Principles of Operating Systems

CMSC 421 — Spring 2018

Project 0

Due by 11:59 PM on Sunday, February 18th

Introduction/Objectives

All of the programming projects in this course will require making changes to the Linux kernel. In addition, all projects will be performed inside of a Virtual Machine for various reasons (uniformity of results, for instance). In order to do the later projects in this course, you must become familiar with the prospects of compiling the kernel inside the VM and creating updates to submit. Every project this semester will involve these steps, and this project is no exception. Project 0 is a very simple project that is designed to familiarize you with the procedures of installing and updating the kernel inside a VM.

In this project, you will build a custom version of the Linux kernel with a modified version string as well as an additional system call. You will also utilize the Git version control system for managing the source code of the project.

This project 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.
- Becoming familiar with the implementation of Linux System Calls.
- Becoming familiar with the process of calling system calls from userspace.
- Using Git for source code revision control.
- Formatting and submitting projects for this course.

Software and Hardware Requirements

Linux on VirtualBox

In order to aid development of the projects in this course, we will be running the projects 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 projects in this class, we assume you will have access to a relatively modern PC that can run the VirtualBox VM. VirtualBox requires an x86 CPU with a decent amount of RAM. In addition, it would be advantageous if the CPU on your host machine supported 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 projects are expected to run in**64-bit mode** on VirtualBox.

VirtualBox is available on the machines in the ITE 240 lab running Linux.

For the purposes of projects in this course, we will be using the amd64 (64-bit) version of the Debian 9.3 Linux distribution. In addition, the custom kernels that will be built in this course will be based on the Linux Kernel version 4.15.

Hard Drives

You must have an external USB hard drive or flash drive if you wish to work in the lab. Even if you do not intend to work in the lab, it would be very useful to have an external storage device for storing backups of your VM image. You will need to have around 32GiB of free space on the device to store one copy of your VM image on it. Also, You will need to format the drive as some type of filesystem that supports files of size > 4GiB. Many flash drives come pre-formatted as FAT32, which does not support files with a size >= 4GiB. **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 ext3 or ext4.

Even if you do not wish to work in the lab, you will still want to make regular backups of your VM images. It is recommended to do so on an external hard drive or flash drive, since your VM images will be relatively large in size and it is always a good idea to store your backups in a different place than where you are storing the data being backed up.

Installing Linux

Perform the following tasks to create the environment that will be used to complete the projects in this course:

1. Install VirtualBox on your computer.
2. Create a Virtual Machine and install Debian into it (see below for some tips while building the VM).

When creating your VM, you should select that you will be running the 64-bit version of Debian Linux in the new VM wizard. Also, when creating your VM, you should increase the disk size from the default to 24-32 GiB. This should give you enough space to build the kernel properly, assuming you don't install a bunch of software within the VM. You should also be sure to give your VM enough RAM to work with while not hampering the rest of your PC. 1GiB of RAM should be plenty, but be sure to not give your VM more than 1/4th the total amount of RAM you have installed in your computer.

You should get the Debian network install CD image from this link to ensure that you will be installing the correct version of Debian.

When installing Debian within the VM, you can use either the normal install or graphical install options -- either is fine. You must be sure to use the 64-bit installation, however (which will be the case if you downloaded the iso from the link in the last paragraph). When you get to the portion about partitioning, you should select the guided partition option and the option to create one large partition on the disk. Finally, when installing packages later in the installation, be sure to select at least one graphical environment from the list if you want a GUI. I personally suggest using Cinnamon or MATE, as they are both relatively lightwieght and will ensure you have ample space to work with later on your virtual hard drive.

Creating an SSH key

In order to work on repositories hosted on GitHub, you will need to have an ssh key set up on your VM and registered on your GitHub account. To do so, run the following command in a terminal in your VM (replacing the `yourumbcemail` part as appropriate):

```
ssh-keygen -t rsa -b 4096 -C "yourumbcemail@umbc.edu"
```

While running this command, you can accept the defaults for just about everything. However, for security, you should provide a passphrase for the key when it asks. This passphrase can be just about anything, but you must make sure to not forget it. You will need your passphrase any time you access resources in your GitHub account via the `git` command line tool.

