Discipline

Design and build swap space management system

CS5003

Assignment 2

Sheikh Muhammed Tadeeb (AU19B1014)

❖ Problem Statement:

Machine configuration to be selected in such a way that you should be able to test the following conditions.

Case1: Only Primary Memory Process Execution

- Report the process graph both memory and CPU execution graph and express and conclude the case in your own words.

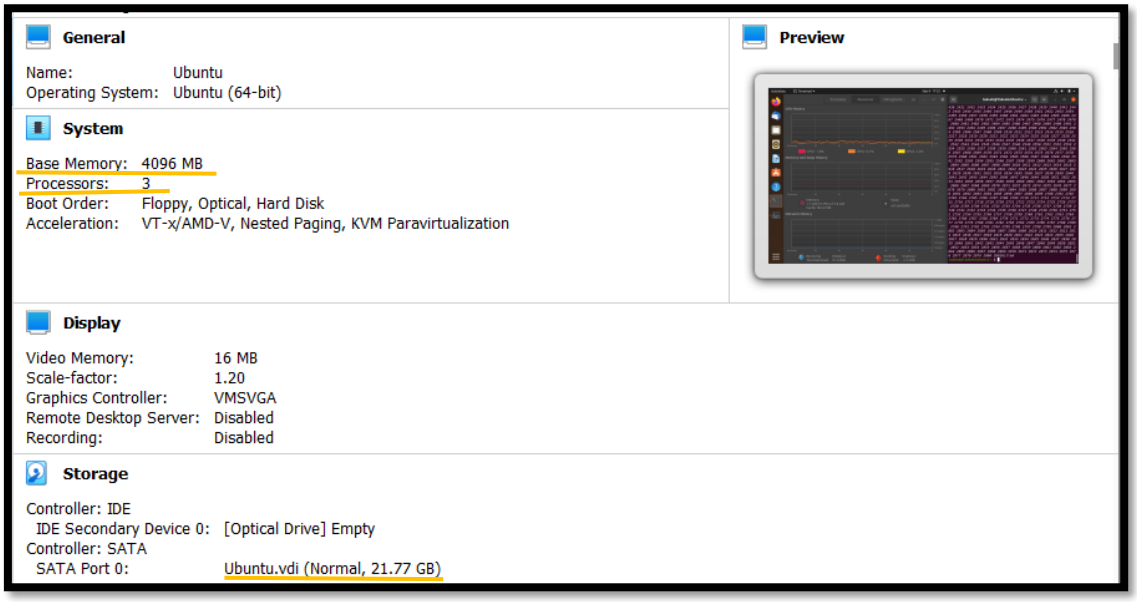
Case2: Allocate both primary and swap memory

-Report the process graph of both memory and CPU execution (Swap space calculation and consideration are required) express and conclude the case in your own words.

Case3: Turn on and off the swap space and analyse the outcome.

❖ Solution:

➢ Machine Configuration:



➢ **Case1**: Only Primary Memory Process Execution

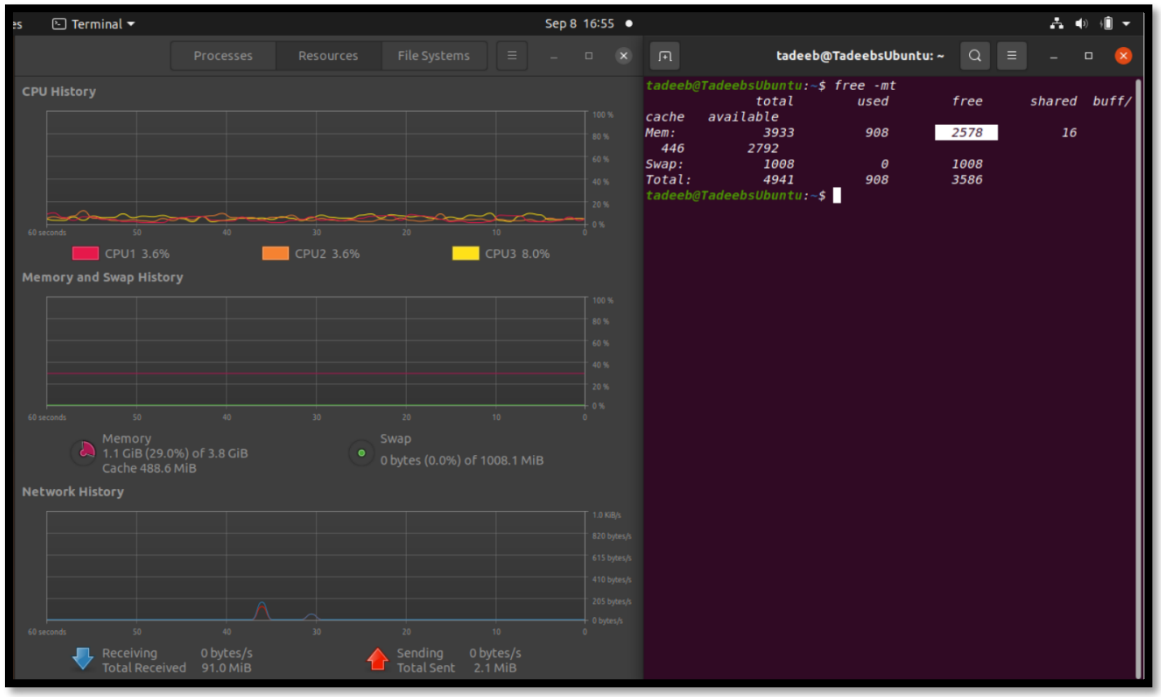


Fig1) Before execution

In the Above figure (Fig1) we can see that the Main Memory is being used about 30% and swap memory is empty i.e., not in use but is switched on.

