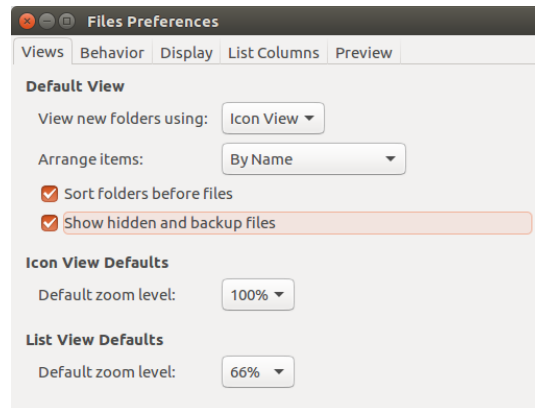
WHEN REACHING THIS POINT, YOU MUST REPEAT ALL THE STEPS ON SECTION 2.1 AND 2.2 ON THE HOST MACHINE AS WELL.

[Screenshot # 3: Create a screenshot of the target machine showing the generated folder with the files as shown above, and add it to your video and report]

SECTION 3:

LINUX KERNEL PATCH AND BUILD

Section 3: Linux Kernel Patch and Build

Section 3.1: Linux Kernel Patching (1)

In this section we proceed to perform a patch on the kernel prior to building it.

AT THIS MOMENT, YOU CAN TURN OFF THE HOST MACHINE.

1. In the Target machine, run the command
   $sudo make menuconfig
   (be sure that the terminal is still located inside /usr/src/linux-4.19.148)

A new window will appear. Check that the options **64-bit kernel** and **Virtualization** are selected. Then save and exit.
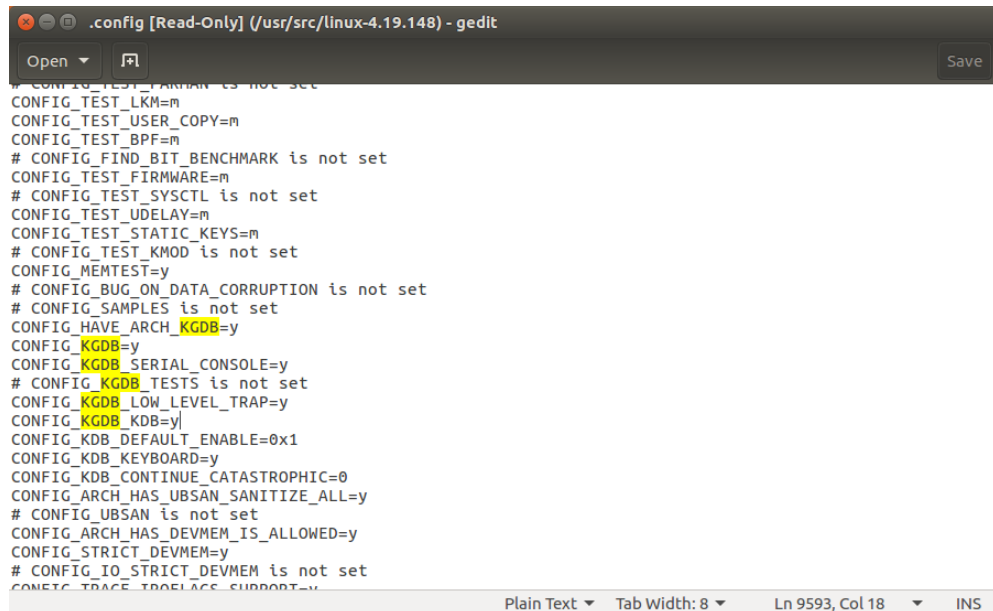
Section 3.2: Linux Kernel Patching (2)

In this section we proceed to enable the necessary debugging tools that we will use in later sections.

1. In the /usr/src/linux-4.19.148 folder, open the .config file.
2. Search for options containing the keyword **KGDB**

Check that the following commands are included and set to value = y

```
CONFIG_FRAME_POINTER=y
CONFIG_KGDB=y
CONFIG_KGDB_SERIAL_CONSOLE=y
CONFIG_KGDB_KDB=y
CONFIG_KDB_KEYBOARD=y
```

In case these commands are missing or not set up, use the command
$sudo nano /usr/src/linux-4.19.148/.config
And add them.

Additional links:
Some useful links for this section are shown below:
- https://www.kernel.org/doc/html/v4.14/dev-tools/kgdb.html
- https://monovm.com/post/35/how-to-search-in-nano
- https://superuser.com/questions/581671/find-next-command-in-nano

[Screenshot # 4 and #5: Include in your report and video at least two screenshots on how you perform the patching.]

Section 3.3: Linux Kernel Build

In this section we proceed to build the Linux kernel.
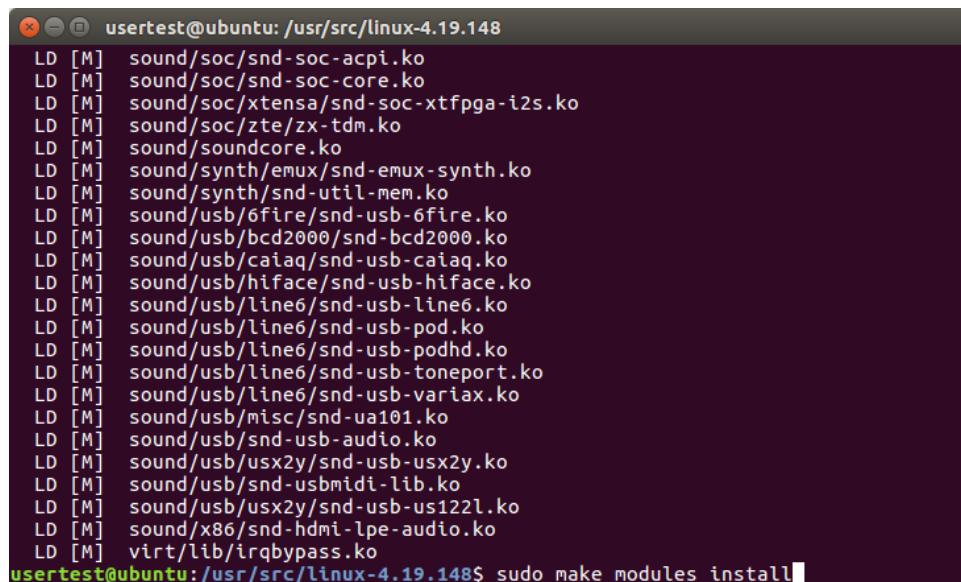
Note: Depending on your computer's resources, the steps in this section **will take a couple of hours to finish.**

