PA1 - Part A

New Attempt

Due Sunday by 11:59pm **Points** 25 **Submitting** a file upload
Available until Sep 5 at 11:59pm

Programming Assignment One - Part A

Introduction

Welcome to the first programming assignment for CSCI 3753 - Operating Systems. In this assignment we will go over how to compile and install a modern Linux kernel, as well as how to add a couple of custom system calls. Most importantly, you will gain experience in setting up an environment that you will use for future assignments.

You will need the following software: **Cloud VM**

All other software and files should be installed on this virtual machine image. It is important that you do not upgrade the virtual machine image as all the assignments are designed and written using the software versions that the VM is distributed with. Also make note that this assignment will require you to re-compile the kernel at least twice, and you can expect the initial compilation to take up 4 hours on older machines. The steps you need to complete this assignment include:

- Configuring GRUB
- Downloading Linux source code
- Compiling the kernel
- Adding a system call
- Writing a test program

Configuring GRUB

GRUB is the boot loader installed with Ubuntu 16.04. It provides configuration options to boot from a list of different kernels available on the machine. By default Ubuntu 16.04 suppresses much of the boot process from users; as a result, we will need to update our GRUB configuration to allow booting from multiple kernel versions and to recover from a corrupt installation. Perform the following:

From the command line, edit the GRUB configuration file

```
sudo vi /etc/default/grub
```

Feel free to use a different editor, if necessary. Make sure following lines are commented out in the configuration file

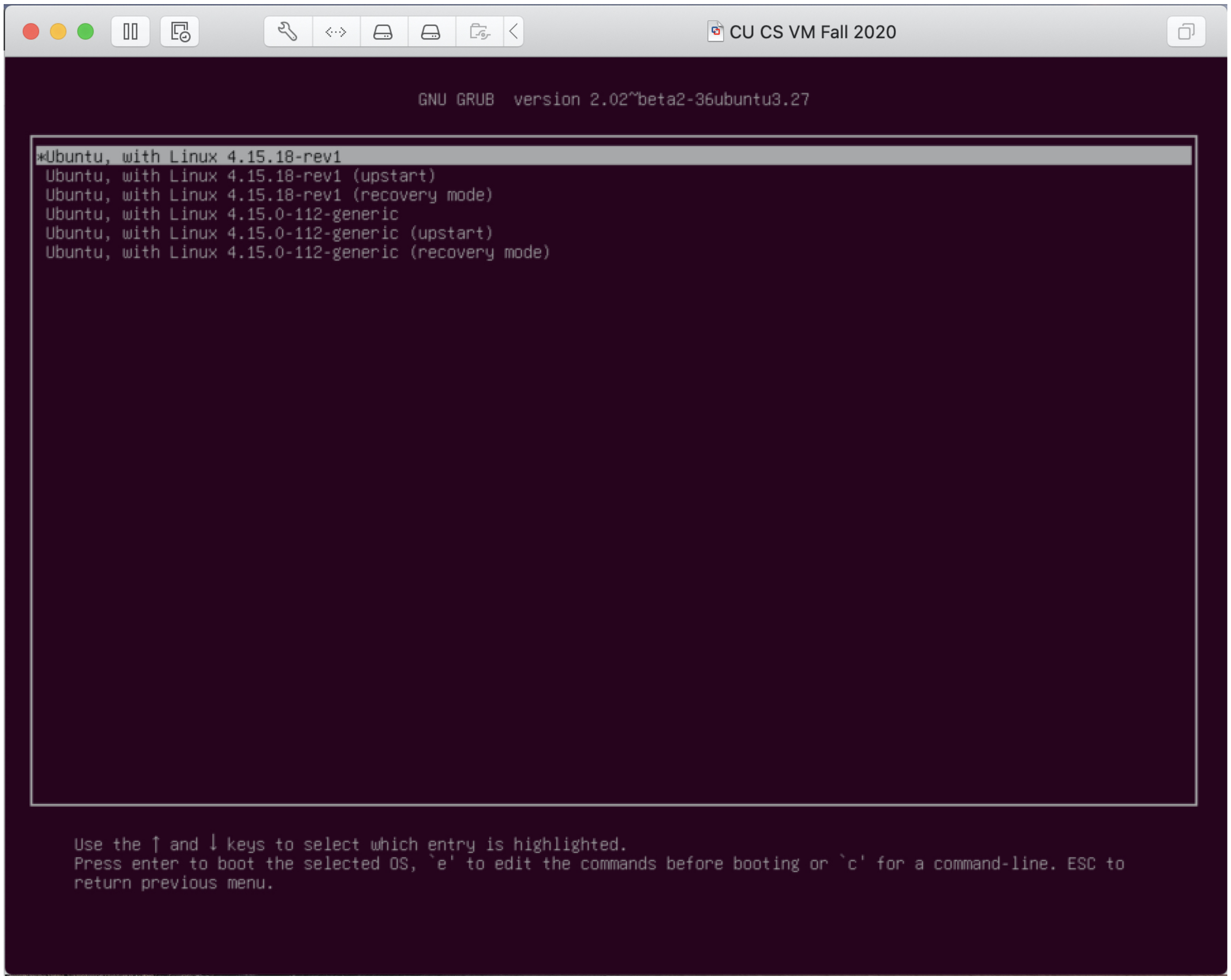
```
#GRUB_HIDDEN_TIMEOUT=0
#GRUB_HIDDEN_TIMEOUT_QUIET=true
#GRUB_TIMEOUT=0
```

Exit the editor and make sure to save your updates

From the command line, update GRUB and then reboot:

```
sudo update-grub
sudo reboot now
```

After rebooting your virtual machine, verify that you see a boot menu similar to Figure 1 (You might need to click first on *Advanced options for Ubuntu* to get to this screen):



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

*Ubuntu, with Linux 4.15.18-rev1
 Ubuntu, with Linux 4.15.18-rev1 (upstart)
 Ubuntu, with Linux 4.15.18-rev1 (recovery mode)
 Ubuntu, with Linux 4.15.0-112-generic
 Ubuntu, with Linux 4.15.0-112-generic (upstart)
 Ubuntu, with Linux 4.15.0-112-generic (recovery mode)

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, `e' to edit the commands before booting or `c' for a command-line. ESC to
return previous menu.
```

Figure 1: Linux GRUB Boot Menu

Once you've verified that you can run alternate versions of the kernel through grub, you're ready to move on to the next step.

Downloading Linux source code

First, you have to create a directory where you'll do the actual recompilation.   To do so, execute the following commands in a terminal window:

```
sudo mkdir /home/kernel
sudo chmod 777 /home/kernel
```

Before we can download the source files for the kernel, we need to enable the appropriate Ubuntu repositories:

```
sudo vi /etc/apt/sources.list
```

Make sure the following lines are **not** commented out:

```
deb-src http://us.archive.ubuntu.com/ubuntu xenial main restricted
deb-src http://us.archive.ubuntu.com/ubuntu xenial-updates main restricted
```

Once the source code repositories are enabled, run an apt update and retrieve the source code:

```
sudo apt update
cd /home/kernel
apt-get source linux-source-4.15.0
```

Compiling the Kernel

Once you have the Linux source code, but before you add your new system call, make sure you can compile the kernel. Typically it takes a long time to compile the kernel the first time. To make subsequent compiles run quicker, install the tool ccache:

```
sudo apt-get install ccache
```

ccache (https://ccache.dev/) is a compiler cache. It is used to speed up recompilation by caching previous results and detecting when the same compilation is being run again. The first compilation will take the usual time, but any recompilations will be much faster if you install ccache first.

The next step is to create the config file for your kernel build. Make sure you run this command in your recently created kernel source directory:

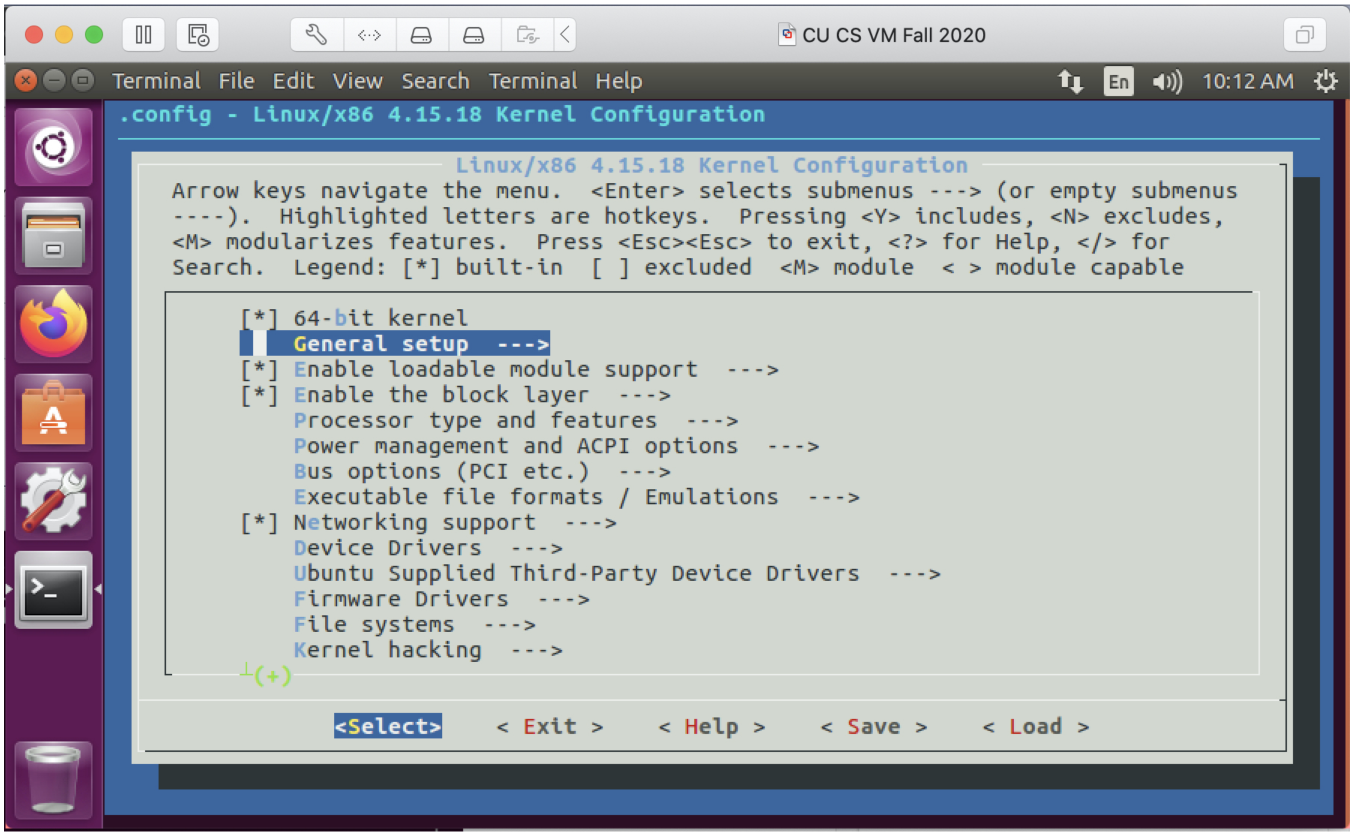
```
cd /home/kernel/linux-hwe-4.15.0
cp /boot/config-$(uname -r) .config
```

The command **uname –r** gives you the current version of the kernel you are using. This ensures you are compiling the kernel source for the same version of the kernel you're currently running on. The command **uname –a** will give you the system architecture information (for example if your OS is 32 or 64 bit).

In this same directory, now execute the command:

```
make menuconfig
```

You can navigate the configuration utility with the tab and arrow keys. Select the **General setup** option (Figure 2) and then select **append to kernel release** (Figure 3). Enter the name of the kernel you will be compiling (**-rev1**, for this assignment). This name will appear on the grub menu when you reboot. Exit out of the menuconfig utility by selecting <**Exit>**, possibly multiple times. Make sure you answer **<Yes>** when it asks if you want to save your changes.

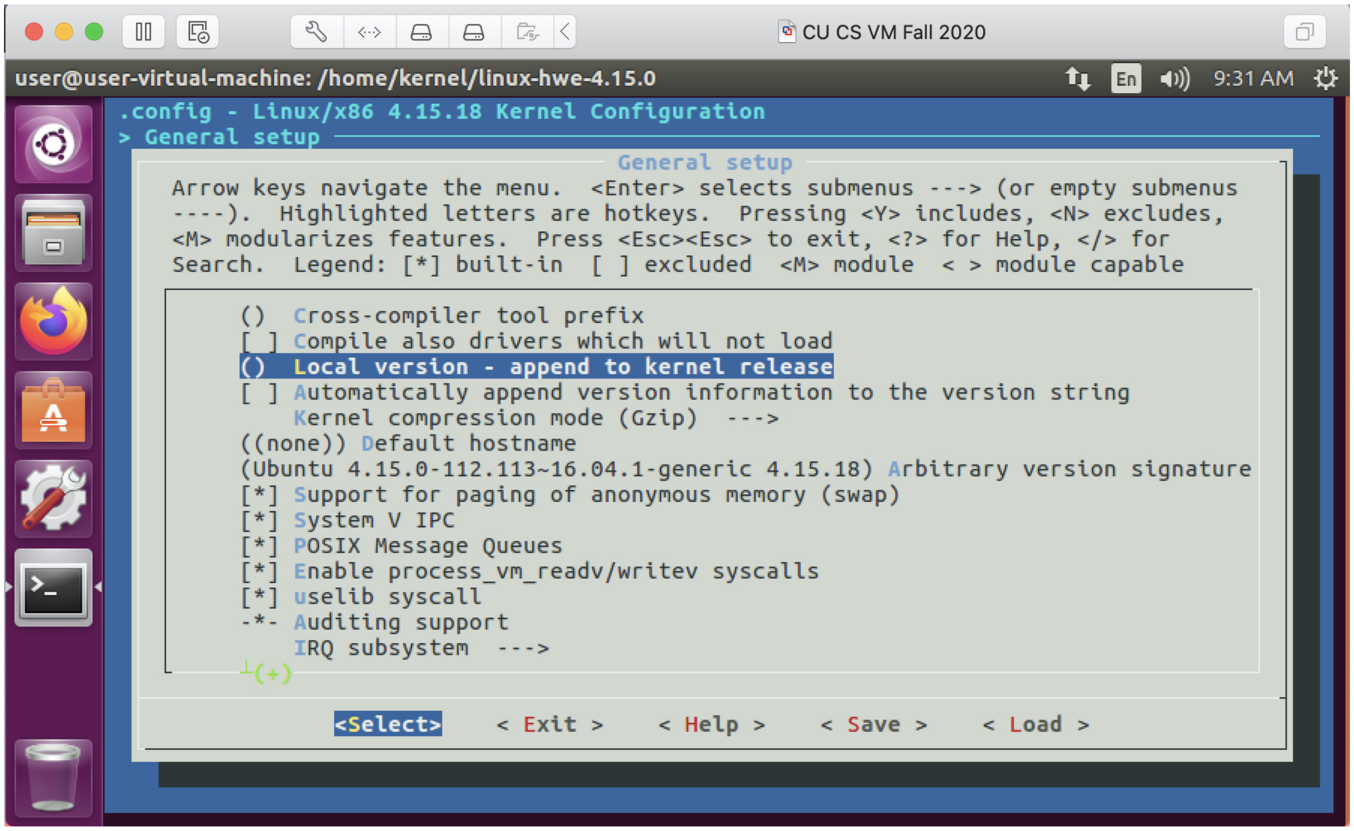


```
.config - Linux/x86 4.15.18 Kernel Configuration
                    Linux/x86 4.15.18 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for
  Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

      [*] 64-bit kernel
          General setup  --->
      [*] Enable loadable module support  --->
      [*] Enable the block layer  --->
          Processor type and features  --->
          Power management and ACPI options  --->
          Bus options (PCI etc.)  --->
          Executable file formats / Emulations  --->
      [*] Networking support  --->
          Device Drivers  --->
          Ubuntu Supplied Third-Party Device Drivers  --->
          Firmware Drivers  --->
          File systems  --->
          Kernel hacking  --->
      (+)

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 2: Configuring the kernel build with menuconfig



```
.config - Linux/x86 4.15.18 Kernel Configuration
> General setup
                              General setup
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for
  Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

       ()  Cross-compiler tool prefix
       [ ] Compile also drivers which will not load
       ()  Local version - append to kernel release
       [ ] Automatically append version information to the version string
           Kernel compression mode (Gzip)  --->
       ((none)) Default hostname
       (Ubuntu 4.15.0-112.113~16.04.1-generic 4.15.18) Arbitrary version signature
       [*] Support for paging of anonymous memory (swap)
       [*] System V IPC
       [*] POSIX Message Queues
       [*] Enable process_vm_readv/writev syscalls
       [*] uselib syscall
       -*- Auditing support
           IRQ subsystem  --->
       ↓(+)

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 3: Configuring the kernel build with menuconfig

Now compile and install the kernel by running the following commands. Remember, these steps could take a long time, so check that your computer doesn't go to sleep in the middle of building your new kernel.

