**Operating Systems HW\_04**

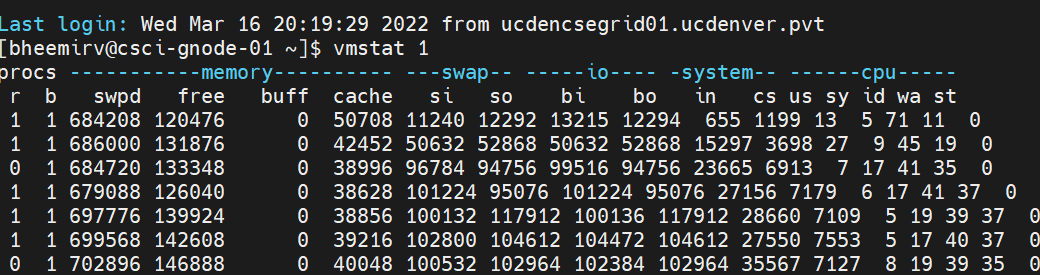
**Question 1:**

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).

**Sol:**

****

From the above table, the CPU Usage Statistics are,

Us - User Time - Time spent running non-kernel code

Sy - Time spent running kernel code.

Id - Idle time - Time spent idle.

Wa - waiting time - Time spent waiting for input or output

st - stolen time - Time stolen from a virtual machine. This is the time a virtual machine has to wait for the hypervisor to finish servicing other virtual machines before it can come back and attend to this virtual machine.

**How do the CPU usage statistics change when running mem?**

By observation, before my process entered into the node, there is another process already running in the node with **us** of 50 when I started running one mem instance **us** has increased to 100. In the same way, before my process started **si,so** are zero and even after my process entered they remained Zero, whereas **id** changed from 50 to 0 and **wa,st** remained zero throughout the process.

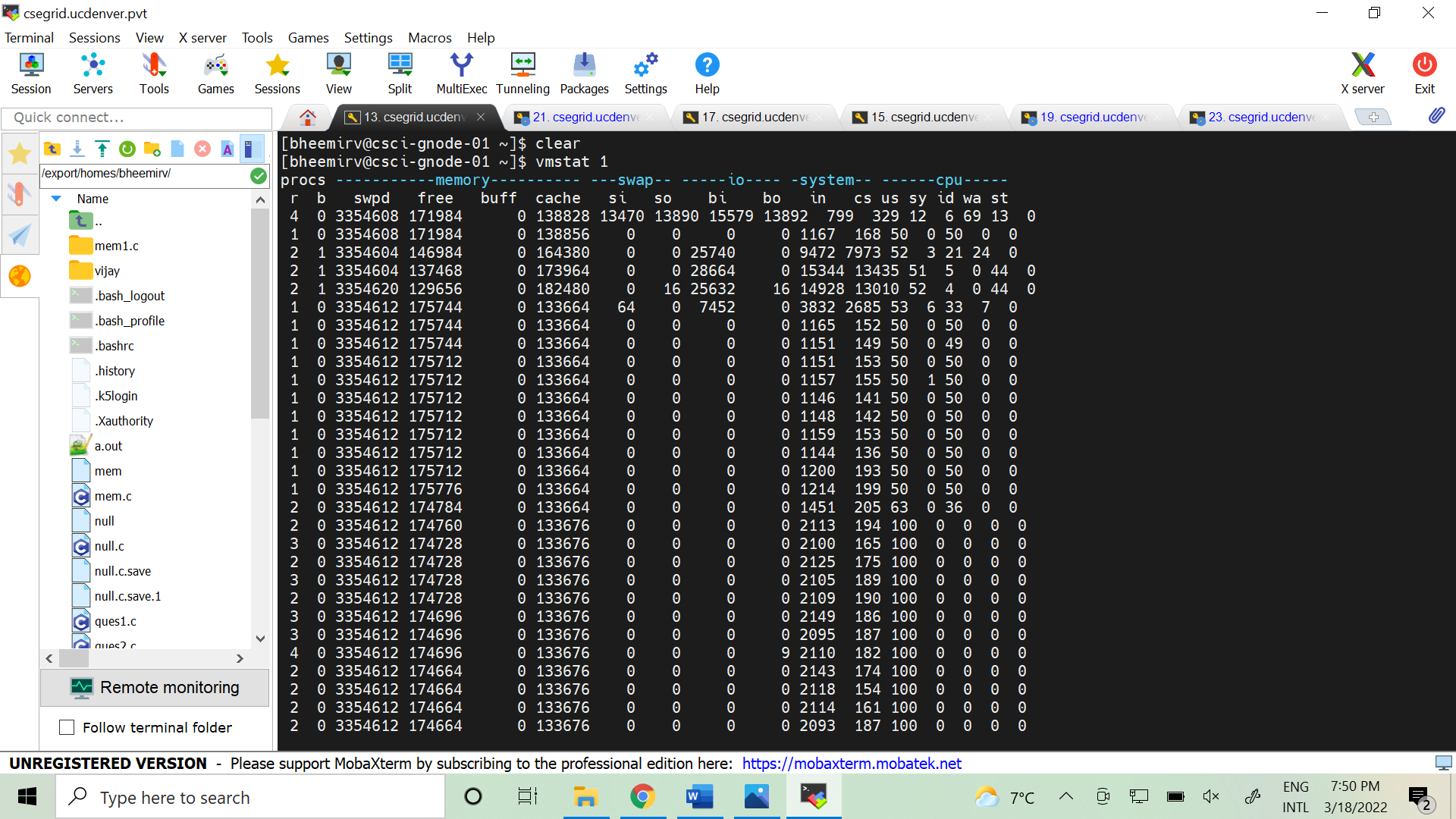
**Do the numbers in the user time column make sense?**

Yes, the numbers in the user time column make sense. This column tells us the amount of time that was taken while running the user code.

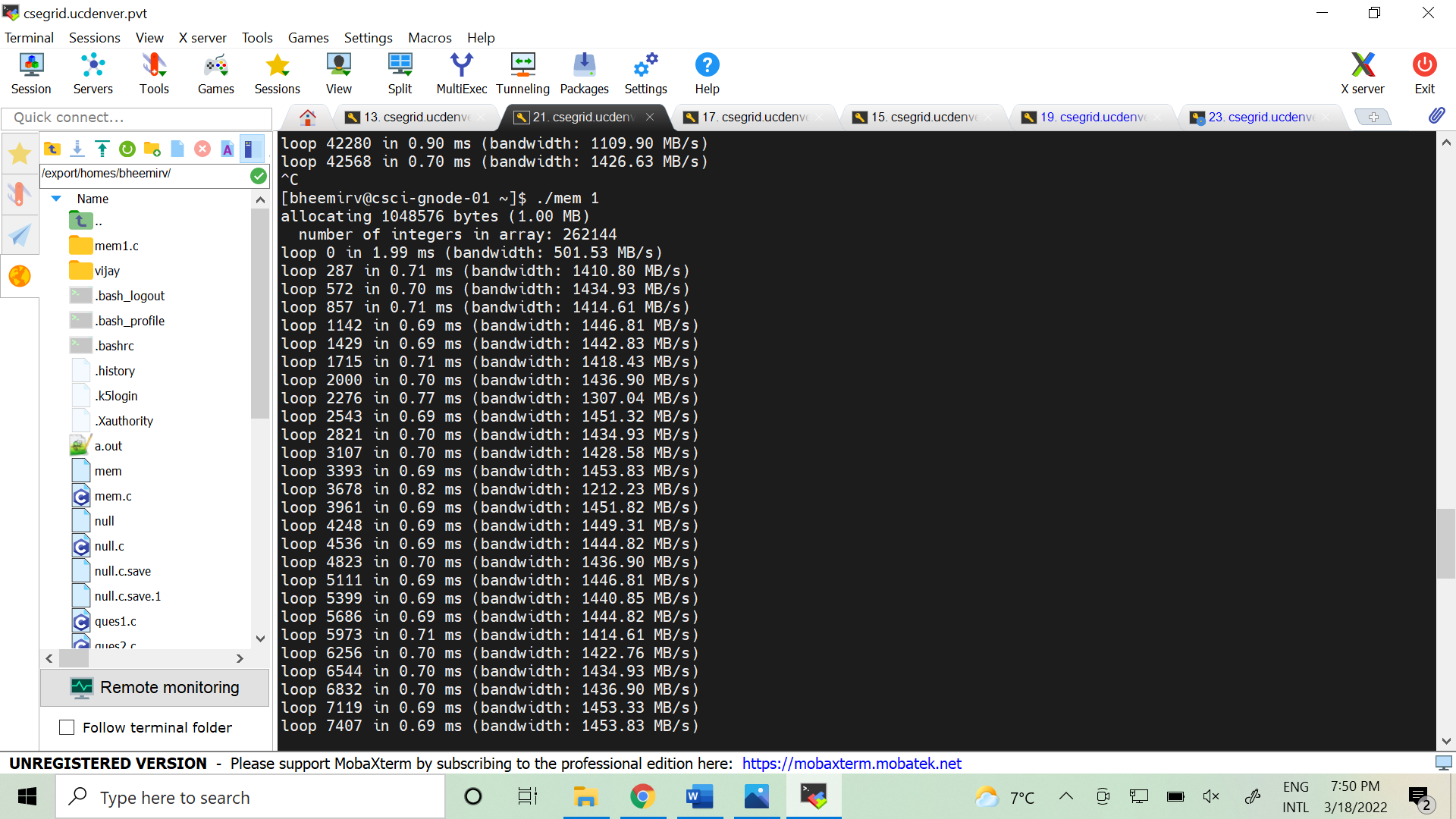
**How does this change when running more than one instance of mem at once?**

When we run more than one instance of mem 1 at a time then **us** column shows the constant value of 100 whereas sy, id, wa, st results in 0. Whenever there is an increase in swap in, there is a slight decrease in **us**.