1. Under the /usr/src/linux-4.19.148 folder, run the command
   $sudo make -j $(nproc)

```
usertest@ubuntu: /usr/src/linux-4.19.148
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo make -j $(nproc)
```

2. When this process is done, run the command
   $sudo make modules_install

```
usertest@ubuntu: /usr/src/linux-4.19.148
  LD [M]  sound/soc/snd-soc-acpi.ko
  LD [M]  sound/soc/snd-soc-core.ko
  LD [M]  sound/soc/xtensa/snd-soc-xtfpga-i2s.ko
  LD [M]  sound/soc/zte/zx-tdm.ko
  LD [M]  sound/soundcore.ko
  LD [M]  sound/synth/emux/snd-emux-synth.ko
  LD [M]  sound/synth/snd-util-mem.ko
  LD [M]  sound/usb/6fire/snd-usb-6fire.ko
  LD [M]  sound/usb/bcd2000/snd-bcd2000.ko
  LD [M]  sound/usb/caiaq/snd-usb-caiaq.ko
  LD [M]  sound/usb/hiface/snd-usb-hiface.ko
  LD [M]  sound/usb/line6/snd-usb-line6.ko
  LD [M]  sound/usb/line6/snd-usb-pod.ko
  LD [M]  sound/usb/line6/snd-usb-podhd.ko
  LD [M]  sound/usb/line6/snd-usb-toneport.ko
  LD [M]  sound/usb/line6/snd-usb-variax.ko
  LD [M]  sound/usb/misc/snd-ua101.ko
  LD [M]  sound/usb/snd-usb-audio.ko
  LD [M]  sound/usb/usx2y/snd-usb-usx2y.ko
  LD [M]  sound/usb/snd-usbmidi-lib.ko
  LD [M]  sound/usb/usx2y/snd-usb-us122l.ko
  LD [M]  sound/x86/snd-hdmi-lpe-audio.ko
  LD [M]  virt/lib/irqbypass.ko
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo make modules_install
```

```
😣➖⬜  usertest@ubuntu: /usr/src/linux-4.19.148
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-dw2102.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-gp8psk.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-m920x.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-nova-t-usb2.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-opera.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-pctv452e.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-technisat-usb2.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-ttusb2.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-umt-010.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-vp702x.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb-vp7045.ko
 INSTALL drivers/media/usb/dvb-usb/dvb-usb.ko
 INSTALL drivers/media/usb/em28xx/em28xx-alsa.ko
 INSTALL drivers/media/usb/em28xx/em28xx-dvb.ko
 INSTALL drivers/media/usb/em28xx/em28xx-rc.ko
 INSTALL drivers/media/usb/em28xx/em28xx-v4l.ko
 INSTALL drivers/media/usb/em28xx/em28xx.ko
 INSTALL drivers/media/usb/go7007/go7007-loader.ko
 INSTALL drivers/media/usb/go7007/go7007-usb.ko
 INSTALL drivers/media/usb/go7007/go7007.ko
 INSTALL drivers/media/usb/go7007/s2250.ko
 INSTALL drivers/media/usb/gspca/gl860/gspca_gl860.ko
 INSTALL drivers/media/usb/gspca/gspca_benq.ko
```

3. After this process is done, run the command
   $sudo make install

```
😣➖⬜  usertest@ubuntu: /usr/src/linux-4.19.148
 INSTALL sound/usb/caiaq/snd-usb-caiaq.ko
 INSTALL sound/usb/hiface/snd-usb-hiface.ko
 INSTALL sound/usb/line6/snd-usb-line6.ko
 INSTALL sound/usb/line6/snd-usb-pod.ko
 INSTALL sound/usb/line6/snd-usb-podhd.ko
 INSTALL sound/usb/line6/snd-usb-toneport.ko
 INSTALL sound/usb/line6/snd-usb-variax.ko
 INSTALL sound/usb/misc/snd-ua101.ko
 INSTALL sound/usb/snd-usb-audio.ko
 INSTALL sound/usb/snd-usbmidi-lib.ko
 INSTALL sound/usb/usx2y/snd-usb-us122l.ko
 INSTALL sound/usb/usx2y/snd-usb-usx2y.ko
 INSTALL sound/x86/snd-hdmi-lpe-audio.ko
 INSTALL virt/lib/irqbypass.ko
 DEPMOD  4.19.148
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo make install
sh ./arch/x86/boot/install.sh 4.19.148 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.19.148 /boot/vmli
nuz-4.19.148
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.19.148 /boot/vmlin
uz-4.19.148
update-initramfs: Generating /boot/initrd.img-4.19.148
```

[Screenshot # 6, #7 and #8: Include in your report and video at least three screenshots of the execution of these three functions, displaying the user with your student ID.]

