CSC 369 Worksheet 6

August 19, 2020

Source: link

1 Homework (Code)

1. The first Linux tool you should check out is the very simple tool `free`. First, type `man free` and read its entire manual page; it's short, don't worry!

2. Now, run `free`, perhaps using some of the arguments that might be useful (e.g., `-m`, to display memory totals in megabytes). How much memory is in your system? How much is `free`? Do these numbers match your intuition?

3. Next, create a little program that uses a certain amount of memory, called `memory-user.c`. This program should take one commandline argument: the number of megabytes of memory it will use. When run, it should allocate an array, and constantly stream through the array, touching each entry. The program should do this indefinitely, or, perhaps, for a certain amount of time also specified at the command line.

4. Now, while running your `memory-user` program, also (in a different terminal window, but on the same machine) run the free tool. How do the memory usage totals change when your program is running? How about when you kill the `memory-user` program? Do the numbers match your expectations? Try this for different amounts of memory usage. What happens when you use really large amounts of memory?

5. Let's try one more tool, known as `pmap`. Spend some time, and read the `pmap` manual page in detail.

6. To use `pmap`, you have to know the process ID of the process you're interested in. Thus, first run ps auxw to see a list of all processes; then, pick an interesting one, such as a browser. You can also use your `memory-user` program in this case (indeed, you can even have that program call `getpid()` and print out its PID for your convenience).

7. Now run `pmap` on some of these processes, using various flags (like -X) to reveal many details about the process. What do you see? How many different entities make up a modern address space, as opposed to our simple conception of code/stack/heap?

8. Finally, let's run `pmap` on your `memory-user` program, with different amounts of used memory. What do you see here? Does the output from `pmap` match your expectations?