Vmstat 1 :

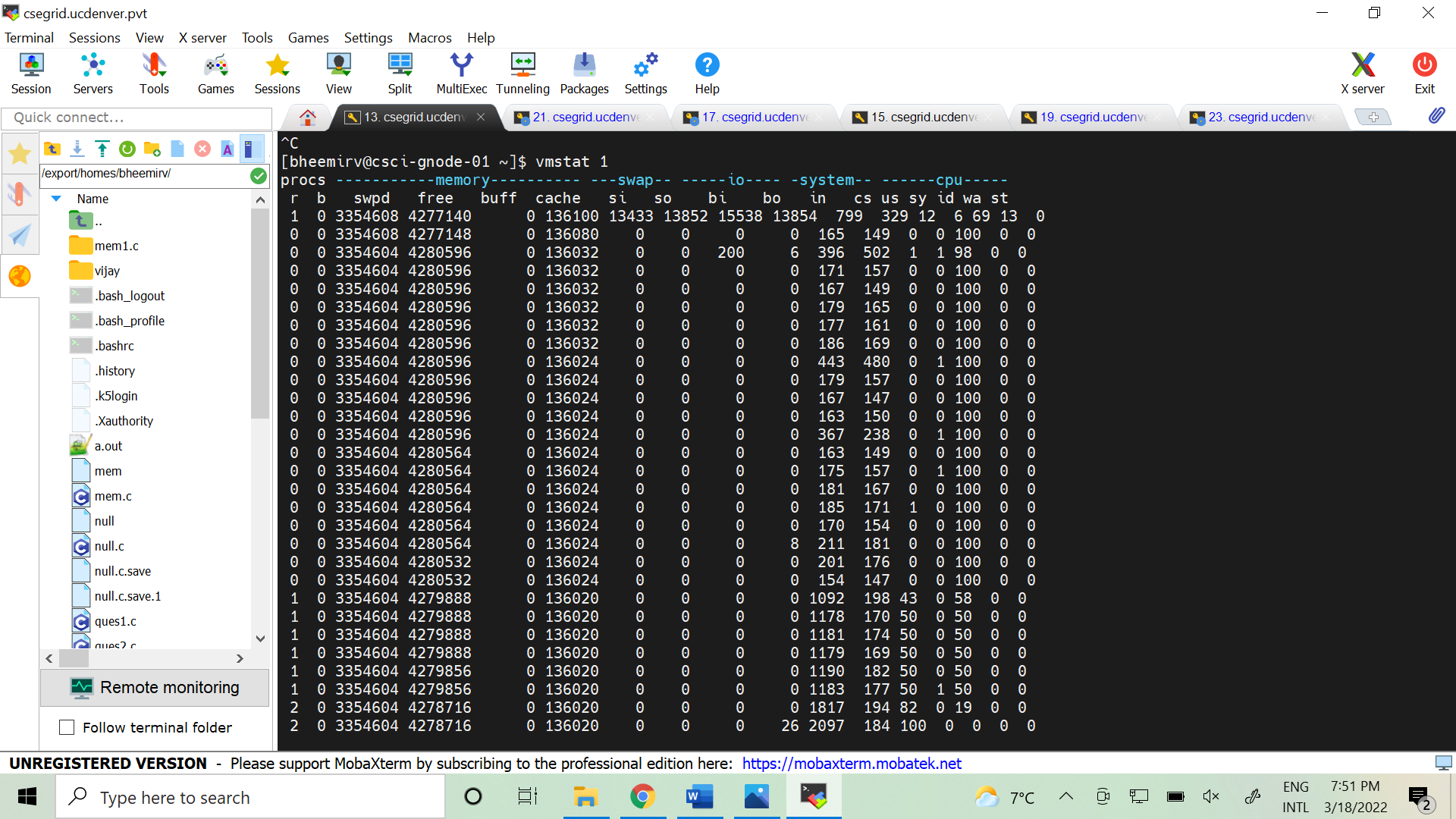


Screenshot 2

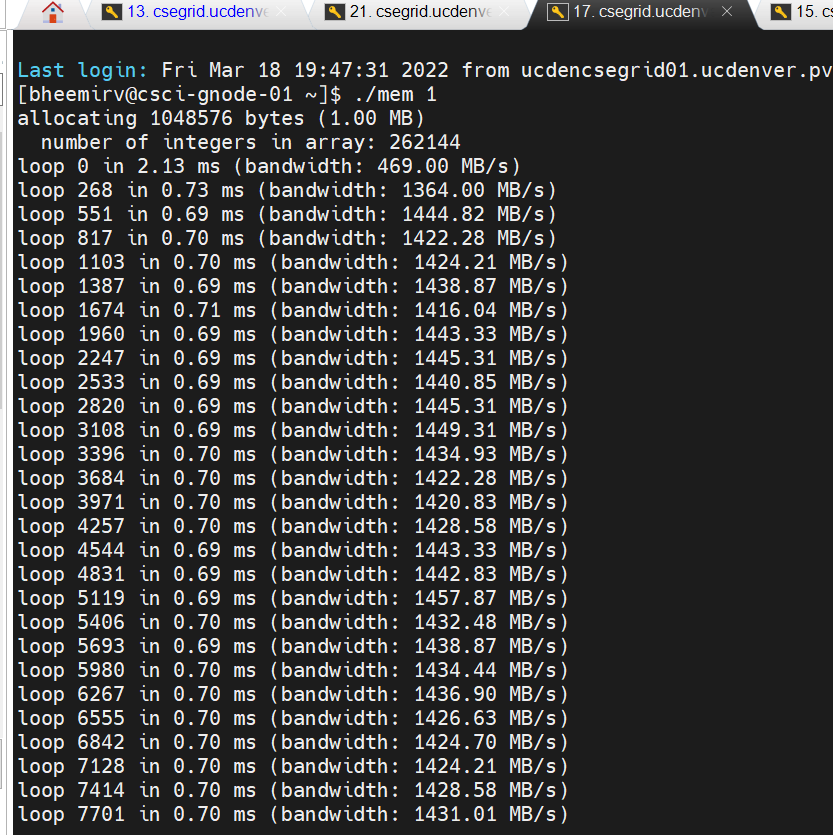


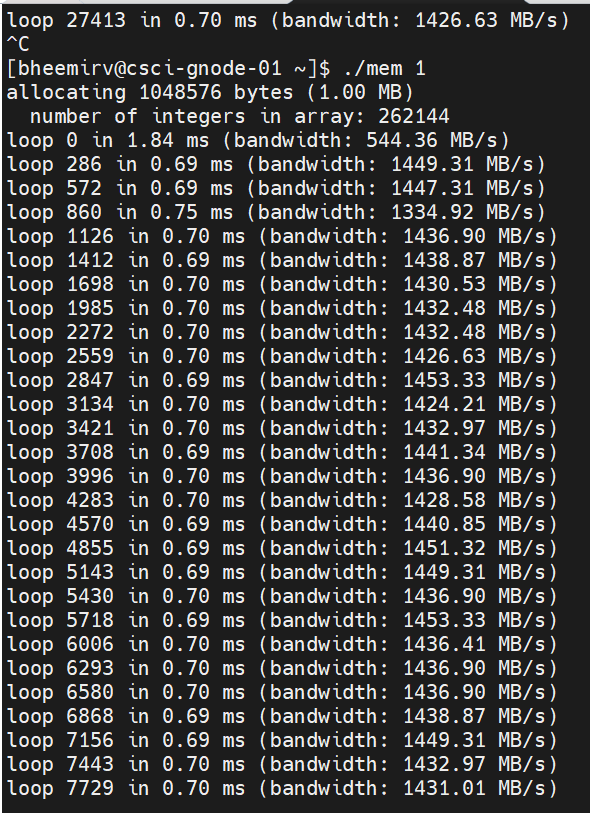
Instance 2:

Screenshot 1

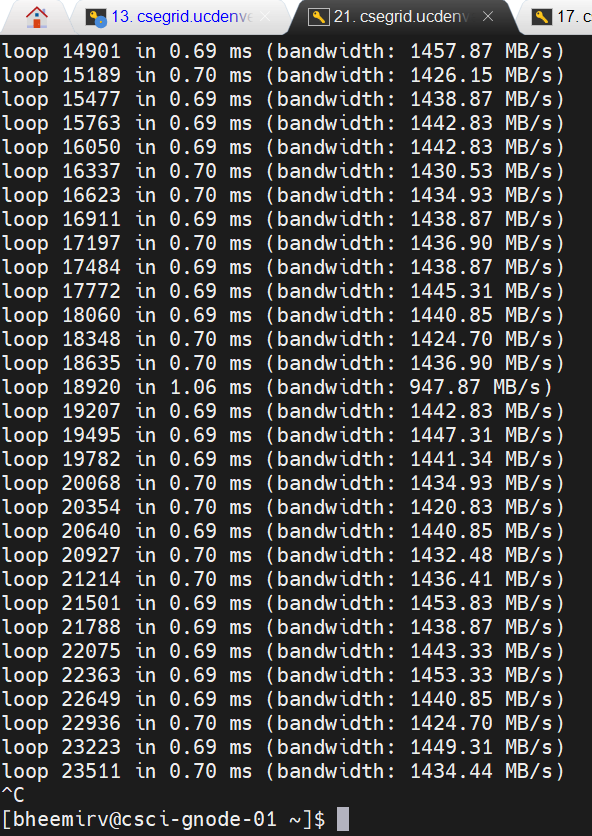


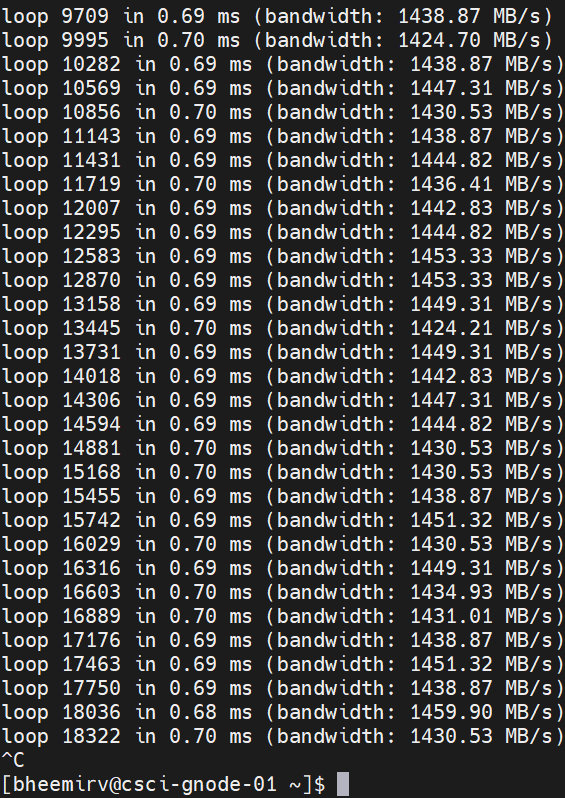
Screenshot 2





When Killed:





**Question 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.

Explanation:

**What do you notice about the values?**

When a single instance of mem 1024 is executed swpd remains constant (3354588).

When a single instance of mem 1024 is executed free memory start taking more space so it will come down from 2930888 to 2813384 and when the process is running it will come to 1878444 and buff and cache columns remain constant. Si & so - bi & bo columns remain zero.

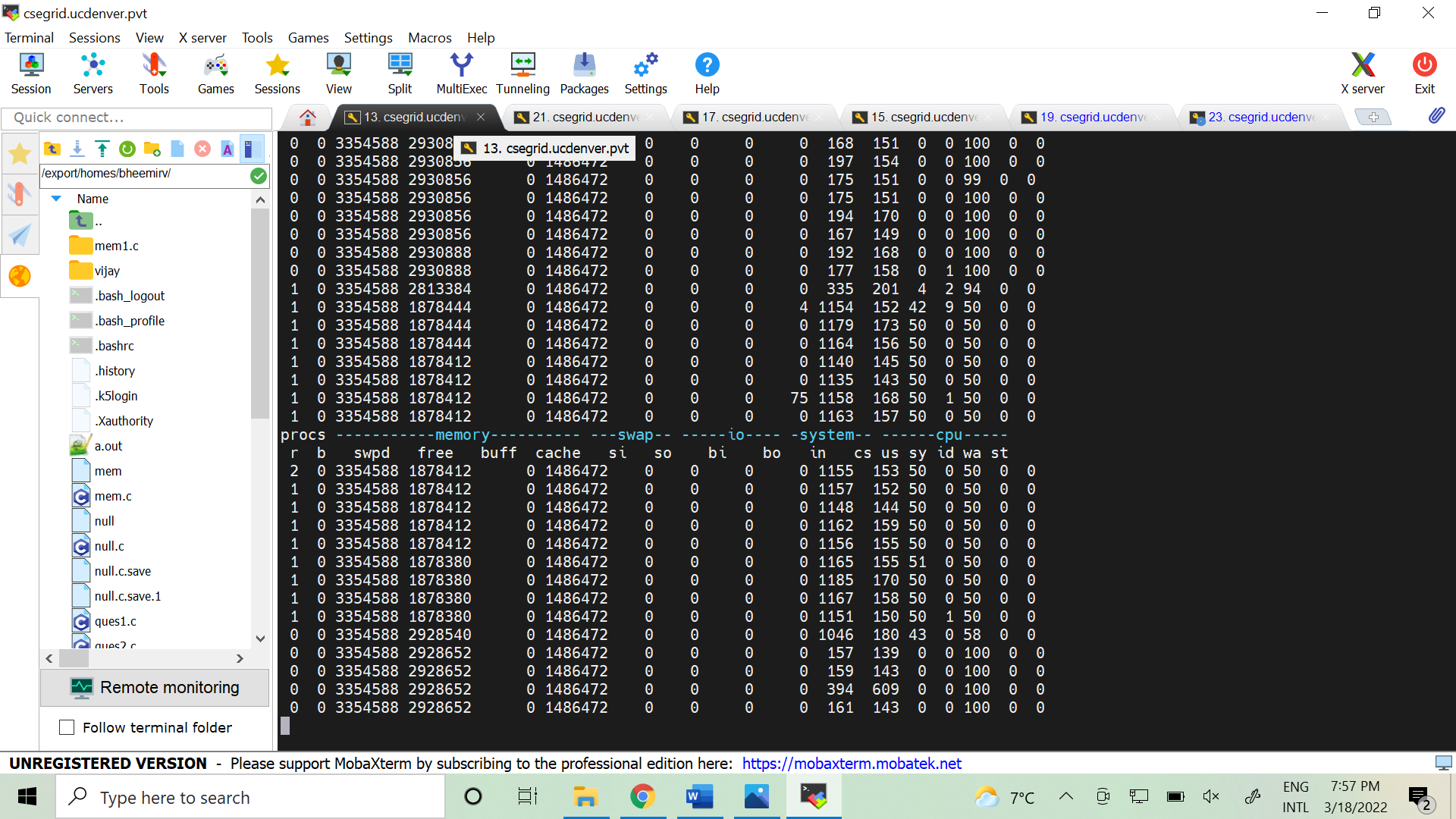
**In particular, how does the free column change when the program exits?**

When the program exits swpd remains the same, and free memory jumps from 1878380 to 2928540. Free memory gradually decreases when the processes running increase. Buff and cache remain constant. Si & so - bi & bo columns remain zero.