4. When the previous step is done, four new files should now exist under your **/boot** folder:
   a. initrd.img-4.19.148
   b. vmlinuz-4.19.148
   c. system.map.4.19.148
   d. config-4.19.148
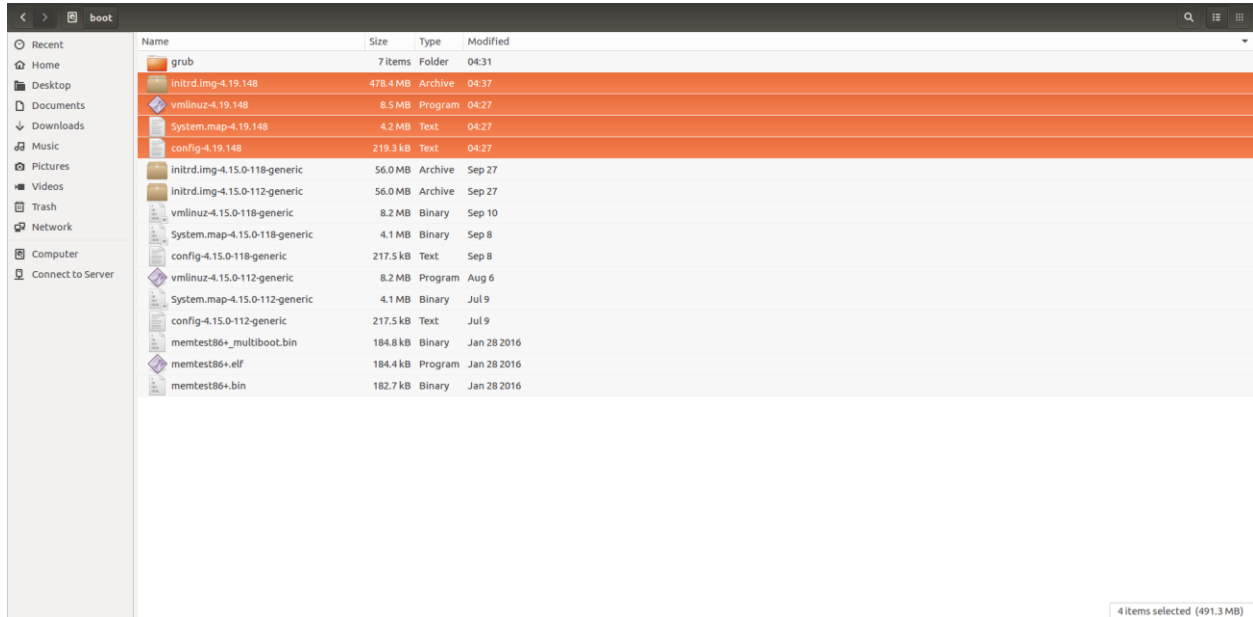
IMPORTANT NOTE:

STARTING FROM THE NEXT SECTION, YOU CANNOT TURN OFF THE TARGET VIRTUAL MACHINE UNTIL IT IS COMPLETELY CONFIGURED.

IF YOU DO SO YOU MAY LOCK YOUR KERNEL, MAKING THE VIRTUAL MACHINE USELESS, AND YOU WILL HAVE TO REPEAT ALL THE PREVIOUS STEPS SINCE SECTION 1.

Section 3.4: Grub update (1)

WARNING: From this point on, you should not turn off your Target virtual machine. Doing so may lock the kernel, rendering the machine unusable.

1. In the Target machine, run the commands
   $sudo update-initramfs -c -k 4.19.148

```
●●● usertest@ubuntu: /usr/src/linux-4.19.148
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.19.148 /boot/vmlin
uz-4.19.148
update-initramfs: Generating /boot/initrd.img-4.19.148
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.19.148 /boot/vmlinuz-4.19
.148
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.19.148 /boot/v
mlinuz-4.19.148
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.19.148 /boot/vmlin
uz-4.19.148
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.19.148 /boot/vmlinu
z-4.19.148
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is se
t is no longer supported.
Found linux image: /boot/vmlinuz-4.19.148
Found initrd image: /boot/initrd.img-4.19.148
Found linux image: /boot/vmlinuz-4.15.0-118-generic
Found initrd image: /boot/initrd.img-4.15.0-118-generic
Found linux image: /boot/vmlinuz-4.15.0-112-generic
Found initrd image: /boot/initrd.img-4.15.0-112-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo update-initramfs -c -k 4.19.148
```

2. Run the command
   $sudo update grub

```
●●● usertest@ubuntu: /usr/src/linux-4.19.148
Found linux image: /boot/vmlinuz-4.15.0-118-generic
Found initrd image: /boot/initrd.img-4.15.0-118-generic
Found linux image: /boot/vmlinuz-4.15.0-112-generic
Found initrd image: /boot/initrd.img-4.15.0-112-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo update-initramfs -c -k 4.19.148
update-initramfs: Generating /boot/initrd.img-4.19.148
usertest@ubuntu:/usr/src/linux-4.19.148$ sudo update-grub
[sudo] password for usertest:
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is se
t is no longer supported.
Found linux image: /boot/vmlinuz-4.19.148
Found initrd image: /boot/initrd.img-4.19.148
Found linux image: /boot/vmlinuz-4.15.0-118-generic
Found initrd image: /boot/initrd.img-4.15.0-118-generic
Found linux image: /boot/vmlinuz-4.15.0-112-generic
Found initrd image: /boot/initrd.img-4.15.0-112-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
usertest@ubuntu:/usr/src/linux-4.19.148$
```

Section 3.4: Grub update (2)