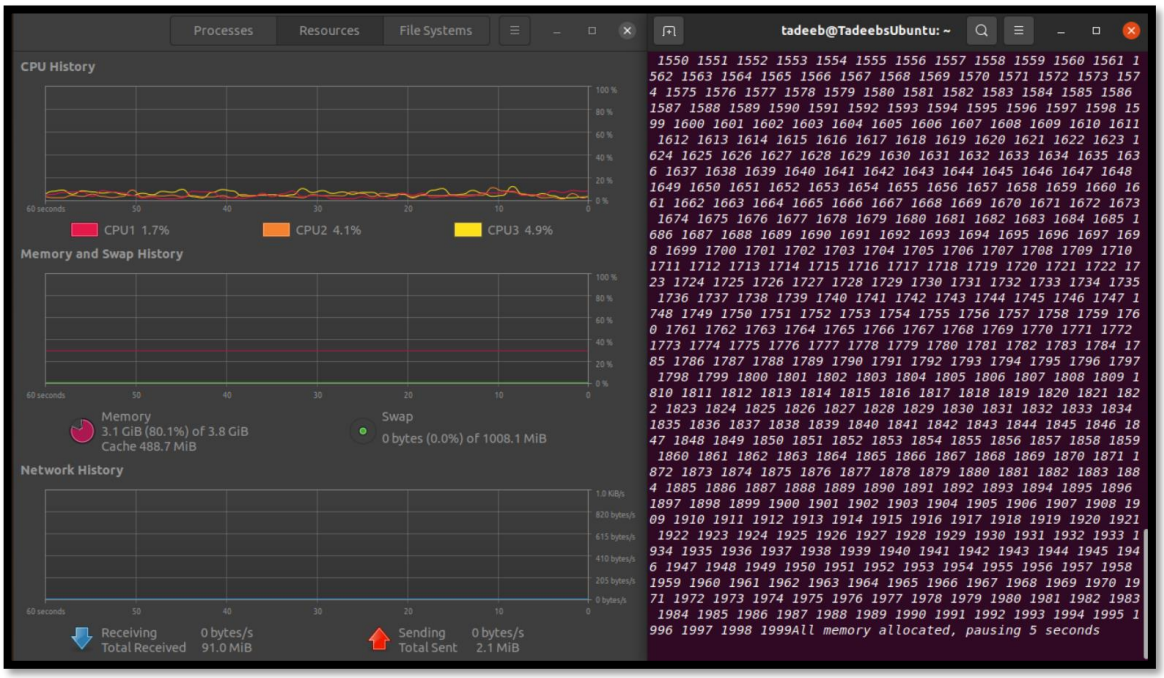


Fig1.1) During execution

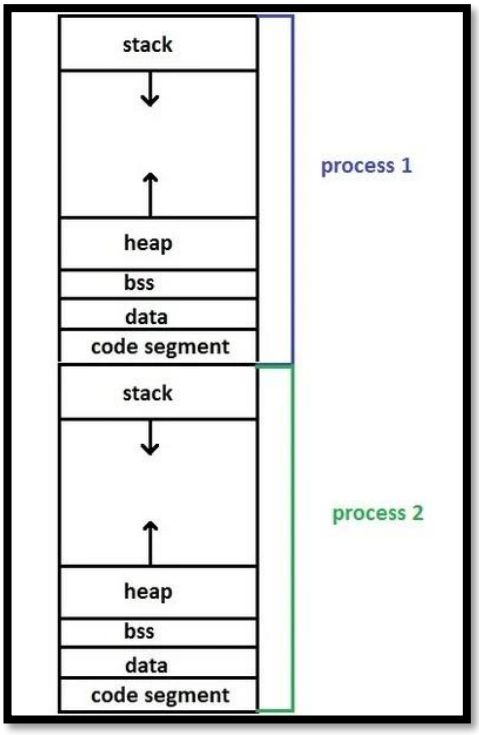
In the above figure (fig1.1) we can see that the main memory is getting filled up.

Fig1.2) After execution

In the above figure (fig1.2) we can see that the main memory is getting filled up.

- **Inference:**

  Here we have around 2GB of unutilized RAM so when we try to fill around 1.5GB then this much memory allocation can be handled by RAM but its utilization will be very high as we can see in the fig 1.2, the utilization is around 90% which makes the processes on the system slow for that duration. Secondly, we can see there is a peak in CPU graphs as well. So, we get to know that as the RAM utilization increases the load on CPU also increases although it gets divided among the different cores.



➢ **Case2**: Allocate both primary and swap memory

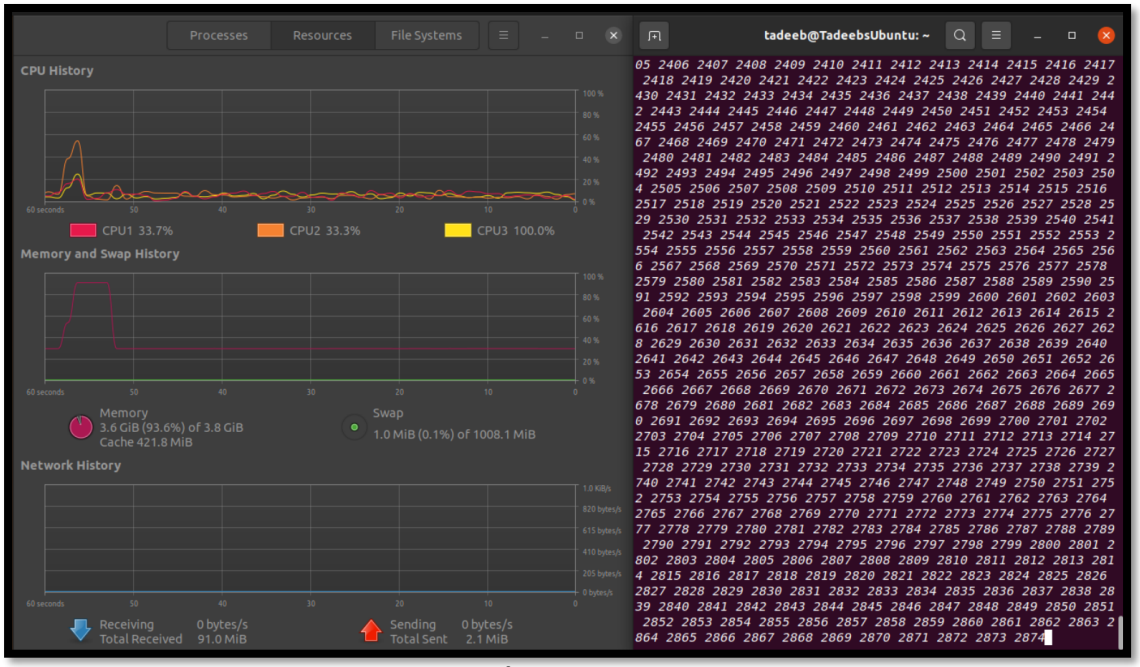


Fig2) During execution

In the above figure i.e., fig2 we can see that when we started to fill our RAM with huge data it gets completely filled.

NOTE: This completely filled RAM has multiple processes running in it but currently we are running this SwapManger process but the RAM is full so it will now move the unused or idle processes to the swap memory which we can see in the below figure 2.1.