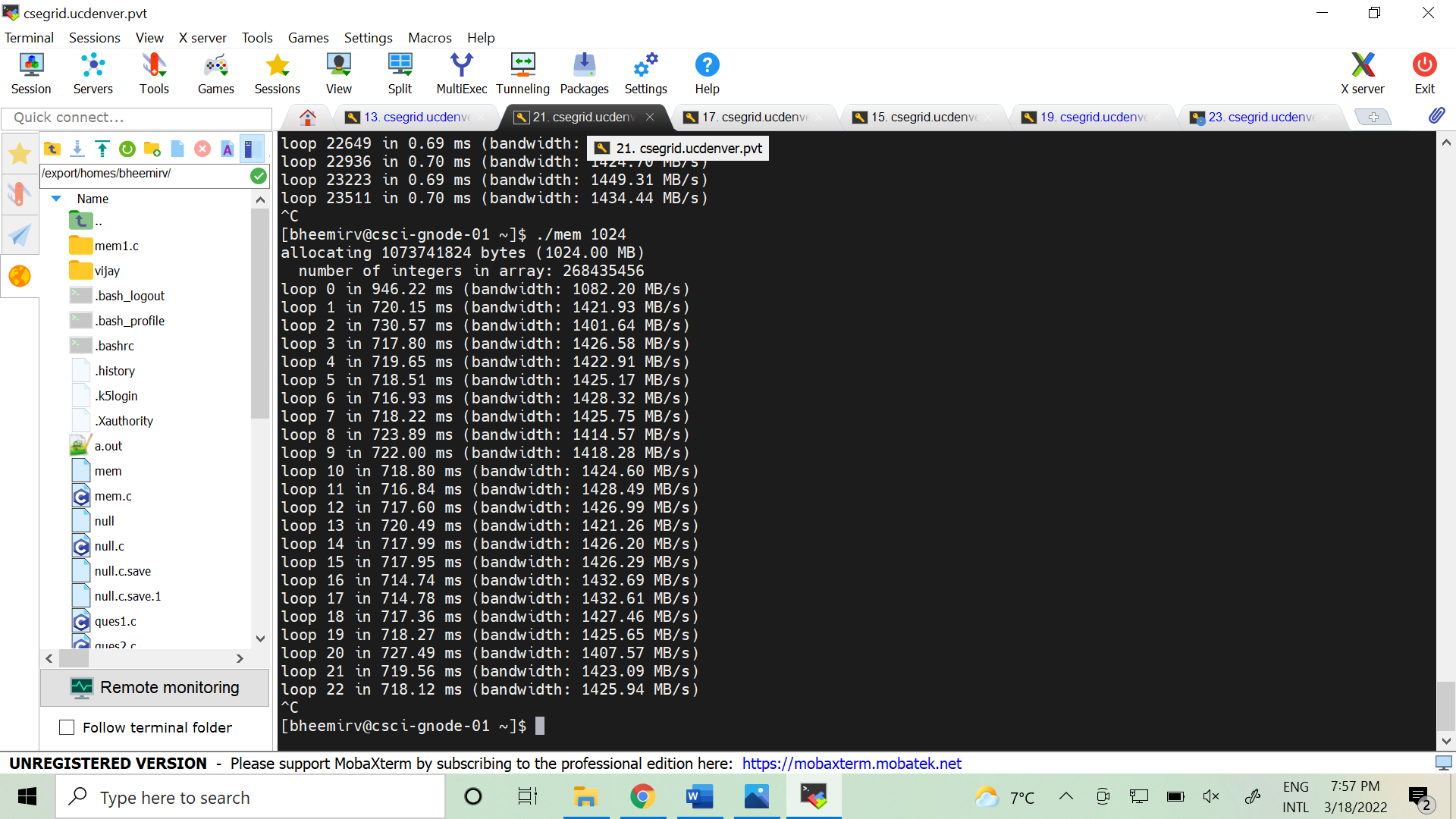
**Does the amount of free memory increase by the expected amount when mem exits?**

When we terminate the process, free memory increases but not by expected value.

Screenshot: vmstat for 1024



Screenshot: mem 1024

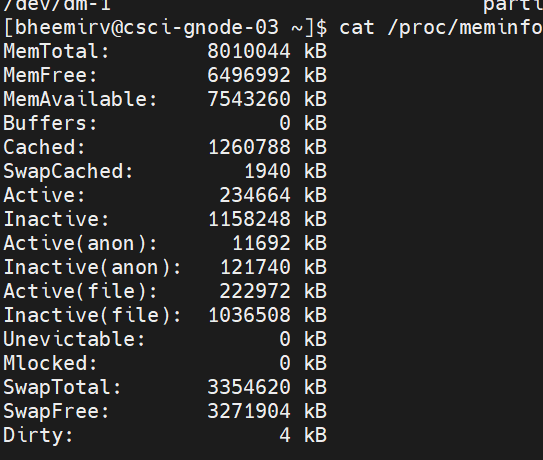


Question 3:

**Q3:** 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 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?)**

Ans:

When I run the command cat /proc/meminfo we got the statistics as,



Total memory is 8GB and free memory is approximately 6.5GB.

**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?)**

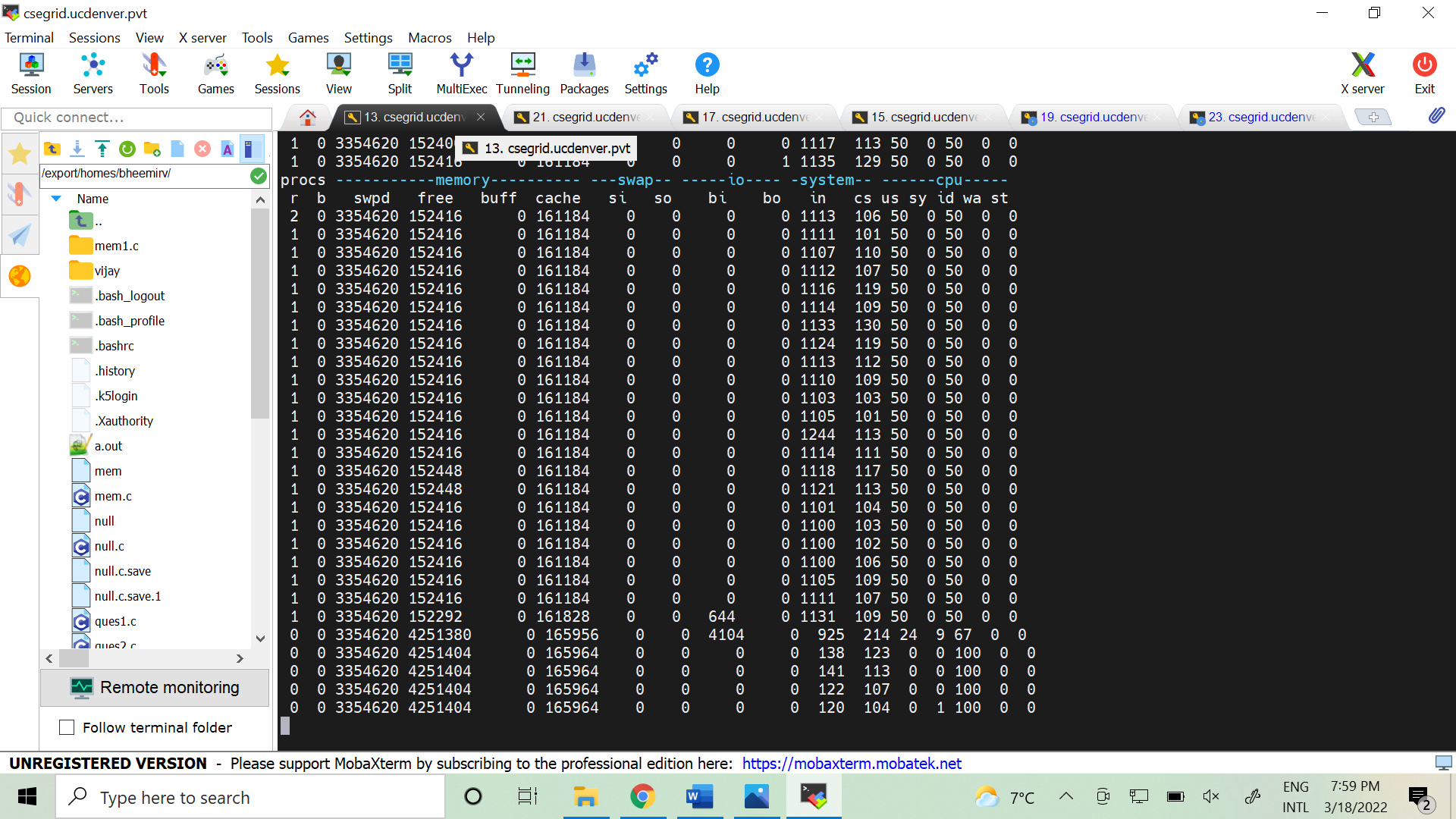
When I run the program using 4000, 5000, 6000 values I didn't get any non-zero values for si-so.

When the program enters Loop 2, Loop 3, and so on, for 4000, 5000, 6000 the si-so values remain constant with zero value.

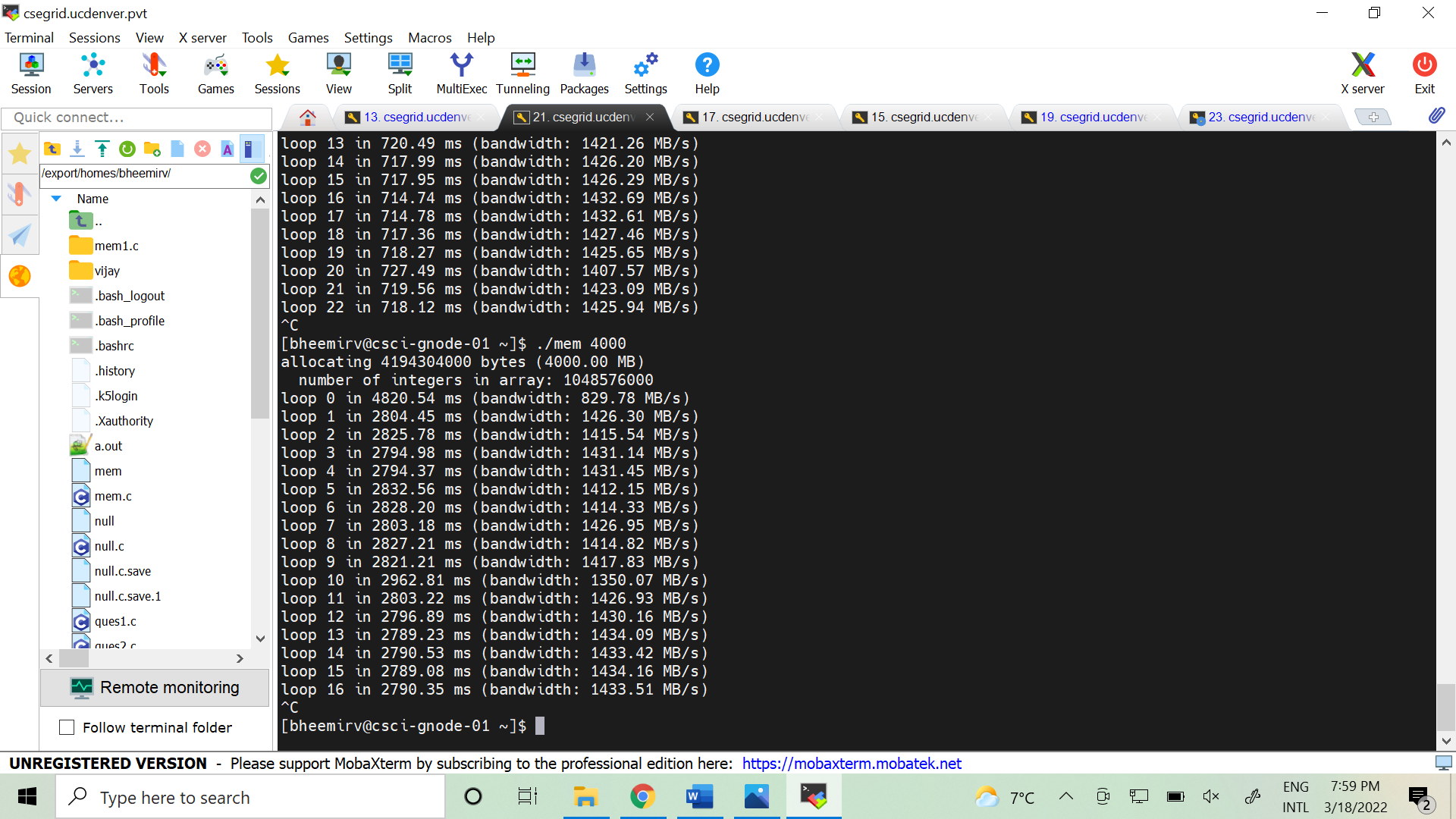
When the program enters Loop 2, Loop 3, and so on, for 4000, 5000, 6000 the data swapped in and swapped out is zero.

The numbers won’t have any order they are scattered.

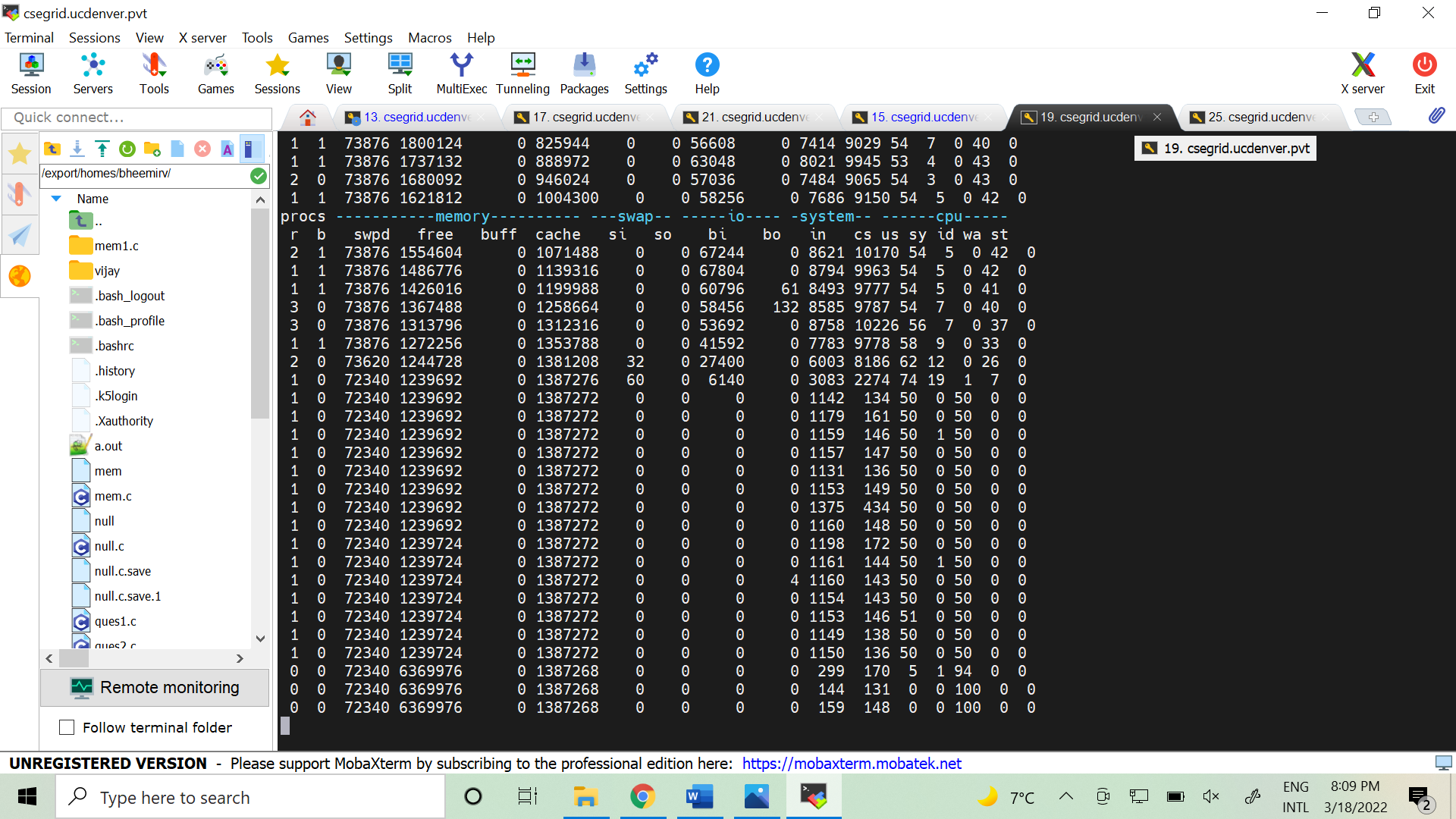
Screenshot: vmstat 1 (4000)



