Greg Matthews
February 2nd 2018

CS 543
Homework #2 – Questions

1. Could you write a command line shell using threads rather than processes? Why or why not? What would be the advantages and disadvantages of this approach?

   Answer:

   Yes, you could write a command line shell using threads, which can be implemented by using the Pthreads, OpenMP, and SSE libraries in C, and even Nvidia's CUDA API using GPU threads. A shell can use these threads to perform operations in parallel, meaning each thread works concurrently to execute a given operation. The one major difference when comparing the use of threads to processes, is that each threads address space is shared amongst threads, meaning all data, variables, and code is shared, therefore utilization of semaphores and mutexes would be necessary if multiple threads are trying to access the same resource while running concurrently, which wouldn't be an issue when using processes by calling fork() because the entire address space is copied, so each forked process has an independent duplicate address space. The main drawback when using processes over threads is the overhead associated with copying an entire address space, rather than having "lightweight" threads that share the same memory, while having separate stacks.

2. How do you resolve name collisions between internal shell keywords and system programs? For example, if there is a system program called "alias" – how would you execute it?

   Answer:

   When given a command to execute, the shell would go through a list of different paths until it finds the necessary executable. The shell would first attempt to call the built-in command by checking internal shell keywords, if there's no match then it checks shell built-ins, then it finally searches for the command in **`$PATH`**. If one wants to call a system program called "alias" when there's already a builtin command called alias, then the shell would call the built-in because it checks there first. To explicitly call the system program that's located in some other directory, which let's say for example, is located in **`/other/directory/location/alias.exe`**, then at the shell prompt we would have to give the entire path name to get the alias system program by typing at the shell: **`other/directory/location/alias.exe`**

3. How would you run a shell under a different usercode? What would you have to require in order to do it? Would it be different just to run a command? (like sudo).

   Answer:

   To run a shell under a different usercode, in regards to Bash this can be done with the following command assuming new usercode is not root:

`/bin/su – username -s /bin/bash` where **`username`** is the different user you're trying to run the shell as, and **`-s /bin/bash`** invokes the Bash shell. To execute the shell as a different user, the password linked to that usercode will need to be supplied. The command would be much different if we were to invoke **`sudo`** overtop the previous command, because instead of asking for the password of the user you're trying to run the shell as, the command would instead ask for the root user's password, essentially bypassing the need to know the lower level user's password.