In this section, we allow the Serial Port communication with KGDB to perform the kernel debug.
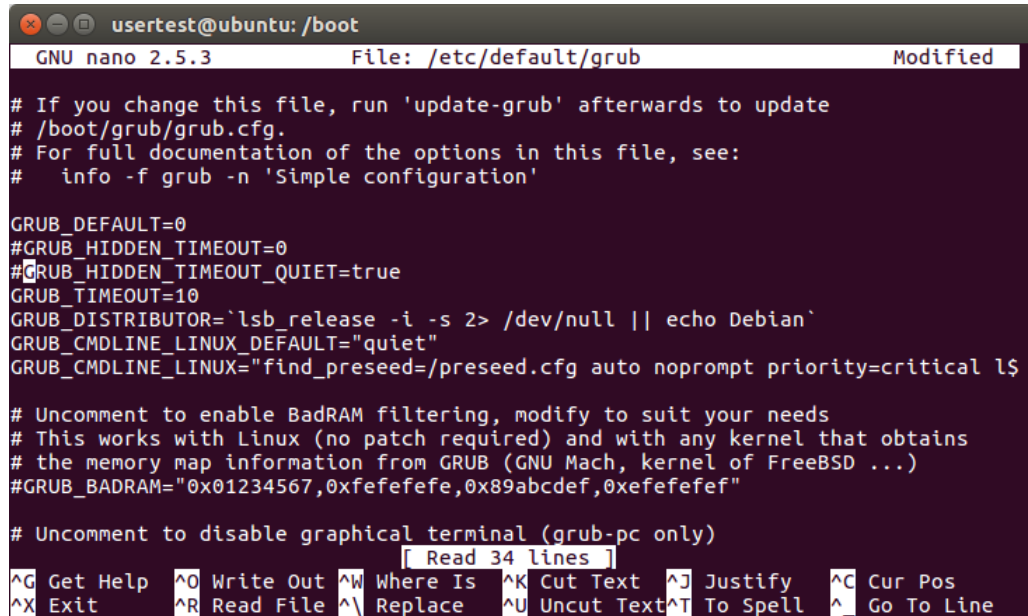
1. Modify the /etc/default/grub file with the command
   $sudo nano /etc/default/grub
   Comment out the following lines
   #GRUB_HIDDEN_TIMEOUT=0
   #GRUB_HIDDEN_TIMEOUT_QUIET=true

```
usertest@ubuntu: /boot
  GNU nano 2.5.3           File: /etc/default/grub              Modified

# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
#GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="find_preseed=/preseed.cfg auto noprompt priority=critical l$

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
                           [ Read 34 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```
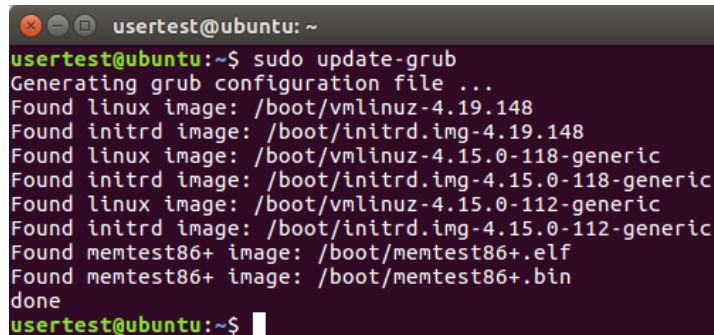
2. Update grub again by running the command
   $sudo update-grub
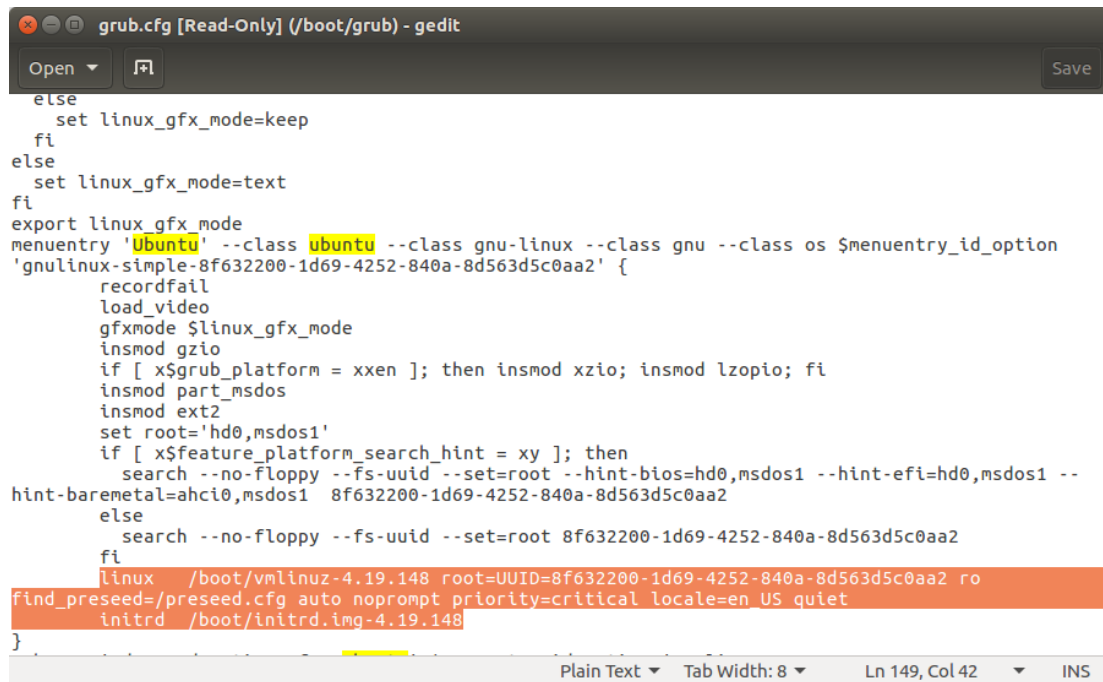
```
usertest@ubuntu: ~
usertest@ubuntu:~$ sudo update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.19.148
Found initrd image: /boot/initrd.img-4.19.148
Found linux image: /boot/vmlinuz-4.15.0-118-generic
Found initrd image: /boot/initrd.img-4.15.0-118-generic
Found linux image: /boot/vmlinuz-4.15.0-112-generic
Found initrd image: /boot/initrd.img-4.15.0-112-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
usertest@ubuntu:~$
```

3. In the Target machine, go to /boot/grub/grub.cfg and find the first entry with the keyword 'Ubuntu'.
   In this entry, find the line starting with the keyword *linux* and check its line number.

4. Using nano, modify this file and add in the line with the keyword *linux* the following commands at the end of it:
   kgdbwait kgdboc=ttyS1,115200
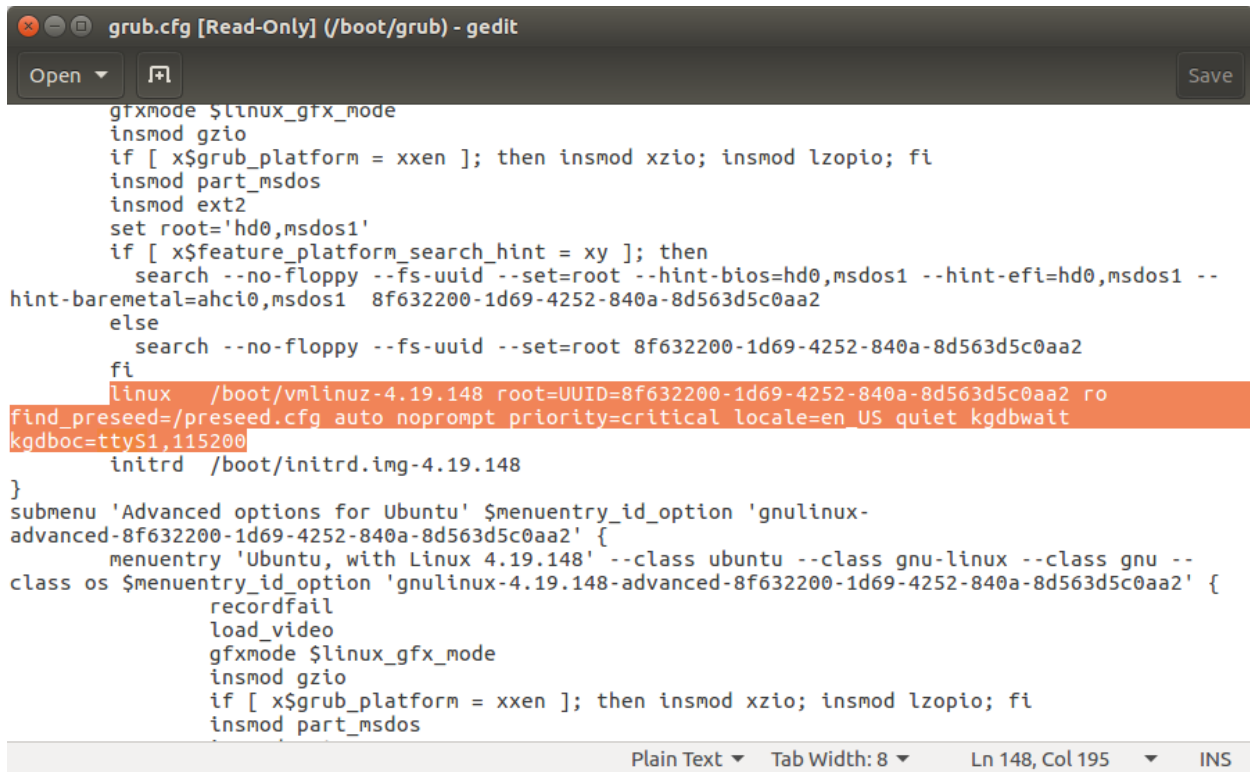
5. After saving this file, its contents should look like the ones below


