ECE391: Computer Systems Engineering **Spring 2017**
Machine Problem 0 **Due: in office hours by Monday, Jan 23**

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 the class, much of the work in the class will be done in the context of a Linux kernel executing on a virtual machine within Windows. Most of the actual programming and development will be done within the Linux virtual machine.

You may want to read the documentation on the tools to familiarize yourself with them before starting this assignment. The tools used in this lab include QEMU, Subversion, GNU Make, and GDB. In addition to the reference materials provided in class, you can find plenty of information online and get hands-on experience by logging in to one of the EWS Linux boxes in Everitt, for example. You should definitely learn to use a Unix editor such as vi(m).

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, with no networking issues, the lab should take no more than around 2 hours. Keep in mind if anything gets corrupted, you will have to start over, so start early.

1. Start by opening the class directory (`\\instdir05.engr.illinois.edu\classes\ece391` or `V:\ece391`) and your work directory (`\\engr-ece-391.engr.illinois.edu\team\ece391\<your NETID>` or `Z:`) under Windows (if you do not see `V:` and/or `Z:`, please map them yourself). Shortcuts to your directory and the directory for all ECE classes should appear in the COMPUTER folder (START/COMPUTER), but you may want to create shortcuts on the desktop by right-clicking on the icons and selecting CREATE SHORTCUT.

2. Run the `ece391setup` script from the class directory under mp0. This script takes about 20 minutes on an unloaded network and will create the directories `vm`, `source`, and `repos`. The `vm` directory will hold the snapshot disks for your VM. The `source` directory will contain the source code for the Linux Kernel. The `repos` directory will hold the subversion repository for your Linux Kernel. This script will create 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 will run your test kernel while the development machine will be running GDB. The script will also setup a Subversion repository that contains the source code for the linux-2.6.22.5 kernel. The source is slightly modified for use in ECE 391. See the Subversion handout or online documentation for information on using Subversion. The script will also create four shortcuts on your desktop: `devel`, `devel_local`, `test_debug`, and `test_nodebug`. The `devel_local` is used for completing mp0 only. The `devel` (after step 11) and `test_(no)debug` will launch their respective virtual machines in QEMU. You will primarily use `devel` and `test_debug`. `test_debug` will be used to launch the test machine and wait for a connection from GDB from the development machine.

   NOTE: you must complete steps 3-11 on the same lab machine and without logging off because of the following issues:

   - The local QCOW file will only be available on that machine. If you switch to a different machine in the lab you will not be able to access the QCOW file.
   - The local QCOW file will be deleted after you log off.

   Most importantly, do not forget to copy the local QCOW file to your work drive in step 11.

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 popup, you need to figure out why. 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. There is a `stderr.txt` file that contains any errors that occurred on the last time QEMU was run. This way, you can debug whatever issue that caused QEMU to not run.

4. Once the QEMU window pops up, you can safely ignore any errors complaining about IPv6 support. If you click in the QEMU window, you will have to press `ctrl-alt` to release the mouse. You can type in the virtual machine without having to click in the virtual machine, simply switch to QEMU using `alt-tab` or click on the title bar. When you see the login prompt, login as `user` with password `ece391`. The `root` account has the same password (feel free to change them, but don't forget them if you do!), 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 (which means, 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 will occur when you log into the machine, but was skipped the first time because the script did not know your NetID. You will 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 it correctly, the class directory will appear under `/ece391` and your work directory 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 the virtual machine snapshot disk files are 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 files 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 check-out of the Subversion repository located at `/workdir/repos/`. This code is located on the EWS server so it is backed up. We highly recommend that you `svn commit` frequently as you make changes. This action will allow you to restore to a previous version and also log why you made certain changes. You only gain this benefit, however, if you comment each revision. See the Subversion handout for more information. **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're ready to compile the kernel. Execute the `make` command in the kernel source directory (`/home/user/build`). Compiling will take around 75-90 minutes, and probably much longer especially if the network is being slow. (particularly if there are a lot of people trying to compile at the same time—don't

wait for the due date!). If the system reports any errors during the process, please delete the source folder in your Z: drive, re-run the setup script in step 2, and execute `make` command again.

11. Now you must copy the local QCOW file to your work drive. First shut down the development machine using `halt`. After the development machine has shut down, 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 login as `user`. We also recommend saving a backup of the `devel.qcow` file, you can make another copy of the file somewhere in the `Z:` drive.

