Introduction to Operating Systems

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

(To be lectured on 2021-10-01)

Prof: Ying-Dar Lin (林盈達)

TA: Ricardo Pontaza

Deadline: 2021-10-29 (Fri) 23:59:59.

Q&A:

If you have any questions, please post it on the E3 forum and it will be answered within two days.

Deliverables:

1. **Demo video** (5 minutes – upload to YouTube – add link in the first page of the report).
2. **Report** (Upload to E3): PDF file with file name of the form **OS_Project01_StudentID.pdf**.
    - Screenshots + one small explanation paragraph per screenshot section.
    - Answers to questions below.

In both the demo video and report, for each screenshot, explain what has been done, and 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 Operating 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 operating 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: https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html
3. Free OBS: https://obsproject.com/

Projects Distribution:

1. **Project 01: Linux Kernel Download, Patch, Compilation, Debugging and Profiling**
   (Ubuntu 16.04.7 **Server** - Kernel **4.4.101** + Ubuntu 16.04.7 **Desktop** – Kernel **4.15.0-142**)
2. Project 02: Linux Kernel System Calls
   (Ubuntu 16.04.7 **Desktop** – Kernel **4.19.148**)
3. Project 03: Dynamically-Loadable Kernel Modules (DLKMs)
   (Ubuntu 16.04.7 **Desktop** – Kernel **4.19.148**)

Table of contents:

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? (Add a figure)
6. For this project, why do we need two virtual machines?
7. In Section 3.2, what are the differences between **make**, **make modules_install** and **make install**?
8. In Section 3.3, 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.
11. What is a kernel function? What is a system call?
12. What is KASLR? What is it for?
13. What are GDB's non-stop and all-stop modes?
14. Explain what the command **echo g > /proc/sysrq-trigger** does.
15. What are these functions: **clone**, **mmap**, **write** and **open?**
16. Why is there no **fork** system call? What is the difference between **fork** and **clone?**
17. **(Questions at the end of section 5)**

Notes:

1. Use the project report check list as your first page of your report. (NEXT PAGE)
2. You can refer to the following tutorial for the previous year's project 01:
    a. Serial Port Communication: https://www.youtube.com/watch?v=SPVRfSvIfrs
    b. SSH Connection: https://www.youtube.com/watch?v=0XB33E_lfdk
    c. Kernel Source code download, compile and KGDB setup: https://www.youtube.com/watch?v=Y5RWhIyIJpM
    d. Creating KGDB breakpoints and how to trigger them: https://www.youtube.com/watch?v=e-RgDwHOlPk
    e. Example of previous year's video: https://www.youtube.com/watch?v=3FV9X3gy1ws https://www.youtube.com/watch?v=UVWqTSn5hPc

These videos are for guidance only, as this year's project uses a different kernel.

Operating Systems Project Report

Project Number (01 / 02 / 03):	
Name:	
Student ID:	
YouTube link (Format youtube.com/watch?v=[key]):	
Date (YYYY-MM-DD):	
Names of the files uploaded to E3:	
Physical Machine Total RAM (Example: 8.0 GB):	
Physical Machine CPU (Example: Intel i7-2600K):	

Checklist	
Yes/No	**Item**
	The report name follows the format "OS_ProjectXX_StudentID.pdf".
	The report was uploaded to E3 before the deadline.
	The YouTube video is public, and anyone with the link can watch it.
	The audio of the video has a good volume.
	The pictures in your report and video have a good quality.
	All the questions and exercises were answered inside the report.
	I understand that late submission is late submission, regardless of the time uploaded.
	I understand that any cheating in my report / video / code will not be tolerated.

SECTION 1:

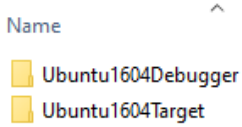
OPERATING SYSTEM INSTALLATION AND ADDITIONAL CONFIGURATIONS

SECTION 1: OPERATING SYSTEM INSTALLATION

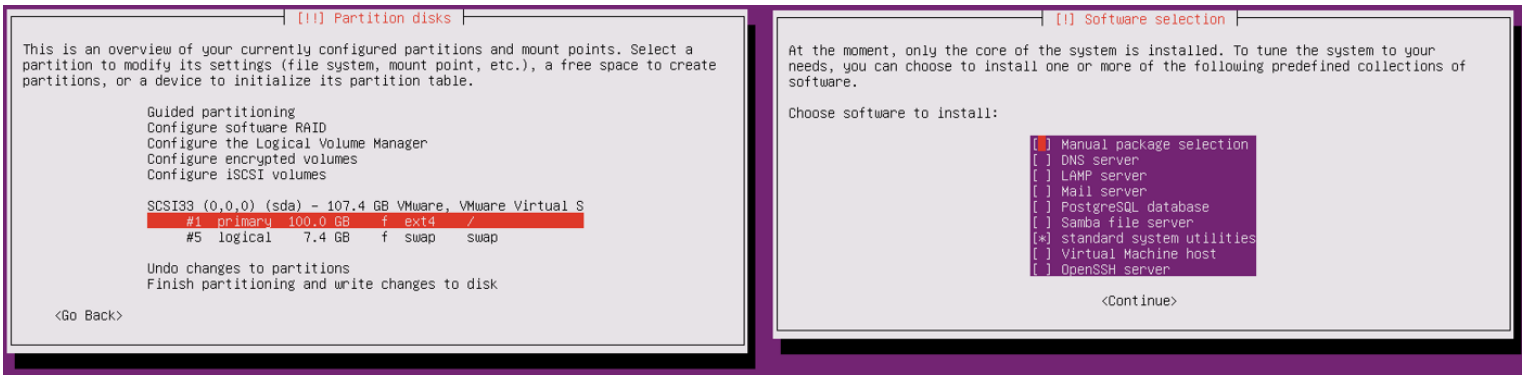
In this section, we will create and configure the virtual machines required for this project.

Section 1.1: Installation

1. Download the **Ubuntu 16.04.7 Server** and **Ubuntu 16.04.7 Desktop** isos.
2. Install VMWare Player: https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html
3. Create a folder called "VHD-VMWarePlayer" in a central location (example C or D), and create two subfolders:
    a. **Ubuntu1604Target** (This machine will contain the Linux Kernel code).
    b. **Ubuntu1604Debugger** (This machine will debug the target).



4. Create the two virtual machines:
    a. Ubuntu1604Target (**Target**):
        i. OS: **Ubuntu Server 16.04.7**
        ii. RAM: **1,600** Mb
        iii. HD: 100 Gb (**Split in multiple files**, and **not preallocated**).
        iv. Username: **usertest+StudentID** (Example: **usertest12345**).
        v. Do a manual partition table, and create the following partitions:
            i. Primary (ext4) – 100Gb – Mount root (/)
            ii. Logical – 7.4Gb - Swap
        vi. Install only the "Standard system utilities".



    b. Ubuntu1604Debugger (**Host**):
        i. OS: **Ubuntu Desktop 16.04.7.**
        ii. RAM: 2,048 Mb.
        iii. HD: 100 Gb (**Split in multiple files**, and **not preallocated**).
        iv. Username: **userhost+StudentID**

AT THIS POINT, YOU SHOULD HAVE TWO WORKING VIRTUAL MACHINES WITH THESE SPECS.

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

NOTES:

1. Do not use Windows Subsystem for Linux (WSL).
2. If you do not know how to create virtual machines, search online for tutorials.

Section 1.2: Target machine terminal colors

It might be hard to read the terminal in the Ubuntu Server edition. So, to change the color palette:

1. Go to the home (~) folder, and edit the file **.bashrc**

```
$ cd ~
$ sudo nano .bashrc
```

2. Search for #force_color_prompt=yes, and uncomment it (remove the # sign).

```
# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
force_color_prompt=yes
```

3. Save the file (control + X, then save).
4. Log out and log in again with **$ exit**.
5. At this point, your terminal should display different colors.

```
usertest@ubuntu:~$ ls -a
.              .bash_logout   bin      libexec   .rnd                          .Xauthority
..             .bashrc        .cache   .nano     .sudo_as_admin_successful
.bash_history  .bashrc-backup etc      .profile  .tracecvent
usertest@ubuntu:~$
```

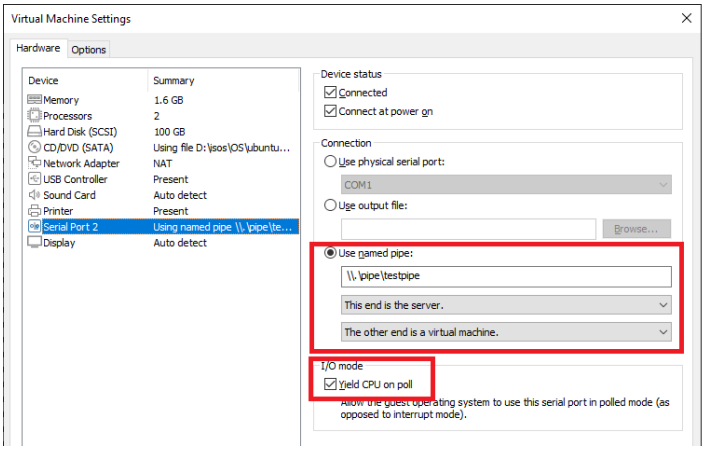
NOTE: If you don't know how to search and save using nano, search online for tutorials.