```
gfxmode $linux_gfx_mode
insmod gzio
if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
insmod part_msdos
insmod ext2
set root='hd0,msdos1'
if [ x$feature_platform_search_hint = xy ]; then
   search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1 --
hint-baremetal=ahci0,msdos1  8f632200-1d69-4252-840a-8d563d5c0aa2
   else
     search --no-floppy --fs-uuid --set=root 8f632200-1d69-4252-840a-8d563d5c0aa2
   fi
   linux    /boot/vmlinuz-4.19.148 root=UUID=8f632200-1d69-4252-840a-8d563d5c0aa2 ro
find_preseed=/preseed.cfg auto noprompt priority=critical locale=en_US quiet kgdbwait
kgdboc=ttyS1,115200
   initrd   /boot/initrd.img-4.19.148
}
submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-
advanced-8f632200-1d69-4252-840a-8d563d5c0aa2' {
     menuentry 'Ubuntu, with Linux 4.19.148' --class ubuntu --class gnu-linux --class gnu --
class os $menuentry_id_option 'gnulinux-4.19.148-advanced-8f632200-1d69-4252-840a-8d563d5c0aa2' {
          recordfail
          load_video
          gfxmode $linux_gfx_mode
          insmod gzio
          if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
          insmod part_msdos
```

6. Turn on the Host machine (with the Serial Port still disconnected).
7. Create a folder called **kgdb-image** inside the /boot/ folder by using the command
   **$sudo mkdir kgdb-image**

8. In the Target machine, verify that there is a **vmlinux** file under /usr/src/linux-4.19.148.
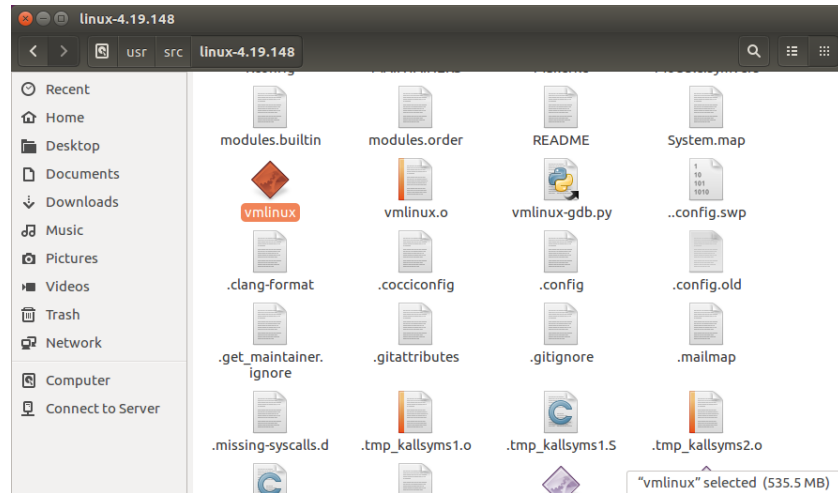
9. **Get inside the kgdb-image folder**, and run the command
   $sudo scp [usertest+studentID]@[target machine IP address]:/usr/src/linux-4.19.148/vmlinux .
   Example:
   $sudo scp usertest12345@192.168.126.136:/usr/src/linux-4.19.148/vmlinux .
   After running this command, the vmlinux file should appear inside the kgdb-image folder.
   (If you have any questions, please refer to section 1.3)
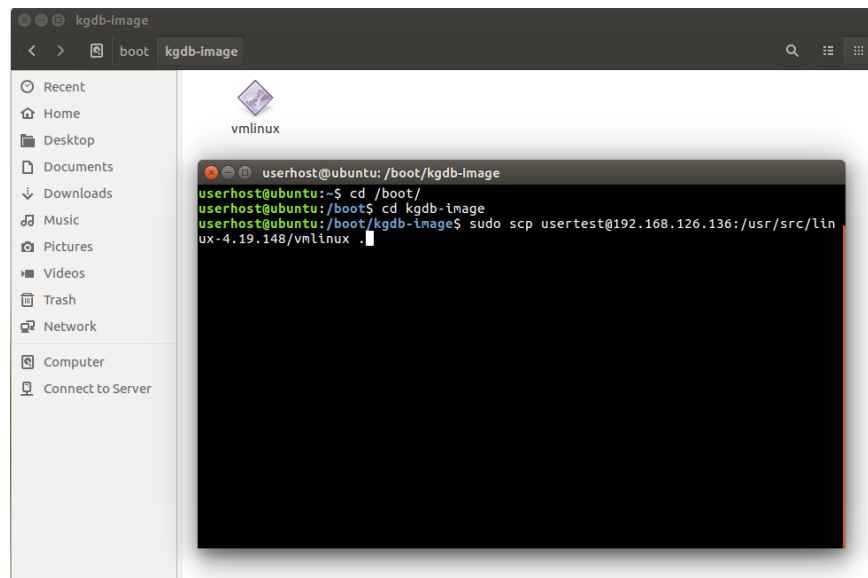
[Screenshot # 9 and #10: Include in your report and video at least two screenshots on how you perform the grub update.]

AT THIS POINT YOU IT IS SAFE TO TURN OFF YOUR TARGET VIRTUAL MACHINE.

Additional links:

- https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html
- https://www.youtube.com/watch?v=67cxIXLCfUk

SECTION 4:

LINUX KERNEL DEBUG [KGDB]

Section 4: Linux Kernel Debug [KGDB]

In this section we cover the basics of kernel debugging.

AT THIS POINT, TURN OFF BOTH VIRTUAL MACHINES AND CONNECT BACK THE SERIAL PORT IN BOTH OF THEM.

FROM THIS POINT ON, YOU MUST TURN ON THE TARGET MACHINE FIRST AND THEN THE HOST MACHINE.

Before proceeding, we present two sections that could help you to solve some problems you may face when turning the Target machine back on again.

(Extra) Section 4.0.1: Possible kernel panic

Depending on your computer, it is possible that the target virtual machine will show the following error when turning it back on again.

[end Kernel panic – not syncing: System is deadlocked on memory]

