CMSC 421: Principles of Operating Systems

Homework 1: Exploring the Linux Kernel

This homework is due on Thursday, September 13, at 11:59:59 PM (Eastern daylight time). You *must* use *submit* to turn in your homework like so:
`submit cs421_jtang hw1 greeting.out 0001-your_commit_message.patch platform-driver.txt hw1.c`

All of your homework assignments must run under 64-bit Linux, specifically Ubuntu 18.04 LTS ("Bionic Beaver"), and all projects will involve making changes to the Linux kernel. In addition, all later assignments **require** that you have a working Linux virtual machine. The projects **require** that you have finished customizing your Linux kernel. In this homework, you will become familiar with the procedures of installing and updating the kernel inside a VM.

> ***If you are unable to customize your kernel for this homework, you will get a zero on the later projects.***

This homework is divided into several parts that will help you perform the following objectives:

- Becoming familiar with the tools used to build a custom Linux kernel image.
- Installing Linux and building a custom kernel.
- Modifying the Linux kernel.
- Creating a *git commit*.
- Exploring the Linux operating system.
- Writing a C program from scratch.

Part 0: Hardware and Software Requirements

Flash Drives and USB Hard Drives

If you choose to work in the lab, you will need an external USB hard drive or flash drive for development as well as for backups. You will need to have at least 40 GiB of free space on the drive to store your VM image on it, so a 64 GiB or larger drive is **highly** recommended. Also, you will need to format the drive to filesystems that supports files of size > 4 GiB. Many flash drives come pre-formatted as FAT32, which do not support file sizes >= 4 GiB. **FAT32 will not work for this reason!** You should format your drive as either NTFS (if you will be using Windows with the drive as well) or a Linux filesystem such as ext4.

Linux on VirtualBox

In order to aid development of assignments in this course, we will be running the submissions under virtualization. Virtualization allows us to run a virtual machine on an actual physical machine. In this way, we can run a second *guest* operating system inside the regular *host* operating system. To do the assignments in this class, we assume you will have access to a relatively modern PC that can run VirtualBox. VirtualBox requires an x86/64 CPU with a decent amount of RAM, at least 4 GiB. In addition, your host CPU should support the x86 virtualization extensions (VT-x for Intel processors, or AMD-V for AMD processors). For more information about hardware and software requirements for VirtualBox, please consult the VirtualBox website. All assignments are expected to run in 64-bit mode on VirtualBox.

VirtualBox is available on the machines in the ITE 240 lab under both Linux and Windows. You can also download it for free from https://www.virtualbox.org/wiki/Downloads to run under your own Windows, Mac, or even Linux host operating system. This assignment was tested using VirtualBox version 5.2.16.

For the purposes of assignments in this course, we will be using the x86 (64-bit) version of the Ubuntu 18.04 Linux distribution. In addition, the custom kernels that are built in this class will be based on the Linux kernel version 4.17 (the latest stable kernel).

Part 1: Install Linux

Perform the following tasks to create the environment that will be used to complete assignments in this course. These instructions are based upon a WikiHow article.

1. Install VirtualBox on your computer. This particular assignment has been tested using version 5.2.
2. For optimal VirtualBox performance, you need to enable virtualization on your computer. The exact steps varies based by computer. See this general guide. (Tablets and convertibles may be incapable.)
3. Download the Ubuntu 18.04.1 LTS x86 LiveCD. Make sure you download the 64-bit version. You can also make a donation to Ubuntu, if you so desire.
4. Scan through the Ubuntu release notes. Probably a lot of it will be incomprehensible. These notes will make more sense as you gain more experience in your Linux career.
5. Create a Virtual Machine for your Ubuntu 18.04 installation.
    1. Using VirtualBox, create a new virtual machine. You must give it a name, such as **421VM**. Give ample memory (at least 2 GiB) and virtual hard disk space (at least 40 GiB).
    2. Your newly created virtual machine requires additional configuration. Set the number of processors to 2 (or more, if your computer is powerful), and increase video memory to 64 MiB.

3. Set the boot device to the Ubuntu ISO image you downloaded above. Power on the virtual machine to boot into the Ubuntu LiveCD.
4. Run the Ubuntu installer. This will take a while, as that the installer will download additional files from the Internet.
5. After installation, shut down your VM. You should now be able to run Ubuntu without using the LiveCD image.
6. Remove the LiveCD from the list of disks in the VM.
7. Reboot the VM, and ensure that it boots into Linux properly.