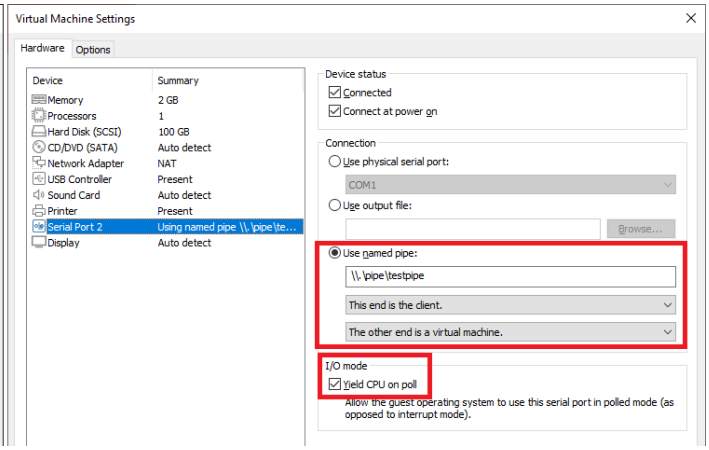
Section 1.3: Serial port communication

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.
- **SSH:** This will be the communication channel to move 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**.

At this point, check if the serial port is enabled or disabled at the corner of your virtual machine. It must be enabled on both machines from now on.

2. With the serial port connected, turn on both virtual machines.
3. In the Host machine, open a terminal and type

$ sudo su (followed by root's password) **# cat /dev/ttyS1**

4. In the Target machine, type

$ sudo su (followed by root's password) **# echo 'test for ttyS1 + studentID' > /dev/ttyS1**

```
usertest@ubuntu:~$ sudo su
[sudo] password for usertest:
root@ubuntu:/home/usertest# echo 'test for ttyS1 12345' > /dev/ttyS1
root@ubuntu:/home/usertest#
```

5. In the Host machine, the message ***test for ttyS1*** + studentID should appear.

```
userhost@ubuntu:~$ sudo su
[sudo] password for userhost:
root@ubuntu:/home/userhost# cat /dev/ttyS1
test for ttyS1 12345
```

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

[Screenshot # 2: Take a screenshot of both messages and add it to your video and report]

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:~$ 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:
  libwrap0 ncurses-term openssh-sftp-server python3-requests python3-urllib3 ssh-import-id tcpd
Suggested packages:
  ssh-askpass rssh molly-guard monkeysphere python3-ndg-httpsclient python3-openssl python3-pyasn1
The following NEW packages will be installed:
  libwrap0 ncurses-term openssh-server openssh-sftp-server python3-requests python3-urllib3
  ssh-import-id tcpd
0 upgraded, 8 newly installed, 0 to remove and 102 not upgraded.
Need to get 817 kB of archives.
After this operation, 5,896 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

```
usertest@ubuntu:~$ sudo ufw allow ssh
Rules updated
Rules updated (v6)
usertest@ubuntu:~$ _
```

3. Enable remote connection in the /etc/ssh/sshd_config file, by using the command

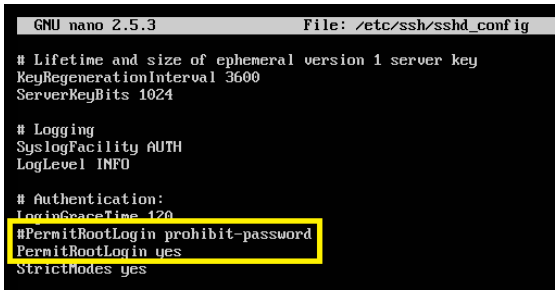
   $ sudo nano /etc/ssh/sshd_config

   And modifying the line

   PermitRootLogin prohibit-password

   To

   PermitRootLogin yes



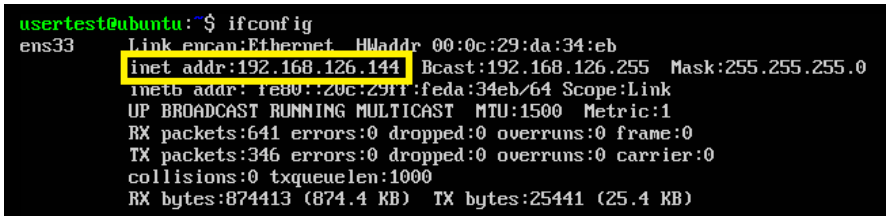
4. Restart the ssh service by using the command

   $ sudo service sshd restart

5. Create a document called **TestDoc+studentID.txt** with the message "*This is a test file*" inside.

6. Get Target's IP address by using the command

   $ ifconfig

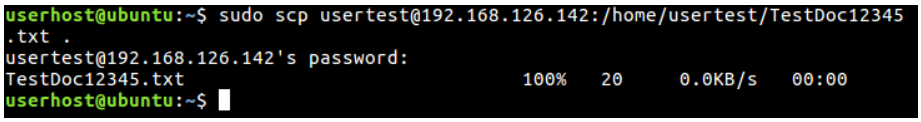


   NOTE: From this point on, any time you see **brackets []** it indicates that you need to replace the content for its value. (Example: [usertest+StudentID] = usertest12345)

7. In the **Host** machine, run the command

   $ sudo scp [usertest+StudentID]@[ip address of target]:/home/[usertest+StudentID]/[TestDoc+studentID.txt] .

   (Note that there is a space and a dot at the end of the command. This means to save the file in the current folder).



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.

SECTION 2:

LINUX KERNEL DOWNLOAD

Section 2: Linux Kernel Download

In this section, we proceed to download the code source for the Linux Kernel we will use in our virtual machines. (Some of the steps, depending on your computer, will take several hours to finish).

Section 2.1: Linux Kernel Download

The following steps must be done in **both machines:**

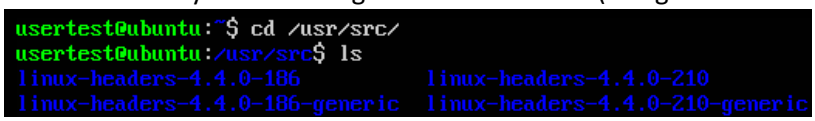
1. Update and upgrade

   $ sudo apt update
   $ sudo apt upgrade

   and install the needed build software by running the command

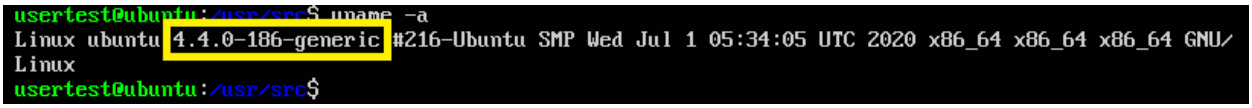
   $ sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev

2. Go to **/usr/src** and check that you have the generic Linux kernel (using the command **ls**).



3. Get the current kernel version by running the command

   $ uname -a



4. Download the Linux Kernel 4.4.101 tarball in terminal, by using the command

   $ sudo wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.101.tar.xz

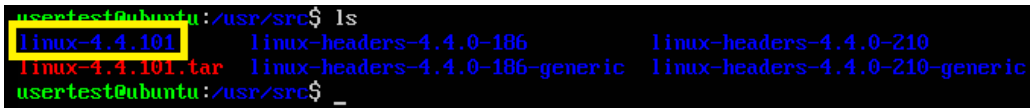
5. Decompress the compressed file by using the command

   $ sudo unxz -v linux-4.4.101.tar.xz

6. Untar the tar file by using the command

   $sudo tar xvf linux-4.4.101.tar

   AT THIS POINT, A NEW FOLDER (linux-4.4.101) SHOULD EXIST UNDER /usr/src/.



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

Section 2.2: Pre-build Additional Configurations

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

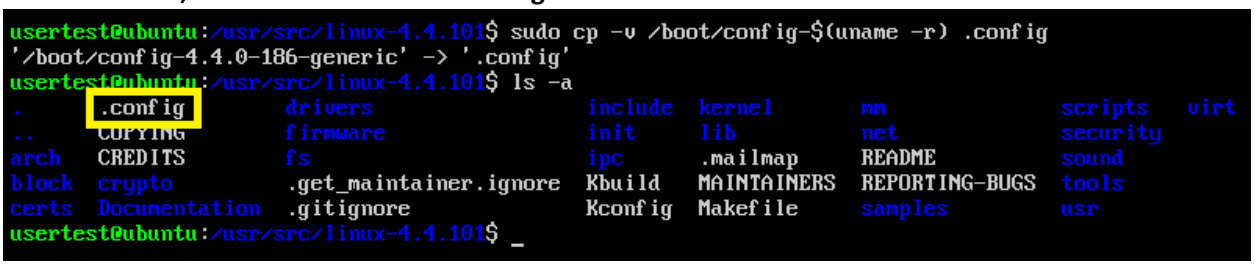
1. Get into the new kernel's folder (/usr/src/linux-4.4.101)
2. Copy the current .config file into this folder by using the command

   $sudo cp -v /boot/config-$(uname -r) .config

   If this step is successful, you should get an output of the form

   '/boot/config-4.4.0-186-generic' -> '.config'

   AT THIS POINT, YOU SHOULD HAVE A .config FILE UNDER THE linux-4.4.101 FOLDER.



