ECE391: Computer Systems Engineering **Fall 2021**
Machine Problem 0 **Due: in office hours by 11:59PM CST Wednesday 1 September**

Preparing your Environment

This assignment will help you to prepare your work environment for the class and will introduce you to the tools that you will be using. As mentioned in class, much of your work will be done in the context of a Linux kernel executing on a virtual machine (VM) within Windows. Most of the actual programming and development will be done within the VM. You will demo all of your work to a staff member on one of the lab machines in ECEB at UIUC.

You may want to read the documentation on the tools—available on the ECE391 web page—to familiarize yourself with them before starting this assignment. The tools used in this lab include QEMU, Git, GNU Make, and GDB. In addition to the reference materials provided on the web page, you can find information online and can get hands-on experience by logging into one of the EWS Linux machines in ECEB using Windows Remote Desktop (you may need to install this application if you use a Mac). See the class web page for additional instructions on connecting to the lab machines. We encourage you to learn to use a Unix editor, such as `vi(m)`, inside the VM. VSCode, Sublime, and GVIM have also been popular with previous students, so you may want to check those out (install them), but they are not available inside the VM. You can, however, read from and write to the same work directory files that you will compile in your VM. Keep in mind that familiarity with Git's command-line interface will be useful both in the demo for MP0 and in general in the course.

In this document, Windows menu and submenu selections are represented as slash-separated sequences in small capitals. For example, START/COMPUTER means to bring up the START menu, then pick MY COMPUTER. Typing as well as Linux names and commands appear in Courier, such as `cat > hat`.

Step by Step Instructions

Please read through this whole section before you begin, and follow the instructions carefully. Done correctly, on an unloaded network (which will not be the case near the deadline!), the entire machine problem takes less than two hours. Note that if anything gets corrupted, you will have to start over, so start early.

1. Start by opening the class directory (`V:\ece391`) and your work directory (`Z:`) under Windows. These should appear under THIS PC (START/THIS PC), but you may want to create shortcuts on the desktop by right-clicking on the icons and selecting CREATE SHORTCUT. If the drives are not mapped on lab machine (in ECEB 3026), you may want to map them yourself using THIS PC/VIEW/MAP NETWORK DRIVE. The `V:` drive should mount `\\ad.uillinois.edu\engr-ews\classes\ece391`, and the `Z:` drive should mount `\\engr-ece-391.ad.uillinois.edu\<your NETID>`.

2. Open the `mp0` folder in the class directory and double click on the `ece391setup` script. This script takes about seven minutes on an unloaded network and creates the directories `vm` and `source` in your work directory. The `vm` directory holds the snapshot disks for your VM. The `source` directory contains the source code for the Linux Kernel. The script creates two virtual machines for you: a development machine and a test machine. The development machine is for writing and compiling code. The test machine is for running your code. When debugging the Linux Kernel, your test machine runs your test kernel, while the development machine runs GDB. The script also creates four shortcuts on your desktop: `devel`, `devel_local`, `test_debug`, and `test_nodebug`. The `devel_local` is used only for completing MP0. The `devel` (after step 11) and `test_(no)debug` shortcuts launch their respective virtual machines in QEMU. You will primarily use `devel` and `test_debug`. `test_debug` launches the test machine and waits for a connection from GDB from the development machine.

Complete steps 3-11 on the same lab machine without logging off. To reduce network load, this lab uses a local file to hold your virtual disk (`devel.qcow`). Other machines can not make use of this file, and the file **will be deleted when you log off**. Be sure to copy the file (Step 11) before you log off. If you make a mistake, return to Step 1.

3. You are now ready to boot your new virtual machines. Start by booting the development machine: double click on `devel_local` on your desktop. If a QEMU window does not pop up, you may want to investigate the failure (this skill will be useful later in the class). Right-click on the shortcut on your desktop. Copy the QEMU directory given in the START IN: of the properties dialog. Go to START/RUN, and paste the path from START IN:. The QEMU directory will appear. The `stderr.txt` file contains error output from the last time QEMU was run, and may help you to identify the problem.

4. Once a QEMU window pops up, wait for the Linux kernel to boot. Ignore errors about IPv6 support. If you click in the QEMU window, you must press `Ctrl-Alt` to release the mouse. You can type in the virtual machine without clicking in the virtual machine: simply switch to QEMU using `Alt-Tab` or click on the title bar. When you see a login prompt, log in as `user` with password `ece391`. The `root` account has the same password, but you should not need super-user privileges often on the development machine.

5. Using your favourite Unix editor, open the `.bashrc` file. Files starting with a period are hidden by default in directory listings; type `ls -al` to see a more detailed listing if you feel the need to see files before you open them. Find the place in which your NetID should be specified according to the comments in the file and fill it in.