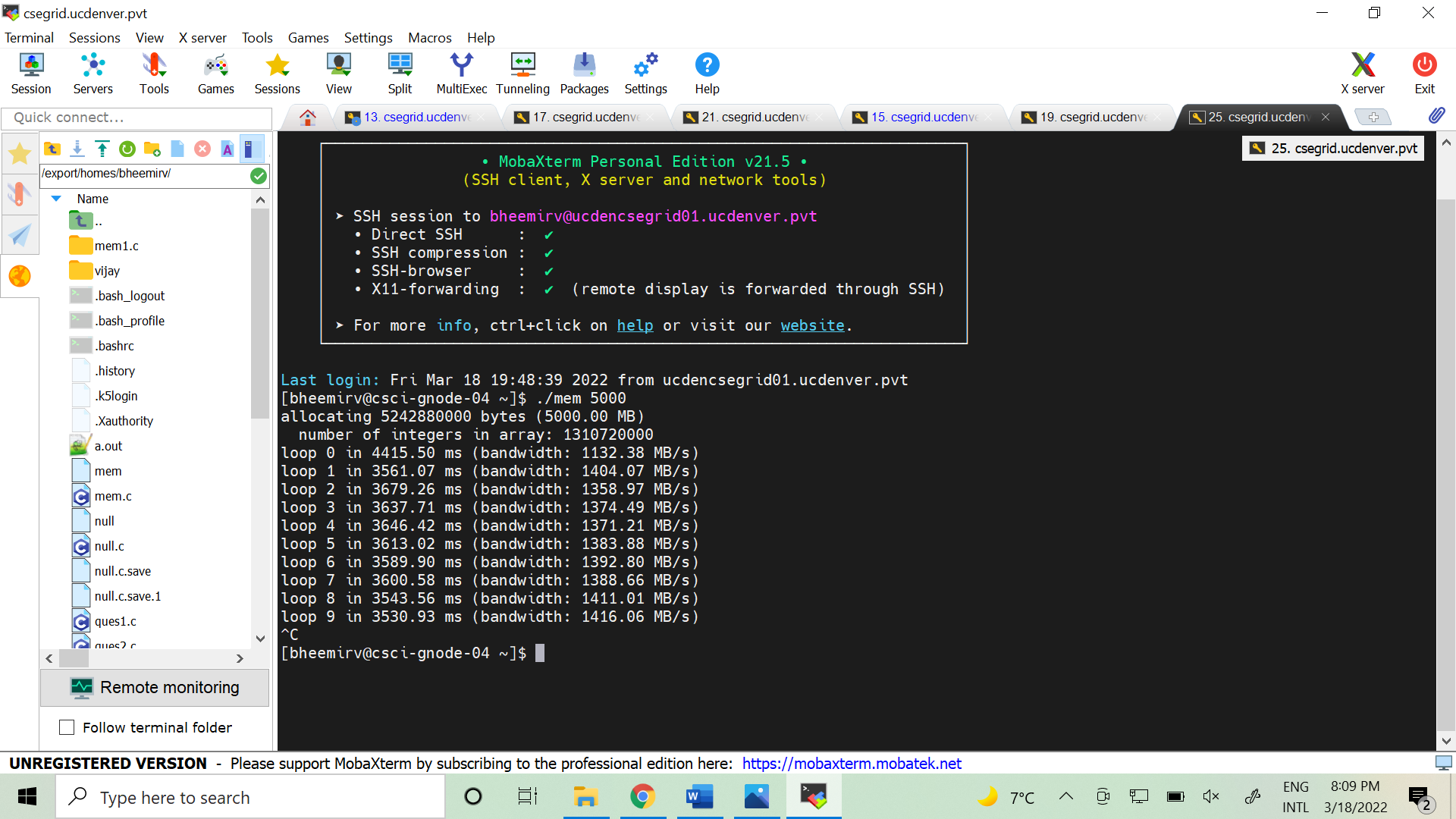
Screenshot: mem 4000



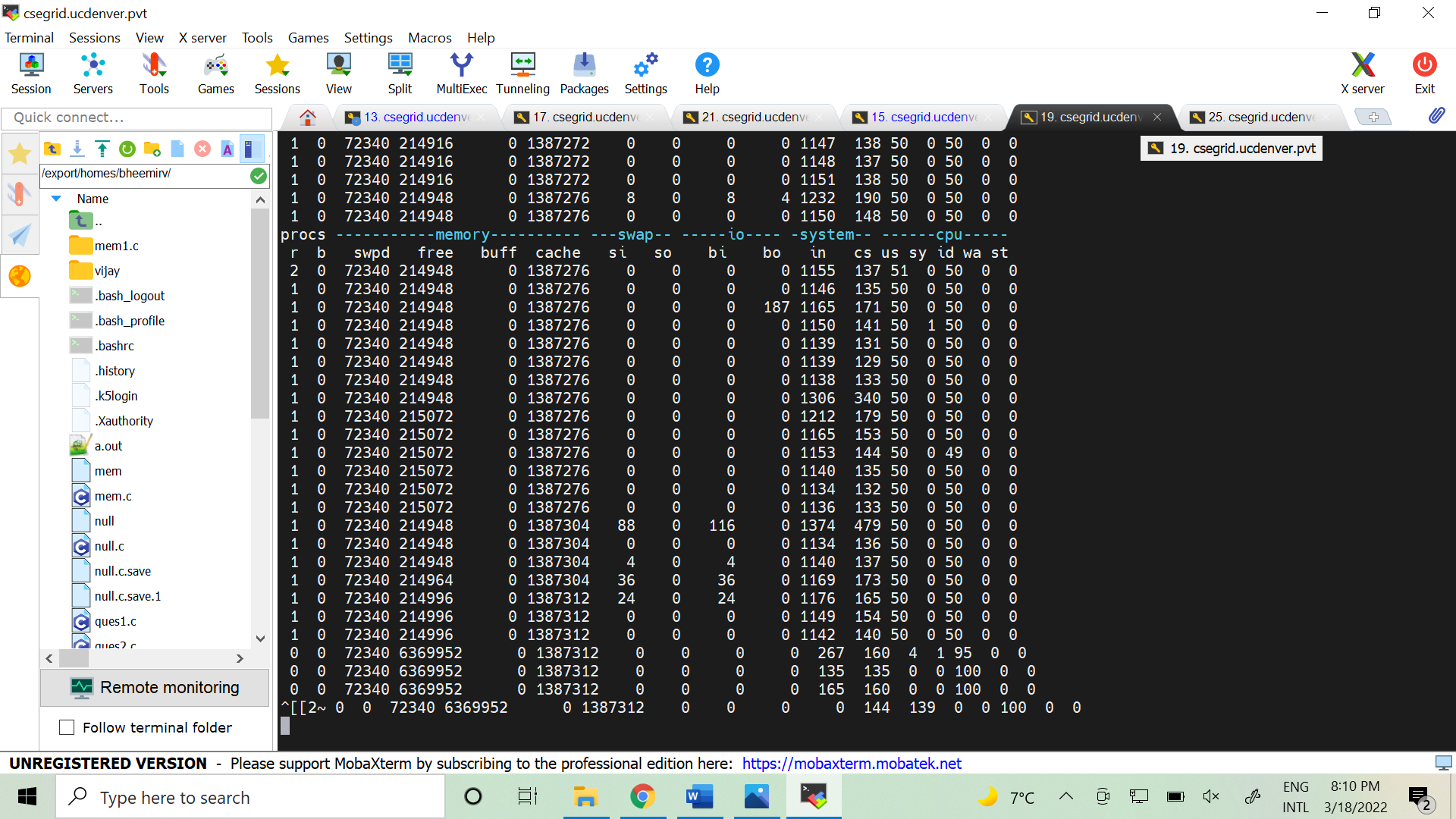
Screenshot: vmstat 1 (5000)



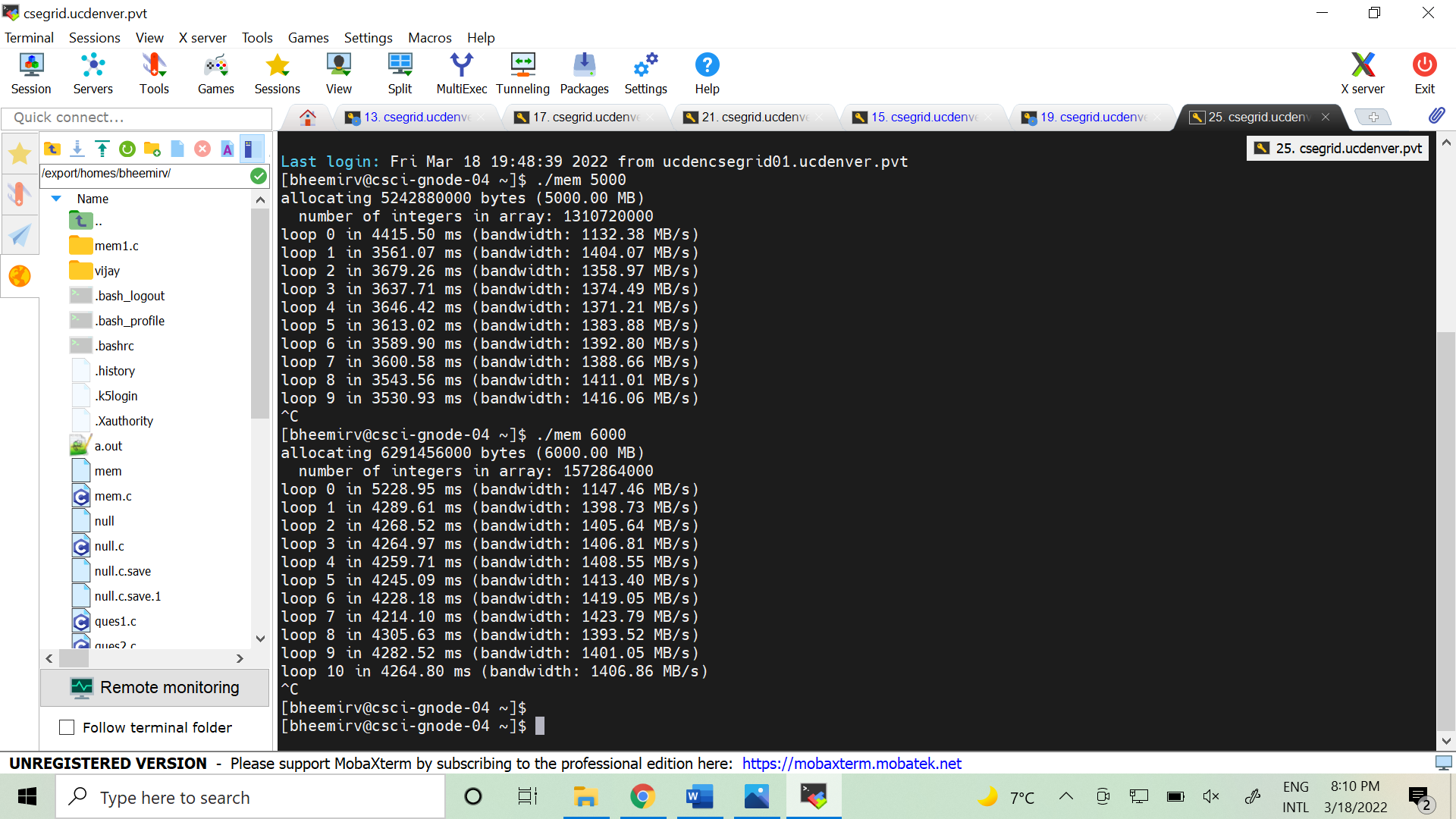
Screenshot: mem 5000



Screenshot: vmstat 1 (6000)



Screenshot: mem 6000



**Question4:**

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?**

**How do they change when mem is running?**

From the above screenshots,

The I/O statistics bi and bo represent the data transfers between virtual memory and block devices.