[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 Hack and Build

Section 3.1: Linux Kernel Hacking

In this section we proceed to patch and prepare the kernel prior to building it.

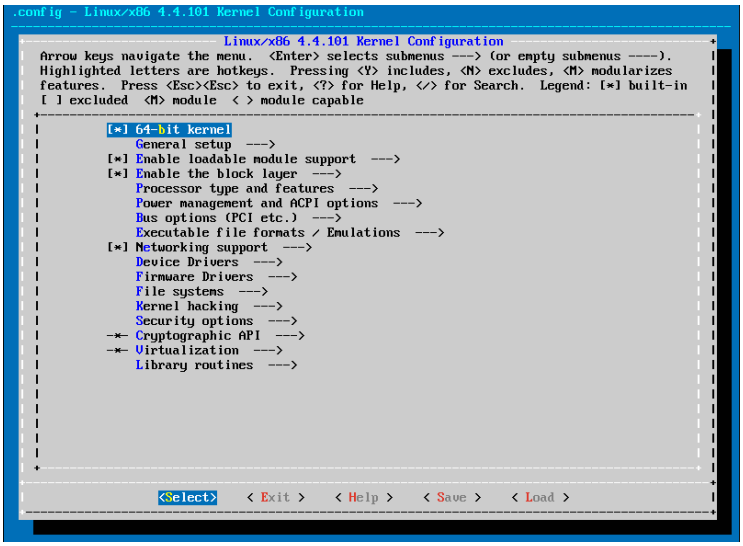
1. In the Target machine, inside the /usr/src/linux-4.4.101 folder, run the command

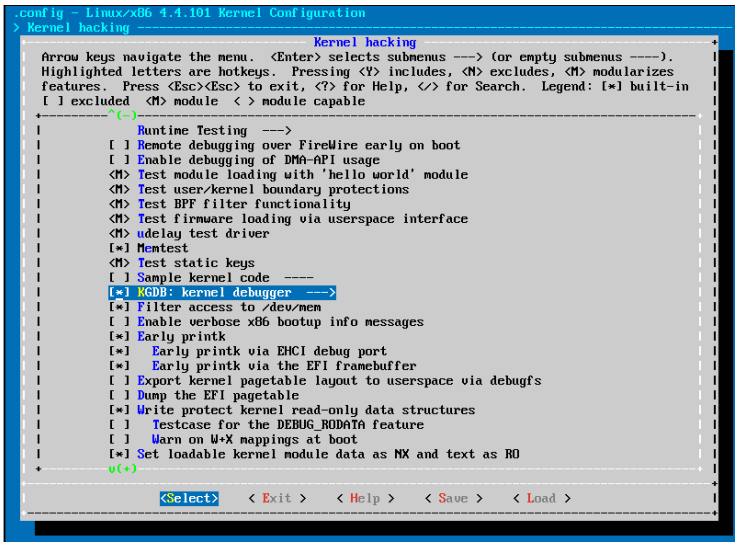
   $sudo make menuconfig

   A new window will appear. Be sure that the following options are selected:

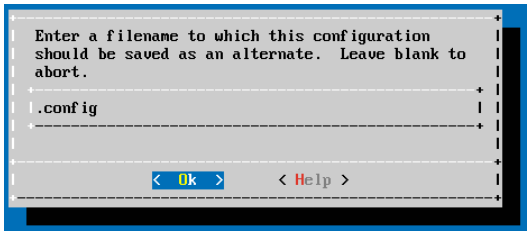
   a. 64-bit kernel

   b. KGDB: kernel debugger (under **Kernel hacking**)



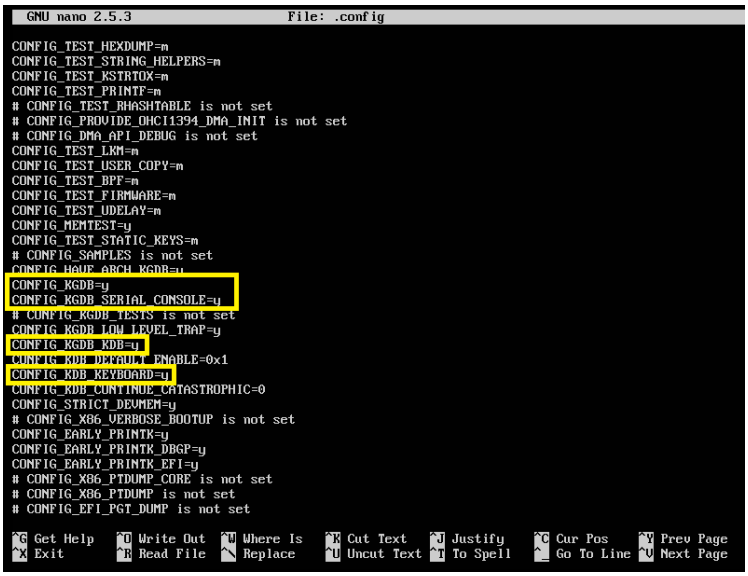


2. Save the configuration as .config



3. In the /usr/src/linux-4.4.101 folder, open the .config file.
4. Search the following commands. They should have 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, add them (be extremely careful to not double-add them)

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

Section 3.2: Linux Kernel Build

In this section we proceed to build the Linux kernel. **(It will take a couple of hours to finish).**

1. Install the **bc** command.

```
$ sudo apt install bc
```

2. Under the /usr/src/linux-4.4.101 folder, clean any pre-compiled files, and build the kernel

```
$ sudo make clean
$ sudo make -j $(nproc)
```

```
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo make clean
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo make -j $(nproc)
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --silentoldconfig Kconfig
  SYSTBL  arch/x86/entry/syscalls/../../include/generated/asm/syscalls_32.h
  HOSTCC  scripts/basic/bin2c
  CHK     include/config/kernel.release
  UPD     include/config/kernel.release
```

3. When this process is done, run the command

```
$ sudo make modules_install
```

```
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo make modules_install
[sudo] password for usertest:
  INSTALL arch/x86/crypto/aes-x86_64.ko
  INSTALL arch/x86/crypto/aesni-intel.ko
  INSTALL arch/x86/crypto/blowfish-x86_64.ko
  INSTALL arch/x86/crypto/camellia-aesni-avx-x86_64.ko
```

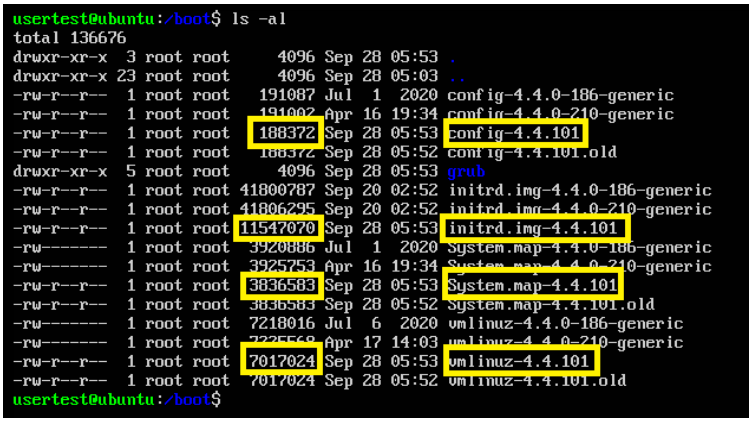
4. After this process is done, run the command

$sudo make install

```
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo make install
sh ./arch/x86/boot/install.sh 4.4.101 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.4.101 /boot/vmlinuz-4.4.101
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.4.101 /boot/vmlinuz-4.4.101
update-initramfs: Generating /boot/initrd.img-4.4.101
```

[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.]

5. When the previous step is done, four new files should now exist under your **/boot** folder (check that their sizes are somewhat big):
    a. initrd.img-4.4.101
    b. vmlinuz-4.4.101
    c. system.map.4.4.101
    d. config-4.4.101



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 BREAK YOUR KERNEL, MAKING THE VIRTUAL MACHINE USELESS, AND YOU WILL HAVE TO REPEAT ALL THE PREVIOUS STEPS SINCE SECTION 2.1.