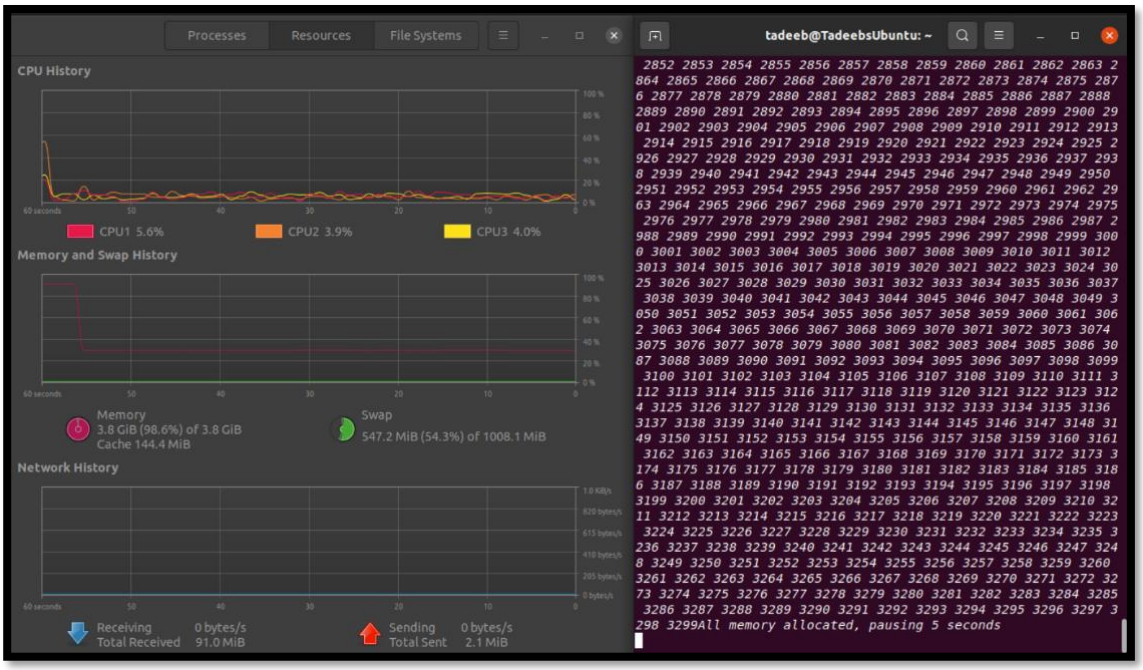


Fig2.1) During execution

In the above figure 2.1 we can see that the RAM got completely filled so to avoid killing of our running process, the idle or unused processes got shifted in the Swap memory (this is a part of HDD and is slower than RAM) and during the process execution 60% of swap memory along with 100% utilization of our RAM.

