**CSCI 4401/5401**

**Principles of Operating Systems**

**Fall Semester 2022**

**Assignment 1 (8/16)**

**Due Sunday, September 4th @ 11:59pm**

**Reading:** Tanenbaum Chapter 1

**Submission Guidelines:**

1. This assignment is worth 100 points for all students.

2. All answers in the form of text (or short commands) and images should be added to a single PDF file named LastName\_FirstName.pdf.

3. Screenshots must use the PrintScreen command so that they are readable.

4. It is your responsibility to make sure all files are readable and submitted on time.

**Submission:**

- This assignment is broken down into five sections. Across these five sections are a total of 9 short answer questions (many of them ask you to submit a screenshot).

**Introduction**

In this assignment, you will study the system-call interface provided by the Linux OS and learn how user programs communicate with the OS kernel via this interface. Your task is to incorporate a new system call into the kernel, thereby expanding the functionality of the OS.

A user-mode procedure call is performed by passing arguments to the called procedure either on the stack or through registers, saving the current state and the value of the program counter, and jumping to the beginning of the code corresponding to the called procedure. The process continues to have the same privileges as before.

System calls appear as procedure calls to user programs but result in a change in execution context privileges. In Linux, a system call is accomplished by storing the system-call number into the EAX register, storing arguments to the system call in other hardware registers, and executing a trap instruction. After the trap is executed, the system-call number is used to index into a table of code pointers to obtain the starting address for the handler code implementing the system call. The process then jumps to this address, and the privileges of the process are switched from user to kernel mode. With the expanded privileges, the process can now execute kernel code, which may include privileged instructions that cannot be executed in user mode. The kernel code can then carry out requested services, such as interacting with I/O devices, and can perform process management and other activities that cannot be performed in user mode.

*Note:* I will use nano as the text editor, but you can use any editor (e.g., vim or emacs) you like. I also named my system call “helloworld”, but you can name your system call whatever you like.

**I. Setting up a Virtual Machine**

**a.** Log into Xen Orchestra using your UNO credentials.

<https://cyber-range.cs.uno.edu/>

**b.** After you log in, I will be able to add you to the class and you can set up your virtual machine. You may have to wait a day or two to be added so make sure you do this right away! You will not be able to continue until you added to the class, and I cannot do that until you log in for the first time.

**c.** On the left-hand side, select “New” -> “VM”.

**d.** Choose: Create a new VM on CSCI4401

**e.** Choose:

Template: “4401\_ubuntu\_server”

Name: “[last name]-[4401/5401]”

You can leave all of the other default settings (e.g., 1 CPU, 4 GB of RAM, 25 GB of disk storage).

Graphical user interface, application

Description automatically generated

**f.** Select “Create” at the bottom. It may take your VM several minutes to start up.

**g.** Go to the “Console” tab and login.

Graphical user interface, text, application

Description automatically generated

**Question 1.** Run **$echo “Hello from [your name]”**. Submit a screenshot of your running VM. Below is an example:

Graphical user interface, text, application

Description automatically generated

**II. Familiarizing Yourself with the Kernel and System Calls**

**a.** Look up your current kernel version with the following command. We will download and use another version later on so you can confirm that you successfully loaded the new kernel.

**$**uname -r

**b.** Run the following command. You should see a config file, System.map, and a vmlinuz file that correspond to your kernel version in the previous question. Later, when we install the new kernel, the (new) config file, System.map, and vmlinuz file will also be in this directory.

**$**ls /boot/

**Question 2.** What is your current kernel version? Hint: it should be in the format 5.x.x.

**c.** Locate your system call files by issuing the following command (note the hyphen at the end):

**$**printf SYS\_read | gcc -include sys/syscall.h -E –

**d.** The system call numbers for the recent version of the Linux kernel are listed in unistd\_64.h. Go to the relevant directory using the previous command (Step c) and issue the following command to look at the system call numbers.

**$**less unistd\_64.h

Text

Description automatically generated

For example, \_\_NR\_close corresponds to the system call close(), which is invoked for closing a file descriptor, and is defined as value 3.

