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?

The user time value increase, the time spend on kernel mode didn't change. We increase 12 user time per instance of mem.

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 program exits? Does the amount of free memory increase by the expected amount when mem exits?

When running ./mem 1024, the free memory will decrease 1024 MB, when kill the process the free memory increase 1024 MB. Yes the amount of free memory increase by the expected amount when mem exits.

3. We'll next look at the swap columns (si and so), which indicate how much 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 4 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?)

When run mem with 4GB, they still give me zero values in swap in/out columns. Because my free memory is 6+GB.

When run mem with 5GB, they still give me zero values in swap in/out columns. Because my free memory is 6GB.

When run mem with 8GB, they give me non-zero values in swap in/out columns. Because my free memory is 6+GB. Threre are total 2GB memory swap in and out. During the second and third are all 2GB, which 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?

Before running mem, the time spent waiting for IO is 0, the time spend running kernel is 0, time spend running non-kernel code is 0, the cpu idle percentage is 100%. After running mem, the time spent waiting for IO is 9, the time spend running kernel is 4, time spend running non-kernel code sometimes is 0, the cpu idle percentage is 87%

5. Now let's examine performance. Pick an input for mem that comfortably fits in memory (say 4000 if the amount of memory on the system 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 loops 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 of memory 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?

```
nick@ubuntu:~/CS5600_hw/week7/chapter21$ ./mem 4000
allocating 4194304000 bytes (4000.00 MB)
  number of integers in array: 1048576000
loop 0 in 3808.24 ms (bandwidth: 1050.36 MB/s)
loop 1 in 547.95 ms (bandwidth: 7299.94 MB/s)
loop 2 in 462.20 ms (bandwidth: 8654.26 MB/s)
loop 3 in 462.35 ms (bandwidth: 8651.53 MB/s)
loop 4 in 483.40 ms (bandwidth: 8274.69 MB/s)
```

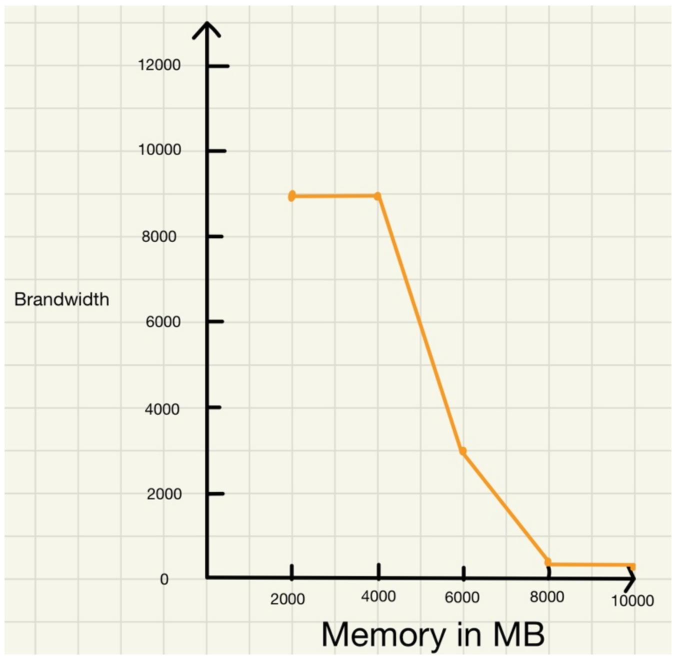
With ./mem 4000
The 0th loop takes 3808.24 ms, the subsequent takes 547.95ms, 462.20ms, 462.35ms etc.

```
nick@ubuntu:~/CS5600_hw/week7/chapter21$ ./mem 9000
allocating 9437184000 bytes (9000.00 MB)
  number of integers in array: 2359296000
loop 0 in 5887.29 ms (bandwidth: 1528.72 MB/s)
loop 1 in 43412.48 ms (bandwidth: 207.31 MB/s)
loop 2 in 45554.79 ms (bandwidth: 197.56 MB/s)
loop 3 in 45854.54 ms (bandwidth: 196.27 MB/s)
```

With ./mem 9000
The 0th loop take 5887.29 ms, loop 1 take 43412.48ms, loop2 takes 45554.79 ms, loop3 take 45854.54ms.

The bandwidth 80 times slower.

Fitting everything into the memory has better performance than constantly swapping.



For everything fits in the memory:
The first loop's performance is less than the subsequent loops.

For not fits in the memory:
Frist loop's performance is better than the subsequent loops.

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?

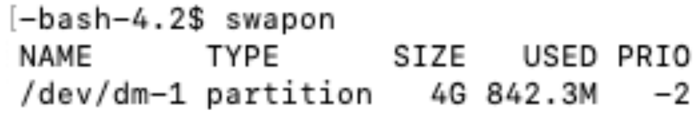
4 GB available swap space.

With ./mem 12000, OS kill the process

With ./mem 13000, memory allocation failed.

7. Finally, if you're advanced, you can configure your system to use different swap devices using swapon and swapoff. Read the man pages for details. If you have access to different hardware, see how the performance of swapping changes when swapping to a classic hard drive, a flash-based SSD, and even a RAID array. How much can swapping performance be improved via newer devices? How close can you get to in-memory performance?

Since I don't have hardware equipment. I also check khoury server only have following:



```
-bash-4.2$ swapon
 NAME      TYPE      SIZE   USED PRIO
 /dev/dm-1 partition   4G 842.3M   -2
```

Thus, in theory the swapping performance heavily rely on the speed persistence storage device. Then RAID array > SSD > hard drive.