CSCI 3753: Operating Systems
Fall 2016

Programming Assignment One
Due Date and Time: 4:00 PM, Wednesday, September 07, 2016

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 custom system call. Most importantly you will gain skills and set up an environment that you will use in future assignments.

You will need the following software:

1. CSCI 3753 Fall 2016 Virtual Machine

All other software and files should be installed on the virtual machine image. It is important that you do not update 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 each compilation to take at least half an hour and up to 5 hours on older machines.

After installing virtualbox, run **sudo apt-get install cu-cs-csci-3753** to install all the necessary packages required for the course.

Assignment Steps

1. Configure the Grub
2. Compile the kernel
3. Add a system call
4. Write a test program that uses the system call

Configuring the Grub

Grub is the boot loader installed with Ubuntu 14.04. It provides configuration options to boot from a list of different kernels available on the machine. By default Ubuntu 14.04 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:

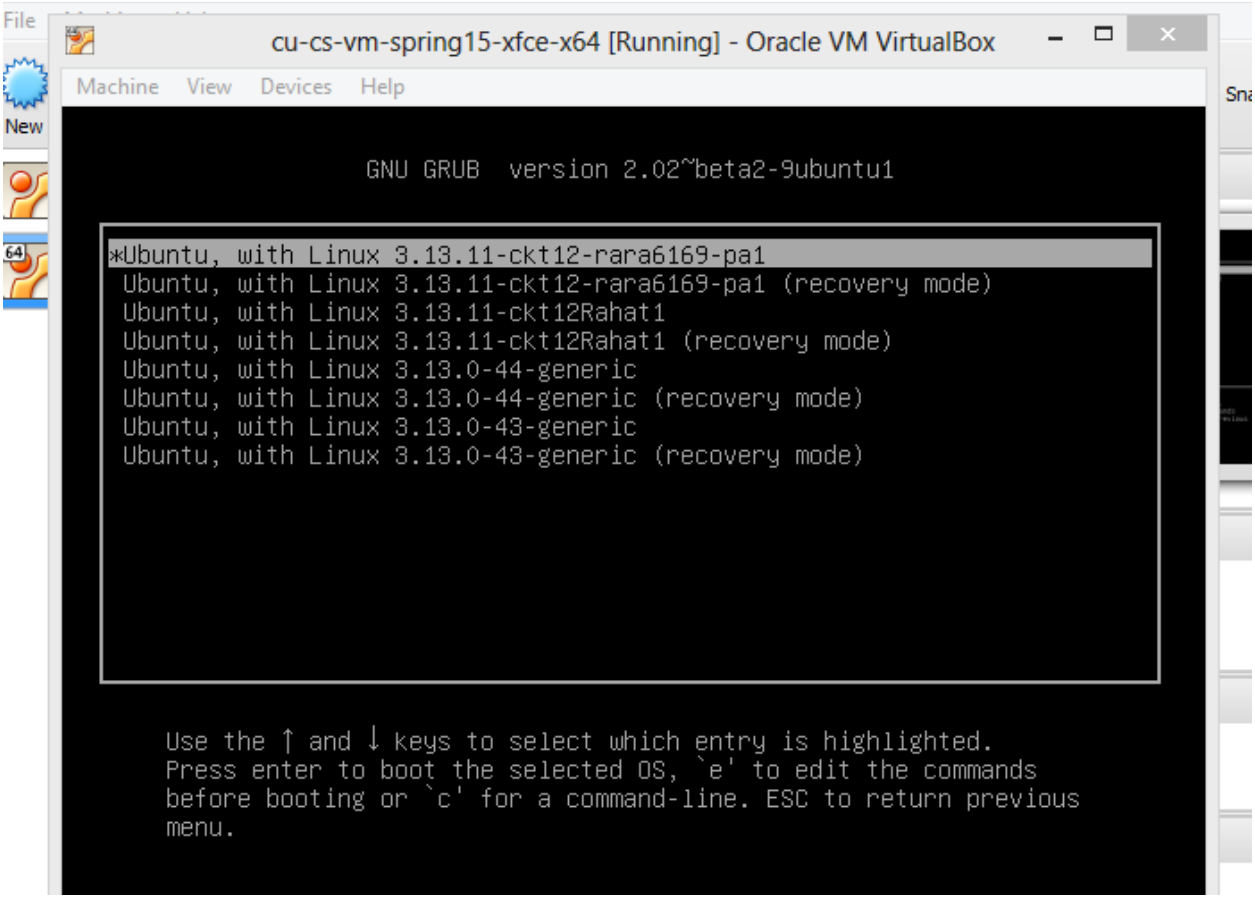

Figure 1: Linux Grub Boot Menu

Step 1:

From the command line, load the grub configuration file:
sudo emacs /etc/default/grub
(feel free to replace emacs with the editor of your choice).

Step 2:

Make the following changes to the configuration file:

1. Comment out:
   GRUB_HIDDEN_TIMEOUT=0
2. Comment out:
   GRUB_HIDDEN_TIMEOUT_QUIET=true
3. Comment out:
   GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
   and add the following new line directly below it:
   GRUB_CMDLINE_LINUX_DEFAULT=""
4. Save updates

Step 3:

From the command line, update Grub: **sudo update-grub**.

Step 4:

Reboot your virtual machine and verify you see a boot menu as shown in Figure 1.

Downloading Linux Source Code

First, you have to download the linux source code that you will be using for this assignment where you will be adding your new system call, compiling the source and then running the kernel with the added system call. For this execute the following commands in the terminal.

1. **cd /home**
2. **sudo mkdir kernel**
3. **cd kernel**
4. **sudo apt-get source linux-image-$(uname -r)**
5. **sudo apt-get install cu-cs-csci-3753** (command to install all the packages necessary for this course)

The command **uname –r** gives you the current version of the kernel you are using. So that means you are downloading the same kernel as your current kernel. The command **uname –a** will give you the system architecture information (for example if your OS is 32 or 64 bit).

Compiling the Kernel

Step 1:

Typically it takes a long time to compile (2-3) hours to compile the kernel source code. To get around this, install ccache by running the following command

sudo apt-get install ccache

ccache is a compiler cache. It is used to speed up recompilation by caching previous compilations and detecting when the same compilation is being done again. The first compilation will take the usual long time but after you make the necessary changes, the recompilations will be much faster after you install ccache.

Step 2:

Now you have to change permission of the debian scripts required to compile the source code. Just run the following commands in the linux source code directory.

sudo chmod a+x debian/scripts/*
sudo chmod a+x debian/scripts/misc/*

or alternatively you can execute all the commands using **sudo**.

Step 3:

The next step is to create the config file. **sudo make localmodconfig** command will create the config file. Run the command in the linux source directory you downloaded and extracted.

Step 4:

In the same folder, execute the command **sudo make menuconfig** . A pop up will be generated. Select the general set up (Figure 2) and then select the option append to the release (Figure 3) and enter the name you want to give to the kernel you will be compiling ,installing and running (for example revision1). This name will appear when you reboot and from the grub you can select it and check your new system call.