6. Use a web browser to ensure that you can connect to the Internet within your VM.
7. Open a Terminal by clicking on the icon in the bottom-left corner. In the search box, enter `terminal` to launch the Terminal program.
8. Update the packages installed on the VM by executing the following command in the Terminal:

   ```
   sudo apt-get update && sudo apt-get upgrade
   ```

   You will need to enter your password to run the operation as **root**.
9. After updating, reboot the VM, to ensure that all updates complete before proceeding. The kernel may be updated by `apt`, and you must reboot to have the kernel updates applied.
10. For this class, you will be compiling lots of C code. Install the required software packages, by running the following commands within the terminal:

    ```
    sudo apt-get install bison flex gcc g++ make libelf-dev libqt4-dev libncurses5-dev libssl-dev perl pkg-config
    sudo apt-get install emacs vim git indent
    ```

11. Install the *VirtualBox Guest Additions*. While running your Linux VM, choose *Insert Guest Additions CD Image* from the Devices menu. Run the installer and reboot for a third time. You can now do things like resize the virtual monitor.

    Be aware you may need to reinstall the Guest Additions each time you update the kernel for this class.

    ---

12. Consider periodically making snapshots of your virtual machine upon completion of each part of this homework.

Many of the commands that you will be running within the VM will require root privileges. There are a variety of methods to elevate your user privileges on Linux. You can use any of the following methods to do so:

```
sudo -s
  (enter your user password when prompted)
  (perform any commands to execute as root)
exit
```

OR

```
sudo sh
  (enter your user password when prompted)
  (perform any commands to execute as root)
exit
```

OR

```
sudo (command to execute as root)
```

The instructions in this and all future assignments will explicitly specify when `sudo` is needed. If it is not needed, do not use the command. Arbitrarily using `sudo` will not magically fix any issues.

Part 2: Obtain Linux Kernel Source Code

Ubuntu 18.04 ships with a 4.15 Linux kernel. As that this class involves learning how operating systems really work, you will upgrade the kernel to a newer version. Follow these steps to obtain the Linux kernel source files:

1. Obtain the Linux kernel source files and unpack them.
   1. Run the following command:

      ```
      git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git linux
      ```

      This last command fetches the Linux source code repository, via `git`, and writes the files into the new `linux` directory. If you are not familiar with the `git` version control system, you should work through the Interactive Git Tutorial before proceeding. If you have already cloned the repository, delete the old repository prior to cloning again.
   2. Switch to the kernel *4.17 branch* by running the following commands:

      ```
      cd linux
      git checkout linux-4.17.y
      ```

      For the rest of this assignment, the sources in the `linux` directory will be referred to as your **working copy** of the kernel, or alternatively, **Linux kernel repository**. If you ever need the original unmodified code, re-run the `clone` command in a different directory; advanced `git` users may also want to create their own branches within the repository.
2. Later assignments require custom kernel code. Follow these instructions to apply those changes to your copy of the kernel.
   1. Configure `git` to allow patching. Run the following, substituting your information:

      ```
      git config --global user.email "gburdell@umbc.edu"
      git config --global user.name "George Burdell"
      ```

      (Note the above is *dash*, *dash*, *"global"*. You will need to type the commands by hand, instead of copying and pasting it.)
   2. Within your working copy of the kernel, run the following commands to obtain *kernel patches*:

      ```
      wget https://www.csee.umbc.edu/~jtang/cs421.f18/homework/hw1/0001-x86-irq-Allow-for-software-triggered-IRQ.patch
      wget https://www.csee.umbc.edu/~jtang/cs421.f18/homework/hw1/0002-x86-Create-kernel-defconfig-for-CS421-VirtualBox-VMs.patch
      ```

3. Apply those patches using `git`:

```
git am 0001-x86-irq-Allow-for-software-triggered-IRQ.patch
git am 0002-x86-Create-kernel-defconfig-for-CS421-VirtualBox-VMs.patch
```

To repeat, none of the commands in this section are to be run under `sudo`. Do not use `sudo` when running any commands in this section, otherwise, you will need to restart Part 2.