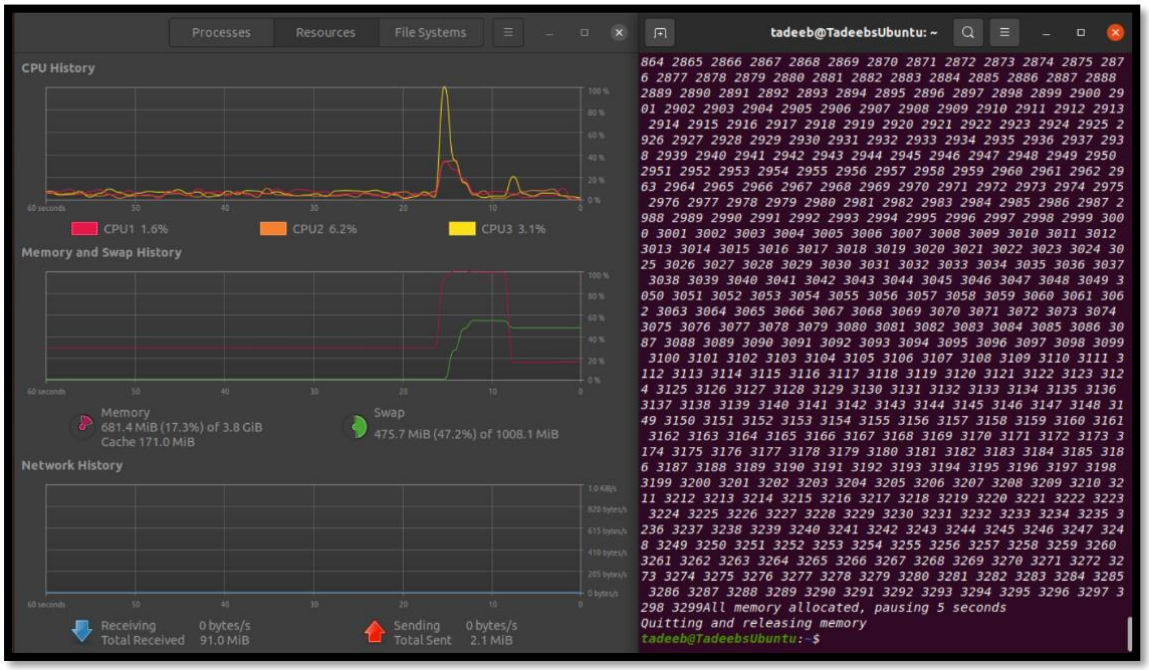


Fig2.2) After execution

Here, in fig 2.2 we can see that once the SwapManager process had been completed, the memory which it was using got released and again we have 70% free RAM to utilize for our future processes. We can also see that our swap memory usage got decreased by 10%, this is due the fact that the processes in the swap memory again got transmitted to our main memory for their execution to get continue.

NOTE: Here swap memory acts as a savior because our executing process didn't got killed, rather the unused processes moved temporarily to swap memory giving free space in RAM to our executing process. Also, we can see that there is a pause of 5 sec when processes were getting transferred from RAM to swap memory and then the memory got released.

- **Inference:**

Swapping is a method in which the process should be swapped temporarily from the main memory to the backing store. It will be later brought back into the memory for continue execution. Backing store is a hard disk or some other secondary storage device that should be big enough in order to accommodate copies of all memory images for all users. It is also capable of offering direct access to these memory images.