```
[    3.333470]  ? md_run_setup+0xbd/0xbd
[    3.333510]  ksys_write+0x52/0xc0
[    3.333542]  xwrite+0x29/0x5a
[    3.333578]  do_copy+0x9b/0xc8
[    3.333603]  write_buffer+0x24/0x34
[    3.333631]  flush_buffer+0x2b/0x84
[    3.333658]  __gunzip+0x275/0x31c
[    3.333685]  ? bunzip2+0x39c/0x39c
[    3.333711]  ? __gunzip+0x31c/0x31c
[    3.333738]  gunzip+0xe/0x11
[    3.333762]  ? md_run_setup+0xbd/0xbd
[    3.333791]  unpack_to_rootfs+0x163/0x2b5
[    3.333821]  ? md_run_setup+0xbd/0xbd
[    3.333849]  ? clean_rootfs+0x182/0x182
[    3.333890]  populate_rootfs+0x5d/0x116
[    3.333919]  ? clean_rootfs+0x182/0x182
[    3.333949]  do_one_initcall+0x4a/0x1c6
[    3.333978]  kernel_init_freeable+0x193/0x233
[    3.334010]  ? rest_init+0xb0/0xb0
[    3.334036]  kernel_init+0xa/0x110
[    3.334063]  ret_from_fork+0x35/0x40
[    3.334160] ---[ end Kernel panic - not syncing: System is deadlocked on memo
ry
[    3.334160]  ]---
```

If you face this problem, turn the machine off, and change the RAM from 2Gb to 4Gb. Turn it back on again and the problem should be solved.

Additional link:

- https://unix.stackexchange.com/questions/492667/compiled-kernel-4-19-will-not-boot-kernel-panic-not-syncing-system-is-deadlo

(Extra) Section 4.0.2: Bypassing Debug mode

This section covers a bypass on the debug mode of the Target machine in case you need to re-do any step from the previous sections.

Because of the steps we performed on Section 3, if you turn on the Target machine, you should reach one the following two windows:

- Only purple screen

- A black screen with the message
  KGDB: Waiting for connection from remote gdb…

But before reaching these two windows, if you performed step 1 and 2 from section 3.4, you should see the grub.

If you need to access the machine, in the previous grub menu, with the option **Ubuntu** selected, press the key **E** in your keyboard. It should bring you to a window like the following one.

Here you can scroll down and look for the line with the keyword *linux* which has the commands **kgdbwait kdgboc=ttyS1,115200** that we added on section 3.4. You can erase these two commands from that line as shown below.