Part 3: Customize Your Kernel

Next, you will perform a simple modification of the official Linux kernel. At startup, your custom kernel will write a message to the *kernel log* using your name. If you have followed all of the instructions up to this point, you should have no trouble doing this.

1. First, view the kernel log:

   ```
   dmesg
   ```

2. The log is rather long. More specifically, view only the first several lines of the log:

   ```
   dmesg | head
   ```

Note the line that begins with `Command line:`. You will change the kernel to display a greeting similar to this:

```
[    0.000000] Linux version 4.17.17+ (tang@CMSC421) (gcc version 7.3.0 (Ubuntu 7.3.0-16ubuntu3)) #2 SMP Sat Aug 18 18:50:52 EDT 2018
[    0.000000] This is jtang@umbc.edu kernel!
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.17.17+ root=UUID=0b48296a-f662-44fb-a8d3-ffe22a81b6d5 ro quiet splash
```

Emphasis added. Change your kernel to display your UMBC email address.

- **Hint 1**: You will have to change at most one file that is distributed with the kernel.
- **Hint 2**: Use `grep` to find the file that logs the command line.
- **Hint 3**: Read how the `printk()` function works.

To begin, you need to compile the kernel and accompanying modules as is.

1. Use the CS421-supplied kernel configuration as a basis of your configuration. Run the following commands:

   ```
   make mrproper
   make cs421_defconfig
   make clean
   ```

2. Now compile and install the kernel. This step will likely take quite a while, maybe an hour on an older computer. Run the following:

   ```
   make -j3
   sudo make modules_install install
   ```

3. While the kernel is compiling, read the book Linux Kernel in a Nutshell.
4. Your working copy of the kernel should now be built and installed. In addition, the install process should have updated the bootloader (`grub`) to boot automatically into your new kernel.
5. Your Ubuntu VM now has two kernels that `grub` can use during booting. By default, `grub` will always select your new kernel. You will need the ability to select the original Ubuntu kernel, in case your customization failed. **As root**, edit `/etc/default/grub`. Change the `GRUB_TIMEOUT_STYLE` from `hidden` to `menu`. Save and quit, then run `update-grub` as root, to update `grub`'s configuration. See the grub documentation for details.
6. Reboot your VM. At the `grub` menu, select the second option, and then your 4.17 kernel. If you do not see it, choose the original Ubuntu 4.15 kernel, then restart at step 1.
7. Make sure your custom kernel boots properly in the VM. Run the following command to display the currently running kernel version:

   ```
   uname -a
   ```

8. Do not forget to reinstall VirtualBox Guest Additions after upgrading your kernel.
9. Now that you have compiled and run an updated kernel, edit the one file to display your greeting. Recompile your kernel starting with step 2 above, then reboot and check your kernel log.
10. Prepare the first part of your homework submission using these commands:

    ```
    mkdir -p $HOME/hw1
    dmesg | head > $HOME/hw1/greeting.out
    ```

Only steps 2 and 5 in this part requires `sudo`. To reiterate a third time, do not use `sudo` when running any other command. Otherwise, you will need to restart at Part 2.

Part 4: Handle Kernel Patches

Your next objective is to learn about *patches* and *diffs*. Since many modifications to the kernel are small (compared to the total kernel source size), such updates are usually distributed in the form of a *patch file*; those patch files describe differences between the original software and the modified code. If you make minor modifications to the kernel (such as for CS421 projects) you will want to create patches containing the *diffs*. Later projects may distribute minor kernel changes in the form of *patches*.

1. Creating diffs is easy with `git`. A good tutorial can be found at https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/diffs.
2. To view the diff you made so far, run the command `git diff`.
3. Next, you will make a *git commit*.
    1. Run `git add` *`filename`*, where *filename* is the name of the file you changed.
    2. Run `git commit`. Using the editor, add a one-line commit message, describing your change. Blank lines and lines beginning with `#` will be ignored. Save and exit.
4. Export your commit like so:

    ```
    git format-patch HEAD^1 -o $HOME/hw1
    ```

5. Now that you have created a patch file, you will apply another patch to your kernel source. Using `wget`, fetch the file https://www.csee.umbc.edu/~jtang/cs421.f18/homework/hw1/0003-x86-Add-handler-for-CS421-HW1.patch.
6. Use `git am` to apply **only** the third patch.
7. Run `make xconfig`. On the top toolbar, click on the last icon to enter *Full View*. In the left panel, navigate to *Device Drivers*, and then expand *X86 Platform Specific Device Drivers*. The above patch added a new so-called *platform device driver* to your Linux source code. It will have a very obvious name. Click on the driver to get more details about that driver.
8. Ensure that the new driver was selected. Quit out of the graphical configuration tool, and save the new configuration. Recompile and install the kernel starting at Part 3 Step 2 above, so as to enable the driver. Reboot your VM.
9. Use your favorite search engine to read what the Linux *virtual file* `/proc/interrupts` does. Examine your `/proc/interrupts` after the reboot.
10. Using your favorite text editor, create the file `$HOME/hw1/platform-driver.txt`. In that file, give the following:
    1. Summary and a brief description of `/proc/interrupts`. This should be about 200 characters or so.
    2. Name of the new platform driver, as given in `/proc/interrupts`.
    3. How this platform driver affected `/proc/interrupts`. This should also be about 100 characters.

For a fourth time, none of the commands in this section requires `sudo`. Doing so will result in your git repository being in an inconsistent state, and you will need to restart at Part 2.

Part 5: My First C Program

Now that you have a working Linux virtual machine, it is time for some programming. View the two virtual files `/proc/ioports` and `/proc/iomem`:

```
cat /proc/ioports
cat /proc/iomem
```

I/O Ports are an archaic system on x86 processors to communicate with peripherals, while I/O Mem is used for modern for memory-mapped devices. In Linux, non-privilege access to these ports only show a device's existence, and not the addresses themselves. Thus, try running the above `cat` commands with and without `sudo`.

Write a program, in C, that takes a single command line argument, a hexadecimal address. Open both files `ioports` and `iomem`. Display all lines whose address range encompasses the passed in value. If no lines contain the device address, display a message.

If no device is given, or if the user gives multiple arguments, or if a non-hexadecimal value is given, then display an error message and quit.

SPECIAL RESTRICTION: As that this is an operating system class, you will learn how the computer actually prints things. For this assignment, you **MAY NOT** use the function `printf()` anywhere within your program. You must find an alternative to displaying strings.

Your program **must** be called `hw1.c`, and it will be compiled on Ubuntu 18.04 as follows:

```
  gcc --std=c99 -Wall -O2 -o hw1 hw1.c
```

(Note the above is *dash*, *dash*, *"std=c99"*, and the other flags likewise are preceded by dashes.) There must not be any compilation warnings in your submission; **warnings will result in grading penalties**. In addition, each code file must be properly indented and have a file header comment, as described on the coding conventions page.

Here is a sample output when run in the VM. Your values may differ. The user need not precede the given address with `0x`.

```
$ ./hw1 1000
ioports:
No devices found.
iomem:
No devices found.
$ sudo ./hw1 0170
ioports:
  0000-0cf7 : PCI Bus 0000:00
    0170-0177 : 0000:00:01.1
      0170-0177 : ata_piix
iomem:
  00000000-00000fff : Reserved
$ sudo ./hw1 0xd00
ioports:
  0d00-ffff : PCI Bus 0000:00
iomem:
  00000000-00000fff : Reserved
$ sudo ./hw1 fee00123
ioports:
No devices found.
iomem:
  fee00000-fee00fff : Local APIC
    fee00000-fee00fff : Reserved
$ sudo ./hw1 1000
```

```
ioports:
  0d00-ffff : PCI Bus 0000:00
iomem:
  00001000-0009fbff : System RAM
$ ./hw1
No address given.
$ ./hw1 abcd 1234
Multiple addresses given.
$ ./hw1 abcdefg
Invalid target address.
```

Other Hints and Notes

- Ask plenty of questions on the Blackboard discussion board.
- At the top of your submitted files, list any help you received as well as web pages you consulted.
- Make sure you submit `greeting.out`, `0001-`*`your_commit_message`*`.patch`, `platform-driver.txt`, and `hw1.c` to receive full credit for this assignment.
- The function `strtoul()` may be useful for Part 5.

Extra Credit

Sorry, there is no extra credit available for this assignment.