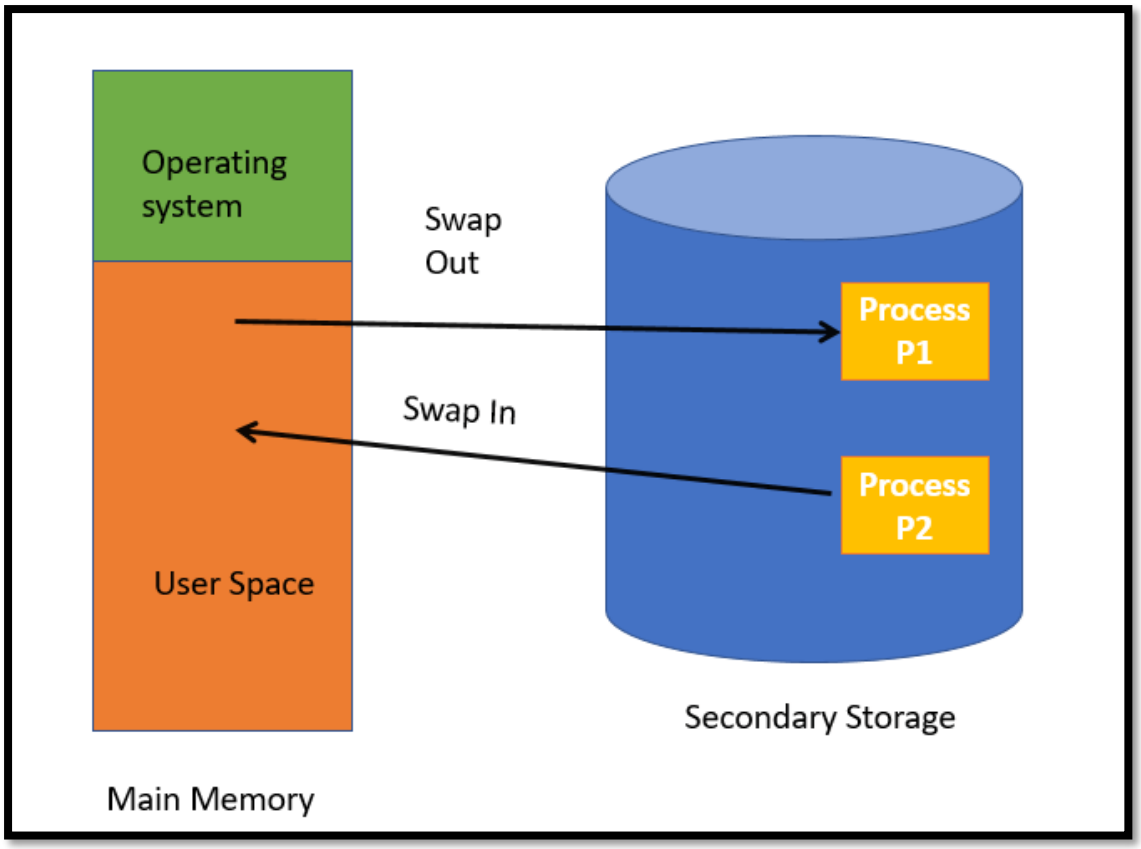


Fig2.3) Pictorial Representation of swapping

- **Case3**: Turn on and off the swap space and analyse the outcome.

Commands used: ***swapon -a*** and ***swapoff -a***

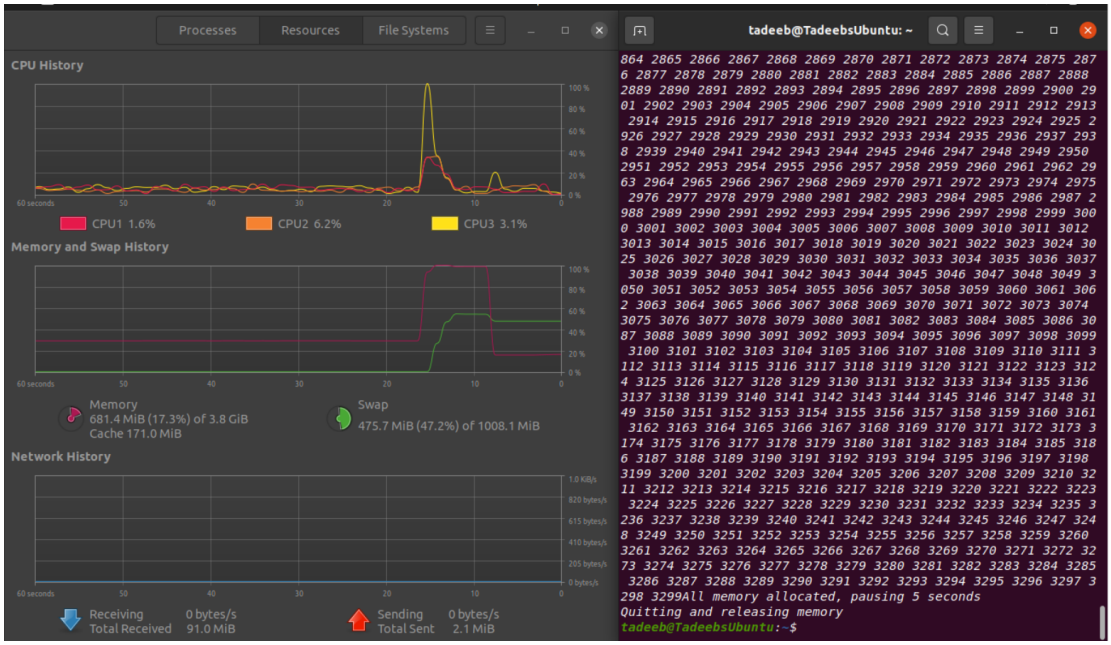


Fig3) During execution

In the above figure (fig3) we can see that the SwapManager process is running and as it required more space so the unused or idle processes got temporarily shifted in the swap memory (as our swap memory is on i.e., swapon -a). We can see that initially there was 100% utilisation of RAM but as soon as swap memory comes into picture the RAM utilisation got decreased to 30% and subsequently the swap memory utilisation got increased from 0% to around 60%.

NOTE: We can see in fig 3 as our swapping takes place the CPU utilisation also gets increased as now the CPU has to decide which process to move to swap memory and which process to keep in the main memory. Also, we can see that there is a pause of 5 sec when processes were getting transferred from RAM to swap memory and then the memory got released.