The CPU utilization statistics are us, sy, id, wa, st.

When the process execution starts blocks are sent to block devices grow much faster than blocks received from the block devices. The system time-st value is changing for some processes whereas id,wt,st,us increases for CPU Statistics.

The I/O statistics are:

**#1. Instance mem 4000 :**

bi-bo - remains 0

us- remains constant as 50 till the process got killed and when the process killed it becomes zero.

sy- remains 0 throughout

id- 50 when the process is running and 100 when the process is killed.

wa- remains 0 throughout

st- remains 0 throughout

**#2. Instance mem 5000 :**

bi-bo - remains 0

us- remains constant as 50 till the process got killed and when the process killed it becomes zero.

sy- remains 0

id- 50 when the process is running and 100 when the process is killed.

wa- remains 0 throughout

st- remains 0 throughout

There are no noticeable changes that occurred in 5000 when compared to 4000.

**#3. Instance mem 6000 :**

**bi-bo -** remains 0

**us-** 50 when the process is running and 100 when the process is killed.

**sy-r**emains 0

**id-** 50 when the process is running and 100 when the process is killed.

**wa-** remains 0 throughout

**st-** remains 0 throughout

There are no noticeable changes that occurred in 6000 when compared to 5000. These are like 4000MB and 5000MB.

Question 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 they 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?**

Ans:

for 4000MB Loop 0 takes 4820.54 ms and Loop 1 takes 2804.45 ms and Loop 2 takes 2825.78 ms

For 5000MB Loop 0 takes 4415.50 ms and Loop 1 takes 3561.07 ms and loop 2 takes 3679.26 ms.

For 6000MB Loop 0 takes 5228.95 ms and Loop 1 takes 4289.61 ms and Loop 2 takes 4268.52 ms.

For 7000MB Loop 0 takes 6485.63 ms and Loop 1 takes 5015.74 ms and Loop 2 takes 4923.62 ms.

For 8000MB Loop 0 takes 8762.72ms and Loop1 takes 6067.10 ms and loop 2 takes 4893.60 ms.

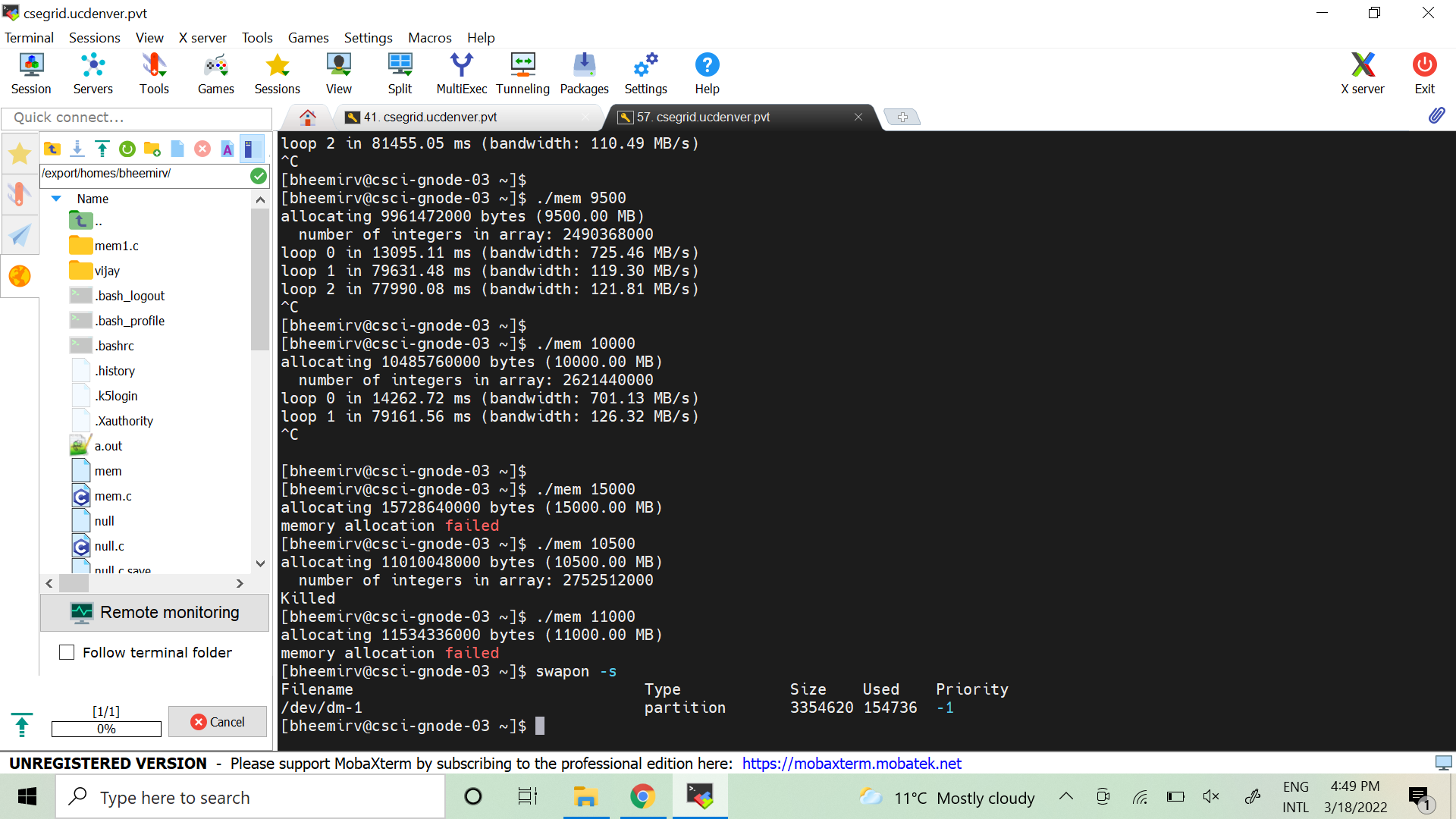
**How long does loop 0 take (and subsequent loops 1, 2, etc.)?**

Loop 0 takes more time than its subsequent loops loop1 and loop2 (below are attached screenshots for 4000,5000,6000,7000,8000) .