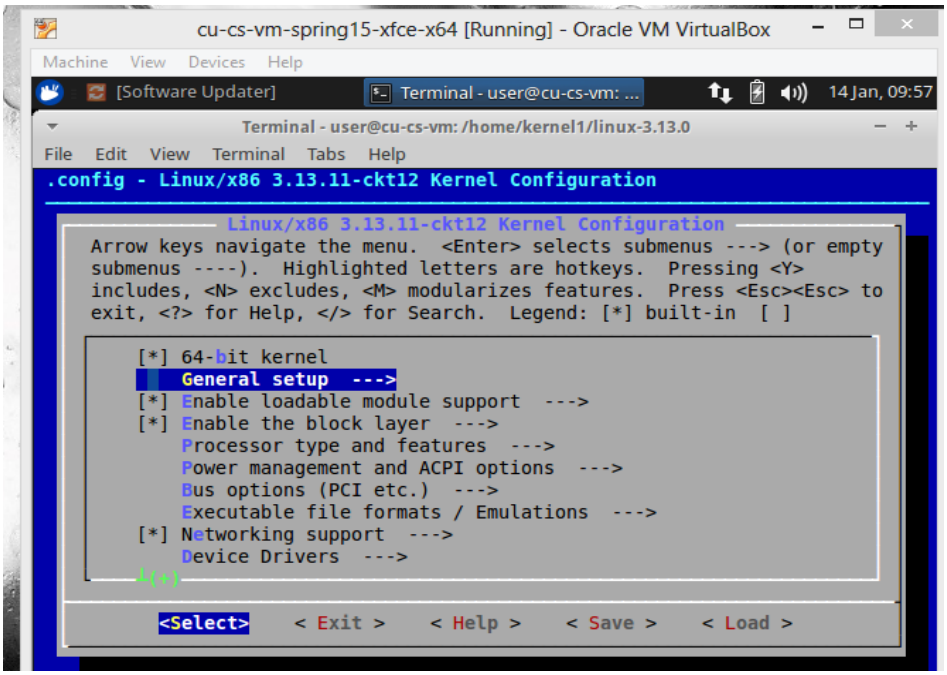

Figure 2: Configuring the menuconfig

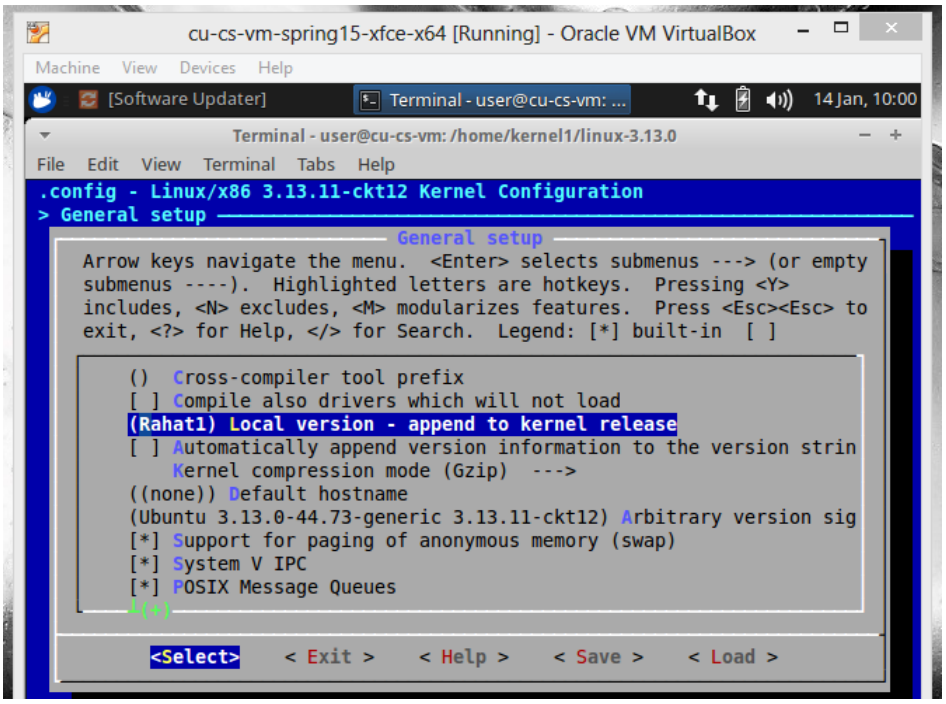

Figure 3: Configuring the menuconfig

Step 5:

Now just run the following commands.
sudo make -j2 CC="ccache gcc"
sudo make -j2 modules_install
sudo make -j2 install
sudo reboot

When the grub option shows up, select the version you have installed with the new name appended.

Adding System Call

Step 1:

Now you have to write the actual system call. First you will see that the linux source code is downloaded in the kernel folder. Go to that folder (for example linux-3.13.0) from the terminal. Then follow these steps.

1. **sudo gedit arch/x86/kernel/helloworld.c** and copy paste the following code snippet and save it

2.

```
#include <linux/kernel.h>
#include <linux/linkage.h>
asmlinkage long sys_helloworld(void)
{
  printk(KERN_ALERT "hello world\n");
  return 0;
}
```

Explanation:

1. Now kernel.h header file makes us able to use the many constants and functions that are used in kernel hacking, including the printk function.
2. Linkage.h defines macros that are used to keep the stack safe and ordered.
3. Asmlinkage is a #define for some gcc magic that 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
4. We have named the function sys_helloworld because all system calls' name start with sys_ prefix.
5. The function printk is used to print out kernel messages, and here we are using it with the KERN EMERG macro to print out "Hello World!" as if it were a kernel emergency. Note that depending on your system settings, printk message may not print to your terminal. Most of the time they only get printed to the kernel syslog (/var/log/syslog) system. If the severity level is high enough, they will also print to the terminal. 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. Open 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. Go to the directory **arch/x86/syscalls** and open the file syscall_64.tbl (if you are using a 32 bit system then syscall_32.tbl). Look at the file and use 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. Ask google if you find any trouble.

Step 4:

Now you will add the new system call in the system call header file. Go to the location **include/linux** and open the file **syscalls.h** and add the prototype of your system call at the end of the file before the endif. Check the file structure or google if you have trouble.

Step 5:

Now recompile the kernel using the instructions given in the previous section.

Step 6:

Now that you have recompiled the kernel and rebooted into the installed kernel, you will be able to use the system call. Write a test c program (check google or type man syscall) to see how to call a system call and what header files to include and what are the arguments. 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. Check sudo tail /var/log/syslog to or type dmesg to check the printk outputs.

Step 7:

If everything until now has been perfect, now you have to write a new system call named simple_add that will take two integer arguments namely number1, number2 and an integer pointer *result. You have to pass these arguments to the system call from your test program. In your system call implementation, you have to printk the numbers to be added, then you have to add those two numbers, store the result in the integer pointer result, printk the result and then print the result again in the test program that you will be running in the userspace. Do all the changes necessary from section 3's step 1 to step 6 again to test this new system call.

References:

1. Textbooks
2. https://help.ubuntu.com/community/Grub2
3. https://help.ubuntu.com/community/Kernel/Compile
4. http://kernelnewbies.org/
5. /kernels/linux-3.13.0/Documentation

Grading

1. 10 percent for compiling the first kernel
2. 10 percent for adding the necessary snippets for new system call
3. 10 percent for compiling with the new system call
4. 10 percent for the test program in userspace that will call the new system call,
5. The following files have to be submitted by the due date
    a. arch/x86/kernel/Simple_add.c
    b. arch/x86/kernel/Makefile
    c. arch/x86/syscalls/syscall_64.tbl
    d. include/linux/syscalls.h
    e. /var/log/syslog
    f. Source code for your test program
    g. A README with your contact information, information on what each file contains information on where each file should be located in a standard build tree, instructions for building and running your test program, and any other pertinent info.