```
            GNU GRUB  version 2.02~beta2-36ubuntu3.27

        insmod ext2
        set root='hd0,msdos1'
        if [ x$feature_platform_search_hint = xy ]; then
            search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1\
 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  8f632200-1d69-4252\
-840a-8d563d5c0aa2
        else
            search --no-floppy --fs-uuid --set=root 8f632200-1d69-4252-840\
a-8d563d5c0aa2
        fi
        linux        /boot/vmlinuz-4.19.148 root=UUID=8f632200-1d69-4252\
-840a-8d563d5c0aa2 ro find_preseed=/preseed.cfg auto noprompt priority=c\
ritical locale=en_US quiet_
        initrd       /boot/initrd.img-4.19.148


    Minimum Emacs-like screen editing is supported. TAB lists
    completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
    command-line or ESC to discard edits and return to the GRUB
    menu.
```

Then by pressing **Ctrl-x**, the virtual machine should restart normally and you should be able to login.

Note:

- This grub modification is not persistent. If you restart the machine, you would have to modify it again.
- If you want this modification to be persistent, you must repeat steps 3 onwards from section 3.4 erasing the commands shown above.

Section 4.1: Debugging

This section covers the basics on how to make the Host machine to debug the Target machine.

1. Turn off both machines.
2. Connect the Serial port in both of them.
3. Turn on the Target machine. After one or two minutes, you should reach either a purple screen or a black screen with the words **KGDB: Waiting for connection from remote gdb…**
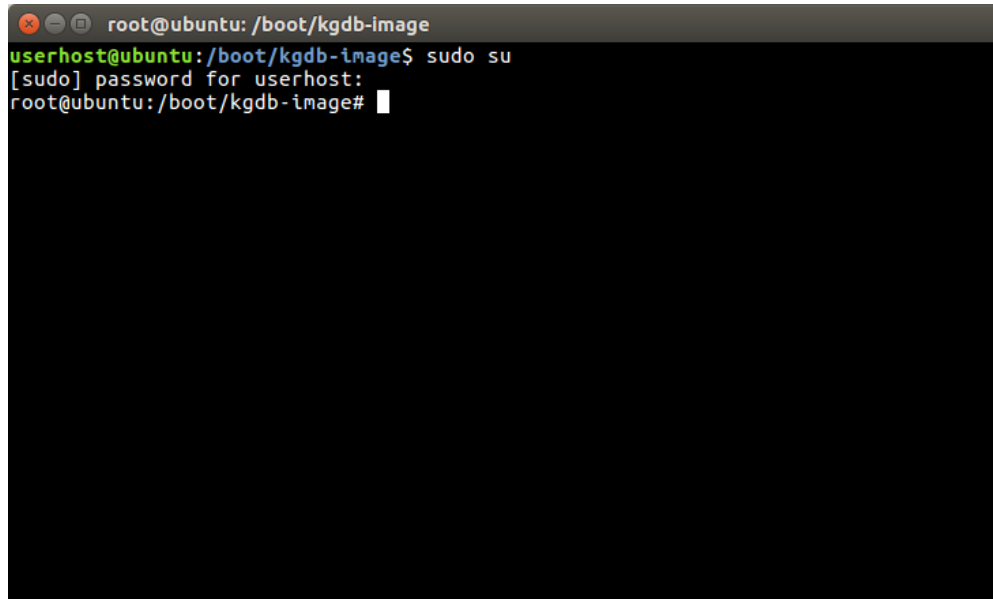

4. Leave the Target machine on. Turn on the Host machine.
5. In the Host machine, in terminal, go to the /boot/kgdb-image folder. Verify that the **vmlinux** file we copied in step 9 of section 3.4 is inside this folder.
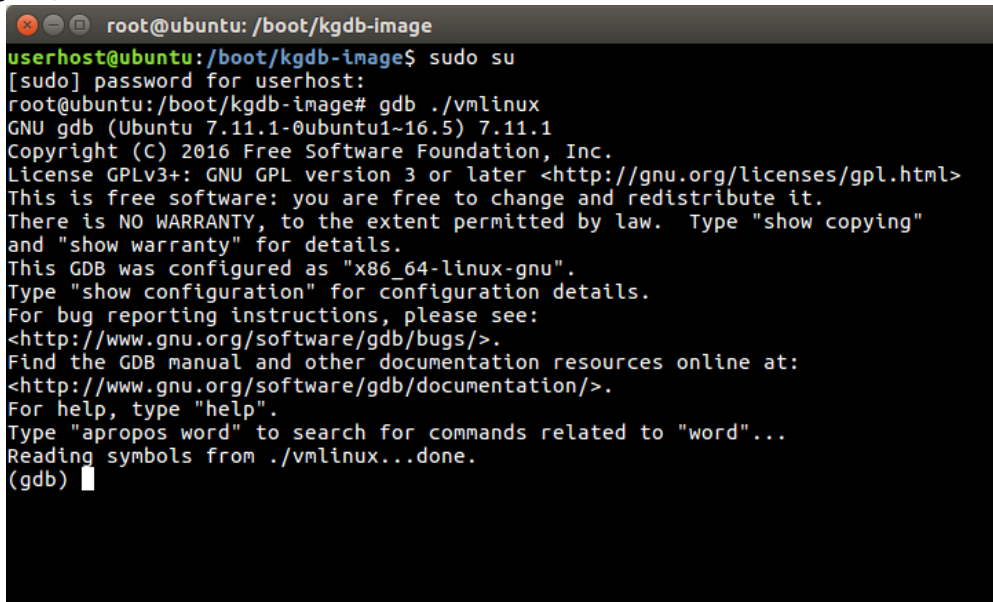
6. Login as **su** with the command
   $sudo su (and then typing the root password).
   The terminal should change from $ to # and all the letters will be white now.

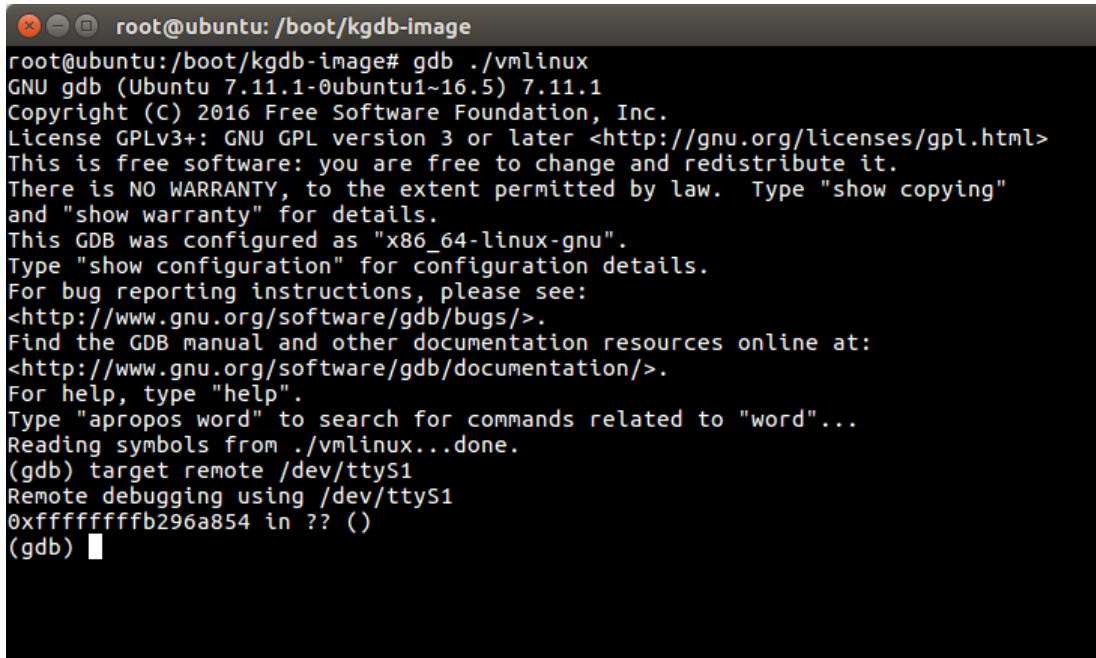
7. Run the command
   # gdb ./vmlinux

AT THIS POINT, YOU SHOULD SEE A MESSAGE SAYING "Reading symbols from ./vmlinux" AND A NEW LINE WITH THE WORD "(gdb)".

This is the gdb tool that we will use to debug the kernel.

8. Connect to the Target machine with the command
   (gdb) target remote /dev/ttyS1
   You should see the words *"Remote debugging using /devttyS1"*, and the terminal (gdb) again.