6. Now source the `.bashrc` script (in other words, use it as a source of commands) by typing either `source .bashrc` or `. .bashrc`. The script will attempt to mount the class directory and your work directory within the virtual machine. Normally, this process occurs when you log into the machine, but was skipped the first time because the script did not know your NetID. You need to type your AD password twice, once for each drive. Please do NOT use the number pad to enter your password: certain versions of QEMU/keyboards do not always map the number pad scancodes correctly. If you type your password correctly, the class directory appears under `/ece391`, and your work directory appears under `/workdir`. If you need to unmount or remount the drives for some reason, such as mistyping your AD password, use the `/root/unmntdrives` and `/root/mntdrives` commands, respectively.

7. Change directory to your virtual machine work directory (`cd /workdir/vm`) and type `ls`. Notice that one of the virtual machine snapshot disk files is now visible within the virtual machine (under `/workdir/vm`). This visibility is analogous to being able to stand outside your body and observe yourself. Changing the file is analogous to conducting surgery on yourself rather than simply observing. If you decide to try it and fail, start again at Step 2.

8. The `/workdir/source/linux-2.6.22.5` directory is a working directory for the Linux source. It is a checkout of the Subversion repository located at `/workdir/repos/`. This code is located on the EWS server so it is backed up. **This step requires no action on your part.**

9. The next step is to build the Linux kernel. To do so, you must configure the kernel, make dependency information, clean up the source directories, and then finally compile the sources. **We have pre-configured your kernel with minimal options**. Kernel configuration allows a wide range of options. Normally, a kernel is configured by the `make menuconfig` command within the kernel source directory (`/workdir/source/linux-2.6.22.5`). You do not need to do this now since we provide a default configuration. The configuration is stored at `/home/user/build/.config` (the Linux kernel requires symlinks and a case-sensitive file-system to build on which is why we use `/home/user/build` to build the kernel while the source is stored on the Engineering file server). After changing the configuration, you must normally re-make the dependency information by typing `make dep`, then clean up the directories by typing `make clean`. **Again, this step requires no action on your part.**

10. Now you are ready to compile the kernel. Execute the `make` command in the kernel source directory (`/home/user/build`). Compiling will take around 50-75 minutes, and much longer if the network is slow (for example, when many people are trying to compile at the same time—don't wait for the due date!). If the system reports any errors during the process (the warnings that are generated are not a problem), delete the `source` folder in your `Z:` drive, re-run the setup script in Step 2, and execute the `make` command again.

11. Now you must copy the local QCOW file to your work drive. First, shut down the development machine: type `halt`, wait for the message `System halted.`, and close the window. After shutting down the development machine, copy the `devel.qcow` file from the local machine (`C:\Users\netid\devel.qcow`) to your workdrive (`Z:\vm\devel.qcow`). After the copy has finished, boot the development machine using the `devel` shortcut on your desktop and log in as `user`. We also recommend making a backup of the `devel.qcow` file—a simple copy operation suffices in Windows. If you corrupt your drive later, you can roll back easily to the virtual disk state after finishing the Linux build in the previous step.

12. From the kernel build directory (`/home/user/build/`) type `make install` to install the kernel into the correct directories. We have customized the Linux install scripts to copy the `bzImage` kernel image and `vmlinux` debug image into the kernel source directory, because the source directory location is accessible to QEMU when you debug your test machine, whereas the build directory resides in the virtual disk of your development machine.

13. Now it's time to debug your new kernel! Start debugging the test machine by double clicking on `test_debug`. (Ignore messages about the the Windows Firewall, if any appear.) This shortcut opens a QEMU window that is paused waiting for a connection from GDB. Switch back to your development machine and make sure that you are in the kernel build directory (`/home/user/build`). Start up GDB on the development machine for the kernel (`gdb vmlinux`). Once GDB has loaded, issue the command `target remote 10.0.2.2:1234`. This command connects GDB to the kernel executing in the test machine. Ignore the warning about not being able to set a breakpoint.

14. Once GDB is connected to your test kernel, list the CIFS (Common Internet File System) code to open a file by typing: `list cifs_open`. Set a breakpoint at this function by typing: `break cifs_open`. Next, allow the kernel to continue execution by typing `c` for continue.

    Switch to the test virtual machine and wait for the kernel to boot, then log in as `user`. Repeat Steps 5 and 6 again in the test machine so that your network drives are mounted. Once the directories are mounted in the test machine, type `touch /workdir/test` and switch back to the development machine. Notice that GDB stopped at the breakpoint when the test kernel tried to open the file. You can now `step` through the instructions and inspect variables in the kernel, just like debugging any other program. To continue, type `c` again (you may need to do this a couple times).

15. The last step is to shut down both machines. On the test machine, shut it down cleanly with the `halt` command. Once it is shut down (The line `System halted.` will be displayed), you may close the window. Next, quit GDB in the development machine by issuing the `quit` command. You may have to press `Ctrl-C` first. Then shut down the development machine, again using the `halt` command.

16. Good work!

Handin

For handin, find any TA in office hours and repeat Steps 13 and 14.