Section 3.3: Grub update

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 4.4.101

```
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo update-initramfs -c -k 4.4.101
update-initramfs: Generating /boot/initrd.img-4.4.101
```

2. Run the command

$ sudo update-grub

```
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.4.101
Found initrd image: /boot/initrd.img-4.4.101
Found linux image: /boot/vmlinuz-4.4.0-210-generic
Found initrd image: /boot/initrd.img-4.4.0-210-generic
Found linux image: /boot/vmlinuz-4.4.0-186-generic
Found initrd image: /boot/initrd.img-4.4.0-186-generic
done
usertest@ubuntu:/usr/src/linux-4.4.101$
```

3. Modify the /etc/default/grub file with the command

$ sudo nano /etc/default/grub

And comment out the following lines

#GRUB_HIDDEN_TIMEOUT=0
#GRUB_HIDDEN_TIMEOUT_QUIET=true

Finally, add “nokaslr” at the GRUB_CMDLINE_LINUX command

```
 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=2
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="nokaslr"
GRUB_CMDLINE_LINUX=""

# 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)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"


^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  ^Y Prev Page
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line ^V Next Page
```

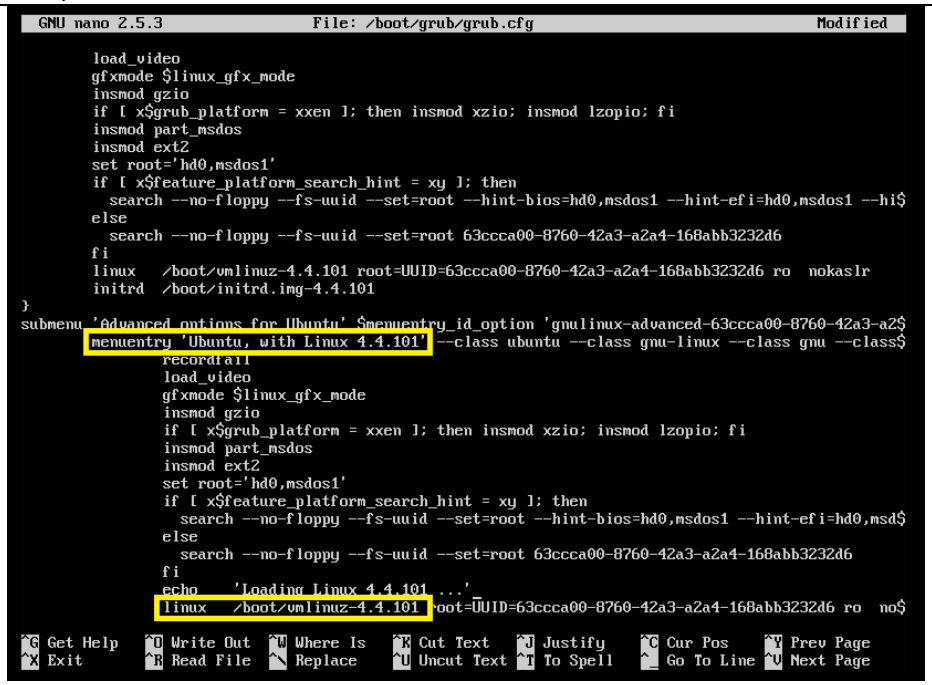
4. Update the grub again by running the command

$ sudo update-grub

```
usertest@ubuntu:/usr/src/linux-4.4.101$ sudo update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.4.101
Found initrd image: /boot/initrd.img-4.4.101
Found linux image: /boot/vmlinuz-4.4.0-210-generic
Found initrd image: /boot/initrd.img-4.4.0-210-generic
Found linux image: /boot/vmlinuz-4.4.0-186-generic
Found initrd image: /boot/initrd.img-4.4.0-186-generic
done
usertest@ubuntu:/usr/src/linux-4.4.101$
```

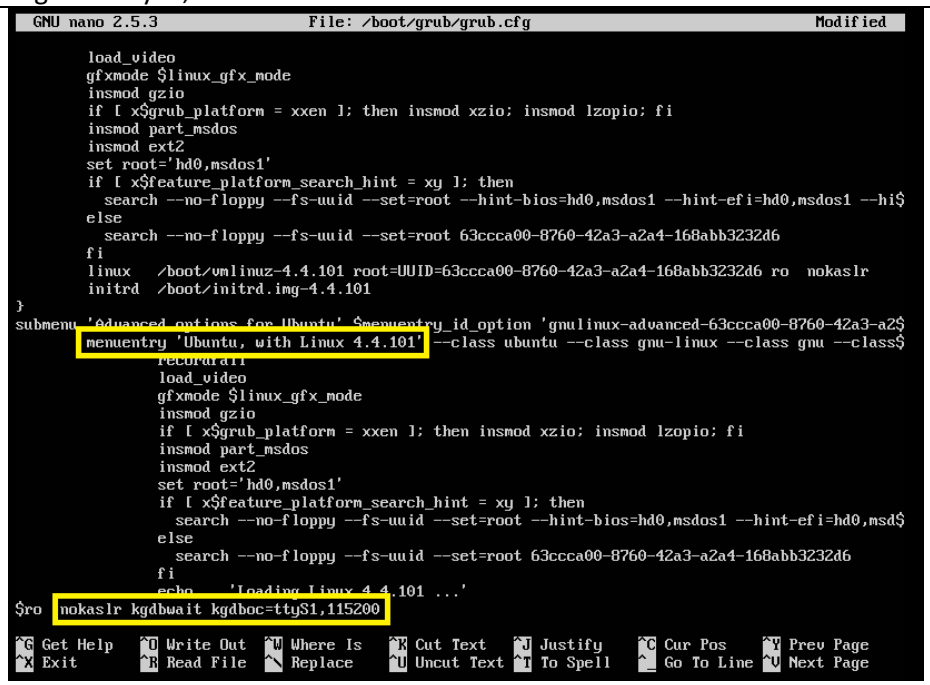
5. Open **/boot/grub/grub.cfg** and find the menuentry 'Ubuntu, with Linux 4.4.101'. Look for

linux /boot/vmlinuz-4.4.101



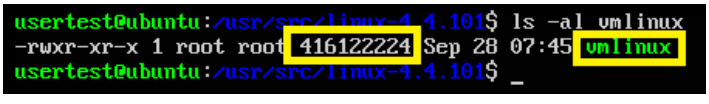
Add the following to the ending of that command

kgdbwait kgdboc=ttyS1,115200



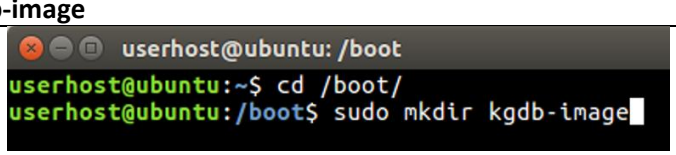
After updating the file, save it.

6. Verify that there is a **vmlinux** file under /usr/src/linux-4.4.101.



7. Turn on the **Host** machine (with the serial port connected).
8. Create a folder called **kgdb-image** inside the **/boot/** folder by using the command

$ sudo mkdir kgdb-image



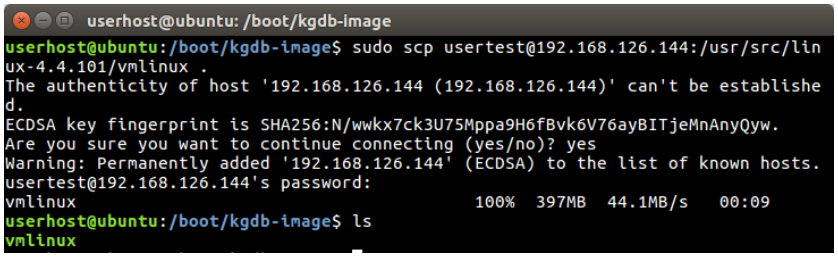
9. **Get inside the kgdb-image folder**, and run the command

$ sudo scp [usertest+studentID]@[target machine IP address]:/usr/src/linux-4.4.101/vmlinux .

(beware of the empty space between "vmlinux" and ".")

Example: *$sudo scp usertest12345@192.168.126.144:/usr/src/linux-4.4.101/vmlinux .*

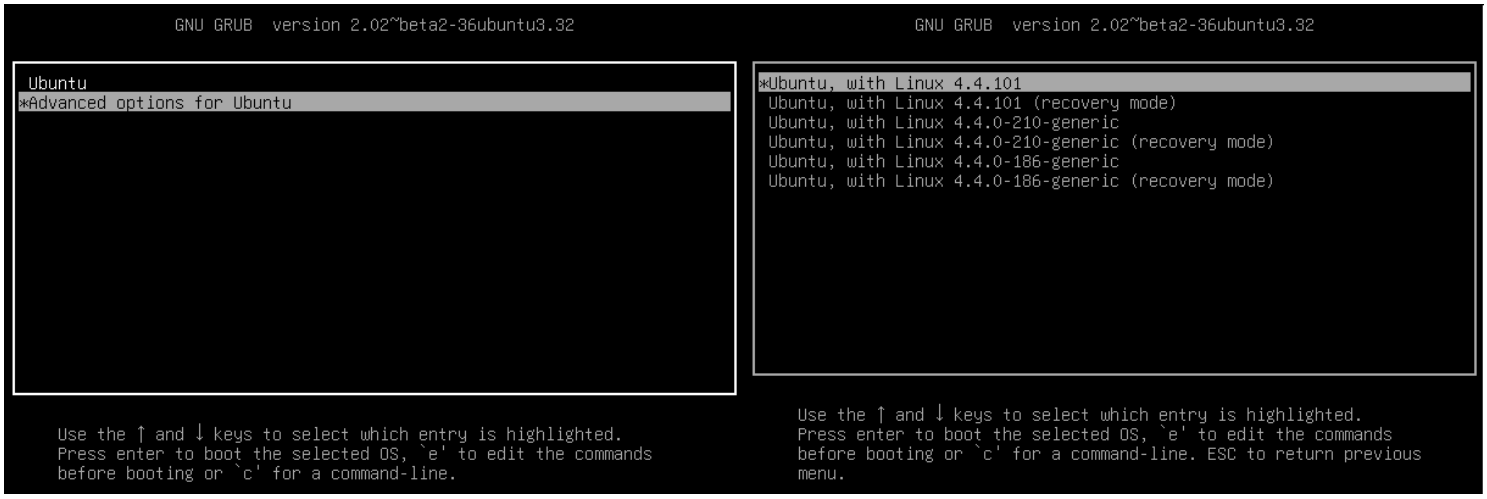
After running this command, the vmlinux file should appear inside the kgdb-image folder. (If you have any questions, please refer to section 1.3)



[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.