```
root@ubuntu: /boot/kgdb-image
root@ubuntu:/boot/kgdb-image# gdb ./vmlinux
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vmlinux...done.
(gdb) target remote /dev/ttyS1
Remote debugging using /dev/ttyS1
0xffffffffb296a854 in ?? ()
(gdb)
```

AT THIS POINT:
- There should be no error message. If there is, please check that the machines can communicate using Serial Port as shown in Section 1.2.
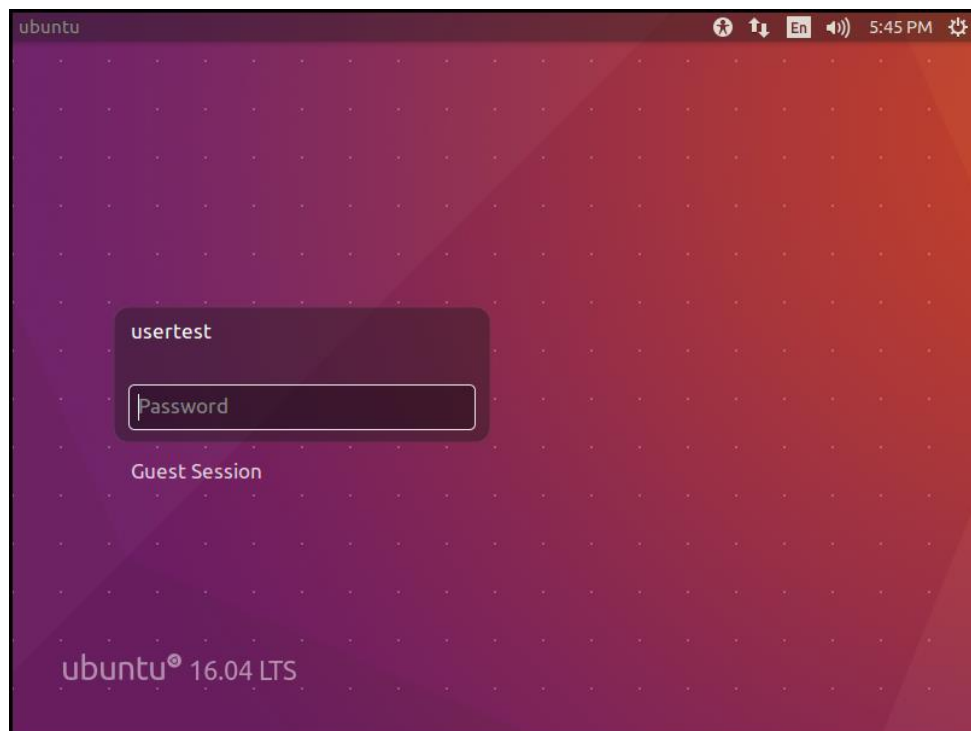- The Target machine still displays the message **KGDB: Waiting for connection from remote gdb…**

We are now connected to the Target machine. We can use GDB commands to debug kernel functions.

9. The target machine will be frozen until we let the kernel execution to continue. In order to do so, run the command
(gdb) continue

When doing so, the Target virtual machine should finish its booting up process.

[Screenshot # 11 and #12: Include in your report and video at least two screenshots on how both virtual machines connect and gdb in action.]