```
cd /home/kernel/linux-hwe-4.15.0
sudo make -j2 CC="ccache gcc"
sudo make -j2 modules_install
sudo make -j2 install
```

Once you've compiled and installed your new kernel, reboot your VM:

```
sudo reboot now
```

When the GRUB option menu appears, select the version with your new kernel name appended. If you can boot into your newly compiled kernel, you're ready to move onto the next part of the assignment.

Adding a system call

In this part of the assignment, you will add a new system call, recompile the source and then run the new kernel with the added system call. To do this go to the directory where you downloaded the linux kernel source, **/home/kernel/linux-hwe-4.15.0**.

Step 1:

Create the file **./arch/x86/kernel/helloworld.c** and insert the following code:

```
#include <linux/kernel.h>
#include <linux/linkage.h>

asmlinkage long sys_helloworld(void)
{
  printk(KERN_ALERT "hello world\n");
  return 0;
}
```

Some notes on the above code:

The **kernel.h** header file contains many of the constants and functions used in kernel programming.

The **linkage.h** header file defines macros that are used to keep the stack safe and ordered.

The **asmlinkage** keyword tells the compiler that the function should not expect to find any of its arguments in registers (a common optimization), but only on the CPU's stack. It is defined in the **linkage.h** header file.

We have named our function **sys_helloworld()** because, by convention, all system call names start with the **sys_** prefix.

The function **printk()** is used to print out kernel messages, and here we are using it with the **KERN_ALERT** macro to print out "Hello World!". Note that depending on your system settings, these messages may not print to your console. Most messages get routed to the file **/var/log/syslog**. If the severity level is high enough, they will also print to the console. Look up printk() online or in the kernel docs for more information.

Step 2:

Now we have to tell the build system about our kernel call. Edit the file **./arch/x86/kernel/Makefile**. Inside you will see a host of lines that begin with obj+=. After the end of the definition list, add the following line (do not place it inside any special control statements in the file):

```
obj-y+=helloworld.o
```

Step 3:

Now you have to add that system call in the system table of the kernel. Edit **./arch/x86/entry/syscalls/syscall_64.tbl**. Look at the file and reference the existing entries to add the new system call. Make sure it is added in the 64 bit system call section and remember the system call number as you will be using that later.

Step 4:

Now you will add the new system call in the system call header file. Edit **./include/linux/syscalls.h** and add the prototype of your system call at the end of the file before the **endif** statement.

Step 5:

Now recompile, install and boot into the modified kernel using the same commands we executed earlier:

```
cd /home/kernel/linux-hwe_4.15.0
sudo make -j2 CC="ccache gcc"
sudo make -j2 modules_install
sudo make -j2 install
sudo reboot now
```

Even with ccache, this can take awhile. Increasing the -j parameter to 4 can sometimes speedup the compilation, but it's best not to go to more than ~2 times the number of CPUs allocated to your VM.

Writing a test program

Now that you have recompiled and rebooted into your modified kernel, you should be able to use the new system call. Write a test program in C (check the manpage for **syscall()** or use google) to see how to call a system call and what header files and arguments to use. The first argument a system call takes is the system call number we talked about before. If everything succeeds, a system call returns 0, otherwise it returns -1. Execute **sudo tail /var/log/syslog** or **dmesg** to check for the output.

Submission

The following file from **/home/kernel/linux-hwe-4.15.0** needs to be submitted to Canvas by the due date:

- ./arch/x86/kernel/helloworld.c
- We'll also confirm the operation of your new kernel and system call by looking at your Cloud VM.

References

https://help.ubuntu.com/community/Grub2 (https://help.ubuntu.com/community/Grub2)
https://help.ubuntu.com/community/Kernel/Compile (https://help.ubuntu.com/community/Kernel/Compile)
https://kernelnewbies.org/ (https://kernelnewbies.org/)