NOTE: From now on, to use the kernel 4.4.101, when booting, you **must** select "**Advanced options for Ubuntu**", and then the "**Ubuntu, with Linux 4.4.101**" option in the grub.



SECTION 4:

LINUX KERNEL DEBUG [KGDB]

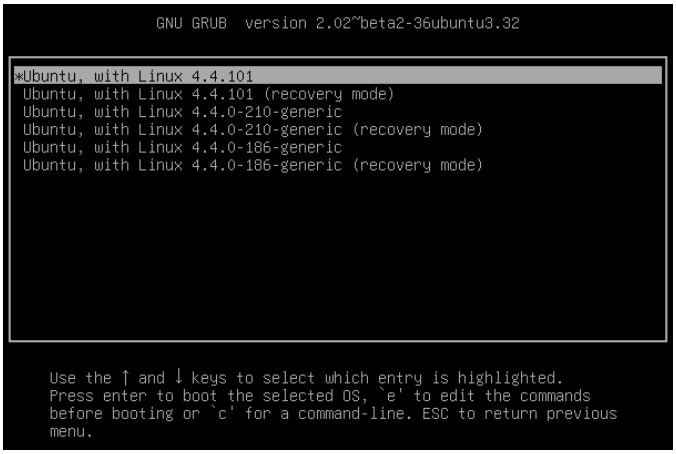
Section 4: Linux Kernel Debug [KGDB]

(Extra) Section 4.0: Bypassing Debug mode

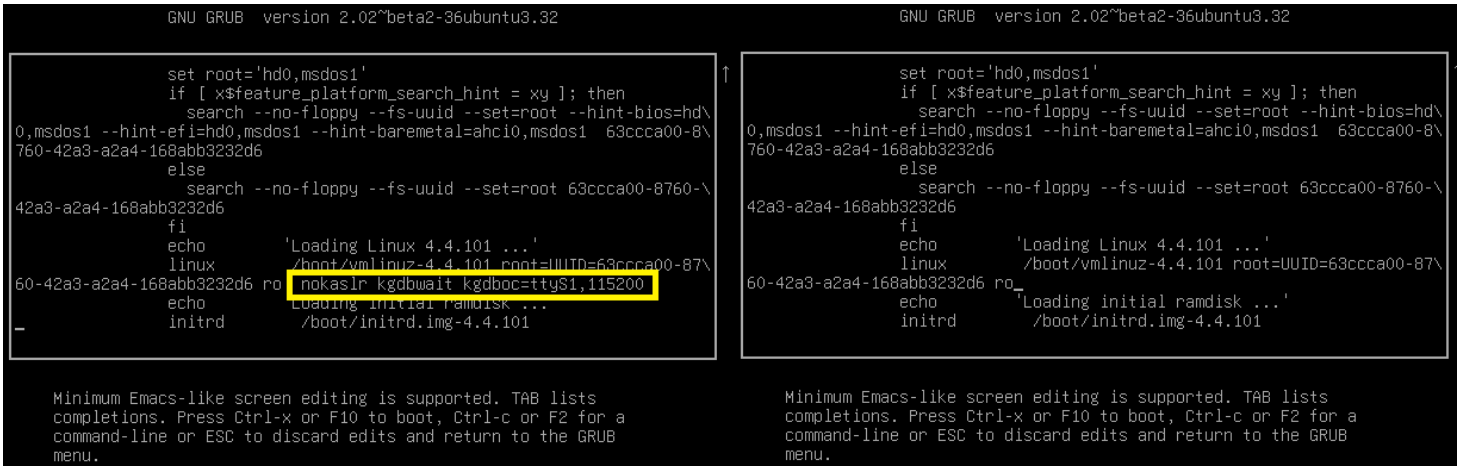
This section covers a bypass on the debug mode of the Target machine in case you need to re-do any step from the previous sections. Because of the steps we performed on Section 3, if you turn on the Target machine with kernel 4.4.101, you should reach a black screen with the message

KGDB: Waiting for connection from remote gdb…

If you need to re-do any step from the previous sections, over the kernel 4.4.101, press the "**e**" key



You will see the menu entry we edited in Section 3.3, step 5. You can delete the **nokaslr kgdbwait kgdboc=ttyS1,115200** parameters, and boot using **Control + X.**



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 Section 3.3 and deleting 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. With both machines off, connect the Serial port in both of them. Then turn them back on.
2. In the **Target** machine, you should reach a black screen with the words **KGDB: Waiting for connection from remote gdb…**

```
[    11.550756] KGDB: Waiting for connection from remote gdb...

Entering kdb (current=0xffff88005f4f0000, pid 1) on processor 1 due to Keyboard
Entry
[1]kdb> _
```

3. 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 3.3 is inside this folder.

```
userhost@ubuntu: /boot/kgdb-image
userhost@ubuntu:/boot/kgdb-image$ ls
vmlinux
userhost@ubuntu:/boot/kgdb-image$
```

4. Login as **su** with the command

```
$ sudo su
```

The terminal should change from **$** to **#** and all the letters will be white now.

5. Run the command

```
# gdb ./vmlinux
```

```
root@ubuntu: /boot/kgdb-image
userhost@ubuntu:/boot/kgdb-image$ sudo su
[sudo] password for userhost:
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)
```

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.

6. 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
userhost@ubuntu:/boot/kgdb-image$ sudo su
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
kgdb_breakpoint () at kernel/debug/debug_core.c:1072
1072            wmb(); /* Sync point after breakpoint */
(gdb)
```

AT THIS POINT: There should be no error message. If there is, please copy the vmlinux file again (as shown in Section 3.3). Also, the **Target** machine should still display 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.

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

(gbd) continue

```
root@ubuntu: /boot/kgdb-image
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
kgdb_breakpoint () at kernel/debug/debug_core.c:1072
1072            wmb(); /* Sync point after breakpoint */
(gdb) continue
Continuing.
```

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

```
Ubuntu 16.04.7 LTS ubuntu tty1

ubuntu login:
```

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

8. Log in into the **Target** machine and **wait for around 45 seconds**. We need to send a signal to the host to re-take control in GDB. As root, run the command

```
# echo g > /proc/sysrq-trigger
```

```
usertest@ubuntu:~$ sudo su
[sudo] password for usertest:
root@ubuntu:/home/usertest# echo g > /proc/sysrq-trigger
```

The **Target** machine will freeze and you will get access again to GDB in the **Host**.

```
root@ubuntu: /boot/kgdb-image
[New Thread 997]
[New Thread 1124]
[New Thread 1216]
[New Thread 1141]
[New Thread 1142]
[New Thread 1173]
[New Thread 1182]
[New Thread 1207]
[New Thread 1316]
[New Thread 1376]
[New Thread 1378]
[New Thread 1380]
[New Thread 1384]
[New Thread 1397]
[New Thread 1399]
[New Thread 1400]