**Question 3.** Submit a screenshot (similar to the example above) of at least ten of your system calls.

**III. Adding a System Call to the Linux Kernel**

**a.** Download and install the essential packages to compile kernels:

**$**sudo apt install git fakeroot build-essential libncurses-dev libssl-dev libelf-dev bison flex xz-utils -y

**b.** <https://cdn.kernel.org/pub/linux/kernel/v5.x/> contains different kernel versions you can download. Select a version to download that is different from your current version. For example, I chose version 5.16.13.

**$**wget <https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.16.13.tar.xz>

**c.** Unpack the tarball you just downloaded

**$**tar -xvf linux-5.16.13.tar.xz

**d.** Go to the directory for your source code (linux-5.16.13/). Create a home directory for your system call (e.g., I named mine “helloworld”).

**$**mkdir helloworld

**e.** Move into that directory (cd helloworld/), and create a C file that has the same name of your system call:

**$**nano helloworld.c

**f.** Write the following code in it, but replace my name with your name:

Graphical user interface, text, application

Description automatically generated

**Question 4.** Submit a screenshot of your system call code.

**g.** In the same directory, create a Makefile for your system call with the following command:

**$**nano Makefile

**h.** Type the following in it:

Graphical user interface, text, application

Description automatically generated

**i.** Next, you will add the home directory of your system call to the main Makefile of the kernel. To do this go back up to the directory for your source code. Search for “core-y” in the Makefile. You will want the second result (at line 1095 in my screenshot) that contains a series of directories (kernel/, certs/, mm/, etc.):

**$**grep -n 'core-y' Makefile

Text

Description automatically generated

**j.** Edit the Makefile by adding the directory of your system call. Note: directories end with a forward slash.

**$**nano Makefile

*Or* you can jump to the line number

**$**nano +[your line#] Makefile

Text

Description automatically generated

**Question 5.** Submit a screenshot of your modified Makefile.

**k.** Open the header file:

**$**nano include/linux/syscalls.h

**l.** Go to the bottom of the file and write the following code just above #endif.

asmlinkage long sys\_[name of your system call](void);Text

Description automatically generated

**m.** Next, you will add your system call to the kernel’s system call table (arch/x86/entry/syscalls/syscall\_64.tbl). Open the table with the following command:

**$**nano syscall\_64.tbl

**n.** Navigate to the bottom. My last system call has a number of 547 so I will use 548 (you should also use N+1).

Text

Description automatically generated

**Question 6.** Submit a screenshot of your syscall\_64.tbl that includes your system call.

**IV. Compiling and Installing the New Kernel**

**a.** Copy over the existing config file from the boot directory.

**$**cp /boot/config-$(uname -r) .config

**b.** Set the default configuration.

**$**make defconfig

**c.** Compile the kernel’s source code. This could take 30 - 60 minutes.

**$**sudo make

**d.** Install the kernel modules and install the kernel.

**$**sudo make modules\_install

**$**sudo make install

**e.** The new kernel config file, System.map, and vmlinuz file should be in the /boot/ directory.

**$**ls /boot/

Graphical user interface, text

Description automatically generated

**Question 7.** Submit a screenshot of your boot directory containing those files.

**f.** Run the following command to enable the new kernel for a boot.

**$**sudo update-initramfs -c -k 5.16.13

**g.** Update the bootloader GRUB.

**$**sudo update-grub

**h.** Reboot your system

**$**shutdown -r now

**i.** When you restart your system (while the console is still black), hit the button in the upper left just above the console that looks like a keyboard and says “Send Ctrl-Alt-Delete” when you hover over it. You should get a screen like the following:

Graphical user interface, text

Description automatically generated

**j.** Select “Advanced options for Ubuntu” and then select your new kernel version.

**k.** One your VM is finished starting up, verify your (new) kernel version.

**$**uname -r

**Question 8.** Submit a screenshot that verifies you are using your new kernel version.

**V. Testing the System Call**

**a.** Write a script that calls your system call. Below is an example:

**$**nano test.c

Text

Description automatically generated

**b.** Compile and run your script:

**$**gcc test.c -o test.o && ./test.o

**c.** If it successfully ran, it prints “System call sys\_hello returned 0”. We can check the kernel log buffer to see if our message is there. This will print many lines, but your message should be near the bottom.

**$**dmesg

Graphical user interface, text, application

Description automatically generated

**Question 9.** Submit a screenshot of your output from $dmesg containing the message from your system call.