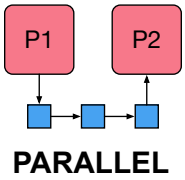

CISC 372: Parallel Computing

University of Delaware, Spring 2022

HW 1

This assignment is due Monday, Feb. 14, at 12:00 PM. Topics: getting set up.

1. Prelude: Logging on to `cisc372.cis.udel.edu`

If you have not already done so, apply for an EECIS account: `https://accounts.eecis.udel.edu/register/`.

Next, you must set up your computer so that you can use it as a terminal to log on to `cisc372.cis.udel.edu`.

Mac Users. Install XCode command line tools, if you have not already done so, by typing

```
xcode-select --install
```

in a Terminal window. If you don't know whether you have already installed command line tools, type the command and it will tell you. Use "Software Update" to make sure the tools are up-to-date. If you can't figure it out, ask us.

Optional: install XQuartz, `https://www.xquartz.org`. This will give you X-window support when you *ssh* to other machines, which can be nice but is not necessary.

Windows Users. *Option 1:* Use the Windows Command Prompt (CMD). It should recognize the `ssh` command. If you want to use your own Windows machine as a development environment, you should also install WSL2, which will give you a Linux environment within Windows:

`https://docs.microsoft.com/en-us/windows/wsl/`
`https://docs.microsoft.com/en-us/windows/wsl/setup/environment`

Option 2: This is the old way, which you can try if the command prompt doesn't work. Download and configure PuTTY + Xming, following instructions at

`https://udeploy.udel.edu/software/putty-with-xming/`

and at

`http://www.udel.edu/it/research/training/config_laptop/puTTY.shtml`

The hostname for your saved session should be `cisc372.cis.udel.edu`. *Note:* Xming gives you X-window support, which can be nice.

Linux Users. You shouldn't have to do anything.

Next, you will log on to `cisc372.cis.udel.edu`. If you are doing this from off campus, you will have to first turn on the UD VPN, using Cisco AnyConnect client, available here:

`https://udeploy.udel.edu/software/anyconnect-vpn/`

For Mac and Linux users, open a Terminal; for Windows, open the command prompt. Type the following:

```
ssh USER@cisc372.cis.udel.edu
```

where `USER` is your EECIS user name. Enter your EECIS password when prompted.

For Windows PuTTY users, use PuTTY as instructed.

If you have logged on successfully, you should see a prompt in the interactive terminal (shell), which should look something like this:

```
siegel@cisc372:~
```

Test it by typing the following at the prompt:

```
pwd
```

It should respond with the path to your home directory, which probably looks something like this:

```
/usa/siegel
```

It may be a different path for you; that's normal and fine.

Next, type:

```
emacs
```

The emacs editor will run in the Terminal window. You can exit emacs by typing in the emacs window:

```
C-x C-c
```

That's control-x followed by control-c. You should now be back in your interactive terminal.

Type:

```
exit
```

to log out. (Alternatively, type C-d.)

If you have made it this far, congrats, you are able to log on and use the machine successfully.

Optional: how to use X windows with ssh. If you have X windows support on your local machine, log back in using this command:

```
ssh -Y USER@cisc372.cis.udel.edu
```

Type:

```
emacs &
```

This should bring up emacs in a new window on your desktop. The `&` means: do the command "in the background." This means that you can continue to use the interactive terminal and the emacs window at the same time. This is very useful, because you can type commands into the terminal (e.g., to compile your program) and keep the emacs window open to edit your program. Again, exit emacs with C-x C-c.

If something goes wrong, type `fg`. This brings the background process into the "foreground." Then you should be able to exit emacs with C-x C-c.

2. Problem 1: Creating/committing directories

Log on to `cisc372.cis.udel.edu`. You should be in your home directory. Proceed as follows:

(1) Check out the course public svn repository:
```
svn checkout svn://vsl.cis.udel.edu/cisc372/372-2022S
```
(2) Check out your private svn repository:
```
svn checkout --username USER svn://vsl.cis.udel.edu/cisc372/372-USER
```
where `USER` is your user name. Use your svn password when prompted.
(3) Change into the local copy of your personal repo (`cd 372-USER`).
(4) Create a sub-directory named `hw` (`mkdir hw`).
(5) Add `hw` to version control (`svn add hw`).
(6) Change into `hw` (`cd hw`).
(7) Create a sub-directory of `hw` named `hw01` (`mkdir hw01`).
(8) Add `hw01` to version control (`svn add hw01`).
(9) Change into `hw01` (`cd hw01`).
(10) Use emacs to create a file in `hw01` named `myhi.c`. This file should be a simple C program that prints `Hello from XXX!` 10 times, where `XXX` is your name. Save the file.
(11) Compile `myhi.c` (`cc -o myhi.exec myhi.c`).
(12) Execute it (`./myhi.exec`) and make sure it works. If there are problems, edit, compile, execute and repeat until it works.
(13) Exit emacs.
(14) Add `myhi.c` to version control (`svn add myhi.c`).
(15) Change back into the root of your local copy (`cd ../..`).
(16) Commit everything (`svn commit`). Enter a meaningful log message when the editor window opens, save it, and exit the editor. *Note:* the command `svn commit` commits all changes in your current directory and subdirectories. It does not commit anything in the parent directory. That's why we first changed into the root of the local copy before issuing this command. This is a good practice in general.

Congrats! This last step has synchronized the local copy of your repository with the global copy stored on the server. All the files that have been added to version control have now been copied there, and they will be there forever! If your personal computer is lost, stolen, or corrupted, at least you know that all of the work that you committed is safe. In fact, it is best to commit *early* and *often*.

Just to make absolutely sure, let's checkout a fresh copy of your repo. First, move into your home directory (`cd ~`). Check out a fresh copy of your personal repo, calling it `TMP`:

```
svn checkout --username USER svn://vsl.cis.udel.edu/cisc372/372-USER TMP
```

Now examine it:

`ls TMP`	[You should see `hw`]
`ls TMP/hw`	[You should see `hw01`]
`ls TMP/hw/hw01`	[You should see `myhi.c`]
`cat TMP/hw/hw01/myhi.c`	[You should see the contents of `myhi.c`]

Finally, remove `TMP` and all of its contents, recursively (`rm -rf TMP`).

Note: this last command using the option `-rf` is *very powerful* and also *dangerous*. It deletes everything in a directory and if you are not careful, you could wipe out your entire home directory.

Grading rubric: 14 points: `hw` and `hw01` directories present (2pt); `myhi.c` present (2pt); `myhi.c` updated to say student name 10 times (4pt); compiles without warnings using `-Wall -pedantic` (2pt); no executables or extraneous files committed (2pt); appropriate log message (2pt).

3. Problem 2: Add some automation

(1) Change into your `hw01` directory.
(2) Create a `Makefile` with the following rules:
   - `myhi.exec`: this compiles `myhi.c` to create the executable `myhi.exec` (unless there is no need to re-compile it)
   - `test`: this executes `myhi.exec`. Of course, if `myhi.c` needs to be (re-)compiled, it will do that.
   - `clean`: this deletes any generated files, which in this case means `myhi.exec` and any file with a name ending in `~`. These latter files are emacs backup files. The command to delete all files of this form is `rm -f myhi.exec *~`
(3) Test your `Makefile`: Try typing `make test` from the command line, and see what happens. Ditto with `make myhi.exec` and `make clean`. If you type `make myhi.exec` twice in a row, what happens? Also, try typing `make myhi.exec`, then edit the file in some trivial way (e.g., add another word to the message being printed), then type `make myhi.exec` again. Note what happens.
(4) Add your `Makefile` to version control and commit it, with a meaningful message. That is all you have to do for this problem.

Grading rubric: 16 points: `Makefile` present (2pt); `make myhi.exec` compiles correctly (2pt) and has correct dependencies (2pt); `make test` works and has correct dependencies (4pt); `make clean` works and has correct dependencies (4pt); appropriate log message from commit (2pt).