In addition, **you will be asked to demonstrate basic knowledge of Git (Note: We will not test Subversion!), GDB, and version control concepts**. Being able to use GDB to debug your code is a critical skill for success in this class. For Git, you may want to read the documentation on the course web page in preparation, particularly if you have not already used Git in previous classes (version control systems are all solving the same problem, so don't panic).

Version control, in the context of this course, is a tool to manage and track changes to your program code. When there are multiple collaborators working on the same project, it becomes a nearly indispensable tool.

For all ECE 391 assignments, **you will be required to learn and use Git**. All your repositories will be stored on Engineering IT's instance of GitLab: `http://gitlab.engr.illinois.edu/`. You may log in using your NetID and password. At the beginning of the semester and before each of the remaining assignments, you will receive an invitation to the ECE 391 group and your repository.

A Couple of Tips

You can keep valuable data in your virtual hard drives, and you may even gain some slight performance advantage by doing so, but if you ever corrupt them, that data is gone. Instead, keep all of your important data under `/workdir` and use your GitLab repository for your source code.

You may want to use a graphical interface and window manager (GNOME), particularly on your development machine. To do so, log in and perform the following steps:

1. Open `/etc/X11/xorg.conf` in a text editor, either as root or using `sudo` to act as root (that is, either log in as root to make this change, or type something like `sudo vi /etc/X11/xorg.conf`).
2. Under the `Device` section, set the `Driver` to `vesa`.
3. Save and close the file.
4. Execute the command `startx`.

When you're done, you may shut down the virtual machine by using the GNOME menu. Be aware of two issues if you choose to use X, however. We will use some graphics card functionality later in the semester, and X may not play nicely with competing applications, so you should not run it on your test machine with those labs (this aspect doesn't matter for the development machine).

Your work directory will be deleted at the end of the semester. Be sure to save any source code that you wish to retain by moving it to your home directory or to your own personal data storage.

If you would like to understand the virtual machine organization and how the environment works, see the figure below or ask a TA.

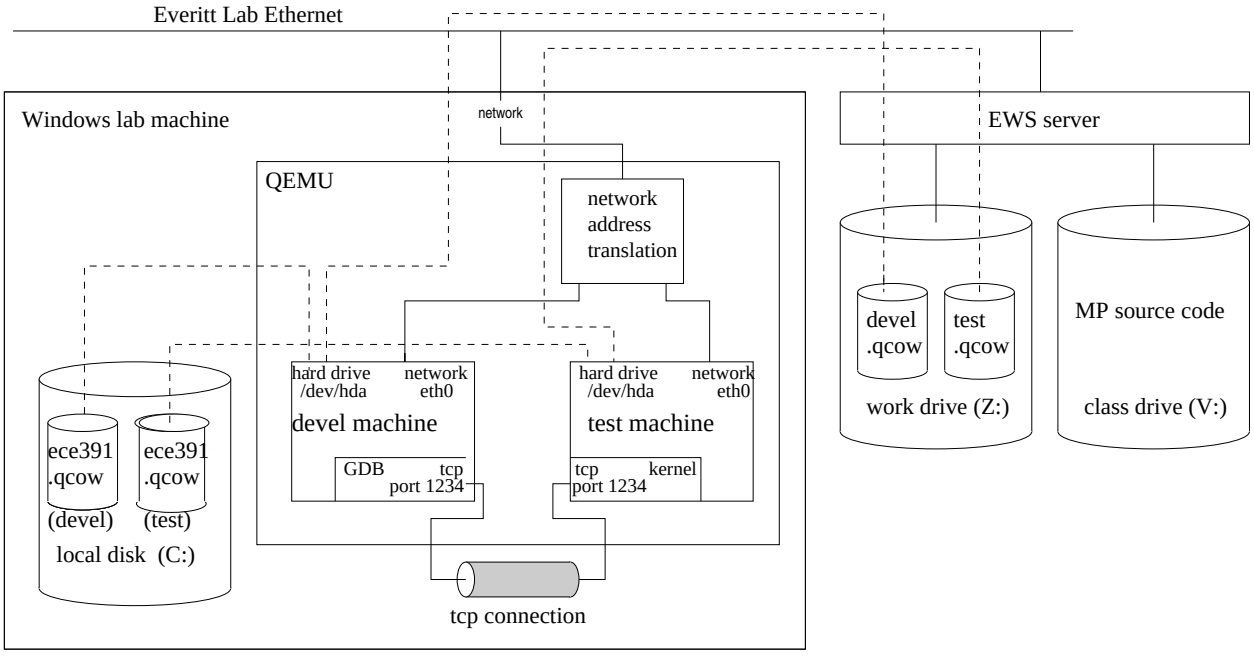

Figure 1: Diagram of the ECE391 execution environment. Class, home directory, and work directory mounts from the two virtual machines occur as network file systems, and thus occur over the virtual machines' Ethernet ports. The connections for these mounts are not shown in the diagram. The hard drive connections from the virtual machines are dotted for clarity; only the ECEB Ethernet and ECE server to storage connections are physical.