As usual, this is also a group project.

3 people per group.

Email me (rahman@cs.fsu.edu) the groups no later than Sept 30.

For this project, we are using Linux kernel version ~~5.3.0~~ 4.19.75

Download the kernel from Canvas (In the Files section, under the folder "Project2_Kernel". It has tar.xz extension, so you have to untar it.

You need to do it in your own laptop.

If you already have a Linux OS installed (say, Ubuntu), you can install the kernel directly in it.

This may give you better performance in terms of speed.

However, we are going to modify the kernel in this project. And it’s very easy to corrupt the OS in the process, and you may not be able to boot.

A safer option is using a Virtual Machine.

Using a VM allows you to quickly recover from kernel errors by using a snapshot (Important, do take system snapshots at regular interval).

VirtualBox is a good virtual machine manager (“hypervisor”). You can use something else if you prefer.

Download a guest OS image (This is a good source: https://www.osboxes.org/virtualbox-images/)

Any version of Linux will do as the guest. I am personally using Ubuntu 18.04 right now (and the commands in the rest of this document will reflect that), but you can choose something else if you wish.

Create a new VM using the downloaded image as the hard-drive.

The more memory and CPU you can allocate to the VM, the faster the kernel will compile.

You may consider giving it half of the available memory, and half of the processor cores (if you have multiple cores).

You can configure the amount of processor and memory allocated to the guest OS from `machine -> settings` menu option in the VM Manger.

You may also want to enable “Shared clipboard”, “Drag/Drop” (set both of these to bidirectional), and set up a Shared Folder.

Shared clipboard will help you to copy and paste text from the VM and the host, and Shared Folder will enable you to easily transfer files between them.

For each of these to work, you need to enable the Guest Additions in VirtualBox. (Want to know how? Google is your friend.)

Shared clipboard and Shared Folder are not essential for this project, but they make life easier.

Ok, now let’s compile a kernel.

First, install some necessary packages:

```
$sudo apt-get update

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

<These is a space after build-esential>

For this project, we are using linux kernel version ~~5.3.0~~ 4.19.75

Download the kernel from Canvas (In the Files section, under the folder "Project2_Kernel".

Extract it in your home directory

Rename the directory to test_kernel (Optional, you can choose to keep it as linux-4.19.75)

Add a symbolic link to your kernel directory from `/usr/src`.

This will make our job easier later when we add our own kernel modules.

```
$sudo ln -s ~/test_kernel /usr/src
```

Now, let's compile the kernel.

Before that, check your own kernel version using `uname -r`

First, you need to configure which modules are to be included in the kernel and which not.

```
$make menuconfig
```

This will bring a graphical window and you can decide which modules to include.

You may save a lot of compiling time by removing unnecessary device drivers and file systems.

But, this is time-consuming, and potentially risky, given that you may also remove something important and end up with a broken kernel.

A safer alternative can be to use your existing old, working kernel.

```
$ cd /usr/src/test_kernel
$ cp .config .config_old
```

[This one backs up any older config file you already had in your test kernel. Chances are high that there will not be any, so the copy will not work. That's fine.]

```
$ cp /boot/config-5.0.0-23-generic .config
$ make oldconfig
```

Here, `config-5.0.0-23-generic` is my existing config. Yours can be named different. Check in your /boot directory.

`make oldconfig` is a utility that reads your existing kernel configuration.

If there is a configuration option in the test kernel that is not found in the existing config file, it prompts the user on what to do. In that case, it will have either Yes or No as default, just accept that.

This will set up a working configuration for you.

```
$ make menuconfig
```

Graphical configuration setup. You don’t really need to change anything.

Save.

Then exit.

```
$ make
```

Compiles the kernel. Depending on your system and configuration, can take HOURS!

If you have multiple cores in your machine / VM, parallelize it by

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

nproc gives the number of available cores. If your VM has 4 processing cores, this will be equivalent to:

```
$ make -j 4
```

Pay close attention to if there is an error. Sometimes there might be some error, but it may still finish compiling. This may cause problems later. You may consider redirecting the terminal output to a file and check that file later (when the compiling is complete) to see if there was any error.

```
$ sudo make modules_install
```

Install module binaries into modules.

Again, check for errors or warnings. There should not be any.

```
$ sudo make install
```

Installs final binary into /boot.

Manually examine the terminal output to see if there was any error. Also, check the output for any mention of `update-initramfs` and `grub` updating. Seeing these two means things are going as expected.

In Debian/Ubuntu, the following commands are optional as `make install` does everything for you. But if there was an error in updating the boot-loader, then you may have to do it manually.

```
$ sudo update-initramfs -c -k 4.19.75
$ sudo update-grub
```

Finally, reboot your system.

```
$ shutdown -r 0
```

At `make install` phase, if you see an error message like this: “`gzip: stdout: No space left on device`”, it means your `/boot` partition is full.

In that case, you need to delete some existing kernel images from the `/boot` partition and then try again.

You should be able to list your installed kernels with

```
$ aptitude search ~ilinux-image
```

You can remove the old kernels, one by one, like this:

```
$sudo apt-get autoremove linux-image-3.2.0-23-generic
```

Make sure you substitute "`3.2.0-23-generic`" with the actual kernel version you want to remove. In your case, it will be more likely to be `5.0.0-23` or `5.0.0-29`.

Once it boots up, you should be in the new kernel. Check it yourself.

```
$ uname -r
```

If this outputs 4.19.75, you are good to go.

If for some reason it still shows your old kernel (say, 5.0.0-23), then you need to manually update your `grub` file. Check the slides under “2 Grub file.pdf” in Canvas on how to do it.

If the kernel was successfully installed, do take a **snapshot** of your VM at this point.

You will inevitably break your kernel while working on the project, this snapshot will save your life then. You can just restore the system to this point then.