

# Homework – Bucket Sort

## Brad Peterson – Weber State University

This homework modifies an existing bucket sort to support multithreading for further speedup. Notable features include:

- Creating and destroying child threads with a fork/join model
- Using a mutex to manage a critical region of code to avoid race conditions
- Create a thread pool model to distribute out work more fairly to any thread ready to work

You need to complete the assignment by implementing three methods inside `bucketSort()`

- **Step 1 – Copying values from the array into the buckets**
- **Step 2 – Sort each bucket**
- **Step 3 – Copy values from all buckets back into the original array**

This assignment focuses only on step 2. No modification (including multithreading) occurs on Step 1 or Step 3. Only Step 2 will utilize parallelism.

### Instructions:

#### Preparation:

Implement a `Step1()`, `Step2()`, and `Step3()` as described in the homework video. Keep `Step2()` in a single threaded approach initially (just a single loop that sorts every bucket). Once you get that working, you can then begin the parallel part.

#### Fork/Join:

Begin

Within `bucketSort()`, after `Step1()` reset the global variable `currentBucket` to 0.

Create an array of thread tracking objects (see lecture). This array should be sized to the number of threads.

You no longer want the master/main thread to invoke `Step2()`. You instead need child threads to do so. Comment out any call to `Step2()`.

Fork and create child threads that start their lifetime in `Step2()`. This requires a for loop, then invoking `thread()`, and for the first argument, passing in the function name in which the threads start (make sure you just give the function name `Step2`, you don't need to give it the parentheses). A second argument into `thread()` is optional, and is mainly useful if you want for debugging purposes to know which thread is doing what work. If you want this debugging feature, pass in the loop iteration variable as the second argument of `thread()`, and likewise modify `Step2()` so it has a parameter to accept that argument. The returned value of each `thread()` call is stored in the appropriate index of the thread tracking object array created from the prior paragraph.

After you fork the threads, create another for loop to join the threads. For each object in the array of thread tracking objects, call its `join()` method.

### **Thread Pool Model in `bucketSort()`:**

Each thread will be running independently in `bucketSort()`. The goal here is for each thread to obtain a valid bucket on which to work, sort that bucket, then loop around again to find a next valid bucket on which to work, etc. Create a simple int variable for a thread's bucket index (each thread gets its own variable independent from the others). Initialize that index value to zero. Create an infinite loop. Inside the loop lock the mutex called `myMutex`. Only one thread at a time can exist in this mutex code region. Copy the global variable's current working bucket index (`currentBucket`) into the thread's bucket index. Increment the global variable bucket index (`currentBucket`). Unlock the mutex. Check if this thread's bucket index is a valid bucket index. If it's not, return out of the function, this thread is done as there is no work left to be found. If it is valid bucket index, have this thread sort that bucket.

### **Compiling:**

On some build systems which use `g++` or `clang`, you may need to include the compiler flag `-lpthread`. For Visual Studio and Microsoft's compiler, no extra work is needed.

### **Debugging:**

Debugging is trickier for multithreaded programs. I strongly recommend using `printf()` and printing messages to screen to indicate the status of what your code is doing at that moment (don't forget the `\n`). The `cout` tool is awful in a multithreaded context, as its output tends to have race conditions and smear with other threads' `cout` output.

