CS 194-24 Lab 0: Introduction

Palmer Dabbelt

January 22, 2013

Contents

The purpose of this assignment is to familiarize you with the development enviornment that you will be using for the remainder of this class. Note that while this assignment **must be completed individually** you will be working in groups for the vast majority of this class. You should form your groups as soon as possible.

1 Virtual Machine Configuration

In order to minimize the pain of setting up a development enviornment we have provided you with a virtual machine image that contains all the tools you will be using in this class. You will be doing all of the development for this class inside of the provided virtual machine.

It's probably best to have a little overview of the enviornment you're about to setup before delving into the setup. You'll be using a VMWare virtual machine as your development enviornment. This VM contains all the tools that will be necessary to complete this class, you'll be spending all of your time inside that virtual machine.

In order to test the kernel modifications you will be making for this class you will be using a second VM, this one based on QEMU. This QEMU VM lives inside the VMWare VM, just like every other tool used in this class. The QEMU VM allows you to boot a kernel containing your modifications without risking the filesystem on your workstation. Figure 1 shows how these two virtual machines are nested.

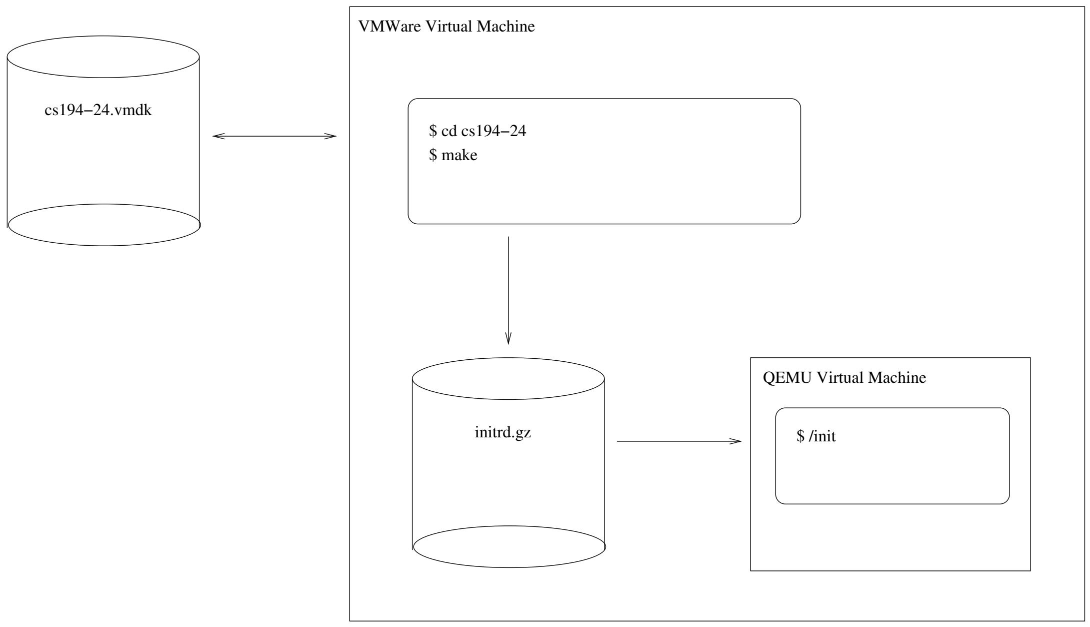

Figure 1: Virtual Machine Layout

1.1 Installing VMWare Workstation

While the virtual machine image we've provided is in a standard format, we can only support running it inside of VMWare Workstation 10. VMWare is already installed on the lab machines designated for this course. It's recommended that your setup VMWare on your personal machine as well.

Obtain the VMWare installation package that cooresponds to your operating system (note that the program will be called "VMWare Fusion" on Mac OSX) and install it on your computer. The Windows download site for VMWare is here

`http://www.vmware.com/products/workstation/overview.html`
while the Mac OS download site is here
`http://www.vmware.com/products/fusion/overview.html`
The EECS department provides VMWare Workstation 10 licenses to all students registered for this class. Unfortunately, we haven't gotten this setup yet so you'll have to apply for a 30-day trial for now. You can apply for a trial key online
`http://www.vmware.com/products/workstation/workstation-evaluation`

1.2 Configuring Your Virtual Machine

Download the provided VM image from the class website. As the image is quite large, it's probably best to do this over a wired connection. The EECS department provides wired AirBears connections that will allow you to quickly fetch the image to your laptop. You can obtain the VM image from the following URL:
`https://cs194-24.cs.berkeley.edu/virtual-image/cs194-24.ova`
The virtual machine should come almost entirely pre-configured. The machine will boot to a graphical login, at which point you should be able to login using the username "`user`" and the password "`user`". At this point all of your work should be done within VMWare.

2 Linux

The bulk of your programming assignments in this course will consist of hacking on Linux. In this section you'll learn how to build and boot a Linux kernel.

2.1 Building a Linux Kernel

Boot the provided virtual machine and login to the user account. Inside your home drive there will be a `cs194-24` folder which contains all the source code you will need for this class (aside from what you will be writing for assignments, of course).

The Linux sources we've provided you have been almost fully configured. Linux uses its own build configuration tool, so in order to familiarize yourself with it your first assignment will be to configure a build with debugging symbols enabled. You can bring up the Linux configuration utility by running

```
$ make -C linux menuconfig
```

A menu of configuration options should appaer. In order to enable debugging symbols, you'll first need to navigate to the "`Kernel hacking`" submenu. You then need to enable "`Kernel debugging`" and "`Compile the kernel with debug info`". If you pay close attention, you'll notice that the option for "`Compile the kernel with debug info`" only appears after you enable "`Kernel debugging`". This happens because the debug info option depends on the debugging option.

It can often be tricky to discover the correct set of dependencies to enable in order for an arbitrary option to appear in the kernel configuration menu. Luckily there is a search feature. You can bring up the search dialog by pressing the "`/`" key. By searching for "`DEBUG_INFO`" you can both the location of that particular configuration option as well as any dependencies it may have.

Exit from the configuration utility and save your changes (you will be prompted when you try and exit). You can build a kernel that uses your new configuration by running

```
$ make linux
```

2.2 Building a Busybox userspace

The above steps build an Linux kernel image, but that itself is not particularly useful. In order to actually use this kernel we need some userspace tools. We will be using the Busybox userspace tools in this class. Busybox is an implementation of the standard POSIX tools that are designed to be as compact as possible, and as such are commonly found in embedded devices. The Busybox project uses the same build system as the Linux kernel. We've already provided a Busybox configuration, so you can build Busybox by running

```
$ make busybox
```

2.3 Building an initrd

In order for the Linux kernel to be able to access the Busybox tools we need to provide a filesystem that contains those tools. In Linux terminology, this is known as an `initrd` (an INITialization Ram Disk). For this class, we'll be using a CPIO formatted archive as our initrd because Linux provides a program to generate properly formatted initrds. Just like everything before, the Makefile is capable of building an initrd for you.

```
$ make .lab0/interactive_initrd.gz
```

2.4 Building the QEMU virtual machine

At this point we have all the code required to boot an entire operating system, but we don't actually have a virtual machine to boot it in. For this class we will be using the QEMU virtual machine to boot and test our kernel images. Source code for QEMU is provided in the `qemu/` directory in the course repository. You can build QEMU by running

```
$ make qemu
```

2.5 Booting Linux in QEMU

In order to test the operating system you've just built without having to reboot your development machine we're going to use yet another virtual machine. In this case we'll be using QEMU because it is simple to script and runs efficiently when nested inside of VMWare. In order to boot the Linux install you just build you can run

```
$ make
$ ./boot_qemu
```

After a few seconds a QEMU window should appear inside of which you should see Linux boot. After a few seconds, a shell will appear where you can interact with all the code you just built. The file `lab0/interactive_config` contains a list of all the tools availiable inside this interactive enviornment. You can add additional tools by following the pattern described within.