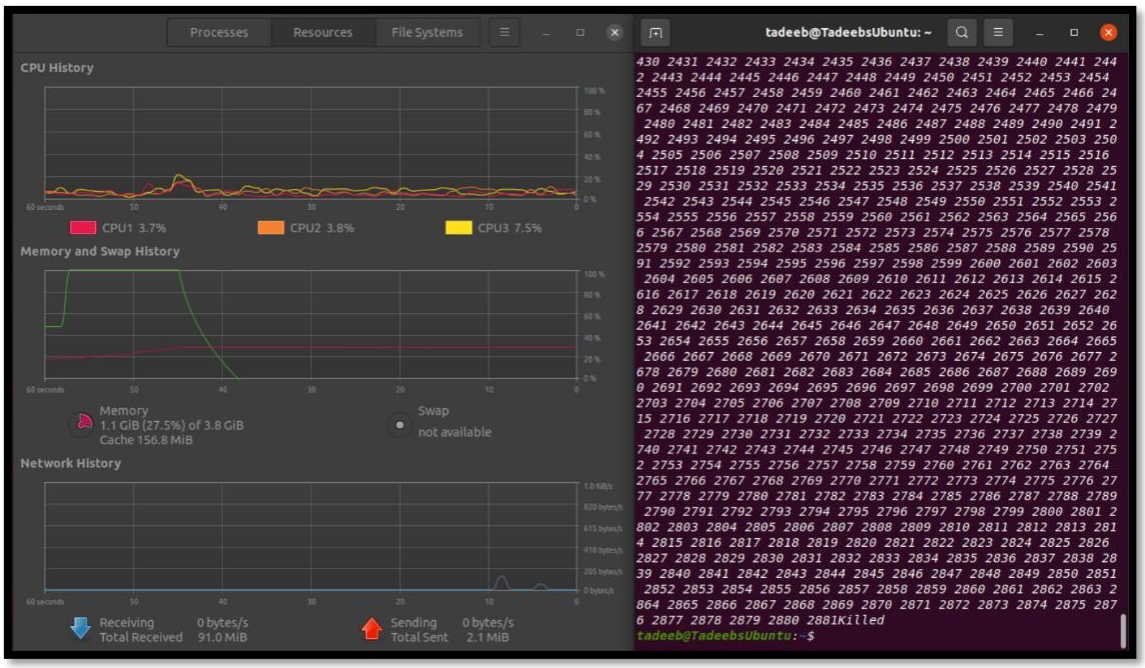


Fig3.1) During execution

In the above figure (fig3.1) we can see that the SwapManager process is running and as it required more space but the unused or idle processes can't be shifted in the swap memory (as our swap memory is off i.e., swapoff -a).

NOTE: We can see in fig 3.1 as our swapping is not taking place the CPU utilisation also remains unchanged as now the CPU has no extra things to decide on. Also, we can see that there is no pause of 5 sec as processes aren't getting transferred from RAM to swap memory and also no memory got released i.e., our system went in thrashing state.

- **Inference:**

1) Here, are major benefits pros of swapping when swapon -a was used:

1) It offers a higher degree of multiprogramming.

2) Allows dynamic relocation. For example, if address binding at execution time is being used, then processes can be swap in different locations. Else in case of compile and load time bindings, processes should be moved to the same location.

3) It helps to get better utilization of memory.

4) Minimum wastage of CPU time on completion so it can easily be applied to a priority-based scheduling method to improve its performance.

2) At any given time, only few pages of any process are in main memory and therefore more processes can be maintained in memory. Furthermore, time is saved because unused pages are not swapped in and out of memory. However, the OS must be clever about how it manages this scheme. In the steady state practically, all of main memory will be occupied with process's pages, so that the processor and OS has direct access to as many processes as possible. Thus, when the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing.

3) A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed. In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finished its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.

❖ Conclusion:

1) To avoid killing of a process, when two process are running say P1 and P2 and if P1 is in idle state and we are using P2 but P2 requires more space in memory so we can free the space of P1 for P2 by moving P1 temporarily in swap memory (although swap memory is slow as it is a part of HDD). Here swap acts as saviour.

2) Secondly if swap is off and our RAM gets Completely filled then we reach thrashing point and our OS kernel has something called as OOM i.e. The "OOM Killer" or "Out of Memory Killer" is a process that our kernel employs when the system is critically low on memory. This situation occurs because processes on the server are consuming a large amount of memory, and the system requires more memory for its own processes and to allocate to other processes.

3) Swap retains the process i.e.; it doesn't kill the process even if both the RAM and swap gets utilized completely.

4) On the other hand, the Kernel kills the process if our RAM gets completely utilised and swap is turned off which make us feel as if our system got hanged for a while.