Worksheet for Monday, January 24th 2022

1. Access R2 using ssh

Open your terminal app (mobaXterm on Windows, terminal on Linux or Mac, or any other terminal app), and use the following command

```
ssh -CX <username>@r2-login.boisestate.edu
```

where <username> if your username assigned by research computing (RC).

- Make sure you change your password (use `passwd` command) from what was provided by RC.
- Test whether you can access R2 from home using VPN. Follow the instructions on how to set up VPN here: https://www.boisestate.edu/oit-network/vpn-services/. If you have problems, please contact OIT. If you use Windows 11, you may need to download an appropriate VPN client from the following link: https://vpn.boisestate.edu/global-protect/getsoftwarepage.esp

2. Copy files

Make a new directory in your home directory using `mkdir` command. Copy the following files:

```
hello.c
run_script_hello.bash
```

from my shared location:

```
/scratch/makopera-shared/comput571/Week_3
```

to your newly created directory. You can use the `cp <from_where> <to_where>` command. If unsure, check the Linux system tutorial here: https://www.boisestate.edu/rcs/cluster-guides/the-linux-operating-system/.

Once copied, go to your directory (`cd <directory_name>`) and list the files which are there, to make sure the copy was successful (`ls`, or `ls -lah`, or `ll` commands).

3. Open and edit files

To open and possibly edit files on R2, you will need a text editor. The OIT people recommend *vim*, and provide a brief overview in the last section of the Linux tutorial here: https://www.boisestate.edu/rcs/cluster-guides/the-linux-operating-system/.

I have a personal preference to *emacs*. You can find an emacs overview article on Canvas in Week 3 module, or check out An Absolute Beginner Guide to Emacs here: http://www.jesshamrick.com/2012/09/10/absolute-beginners-guide-to-emacs/

Both editors are very powerful and ubiquitous on computing systems. Using them is a learning curve, but they are very powerful. Whichever you choose will serve you well. There is one thing you need to know, though. People working in scientific computing world fall into two categories: emacs people and vim people. There are slight animosities between them, with a good amount of banter flying both ways at computing conferences :) So… feel free to use vim, but we can't be friends then ;)

Once you decide which editor you want to use, open the `hello_world.c` file and examine it. It is the most basic C program, which only function is to print a hello message. Change the message to whatever you want using the editor, save the file and exit the editor.

4. Compile the hello_world program

Before you can run your hello_world code, you need to compile it, that is translate from a human-readable code to a computer-executable program. This is achieved using a program called compiler. R2 has a number of compilers available. We will use one of the most popular ones called gcc.

First, you will have to load the compiler module, so you can use it (this is equivalent to installing a program on your computer, but will not take as long). Type:

```
module list
```

to see a list of modules which are loaded for you automatically. It may be empty, or may contain some pre-loaded modules. Next, load gcc using

```
module load gcc
```

and check again the list of loaded modules. You should see gcc on the list now. You can check the list of all available modules by typing

```
module avail
```

You can also read more on the module system here: https://www.boisestate.edu/rcs/cluster-guides/loading-software/

Now you are ready to compile your hello program. From the directory where you have saved your hello_world.c, type:

```
gcc hello_world.c -o hello
```

which will take the code you have in `hello_world.c` and create an executable file called `hello`.

5. Run the program using the submission system

On your own unix-based computer, you can simply type the executable program name (i.e. `./hello`) and run it. On a cluster computer like R2, where there is many other users, all wanting to used shared resources, you have to submit a job to a queuing system, which decides which job is executed when. Before you do that, take a look at the queue by typing `squeue`. R2 will display the current queue. The currently running jobs will have time and resources assigned, while jobs waiting to be run will have a reason for the wait displayed in the last column. Sometimes it will be the lack of resources requested by the job (i.e. more processors than is currently available), and sometimes it will be the priority, meaning the person requesting resources is lower in priority queue than others. This may happen because that person has already used a lot of resources and others get priority at the moment. You will also see a column for “partition”, which means which queue the job was submitted to. The default queues `defq`, but we will mostly use `eduq` and `shortq`, as they have much higher turnaround time and priority than others.

To submit a job to a queue you first need a submission script. Open the file `run_script_hello.bash` using your favorite text editor. You will see a bunch of parameters which can be adjusted.

`#SBATCH --time=00:10:00` - how much time we need for the job (hh:mm:ss)
`#SBATCH --tasks=1` - how many processes (cores) we need
`#SBATCH --partition=shortq` - which queue to submit to
`#SBATCH --job-name=hello` - what is the job name visible in the queue
`#SBATCH --output=hello.o%j` - file name to which direct output of the program. The %j will be replaced by a job number assigned by the system, so you will have a different output for each run of the same program

The line below will run the program hello using only one processor. The number of processors is defined by the `-np` option. We cannot ask for more processors than what we have defined in the `--tasks` option above.

```
mpirun -np 1 ./hello
```

Submit the script by typing

```
sbatch run_script_hello.bash
```

If you are quick, you may see your job either waiting in the queue or being run by typing `squeue`.

Once the job is completed, you should see a new file `hello.oxxxxxxx`, where instead of `xxxxxxx` you will see the job number. Open the file using a text editor, or simply check the contents using

```
tail <file_name>
```

You should see a message you put in the `hello_world.c` code printed in that file. There may be some warning messages as well, or possibly error messages if something went wrong.

6. Use multiple processors

Now time to have some fun. Edit the run script and change both `--tasks` and `-np` options to a number of processors that you want. I would not go too crazy, because if you ask for too many processors, your job will wait for a very long time. Try starting with something small, like `-np 4`, but feel free to experiment. Remember that you cannot ask to run the program with the `-np` option with more processors that you have asked for in the `--tasks` option, but it is ok (if a bit wasteful) to run with fewer processors than you asked for. The queuing system will allocate resources (and decide on the wait time) based on the `--tasks` value.

Check how does the output file looks like for this job. How many times did the message appear? Why do you think that is?