2.6 Debugging Linux with GDB

While it's possible to use GDB to debug a running Linux kernel, it is a bit quirky. Luckily, a script has been provided that works around most of these quirks for you. Running

```
$ ./boot_qemu --debug
```

will boot up your Linux kernel under QEMU with a GDB session attached to it. The GDB session will launch in a seperate terminal window which you can use to interact with Linux. At this point, using GDB to debug the kernel is very similar using GDB to debug a userspace application. If you are unfamiliar with GDB then now would be a good time to read the manual – GDB will prove to be an invaluable tool during this class.

3 A simple Linux change

In order to verify that you're actually running the results of your build you need to make some sort of change to the Linux sources. One simple change is to modify the kernel's version number. In order to do this, you only have to change a single line in the Linux Makefile. Simply set the "`EXTRAVERSION`" variable to "`cs194`". The following patch describes the change (if you're unfamiliar with the output of `patch`, you should read up about it – it'll prove exceedingly useful for the course)

```
diff --git a/linux/Makefile b/linux/Makefile
index fbf84a4..c60d949 100644
--- a/linux/Makefile
+++ b/linux/Makefile
@@ -1,7 +1,7 @@
 VERSION = 3
 PATCHLEVEL = 12
 SUBLEVEL = 8
-EXTRAVERSION =
+EXTRAVERSION = cs194
 NAME = One Giant Leap for Frogkind
```

At this point you should be able to boot the provided Linux image and verify that the kernel version number (one of the first few strings printed out at boot) contains "cs194". We have provided a simple initrd that will shut down your QEMU instance as soon as it finishes booting that you can run by typing

```
$ ./boot_qemu --stdio --initrd .lab0/version_initrd.gz
```

4 Git

For this class we will be using the git version control system, which is a common open source VCS. If you are completely unfamiliar with git now would be a good time to read some documentation. You'll want to me familiar with the terminology used by git as you'll be using git to submit your assignments this semester. We'll be using a fairly advanced git configuration here: you'll need to understand how multiple branches and multiple remotes work.

4.1 Redmine

This course uses Redmine for project management. Redmine allows you to manage a private git repository to store your code in (as well as group shared repositories for the group projects), a wiki, and a bug tracker. The course's Redmine server can be accessed using your EECS LDAP credentials at `https://cs194-24.cs.berkeley.edu/redmine/`.

The following instructions should demonstrate how to setup your account on the Redmine server for repository access. The redmine website has a number of tutorials availiable demonstrate its more advanced features: `http://www.redmine.org/projects/redmine/wiki/Tutorials`. While we won't be requiring that you use any of these features, they may prove useful for the course. Specifically, we suggest considering using your group's Redmine wiki for the design documentation you'll be producing for later labs.

4.2 Repository Management

You will find that the provided `cs194-24` folder in the virtual machine's home directory is a git repository. There are three important repositories you'll need to know about for this class:

- `git://cs194-24.cs.berkeley.edu/cs194-24-spring-2014.git`: The public repository for this class. New assignments will be distributed via commits to this repository. The git repository in your virtual machine already contains a remote setup called `public` for this repo.
- `ssh://git-cs194-24@cs194-24.cs.berkeley.edu/student-parent/USERNAME.git`: Your personal repository, stored on the class's server. You are the only person (aside from class staff) who was read or write access to this repository. This is where you'll be submitting this assignment, and is a place to stage work before showing it to your group. This will be refered to as the `personal` repository in the future.
- `ssh://git-cs194-24@cs194-24.cs.berkeley.edu/group-parent/GROUPNAME.git`: Your group's repository, stored on the class's server. Only your group members (in addition to class staff) have read or write access to this repository. This will be refered to as the `group` repository in the future.

All these branches and repositories are managed by a centralized Redmine server that you can login to using your university credentials. The management server is located at `https://cs194-24.cs.berkeley.edu/redmine/`.

Login using your university username and password (your username is your email address without the domain). The remainder of this section will walk you through setting up your virtual machiane such that it is able to communicate with the class's Redmine server.

4.3 Branching Strategy

We will be using git to manage your assignment submissions for this class, which means the course staff will have access to your entire commit history. In order to keep things organized, we've decided to enforce a particular branching strategy. Your git repository should contain at least two branches, but you can have as many as you feel are useful.

- `master`: Your default development branch. Commits that exist solely on this branch will not be considered for grading. `master` is a good place to put code that works, but that you're not ready to have graded yet. For example: if you're working on an assignment and you've written some test cases but haven't yet implemented them in Linux, `master` would be a good place to push that commit.
- `release`: The branch of commits to be graded. When you commit to this branch, a run of the autograder will be triggered (note that there is no autograder for Lab 0). It's important to keep this branch clean both because we will be grading your commit history and because you don't want to overload the autograding server.

We're also going to ask that you provide merge messages when you merge to the `release` branch, which will make sorting out your solutions to each assignment much simpler. The following command will merge `master` to `release` while allowing you to type a merge message. A sane merge message might be something like "solution to lab 0".

```
$ git checkout release
$ git merge master --no-ff
```

Merging in this way will provide a single commit for each solution you expect to be graded, which will make tracking the results of your autograder runs much simpler. If you follow this merging strategy then you can get a list of all your submissions by running the following command.

```
$ git log release --merges
```

4.4 Git Configuration

Before you can actually submit your assignment, you'll first need to configure your git installation. You'll first need to inform git of your name and email address so it can include these in your commits. Be sure to use your real name and email address, as this is what will identify you to your group members in the next assignment. Input your name and email as follows:

```
$ git config user.name "My Name"
$ git config user.email "me@eecs.berkeley.edu"
```

You'll now need to generate a SSH keypair. SSH keypairs, like all public key encryption schemes, consist of two parts: the public key and the private key – if you're unfamiliar with how to use SSH public key encryption you should read up on it now. In SSH terminology, the public key will be named with a "`.pub`" extension. You can generate a SSH keypair by running

```
$ ssh-keygen
```

which will prompt you a few times. If you select all the defaults you'll end up with a private key located at `$HOME/.ssh/id_rsa` and a public key located at `$HOME/.ssh/id_rsa.pub`. It's important to keep your private key to yourself, but your public key is not sensitive.

Go to the class Redmine site and add your public key to the list of keys that have access to your account. That list lives inside your account settings (there's a link on the top-right after you login).

You'll now need to add your personal git repository (which lives on our Redmine server) as a remote of the repository that lives in your VM. On the class's Redmine server, find your project (there is a projects link at the top of the page, you project will reside within the "Students" project).

Find the repository URL on your project page (it's on the left side, about half the way down), and add it to the repository that lives inside your VM by running something like

```
$ git remote add personal ssh://git-cs194-24@cs194-24.cs.berkeley.edu/student-parent/USERNAME.git
```

4.5 Submitting Changes for Grading

As previously mentioned, we will be using git to submit code in this class. As you now have a small feature implemented (the extra version number you added previously), now is a good time to test the class' submission mechanism.

You'll first want to commit your code to the master branch. Be sure to use an informative commit message as it will be visible to the course staff.

```
$ git checkout master
$ git add .
$ git commit
```

You now need to create a merge commit on your release branch. While this is overkill for a single commit, it will become useful later when your implementations consist of many commits

```
$ git checkout release
$ git merge master --no-ff
```

At this point you have everything committed to your local repository, all that's left to do is inform the Redmine server about your changes.

```
$ git push personal master
$ git push personal release
```