**Homework Wan Huzaifah bin Wan Azhar**

**Answer:**



Sys and us increased while id decreasing when running mem.c with 1MB. Sys and us is increasing because us, which is user time is time spent running user code runs the mem.c program. Sys is time spent running kernel code. It increase because of the malloc function in mem.c, which allocates memory by requesting it from OS. Lastly, Id stands for time spent idle. Idle is at 99% if the mem.c is not running while it is decreasing when it is running because less time is spent idling and running the program.

When running two mem.c at the same time, the user time doubled. This make sense because the user is running two program simultaneously, and thus taking more time on the system.



Swpd (the amount of virtual memory used) remains at 0 when running ./mem 1024. However, free memory is decreasing when running the program. Killing the program increased free again. The most surprising thing is that free memory increased exactly to the amount before the program is running. So yes, the amount of free memory increased as expected it should be.



As the total of my OS is 4GB, I first run mem.c using 2GB. The swap value, si and so does not change because there is still free memory. Running mem.c using 3GB forces the OS to swap to disk. Free memory never reaches zero but si is increasing by a fair amount (< 1000), which tells the amount of memory swapped in from disk. At this point, the system starts to lag. Running mem.c using 3.5GB shows a big number of si and so.

Swap out (so) on the first loop is 140K, to 10K on second loop to 0 on third and next loop. This make sense as page need to be swapped out so that it mem.c program can allocate more than free space. Swap in (si) is stabilising. It is not 0 but it increase and decrease between 100 to 1000. I assume this works as intended so that swap out can allocate more memory.



Running 2GB does not change much on Cpu and IO. Running 3.5GB shows that block out has the same value as swap out. This is because swapping is using IO to disk so it move the page into the disk. Block out (bi) increase for 4 loops but then returns to 0.



When comfortably allocating memory below its total memory, bandwidth allocation is fast, which is around 7000MB/s. However, when allocating memory comfortably beyond total space, the average bandwidth dropped to 6000MB/s. The difference of 1000MB/s is huge.

The performance of the first loop of 3500 (where everything does not fit in memory) is 2586MB/s and increases to 7000, but it keeps dropping to 6000 and 5000 and stabilising around 5000 and 6000 and rarely does it reaches more than 7000 MB/s.

The performance of the first loop of 2000 (where everything does fit in memory) is 2641MB/s and increases to 7000 and maintaining the up and down around 7000.

This indicates that speed actually decreases when swapping is used, which make sense.



When allocating mem with very high value of 100000, the program will exit with memory allocation failed. This is because when swap space and free space cannot allocate the memory needed, the OS will not allocate it.

For 4GB system, I cannot start mem with 5900 MB, which exceeds the main memory and swap spaces total.