Now that you've created your key, you must associate that key with your GitHub account. To do so, open up the web browser (it may be either Iceweasel or Firefox, depending on what options you installed at setup time) inside your VM, and go to github.com. Upon logging in with your GitHub credentials, visit the SSH settings section of your GitHub user settings. Use the "New SSH key" option, give the key a title ("421 VM Key" works nicely, if you can't think of anything better), and then run the following command in a terminal and carefully copy/paste the output into the Key box on the page, and hit the Add SSH key button.

```
cat ~/.ssh/id_rsa.pub
```

Obtaining `root` Privileges

Many of the commands that you will be running on the Linux installation within the VM will require `root` privileges. There are a variety of methods to elevate your user privileges on Linux. From a regular terminal, you can use `su` to do so:

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

If only one command needs to be run as root, you can alternatively do things this way (assuming the command to be run is simply `command`):

```
su -c "command"
(enter your root password when prompted)
```

It is very important that you pay careful attention to what steps should be run as root. In these instructions, we specifically mention which steps should be run as root. Anything else should not be run in a root shell, and should be run as your normal user account. If you fail to do so, you will have problems with your build.

Building a Custom Linux Kernel

The next part of this project is to obtain the Linux kernel sources and build a customized kernel. Follow this set of steps to do that:

1. Update all packages on your installation.
   In a terminal, **as root**, run the following commands:

   ```
   apt-get update
   apt-get upgrade
   apt-get dist-upgrade
   reboot
   ```

   Wait for the VM to reboot itself.
2. Install the required software packages used for kernel development.
   A. In a terminal, **as root**, install the required packages with the following command:

      ```
      apt-get install build-essential qt-sdk nano patch diffutils curl fakeroot git pkg-config gedit libssl-dev libncurses5-dev libelf-dev
      ```

3. Obtain the Linux Kernel sources and unpack them into the appropriate directory.
   A. In a terminal, **as root**, run the following commands:

      ```
      cd /usr/src
      chmod 777 .
      ```

B. In a terminal, run the following commands as your normal user (not as root). Replace yourusername with your github username, yourumbcemail with your UMBC email account, and make sure to put your full name where suggested.

```
cd /usr/src
git config --global user.email yourumbcemail@umbc.edu
git config --global user.name "Your Full Name"
git clone git@github.com:UMBC-CMSC421-SP2018/project0-yourusername vanilla-project0
cp -ra vanilla-project0 project0
```

C. For the rest of this project, the sources in the directory /usr/src/project0 will be referred to as the **working copy** of the kernel. The sources in the directory /usr/src/vanilla-project0 will be referred to as the **vanilla copy** of the kernel. **Never make any modifications to the vanilla copy of the kernel! They are kept here in case you mess something up!**.

4. Build and install your customized kernel.
   A. Read the Kernel Rebuild Guide. We will be using Linux 4.15 for this project, which is handled very similarly to Linux 2.6.x. **You should disregard the building instructions in the Kernel Rebuild Guide, as you will be following the instructions in this project description instead!**
   B. You may wish to review Linux Kernel in a Nutshell as well.
   C. Give your working copy of the kernel a *unique* version string of the form 4.15.0-cmsc421project0-USERNAME . Where USERNAME is your **UMBC** username (your UMBC email username -- **NOT your GitHub username, if they are different**). This is to ensure that our customized kernel does not interfere with the modules or other files of any kernels installed by the Debian package manager. If you have followed all of the instructions up to this point, you should have no trouble doing this. **You must change at least one file in the kernel source code to do this for this project!**
   As an example, if I were doing the project, this would be what should be printed out by uname -a (the hostname on my VM is akagi-vm):
   ```
   Linux akagi-vm 4.15.0-cmsc421project0-lsebald1+ #1 SMP Tue Feb 6 03:13:37 EST 2018 x86_64 GNU/Linux
   ```
   Your version string must be all lowercase, otherwise you will not be able to build the kernel properly for Debian.
   D. Add a "Hello World" system call to the kernel, performing the system call implementation and kernel modification steps in this tutorial. You should be sure to run the test program and verify that the system call actually was able to run successfully **only after you build, install, and reboot into your new kernel**.
   E. Configure and compile the custom kernel and its modules. This step will likely take quite a while. Run the following commands in a terminal:

   ```
   cd /usr/src/project0
   make mrproper
   make xconfig
   make localmodconfig
   make KDEB_PKGVERSION=0 bindeb-pkg
   ```

   During the make xconfig step, you should go into the Library routines section of the configuration and make the "CRC32c CRC algorithm" or "CRC32c (Castagnoli, et all) Cyclic Redundancy-Check" option be built-in to the kernel instead of being built as a module (make it so it has a checkmark, not a dot in the box next to it). Once you've done that, hit the floppy disk icon to save the configuration. If the make localmodconfig step prompts you at any point, you can just hit Enter to accept the defaults.
   F. Now that the kernel has been built, you must install the kernel image. To do so, run the following command, **as root** from the /usr/src directory (substituting your UMBC username as appropriate, of course):

   ```
   dpkg -i linux-image-4.15.0-cmsc421project0-USERNAME+_0_amd64.deb
   ```

   G. Your working copy of the kernel should now be built and installed. In addition, the install process should have updated your bootloader configuration to add the new entry to it (and made it the default kernel option too).
   H. Reboot your VM, and select your customized kernel from the GRUB boot menu
   I. Make sure your custom kernel boots properly in the VM. Run the following command to check the customized kernel:
   ```
   uname -a
   ```
   J. Verify that the output from the uname command shows the correct version string, as specified earlier in the project.
   K. If your the output from the last command does not match the version string format provided, try again. Note that this will change the #1 in the version string (right before the date that the kernel was compiled on). This is not important and will not affect your grade in any way. Generally, when rebuilding the kernel, you should only have to run the make KDEB_PKGVERSION=0 bindeb-pkg portion (and then install it with dpkg as root). **You should not generally have to re-run make mrproper after you have run it once for this assignment!**
   L. Verify that the Hello World system call can be called correctly from userspace by performing the steps in the "Testing Your New System Call" section of the Hello World system call tutorial. If this does not work properly, ensure that you have followed all steps in the tutorial and the compilation steps of this assignment. Also, be sure that you didn't make any mistakes in creating the test program.

Submission Instructions

You have already initialized and created your git repository early in this project (when you ran the git clone command earlier). Now it is only a matter of updating the copy of the git repository on your local VM and pushing the update up to the GitHub server.

First, ensure that your changes are all reflected in the local git repository. Run the following command in a terminal in your /usr/src/project0 directory and ensure that any files you have modified within the kernel sources are included in the section labeled "Changes not staged for commit": git status.

Next, you should ensure that the changes you made are accurately reflected in the modified data. To do so, run the command git diff and make sure that every change you made to the kernel source code is shown. You should be able to scroll back and forth through the diff output with the arrow keys on your keyboard, if needed.

Now that you have ensured that all the changes you made are reflected in the git repository's status, you must add the changes to your local repository and commit the changes. Do so with the following commands (replace `FILESYOUMODIFIED` with the filename of any files you modified -- the output to the `git status` command earlier should help):

```
git add FILESYOUMODIFIED
git commit
(Give a good commit message here describing your changes)
```

Now that you have committed your changes to your local git repository, you may wish to ensure that the commit has happened properly. To do so, run the `git log` command and make sure that your commit shows up at the top of the list.

Now that your commit is in your local repository, it is time to push the changes up to your GitHub account. To do so, run the `git push origin master` command. You should now be able to open your account on the GitHub website and see that your changes are pushed to the repository.

Please fill out this form to inform the TAs of your GitHub username. If you do not, we may not be able to grade your assignments, as we will not know which assignment belongs to whom.

Congratulations! You have completed Project 0!

Cleaning up...

After submitting your project, you should clean up your VM to ensure that you do not run out of space later on. To do so, run the following commands in a terminal.

```
cd /usr/src/project0
make mrproper
cd /usr/src
rm -rf vanilla-project0
```

What to do if you want to get a 0 on this project:

- Not pushing changes to your GitHub repository by the project due date.
- Excessive unnecessary changes made to the kernel sources.
- You have not included either of the two changes required by the project.
- Extraneous files are included.
- Not informing the TAs what your GitHub username is.

Please do not make us take off points for any of these things!

Last modified Tuesday, 06-Feb-2018 21:00:39 EST