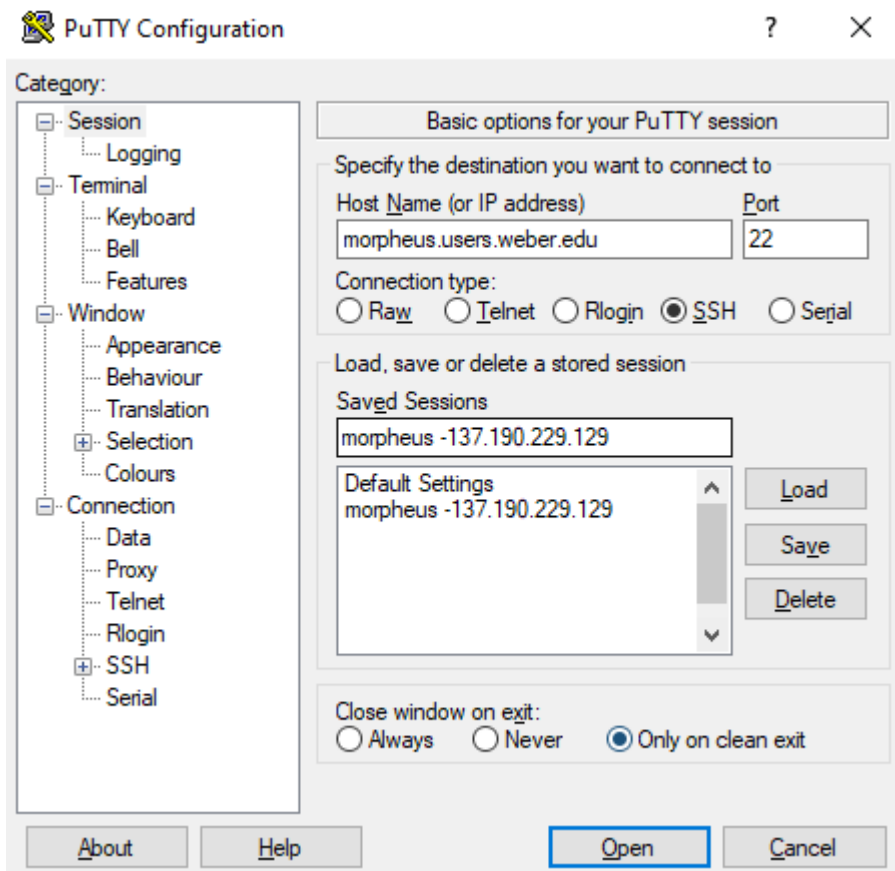
You can use the visual step-by-step debugger, but depending on the IDE, trickier things can happen as you step over or step into.

I also noticed many students were not taking advantage of `printAllBuckets()`. I would recommend calling `printAllBuckets()` in `bucketSort()` after the join step is complete.

### **Running:**

Run the assignment locally. Once you are confident it works, then change it to an optimized mode. (Release mode in Visual Studio or the `-O3` flag for those using a command line approach with `g++` or `clang`.) Take a screenshot of the result of your best program times.

Next, you must run your program on a high-performance server. The server is behind Weber's firewall at the address is `morpheus.users.weber.edu`. I assume most of us are not inside Weber's firewall, but outside. Thus, the easiest solution to connect to the server behind the firewall is to install the Weber State VPN tool ([https://www.weber.edu/help/kb/VPN\\_Install.html](https://www.weber.edu/help/kb/VPN_Install.html)), and then log into the VPN with your normal Weber State credentials. Once the VPN connection is running, I recommend using Putty (<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>) or another ssh tool to verify you can connect to `morpheus.users.weber.edu`. A screenshot of Putty is found below:



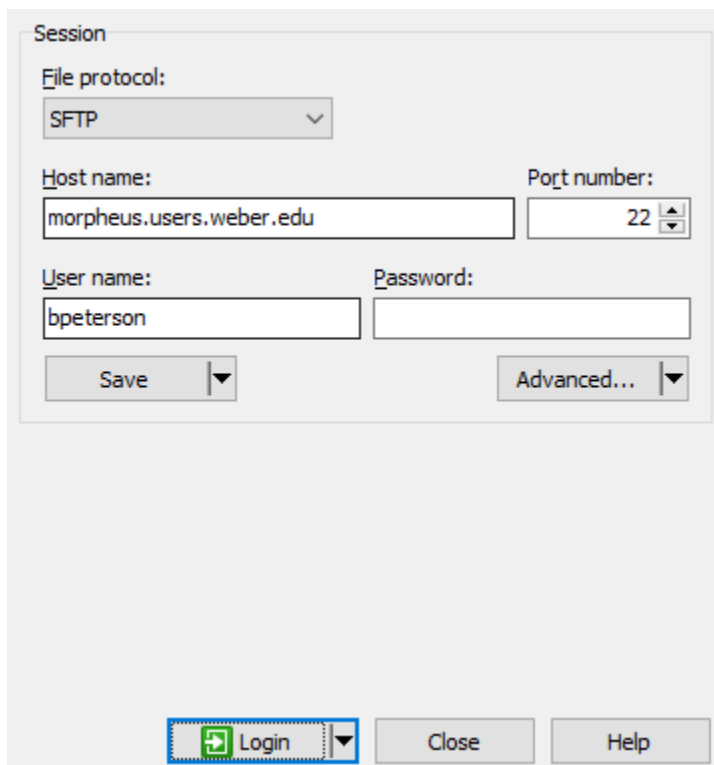
After entering the IP address and clicking Open, the first time your ssh tool should confirm to save a set of keys, just accept that. (Note that ssh does not display any typed password characters, even as asterisks.)

Your username is your initials (lower case) followed by the last 5 digits of your W#. For example, if your name is John Smith and your w# is w00012345 then your username would be js12345. Your password will be your first name (first letter capitalized) followed by "cs!". For example, if your name is John Smith and your w# is w00012345 your password would be Johncs!. One important thing to take note of here is that the usernames and passwords are created from registration data. Try your full first name instead of short versions, i.e. Kimberly instead of Kim.

Once you verify you can log into the server via a command line prompt, you should see a screen like this.

```
bpeterson@morpheus: ~  
Memory usage:      0%  
Swap usage:        0%  
Temperature:       35.0 C  
Processes:         745  
Users logged in:   1  
IPv4 address for enp5s0: 137.190.229.129  
IPv6 address for enp5s0: 2001:1948:216:10:aa5e:45ff:fe3e:6bb1  
  
* Introducing self-healing high availability clustering for MicroK8s!  
  Super simple, hardened and opinionated Kubernetes for production.  
  
  https://microk8s.io/high-availability  
  
138 updates can be installed immediately.  
0 of these updates are security updates.  
To see these additional updates run: apt list --upgradable  
  
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your  
Internet connection or proxy settings  
  
*** System restart required ***  
Last login: Sat Nov 14 01:49:41 2020 from 137.190.250.192  
bpeterson@morpheus:~$
```