12. From the kernel build directory (`/home/user/build/`) type `make install` to install the kernel into the correct directories. We have customized the install 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. Next we will start debugging your built kernel. Start debugging the test machine by running `test_debug`. (Ignore any messages about the the Windows Firewall if there is any.) This will open a QEMU window that is paused waiting for a connection from GDB. Now switch back to your development machine and make sure you are in the kernel build directory (`/home/user/build`). Start up GDB on the development machine for the kernel (`gdb vmlinux`). Once GDB is loaded, issue the command `target remote 10.0.2.2:1234`. This will connect GDB to the kernel 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`. You will need to repeat step 5 and 6 again in the test machine so that your network drives are mounted in the test machine. Once the directories are mounted, in the test machine, type `touch /workdir/test` and switch back to the development machine. You will see GDB has stopped at the breakpoint when the 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. During handin, find a TA and repeat steps 13-14. In addition, you will be asked to demonstrate basic knowledge of Subversion and version control concepts. You may want to read the Subversion handout as preparation for this step.

16. 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 you may quit GDB in the development machine by issuing the `quit` command. You may then shut down the development machine with the `halt` command.

17. Good work!

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 Subversion 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`
2. Under the ”Device” section, set the Driver to ”vesa”
3. Save and close the file
4. Run the command ”startx” . . .

When you're done, you may shutdown 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). Performance wise, it is best if you use the editor of your choice in windows to edit source code, the development machine to compile the code, and the test machine to debug and test the code.

Your work directory will be deleted at the end of the semester. Be sure to save any source code 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 setup and how the environment works, see the figure below or ask a TA.

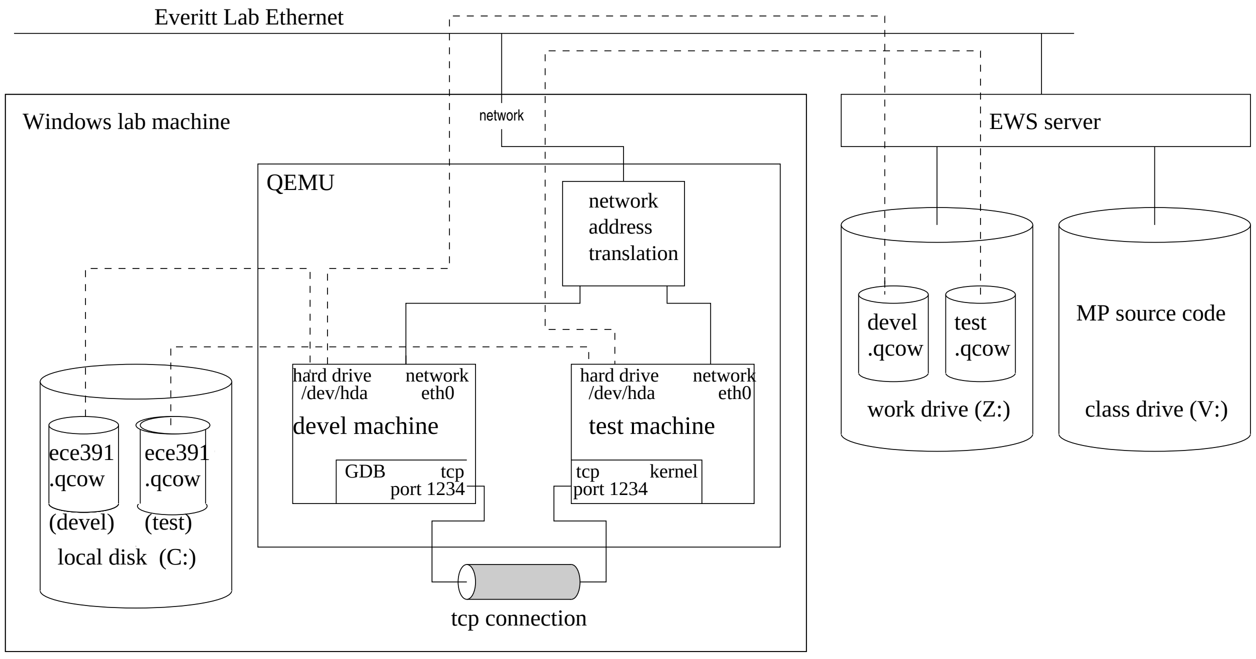

Figure 1: Diagram of the ECE391 execution environment. Class, home directory, and work directory mounts from the two virtual machines occur as network file systems, and thus occur over the virtual machines' Ethernet ports. The connections for these mounts are not shown in the diagram. The hard drive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