Thread 41 received signal SIGTRAP, Trace/breakpoint trap.
[Switching to Thread 1401]
kgdb_breakpoint () at kernel/debug/debug_core.c:1072
1072            wmb(); /* Sync point after breakpoint */
(gdb)
```

KGDB will debug **KERNEL FUNCTIONS**. In the kernel 4.4.101 source code folder, check for the **/arch/x86/entry/syscalls/syscall_64.tbl** file. This has the list of the kernel functions with its system call number.

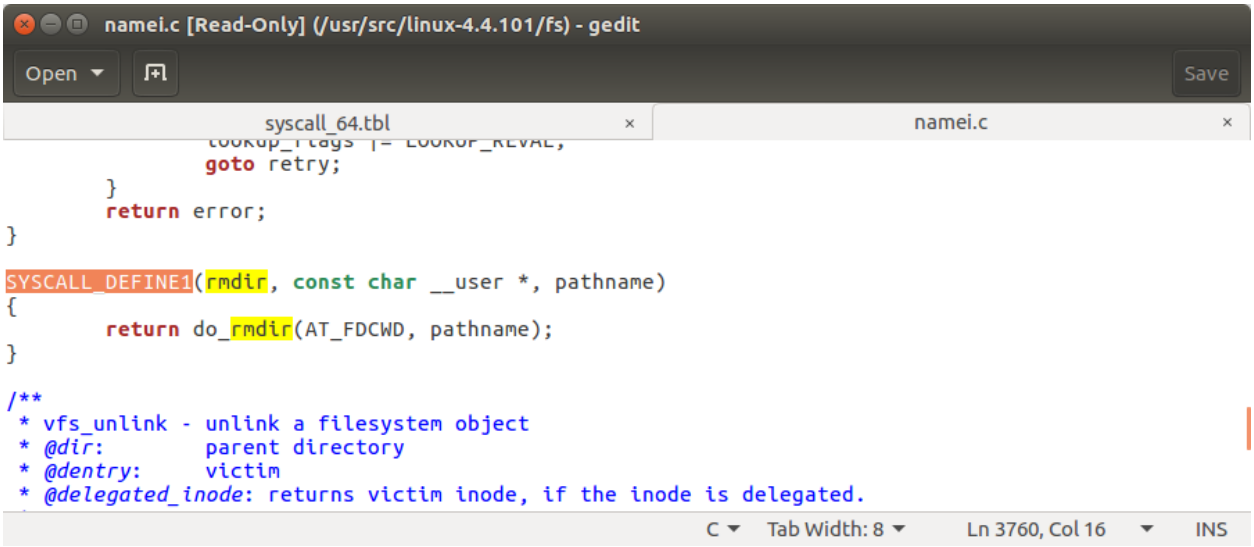
```
syscall_64.tbl [Read-Only] (/usr/src/linux-4.4.101/arch/x86/entry/syscalls) - gedit
Open   Save
syscall_64.tbl   namei.c
63      common  uname           sys_newuname
64      common  semget          sys_semget
65      common  semop           sys_semop
66      common  semctl          sys_semctl
67      common  shmdt           sys_shmdt
68      common  msgget          sys_msgget
69      common  msgsnd          sys_msgsnd
70      common  msgrcv          sys_msgrcv
71      common  msgctl          sys_msgctl
72      common  fcntl           sys_fcntl
73      common  flock           sys_flock
74      common  fsync           sys_fsync
75      common  fdatasync       sys_fdatasync
76      common  truncate        sys_truncate
77      common  ftruncate       sys_ftruncate
78      common  getdents        sys_getdents
79      common  getcwd          sys_getcwd
80      common  chdir           sys_chdir
81      common  fchdir          sys_fchdir
82      common  rename          sys_rename
83      common  mkdir           sys_mkdir
84      common  rmdir           sys_rmdir
85      common  creat           sys_creat
86      common  link            sys_link
87      common  unlink          sys_unlink
88      common  symlink         sys_symlink
89      common  readlink        sys_readlink
90      common  chmod           sys_chmod
91      common  fchmod          sys_fchmod
92      common  chown           sys_chown
93      common  fchown          sys_fchown
94      common  lchown          sys_lchown
95      common  umask           sys_umask
96      common  gettimeofday    sys_gettimeofday
97      common  getrlimit       sys_getrlimit
Plain Text   Tab Width: 8   Ln 93, Col 50   INS
```

9. We will create a break point for the **rmdir** function, which will be triggered when we delete a folder.
   Look online for at least 2 tables showing where each kernel function is implemented. An example is https://filippo.io/linux-syscall-table/, but keep in mind that some functions inside the syscall_64.tbl file might be missing in the online tables. (Check for Linux system calls only). In the table, we find that the rmdir function is implemented in the **fs/namei.c** file. (These paths are inside the kernel folder).

80	chdir	sys_chdir	fs/open.c
81	fchdir	sys_fchdir	fs/open.c
82	rename	sys_rename	fs/namei.c
83	mkdir	sys_mkdir	fs/namei.c
84	rmdir	sys_rmdir	fs/namei.c
85	creat	sys_creat	fs/open.c
86	link	sys_link	fs/namei.c
87	unlink	sys_unlink	fs/namei.c

[Screenshot # 13: Create a screenshot showing the syscall tables you found online.]

In the namei.c file, check for an entry of the form **SYSCALL_DEFINE[N](rmdir,…)**, where [N] is an integer number. In this case, we find the entry as below



We see that the rmdir function calls the **do_rmdir** kernel function, and has the pathname as one of the parameters. We will create a breakpoint in this function.

10. We create a breakpoint with the command

```
(gdb) break do_rmdir
```

```
(gdb) break do_rmdir
Breakpoint 1 at 0xffffffff8121bad0: file fs/namei.c, line 3702.
(gdb)
```

It will create a breakpoint and give to you its ID (in this case is 1).

11. Type

```
(gdb) continue
```

and the Target machine will unfreeze. When you go back to the Target, if it is still frozen, come back to GDB and check if a breakpoint was hit. Type **continue** until the target machine is responsive again. If too many break points are hit, type

```
(gdb) delete
```

to delete all breakpoints, and

```
(gdb) continue
```

Wait for around 1 minute, and repeat steps 8, 10 and 11 again.

```
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) continue
Continuing.
```

12. Now with a responsive Target machine, create a directory called testfolder with the command

```
# mkdir testfolder
```

And then delete it with the command

```
# rmdir testfolder
```

As soon as you hit Enter, the **Target** machine should freeze

```
root@ubuntu:/home/usertest# ls
TestDoc12345.txt
root@ubuntu:/home/usertest# mkdir testfolder
root@ubuntu:/home/usertest# ls
TestDoc12345.txt  testfolder
root@ubuntu:/home/usertest# rmdir testfolder_
```

And the **Host** machine should show that a breakpoint was reached

```
(gdb) continue
Continuing.
[New Thread 1490]
[Switching to Thread 1490]

Thread 199 hit Breakpoint 2, do_rmdir (dfd=-100,
    pathname=0x7fffd286b8d4 "testfolder") at fs/namei.c:3702
3702    {
(gdb)
```

13. Print the *pathname* parameter for the **rmdir** function with the function