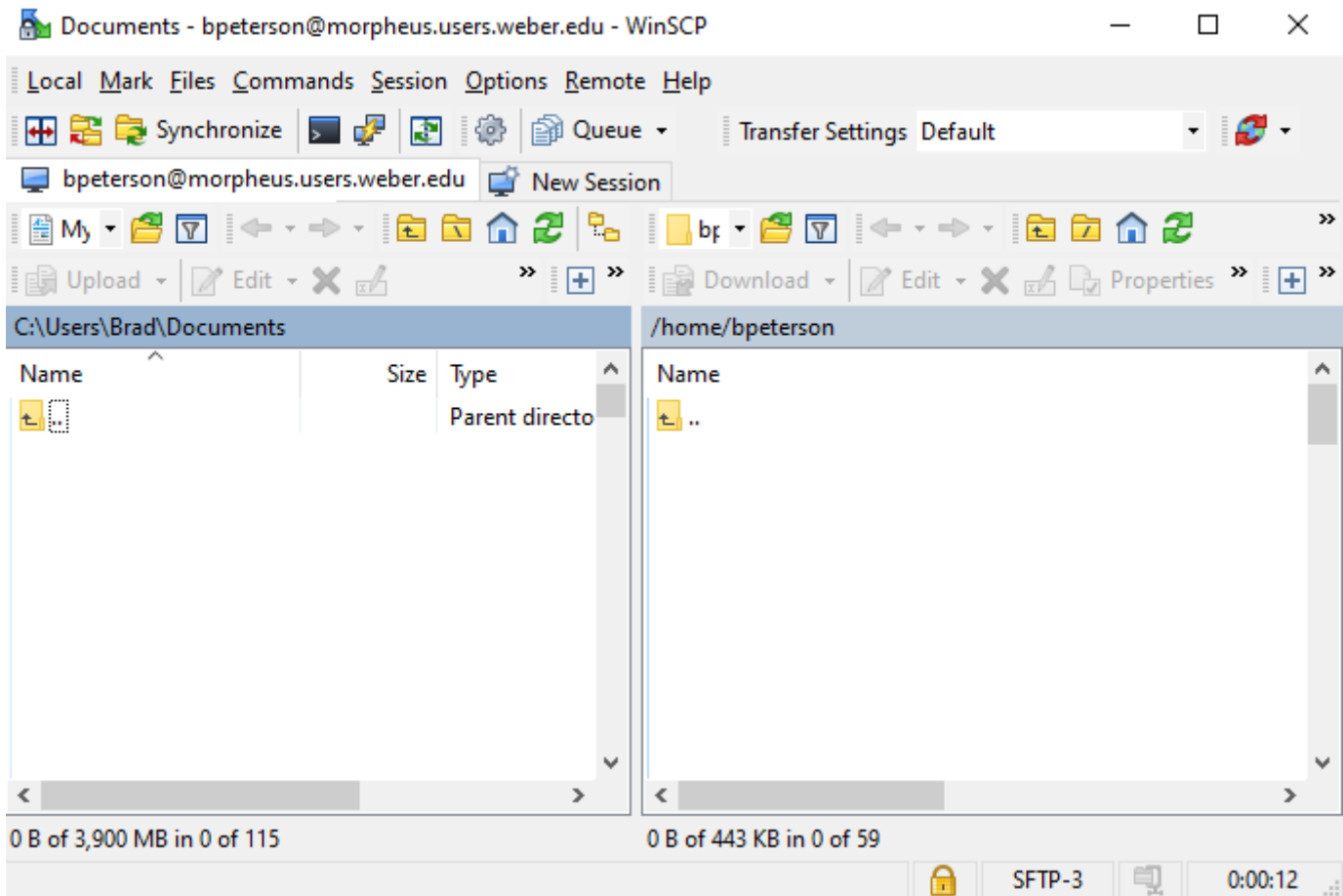
Your next goal is to copy your cpp file over to this server. WinSCP and FileZilla are both great GUI tools to help transfer the data from your machine to the server. You can also use a command line approach, which I'll describe after. For WinSCP or FileZilla make sure it is set to SFTP or SCP (it just can't be FTP), and then put in the IP address. An example of valid WinSCP settings is given below:



The image shows the WinSCP 'Session' dialog box. It contains the following fields and controls:

- File protocol:** A dropdown menu set to 'SFTP'.
- Host name:** A text box containing 'morpheus.users.weber.edu'.
- Port number:** A spinner box set to '22'.
- User name:** A text box containing 'bpeterson'.
- Password:** An empty text box.
- Buttons:** 'Save', 'Advanced...', 'Login' (with a green icon), 'Close', and 'Help'.

An example of a connection through WinSCP is shown below:



From here, copy your cpp file from your local machine to the server (the right side).

If you don't want to use a GUI tool to copy the file, and your local machine is running Linux or Mac, this command line statement will copy it over for you:

```
scp myAssignmentFile.cpp myaccount@morpheus.users.weber.edu:~
```

Now go back to your command line tool that's connected to the server (Putty or ssh), and verify that the assignment is there by simply running the Linux list command `ls`. You should see your cpp file in there.

Once the file is copied over, you can now compile your program with:

```
g++ yourfile.cpp -O3 -o yourfile.x -lpthread
```

1. `g++` is the compiler on the server. The file `yourfile.cpp` is your cpp filename. The `-O3` is for high optimization. Make sure it is `-O3` with a capital O (not a zero). The `-o` means output to this file. The `yourfile.x` is the name of the program it built, and you can name this whatever you want. The `-lpthread` means to link against the thread library.
2. A filename with spaces requires you enter it in like so: `g++ Bucket\ Sort\ summer\ 2020.cpp -O3 -o myprogram.x -lpthread`. Spaces must be escaped with a backslash.

Run the program: `./yourfile.x`

**Reporting:**

Create a document that has two screenshots. One from the output of running your homework on your local machine. A second screenshot of the output from this CS Linux server. Make sure you are on RELEASE mode in Visual Studio and you used the -O3 flag when using the g++ or clang compiler.