CSCI 5573 Homework Assignment 4 Spring 2022

Due Date: 11:59 pm, March 16 (Wed), 2022

1) Disclaimer

This homework is created based on our textbook, Operating Systems: Three Easy Pieces, written by Remzi and Andrea Arpaci-Dusseau at the University of Wisconsin.

2) Goal

This homework introduces you to a new tool, **vmstat**, and how it can be used to understand memory, CPU, and I/O usage. Read the associated README and examine the code in mem.c before proceeding to the exercises and questions below.

3) Requirement

You must test mem.c code and tools (**vmstat** and **swapon**) on the CSE department Linux cluster, csegrid.ucdenver.pvt. Download mem.c code and README from the class Canvas.

4) What to submit & how to submit

a) Answer all questions and add print output for each question.
b) Upload all required materials to the class Canvas.

5) Questions

1. First, open two separate terminal connections to the *same* machine, so that you can easily run something in one window and the other.

   Now, in one window, run **vmstat 1**, which shows statistics about machine usage every second. Read the man page, the associated README, and any other information you need so that you can understand its output. Leave this window running vmstat for the rest of the exercises below.

   Now, we will run the program **mem.c** but with very little memory usage. This can be accomplished by typing **./mem 1** (which uses only 1 MB of memory). *How do the CPU usage statistics change when running mem? Do the numbers in the user time column make sense? How does this change when running more than one instance of mem at once?*

2. Let's now start looking at some of the memory statistics while running **mem**. We'll focus on two columns: *swpd* (the amount of virtual memory used) and *free* (the amount of idle memory). Run **./mem 1024** (which allocates 1024 MB) and watch how these values change. Then kill the running program (by typing control-c) and

watch again how the values change. *What do you notice about the values? In* particular, *how does the free column change when the programexits? Does the amount of freememory increase by the expected amount when mem exits?*

3. We'll next look at the *swap* columns (*si* and *so*), which indicate howmuch swapping is taking place to and from the disk. Of course, to activate these, you'll need to run mem with large amounts of memory. First, examine how much free memory is on your Linux system (for example, by typing cat /proc/meminfo; type man **proc** for details on the /proc file system and the types of information you can find there). One of the first entries in /proc/meminfo is the total amount of memory in your system. Let's assume it's something like 8 GB of memory; if so, start by running **mem 4000** (about 1 GB) and watching the swap in/out columns. *Do they ever give non-zero values? Then, try with 5000, 6000, etc. What happens to these values as the program enters the second loop (and beyond), as compared to the first loop? How much data (total) are swapped in and out during the second, third, and subsequent loops? (do the numbers make sense?)*

4. Do the same experiments as above, but now watch the other statistics (such as CPU utilization, and block I/O statistics). *How do they change when mem is running?*

5. Now let's examine performance. Pick an input for mem that comfortably fits in memory (say 4000 if the amount of memory on thesystem is 8 GB). *How long does loop 0 take (and subsequent loops 1, 2, etc.)?* Now pick a size comfortably beyond the size of memory (say 12000 again assuming 8 GB of memory). *How long do the take here? How do the bandwidth numbers compare? How different is performance when constantly swapping versus fitting everything comfortably in memory? Can you make a graph, with the size ofmemory used by mem on the x-axis, and the bandwidth of accessing said memory on the y-axis? Finally, how does the performance of the first loop compare to that of subsequent loops, for both the case where everything fits in memory and where it doesn't?*

6. Swap space isn't infinite. You can use the tool ***swapon*** with the -s flag to see how much swap space is available. *What happens if you try to run **mem** with increasingly large values, beyond what seems to be available in swap? At what point does the memory allocation fail?*