```
(gdb) print pathname
```

```
Thread 199 hit Breakpoint 2, do_rmdir (dfd=-100,
    pathname=0x7fffd286b8d4 "testfolder") at fs/namei.c:3702
3702    {
(gdb) print pathname
$1 = 0x7fffd286b8d4 "testfolder"
(gdb)
```

We should get the pathname of the folder we just deleted.

[Screenshot # 14: Create a screenshot showing the pathname parameter value. (It should include your student ID)]

14. Type

    (gdb) continue

    to make the Target machine responsive again.
15. **[Do it yourself]** Create a breakpoint for the **mkdir** function. Look for where it is implemented and create your own scenario to trigger it.

    Required to explain in both the report and in the video:

    a. Which parameters the kernel function has.
    b. The Target machine **must be totally on** and responsive before hitting the breakpoint.
    c. You must display the value of the parameter that the function receives.
    d. You must include 3 screenshots showing:
        i. How you created the breakpoint.

[Screenshot # 15: Show the commands used to create the breakpoint and the result].

ii. The host machine hitting the break point, and the value of at least one parameter.

[Screenshot # 16: Show both machines, the Target machine should be frozen and the Host machine should have the breakpoint]

iii. The Target machine working again, with the action totally finished.

[Screenshot # 17: Show both machines, the Target machine should have executed the command successfully, and the Host should show (gdb) continue].

SECTION 5:

PROFILING KERNEL FUNCTIONS

Section 5: Profiling Kernel functions

In this section, we will install the required tool to profile the kernel, and proceed to show how to profile and interpret some example functions. AT THIS POINT, YOU CAN TURN OFF THE TARGET MACHINE.

Section 5.1: Profiling tool installation

In this section, we will build the profiling function **perf**.

1. Update and upgrade your **Host** machine

   ```
   $ sudo apt update
   $ sudo apt upgrade
   ```

   After upgrading, restart your machine.
2. Check your kernel version. It should have the word “generic” in it. If it does not, enable the grub and log into one of the generic kernels (Section 3.3, steps 3 & 4, without “noasklr”).

   ```
   userhost@ubuntu:~$ uname -r
   4.15.0-142-generic
   ```

3. Install perf by running the commands

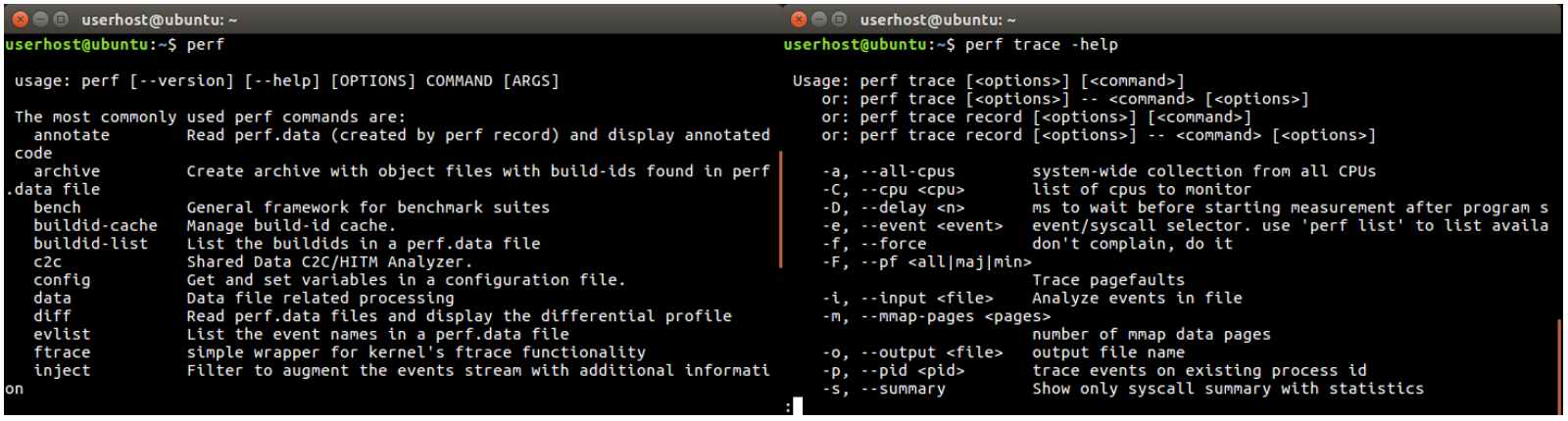
   ```
   $ sudo apt install linux-tools-common
   $ sudo apt install linux-tools-$(uname -r)
   ```

4. At this point, the perf tool should be created. Type

   ```
   $ perf
   ```

   to verify that the tool was built successfully.
   Also check that the perf trace command was installed, by running the command

   ```
   $ perf trace -help
   ```

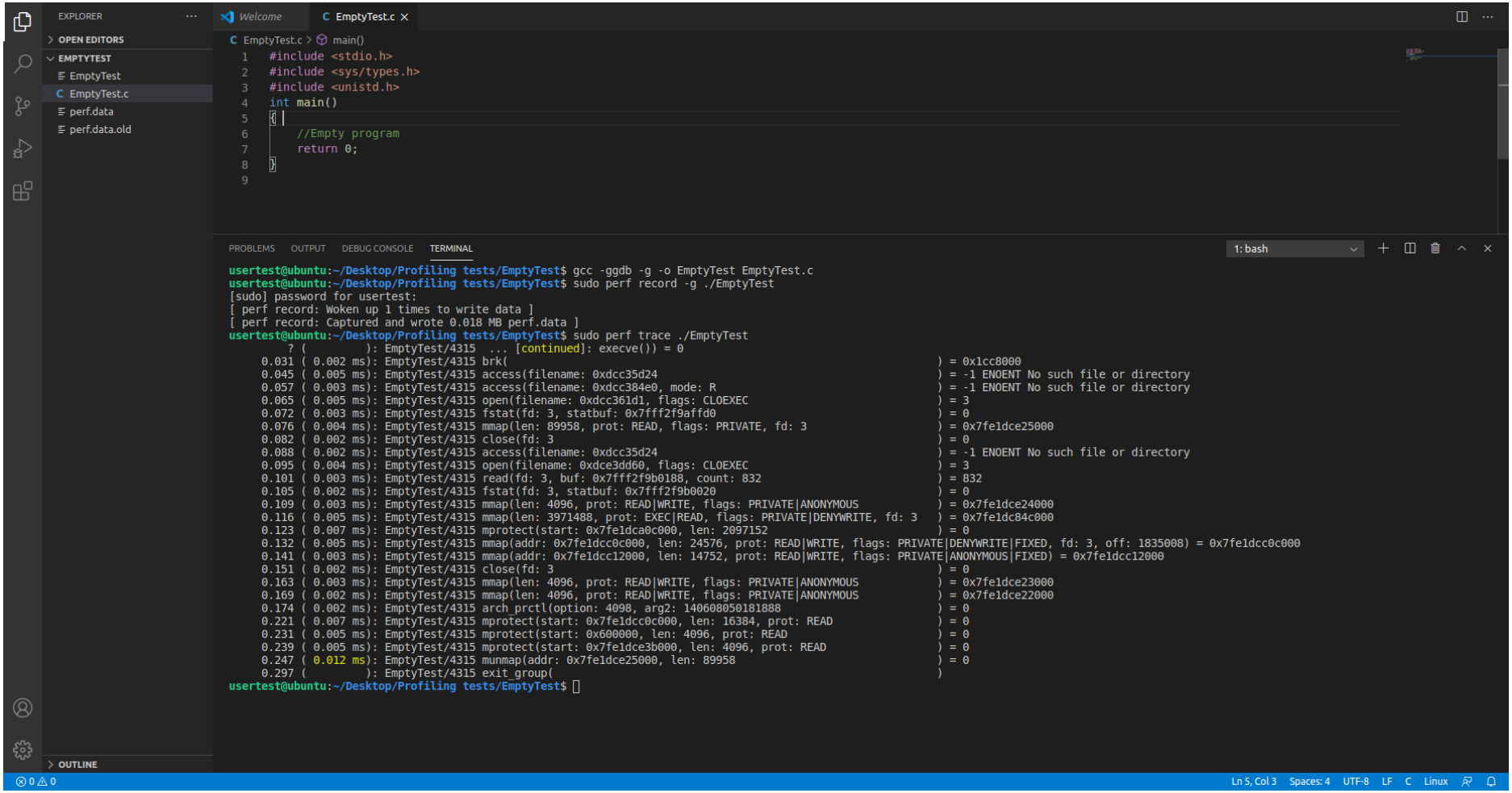


[Screenshot # 18: Create a screenshot showing the perf tool commands like in the previous screenshots. Your student ID must be visible.]

Section 5.2: Profiling functions

In this section, we will show how to use **perf**. Perf must be executed as root. The programs used in this section are available in E3.

1. Create a folder called **Profiling tests** in your Home directory.
2. Create two folders: **emptyTest** and **fileCopyTest**.
3. For **emptyTest**, download **emptyTest.c** from E3.



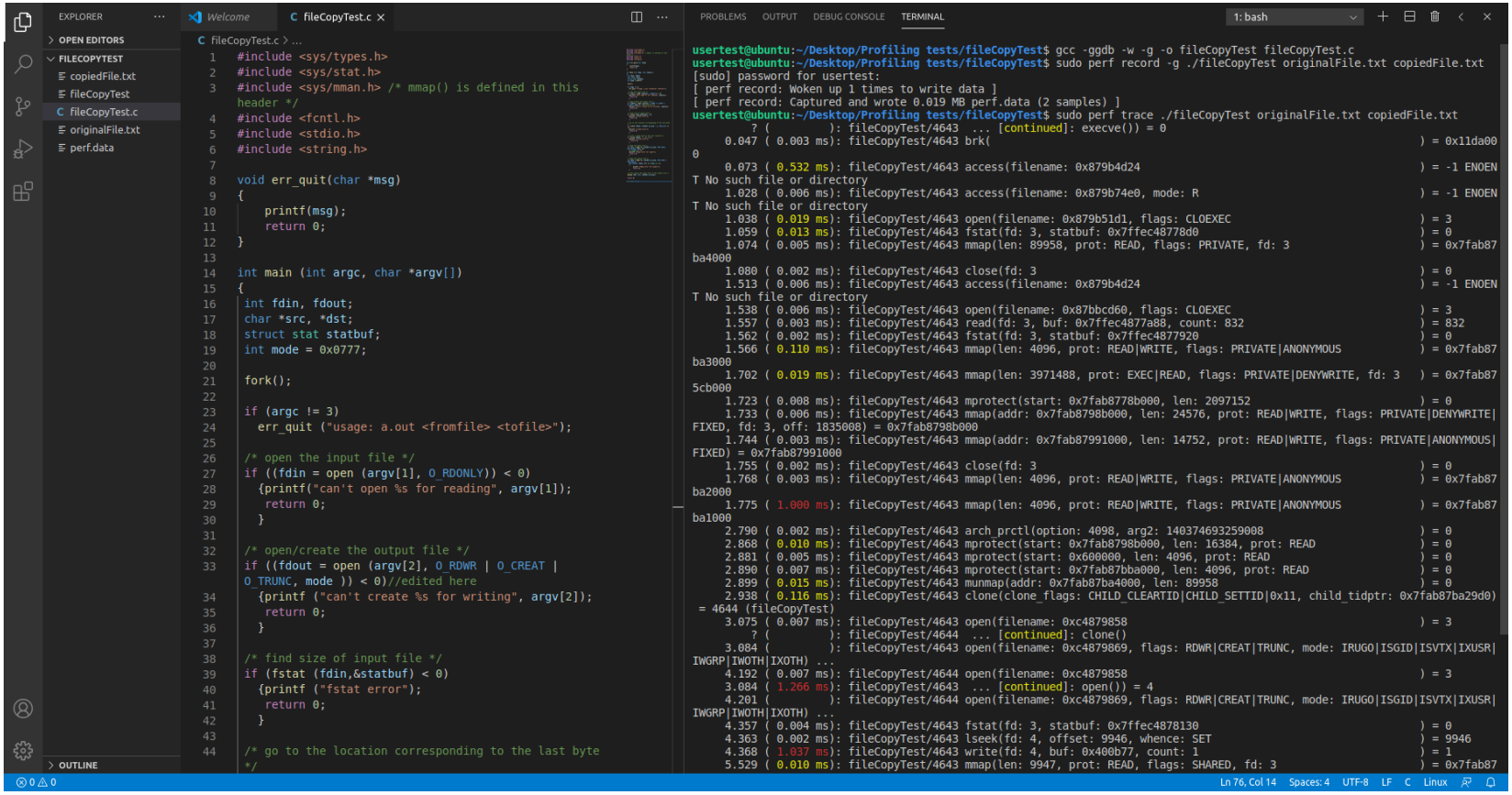
4. Run the commands

