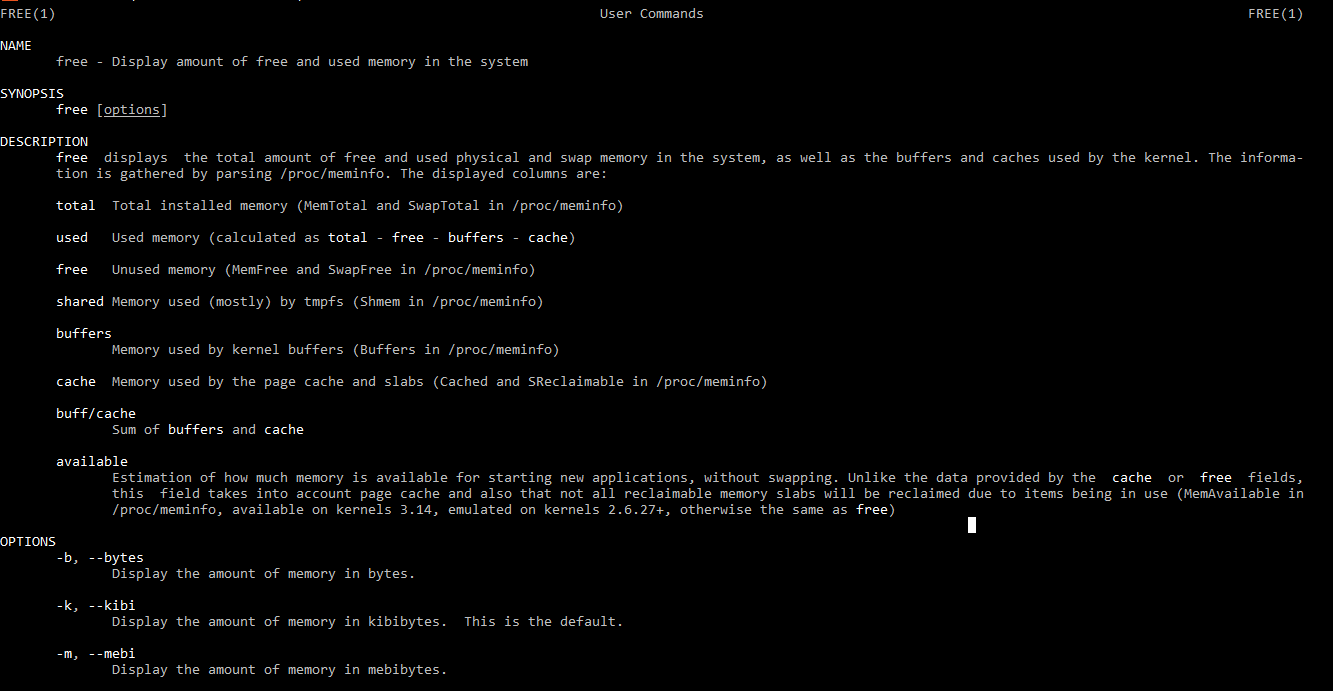


Lab Tasks: In this lab, we’ll learn about a few useful tools to examine virtual memory usage on Linux-based systems. This will only be a brief hint at what is possible; you’ll have to dive deeper on your own to truly become an expert (as always!).

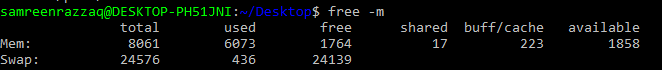
Tasks:

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; write down what you learnt.

Free Displays free and used physical and swap memory in the system. And when we parsing /proc/meminfo, is displays the free memory, total memory, used memory, unused memory, shared memory, memory use by Kernal buffers, memory use by page cache and slabs, sum of buffer and cache, available memory to start new application. And free also displays the amount of memory in bytes, kilobytes and mebibyes. We can see this:



1. 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?

