Kernel Virtual Machine

In this meeting, we will use several tools to (1) create a virtual disk, (2) add a file system, (3) install Ubuntu system on that file system, (4) configure Ubuntu, and (5) run our VM using KVM and a kernel image.

For this meeting, we will use `man` and online documentation to figure out the necessary steps. As soon as you complete a step, please put the commands you used on the wiki *using code formatting*. The end goal of today is for all of us to have running VMs, and for someone to be able to open the wiki and copy and paste (documented) code into their terminal to achieve the same results.

If the command is already on the wiki, compare your commands to what is there. There are often many ways to complete the same task, so document any differences. If you think something is incorrect, ask the person who added it to confirm. In this way, we can all help each other.

Modules

There are several packages we need to install on our systems. Many of the packages we use provide us stable versions of user-level programs. Some, however, install kernel modules and device drivers. For example, you may install packages provided by nvdia for your graphics card.

- What is a kernel module?
- How do you check which modules are "loaded"? (Hint: `ls` lists files in a directory, `lsusb` lists USB devices, `lspci` lists all PCI devices, ... The command might start with `ls`)

Use `lsmod` to see which modules are loaded in your kernel.

Installing The Right Packages

There may be additional packages that are required, but you can start by using

```
$ sudo apt install ...
```

to install `qemu-kvm` (which also installs: qemu-utils, qemu-system-common, qemu-system-x86, ...). If you have the virtualization features of your hardware enabled in BIOS (they should be by default in most modern systems), and if your kernel is configured properly, you should see two new kernel modules after installing this package. Use the `lsmod` command again.

```
$ lsmod | grep kvm
```

- You should now see two new kernel modules. What are they?
- Do you think you might see different modules on a different system?

Also install the `debootstrap` package, which we will need to install Ubuntu on our virtual disk, and the `e2fsprogs` package, which will let us create `ext4` file systems.

Virtual Disks

From our reading, we saw that a virtual disk is "the virtual machines physical representation on the disk of the host machine. A virtual disk comprises either a single file or a collection of related files. It appears to the virtual machine as a physical hard disk."

To provision a virtual machine, we need to first create a virtual disk. There are probably tools that complete the whole process automatically (`libvirt`?), but we will do it step by step (we will still use tools for some of the steps—you could get even more low-level if interested).

Our first task is to create a virtual disk file. Your file system tree can be composed of many file systems, each on its own disk(s)/partition(s). For this step, we will just create a single virtual disk with a single root file system.

Use the utility `qemu-img` to create a 16GB file called `virtdisk.raw`. Create `virtdisk.raw` using the *raw* format. The `qemu-img` utility does much more than *create* disk images. Take a second to explore its man page because it is very handy.

- Update the wiki with the command you typed to create a virtual disk. Format this using code styling. Also include the meaning of the flags and options in enough detail that someone who reads your documentation could change the parameters to control its behavior.

Right now `virtdisk.raw` is an empty file. Let's explore it.

- Check its size using `ls -lh`. How big is it?
- Check its size using `du -h`. How big is it?
- Why are these sizes different?
- Use less to look at the first screenful of our file. What do you see?

Now we will install the ext4 filesystem on our newly created virtual disk file. Please read the manual page for `mkfs.ext4`. Use the '\' keyboard binding to search for the `-F` flag. Document the following command on the wiki in text a little more comprehensible than the manpage. Are there better commands to do the same thing?

```
$ mkfs.ext4 -F virtdisk.raw
```

Now we have a file system. And not just any file system, but `ext4`, the default Ubuntu file system. I'm feeling pretty good about this. But let's look at our virtual disk file again.

- Check its size using `ls -lh`. Has it changed?
- Check its size using `du -h`. Has it changed?
- Use less to look at the first screenful of our file. What do you see and why? (If you don't know, ask someone who took OS to explain it to you. If you took OS and don't know, then you are in the same boat I was after I took OS. We can work it out on the board.)

Now we want to mount our virtual disk file (which has a file system on it) as if that file was actually a block device.

- What is a block device?

We are going to use something called a loop device. A really useful tool for loop devices is `losetup`. You should know how this tool works. For right now, however, we will just mount our virtual device file using an option to the `mount` utility that takes care of setting up the loop device for us. (But I encourage you to use `losetup` if you would like.)

To mount a filesystem, the first thing we need is a mount-point. A mount-point is just a directory in our filesystem namespace where we will attach a filesystem. Create a directory called `mnt/`. To mount our virtual disk on `mnt/`,

```
$ mount -o loop $virtdiskfile $mntpnt
```

(Please document this command on the wiki.)

Now type `mount` and examine the output. You should be able to `cd` to your mountpoint and perform commands like `ls`. Even though it is a file, it appears as if it were a file system. *Do NOT* edit the file while it is mounted.

Now we will populate our virtual disk with an Ubuntu system. Think back to meeting 01 when we first set up our panics. We read about Linux distributions. Ubuntu is a Linux distribution based on Debian, and because of their similarity, Debian tools are often used in Ubuntu as well. Another fact that is hopefully clear at this point is that systems builders are very clever people. One symptom of this cleverness is the names they choose for their projects: `debootstrap` is a program to bootstrap a Debian system.

Please document on the wiki the `debootstrap` command you would use to:

- Use the amd64 architecture (We are using Intel processors, why would we use amd64?)
- Install packages in addition to the default Ubuntu system: (I always use openssh-server, emacs, git, python, and build-essential. Recommend anything else you think is important).
- Install the 16.04 release (Xenial Xerus)

Then use that command to install an Ubuntu system on your mounted virtual disk. It will take a long time. While you wait, you can continue where you left off from Monday or help your friends.

At this point your virtual disk file has an Ubuntu 16.04 installation, and it is still mounted. Before unmounting it, we will edit one of the configuration files that we are familiar with from last meeting. However we will do something new. By default, your system has a user named root. However, we did not define a password for that user. So that we can log-in for the first time, we will do something dangerous.

- Make it so that the root user does not need to use a password to login to your newly install virtual disk's OS. (Hint: this involves editing `/etc/passwd`).

Now you should unmount your virtual disk.

Booting your virtual machine using KVM

There are many parameters to KVM. We will explore more of them today, and even more when we build our own kernels. Here are some basic commands to get you started.

```
qemu-system-x86_64 -smp 2 -cpu host -m 2G \
    -drive format=raw,file=virtdisk.raw \
    -kernel /replace/with/path/to/bzImage \
    -enable-kvm \
    -append "root=/dev/sda" \
    -curses
```

Before executing any command, you should understand it. Please document this command on the wiki. What do each of these parameters do?
There is one important piece missing. The `-kernel` flag takes an argument: a compressed kernel image. For today, I compiled a kernel for us. I started from defualt Ubuntu configuration and added a few *virtio* drivers to take advantage of *paravirtualization*. You may find that kernel on the Department Linux machines in

```
/home/cs-local-linux/11/bzImage
```

Please copy this file to the appropriate place and adjust the command.

- Which utility did you use to securely copy a file from one machine to another? Please document it.

Uh-oh!

Sometimes things go wrong. That is one of the great things about virtual machines—if something goes wrong, you can just close the application and try again. There is also a qemu menu that lets you control your VMM. `Alt-2` will bring up the menu, and `quit` will exit qemu-kvm.

Challenges

- Add a user so that you don't log-in as root. Then add that user to the appropriate groups so that you can use `sudo`.
- Try to get basic networking to work so you can `ssh` to/from your VM and from your VM to the world. As a hint, here are the commands that will set up a basic network interface card (NIC) and then forward TCP traffic from one port on your host to a port on your guest.

  ```
  -net nic,model=virtio,macaddr=52:54:00:12:34:56 \
  -net user,hostfwd=tcp::${GUEST_SSH_PORT}-:22 \
  ```

  (Hint: from within your VM, find the name of your guest's NIC. Then edit the `/etc/network/interfaces` of your virtual disk so that it has an interface named ¡NIC¿ that uses dhcp.)

  ```
  auto lo
  iface lo inet loopback
  auto <nic name>
  iface <nic name> inet dhcp
  ```

- Add your public `ssh` key to the virtual disk's `authorized_keys` file (you may need to create some directories)
- One advantage of KVM is that it uses *paravirtualizaton*. Try to use the virtio interface for your root filesystem's device (Hint: in addition to specifying an additonal drive option, you may also have to change the kernel boot parameters in the `append` flag)