```
$ gcc -ggdb -g -o emptyTest emptyTest.c
$ sudo perf record -g ./emptyTest
$ sudo perf trace ./emptyTest
```

5. Copy the result of the trace command, and save it in a text file (call it **emptyTest.txt**).

From the previous image, you can see that perf record generates several lines of code for an empty file. These lines are common in any profiled compiled file. We will show how to ignore them.

6. For **fileCopyTest**, download the **fileCopyTest.c** from E3.



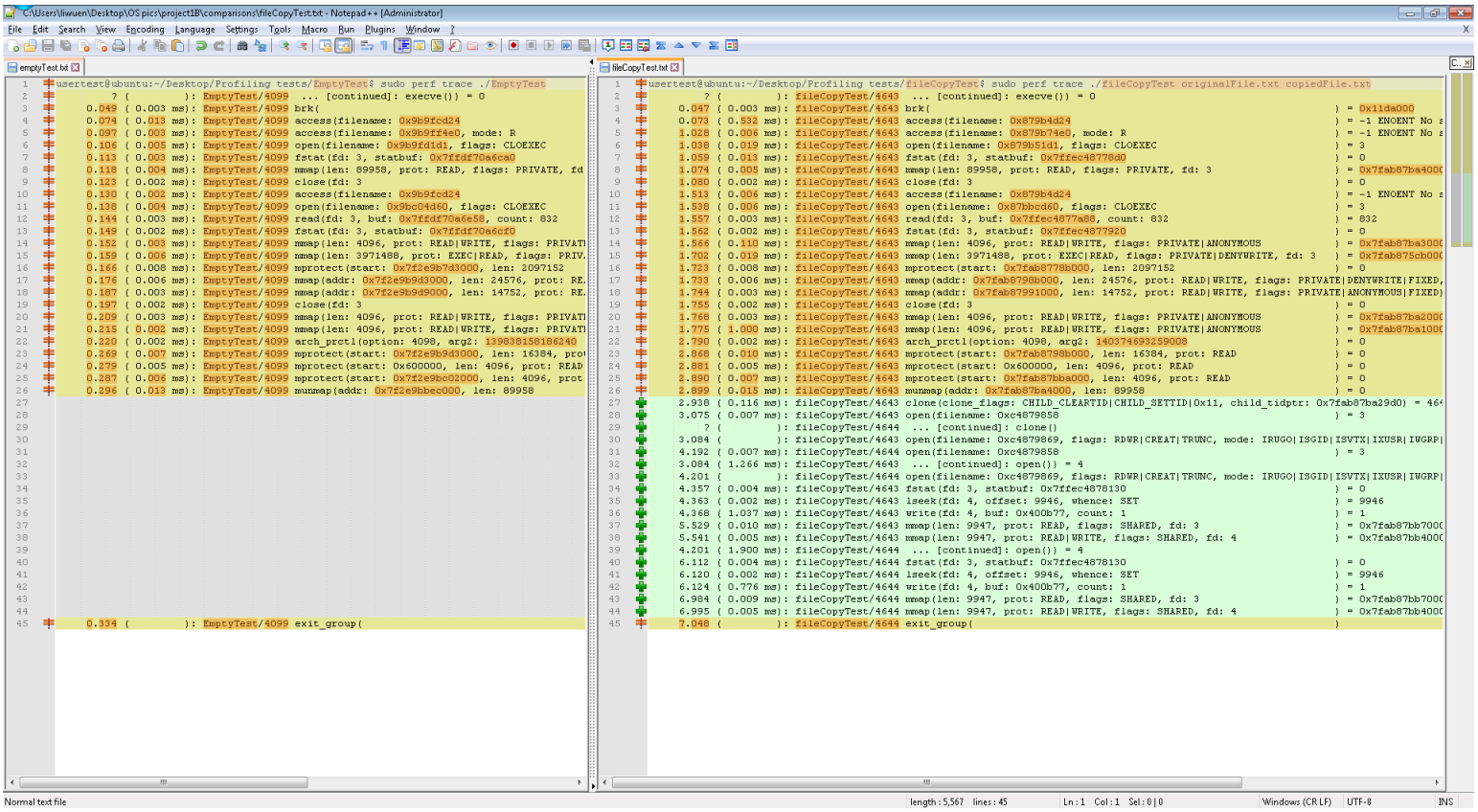
This program copies one text file into another file and uses **fork, mmap, write,** and **printf.**

7. Download the **originalFile.txt** from E3 and place it in the same folder as **fileCopyTest.c**.
8. Run the following commands

```
$ gcc -ggdb -w -g -o fileCopyTest fileCopyTest.c
$ sudo perf record -g ./fileCopyTest originalFile.txt copiedFile.txt
$ sudo perf trace ./fileCopyTest originalFile.txt copiedFile.txt
```

[Screenshot # 19: Create a screenshot showing these files and the trace results, as shown above. Your student ID must be visible.]

9. Using Notepad++, compare both files.



We used the Notepad++ **compare** plugin. In case you cannot install Notepad++ or wish to use another tool to display the differences between two text files, please feel free to do so, but be sure that the differences are clear enough. If you want to use Notepad++, here it is explained how to turn on the compare plugin:
http://www.technicaloverload.com/compare-two-files-using-notepad/

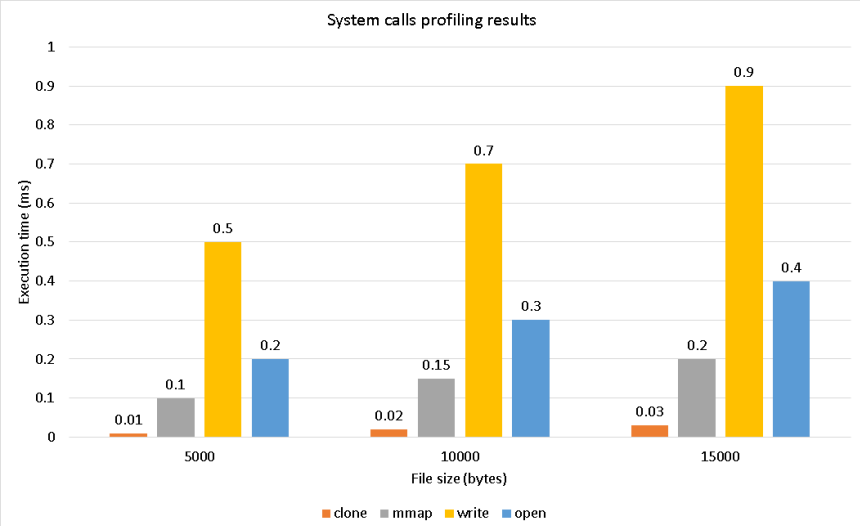
[Screenshot # 20: Create a screenshot showing the differences between these files.]

We can see that **fileCopyTest.txt** has some extra lines which are invocations of the clone, mmap, write and open system calls. By using these lines, we can calculate the average execution time of each function for this scenario.

Questions to answer (in both the report and video):

a. Will the functions' execution time be longer if the file is bigger?
b. How is the behavior of each function? Sort them from slowest to fastest.
   (Example -from fastest to slowest- : clone, mmap, open, write).
c. **Create a graph of file size (in bytes) vs. execution time (ms) of these four functions, using 3 different file sizes.**
   [Screenshot # 21: Create a graph using an input file with 5,000 – 10,000 and 15,000 bytes of size and group the results per file size and clearly show each function, as shown below].

   An example of the expected graph is shown below



(This is a dummy graph that only shows what is expected: **file size** vs. **execution time.** The real behavior of the functions is not reflected in this example.)

d. Perf also has the **report** command:

$ **sudo perf report**

   Explain:
   i. What is it for?
   ii. For **fileCopyTest**, show and interpret the results.

[Screenshot # 22: Take a screenshot of the report result you obtained, and give an interpretation to the results].

e. Perf has more commands (Section 5.1 step 4). Select another command **(besides report, trace and record)**, explain what is it for and show how to use it. Create your own scenario.

[Screenshots # 23 and #24: Take two screenshots of another command of the perf tool. Show how to use it and give an interpretation of the results. Create your own scenario on how to use it].