4. Problem 3: Telling Subversion to ignore things

(1) Move back into your `hw01` directory and type `make test`.
(2) Type `svn status`. The output should indicate that you have at least one file (`myhi.exec`) which is *not* under version control. That's good—you don't want it to be under version control. (Why?) However, it can get annoying when svn continually tries to remind you about this file, and when there are many files involved, it can be difficult to see if there are files in the list that you really *do* want to add to version control. The solution to this problem is to tell Subversion that certain files should be *ignored*. Do this as follows:
(3) Type

    ```
    svn propedit svn:ignore .
    ```

    (Note: the `.` on the end is very important. Don't leave it out!) This means that you are going to edit a *property* named `svn:ignore` associated to the current directory. An editor window should open with an empty buffer. Add to this buffer one line for every filename (or filename pattern) that you want ignored. In this case, you should add 2 lines:

    ```
    *.exec
    *~
    ```

    Save the file, and exit the editor. *Note:* you can specify your default editor by setting the environment variable `EDITOR` in the appropriate bash startup file. For example, the line `export EDITOR=emacs` in `.bash_profile` will make `emacs` the default editor.
(4) Type `svn status` again, and this time you should *not* see the ignored files. You will however see something that indicates there is an uncommitted change to the current directory—this is the change your made to the (invisible) `svn:ignore` property file.
(5) Commit that change as usual by typing:

    ```
    svn commit
    ```

    and enter a log message explaining what you did and why. Save it, and exit the editor.

Grading rubric: 8 points: subversion *ignore* property set correctly (6pt); appropriate log message (2pt).

5. Problem 4: Get XSEDE Account

Direct your web browser to `https://portal.xsede.org`.

Click "Create Account." Fill out an account application form. Use your UD user name; if it is not available, you will have to make up a different user name.

Create a plain text file in the root directory of your private Svn repository named `XSEDE`. Put your new XSEDE user name in this file, followed by a newline. The file should consist of exactly one line, which contains your XSEDE user name, and nothing else. Add this file to version control and commit it.

After you do this we will add you to the class Bridges-2 allocation, which will allow you to start using Bridges-2. This may take a day or two.

Grading rubric: 8 points: XSEDE file present in correct place and contains valid user name and nothing else (6pt); appropriate log message (2pt).

6. Problem 5: Power

Assume the ideal model of the relation beween CPU power consumption and clock frequency (see notes from Introductory lecture).

A certain CPU core consumes 100 Watts of power when running at a frequency of 2.0 GHz. We will make an 8-core version of this CPU. What frequency will the cores of the new CPU have to run at in order to consume the same power (100W) as the original single-core version?

Put your answer in a file named `power.txt` in your `hw01` directory. The file should have the following form:

```
New frequency (GHz): XXXX
Explanation:
  <show the math or give some explanation on how you arrived
   at your answer>
```

where `XXXX` is your answer expressed as a floating-point number.

Add and commit the file `power.txt`.

Grading rubric: 14 points: file is present and in correct place (2pt); correct frequency (5pt); correct explanation (5pt); appropriate log message from commit (2pt);

Appendix: Installing WSL2 on Windows: Old Way

This is here mainly for older releases of Windows 10. (For newer releases, it is as easy as typing `wsl --install`.)

Download Windows Update Assistant: go to

`https://www.microsoft.com/en-us/software-download/windows10`

and click "update now".

After update (and several reboots), enable WSL: from an elevated powershell (win-x then "Windows Powershell (admin)"), type (on one line):

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux
   /all /norestart
```

Enable Virtual Machine Platform: from an elevated powershell, type:

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

Restart the computer.

Upgrade the Kernel (I got an error when I did not do this): download and install

`https://docs.microsoft.com/en-us/windows/wsl/wsl2-kernel`

Set WSL2 to default: from an elevated powershell, type:

```
wsl --set-default-version 2
```

If new to WSL, install any Linux distro from the windows store (suggest Ubuntu 20.04 LTS):

`https://www.microsoft.com/en-us/p/ubuntu-2004-lts/9n6svws3rx71?activetab=pivot:overviewtab`

If you already have a distro you want to keep, but upgrade:

```
wsl.exe --set-version <distroname> 2
```

where `<distroname>` is the name of the version to upgrade. You can get this by issuing the command `wsl -l -v` at the powershell prompt.

Launch: type `ubuntu2004` in cmd.

Update packages: type `sudo apt-get update`