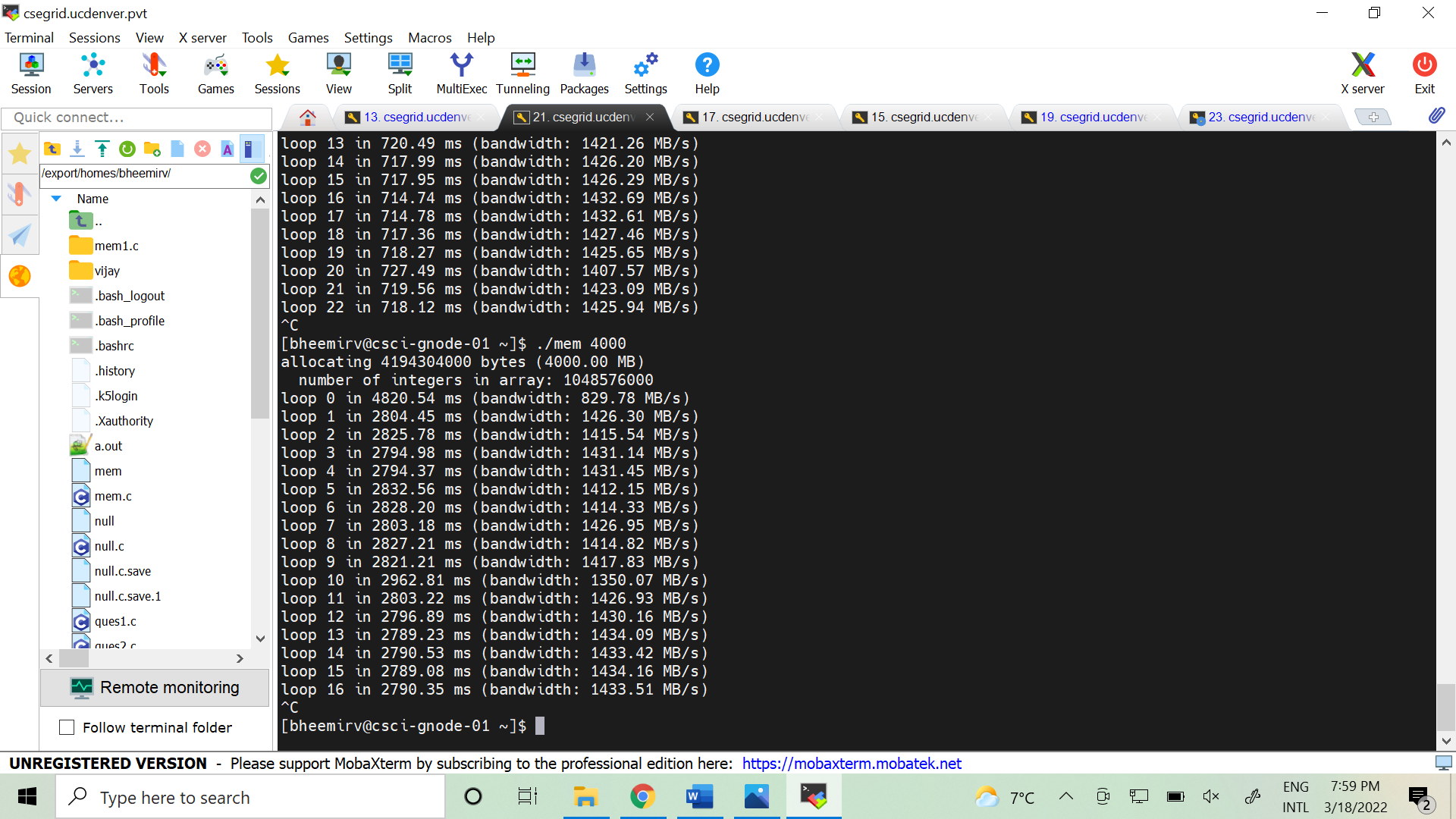
**Now pick a size comfortably beyond the size of memory (say 12000 again assuming 8 GB of memory). How long do they take here? How do the bandwidth numbers compare?**

Memory allocation killed at 10500MB and failed at 11000MB, my last execution is at 10000MB, here Loop 0 takes 14262.72 Milliseconds and with a bandwidth of 701.13 MBPS. Similarly, Loop 1 takes 79161.56 Milliseconds with a bandwidth of 126.32 MBPS and so on. The maximum available memory on the node is 8000. When I am going higher than the allocated memory it is giving message of Memory Allocation Killed or failed. The bandwidth numbers are decreasing. When Swap in and swap out are high then the performance will get decrease.

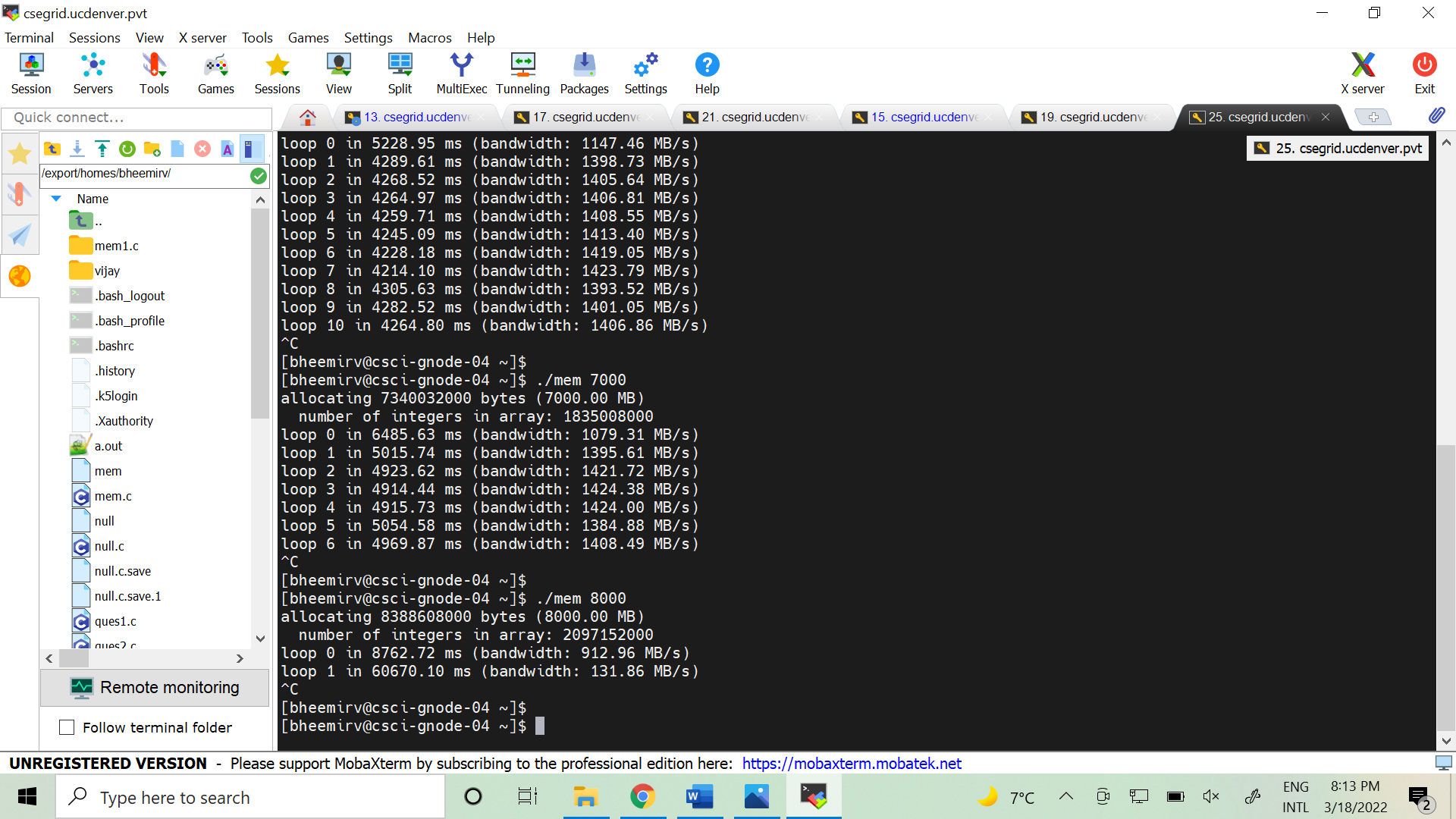
Memory Allocation Failed Screenshot:



For 4000:



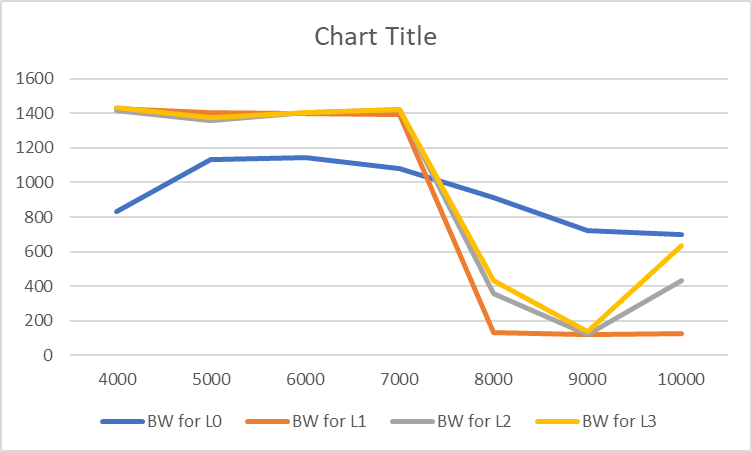
For 7000 & 8000



**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?**

When there is constant swapping, the bandwidth is low, and performance decreases.

**Graph:**

****

**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?**

The performance of the first loop is less than the other loops in the case where everything fits in memory and the performance of the first loop is high when compared to other loops in case where all memory doesn't fit.

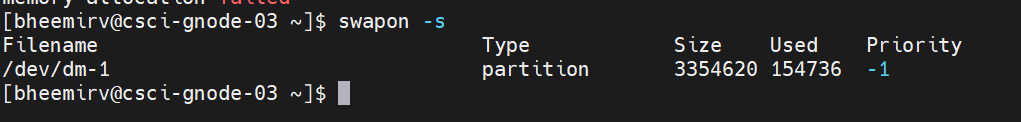
**Question 6:**

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 the swap? At what point does the memory allocation fail?**

Ans:

Swap limit: swapon -s

The swap size is around 3.1GB and used memory is 0.1GB.



If you try to run **mem** with increasingly large values it will result in failure.

Memory Allocation failure:

Observation:

The memory allocation failed at 11000 MB, but before that, the program starts to get killed at 10500 MB.