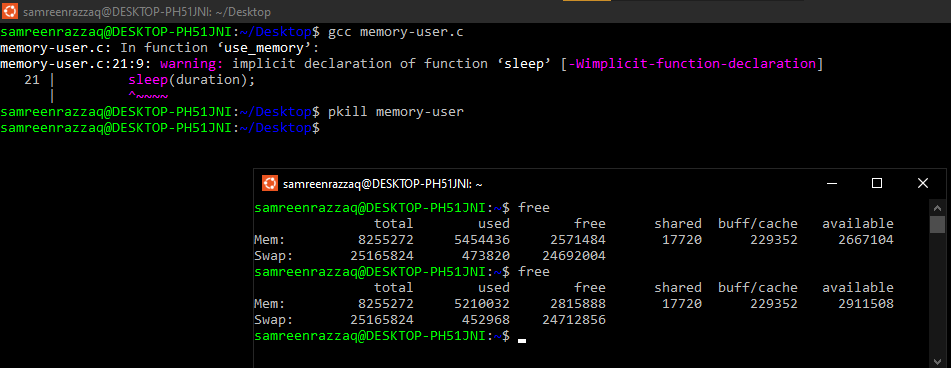
* Total memory: 8061, Used memory: 6073

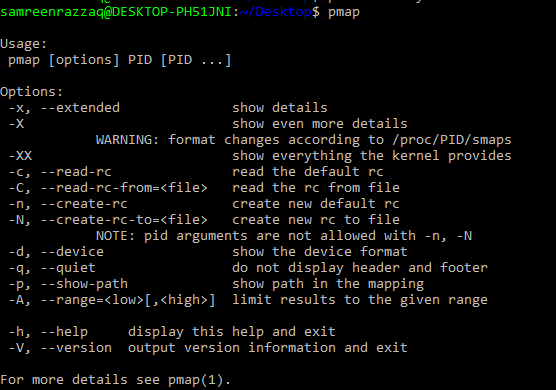
Yes! These values is matches with my system information.

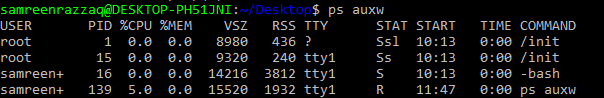
1. Now run C program memory-user.c, while running your memory-user program, also (in a different terminal window, but on the same machine) run the free tool. How the memory usage do totals change when your program is running? How about when you kill the memory-user program? Do the numbers match your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

* Memory usage in different terminal window = 5454436
* Memory usage when we kill the memory-user program = 5210032

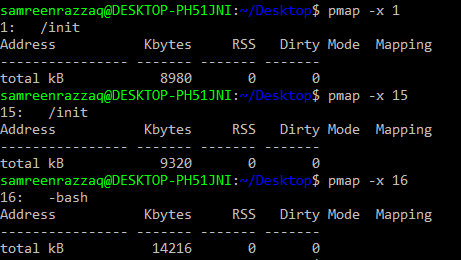
We see that usage memory is different in both cases. By killing the program, memory usage is less than without killing program.



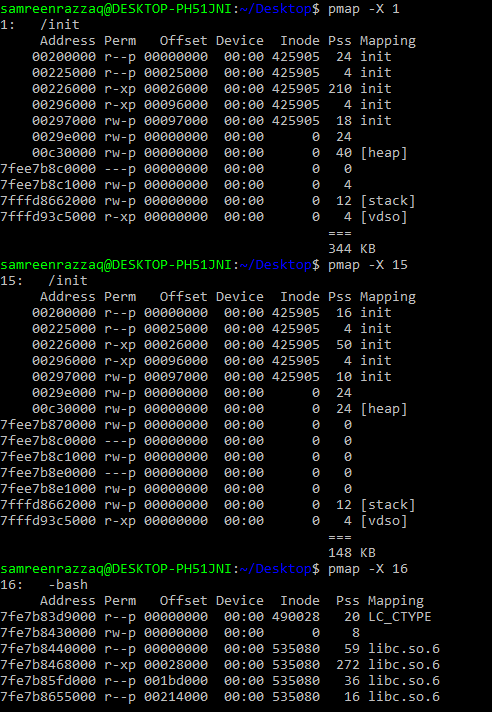
1. Let’s try one more tool, known as pmap. Spend some time, and read the pmap manual page in detail.
2. To use pmap, you have to know the process ID of the process you’re interested in. case (indeed, you Thus, first run ps auxw to see a list of all processes; then, pick an interesting one, such as a browser. You can also use your memory-user program in this can even have that program call getpid() and print out its PID for your convenience).

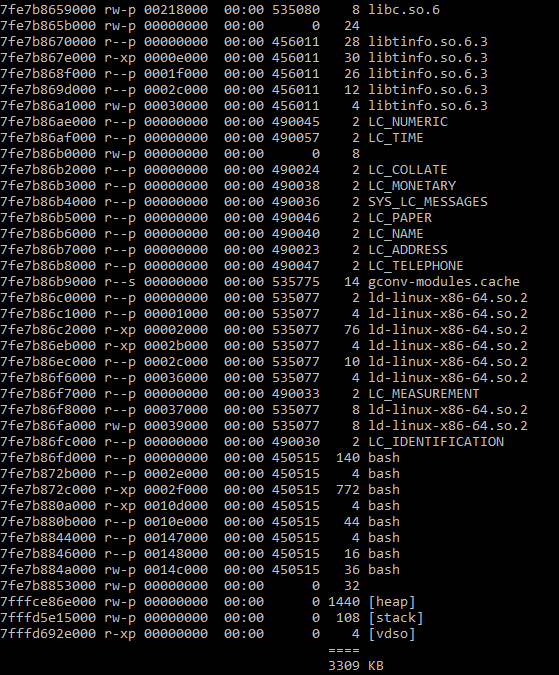


1. Now run pmap on some of these processes, using various flags (like -X) to reveal many details about the process. What do you see? How many different entities make up a modern address space, as opposed to our simple conception of code/stack/heap?

By using -x:

By using -X:





1. 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?

We see the changing of memory usage (in terabytes) in both cases (before and after running pmap with different amounts). And yes! pmap values matches the expectations. We can see:

