GRADUATE SYSTEMS – PA01 – MT25036 - Report

Part A: Write two C programs - Process and Thread Based.

1. **Program A:** We created K number of child processes using `fork()` through the main parent process. These child processes are capable of running any function that is passed.
    a. The parent process calls fork() repeatedly to create K child processes.
    b. Each child process runs exactly one worker function. Exits on finish. Parent waits till every child process exits.
2. **Program B:** We created K number of threads using <pthread> library using the parent main process. Each of the K threads are capable of running any worker function passed through them.
    a. Create K threads using pthread_create(…)
    b. Threads are synchronized using pthread_join(…)

In the beginning we hardcoded it to create 2 processes and 2 threads, later changed to a user dependent requirement.

Part B: Three worker functions – Design Cpu, Memory, Io intensive functions

1. Cpu() function
    a. Performs matrix multiplication.
    b. Initializes two 50×50 matrices with constant values, then multiplies them 6 x 1000 times, storing results in a third matrix.
    c. This create load on CPU as we are doing floating point operations.
2. Mem() function
    a. Allocates 1MB (1024 * 1024 Bytes) memory in heap and initialize with the same value in all.
    b. Adds 5 to every number 6000 times forcing to put load on repeated memory access.
3. Io() function
    a. Write a 64KB(65536 Bytes) chunk of data into a text file 6 x 1000 times.
    b. Makes the disk work hard by doing many write operations.

Part C: 6 Combinations: Measure Resource Utilization

For the 6 different combinations formed where Program A and Program B run 3 different types of worker functions each. Program A has 2 processes and Program B has 2 threads. For each combination we measured these statistics:

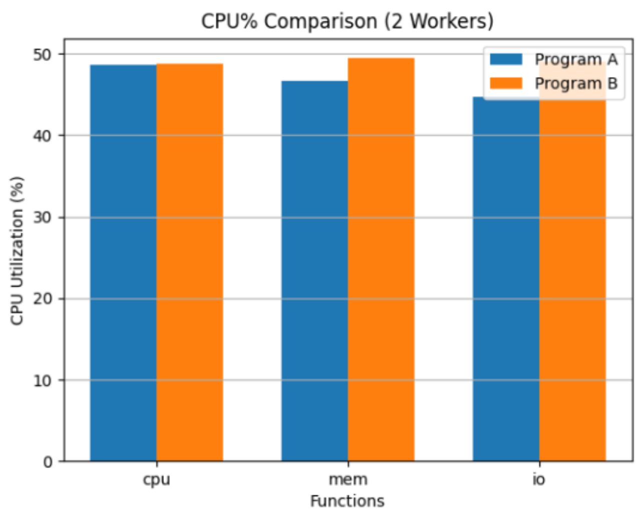
- **CPU%**: Processor the program uses while running each combination.
- **MEM%**: RAM the program uses while running each combination.
- **Disk I/O**: How much disk activity occurs. This included disk write, read and overall utilization. It gets divided into 3 parts –
    - **Disk Write Speed** (in kBps)
    - **Disk Read Speed** (in kBps)
    - **Disk Utilization%** (Most important metric)
- **Execution Time**: Time (in secs) the program takes to complete.

We used a bash script to automate running the combinations and measuring each metric. Here are some of the things:

- **taskset** to pin programs to a single CPU core.
- **top** to gather CPU and memory usage.
- **iostat** to get disk activity statistics.
- **time** to measure execution time of each program.

Let us now look at some graph and analyse what we observe there.

1. CPU% by both the programs in different combinations.

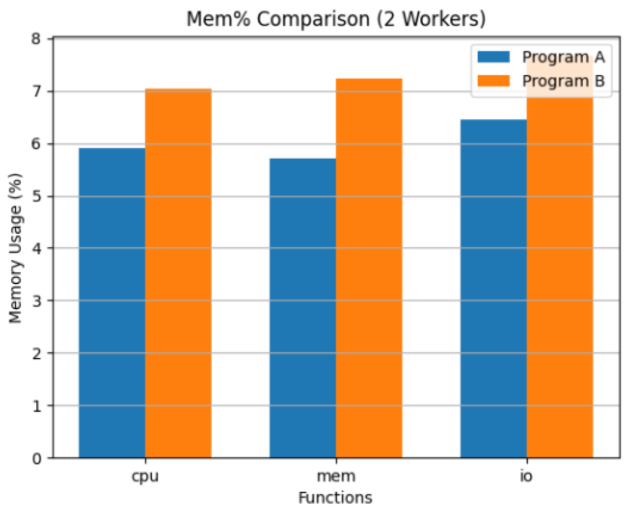


Observations:

- Both process based and thread-based programs utilize close percentage (44% to 49%) of CPU across all the worker functions with thread-based programs utilizing slightly higher than the process-based ones.
- CPU% is seen a tad bit higher in using cpu() and mem() worker functions than in using io() function.

The lower percentages suggest that programs have been running on a single core and not utilizing all available ones as taskset restricts them to one.

2. Memory % utilization by both programs

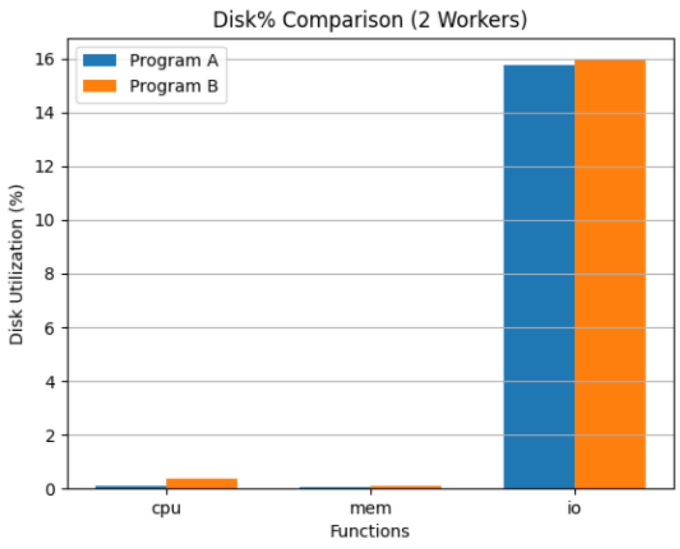


Observations:

- Memory Utilization is higher in thread-based program than in process based across all three worker functions. It is about 5.9% to 6.4% in using program A and about 7% to 7.6% in using B.
- IO function slightly higher than cpu and mem.

Threads share memory space, which is observed by the higher memory%, while forked processes have separate isolated memory spaces.

3. Disk Utilization by both programs across all combinations

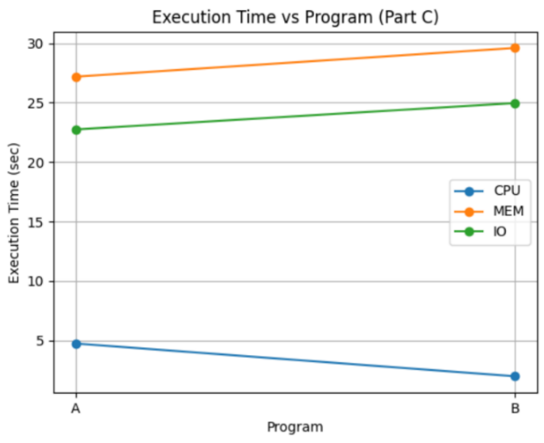


Observations:

- Io function utilizes the most disk with both program A and B with 15.7% with A and 15.9% with B as it should since we are writing into a text file continuously, into disk.
- Other functions show low disk utilization and those functions don't access the disk like the io function.

The I/O function writes 64KB into a file 6 x 1000 times. It creates heavy disk load, while CPU and memory functions barely touch the disk.

4. Execution time across all combinations



Observations:

- Thread-based program B is faster on CPU function (1.98s vs 4.74s) than process-based program A. B is 2.4 times faster.
- For Memory function, A takes 27.19 and B takes 29.61. B is 2.4 secs slower.
- For IO function, A takes 22.75 secs while B ends up taking 24.97 secs.

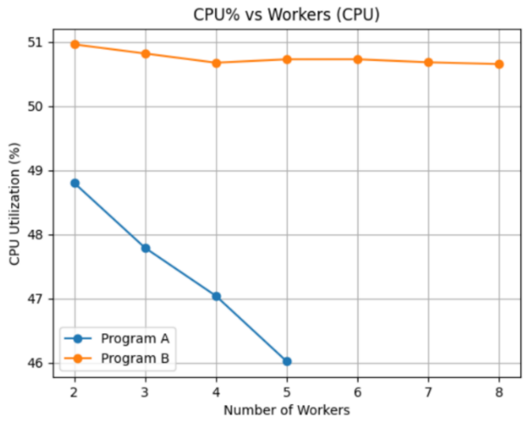
I think for CPU-intensive program, B took a lot less time than A is because of the overhead A has on creating processes using fork(). Threads have a lower creation overhead.

<u>**Part D: Measure Resource utilization if No. of workers increased**</u>

We are going to do the same things we did in C but here we would increase the number of processes or threads needed to be created.

- We made changes to the part A programs making them take K as an argument (K = #workers)
- We will test program A with 2, 3, 4, 5 number of processes and program B with 2, 3, 4, 5, 6, 7, 8 number of threads across the same 3 worker functions.
- We are using the same tools we used in C but the new script creates many new combinations to test.

1. CPU Utilization

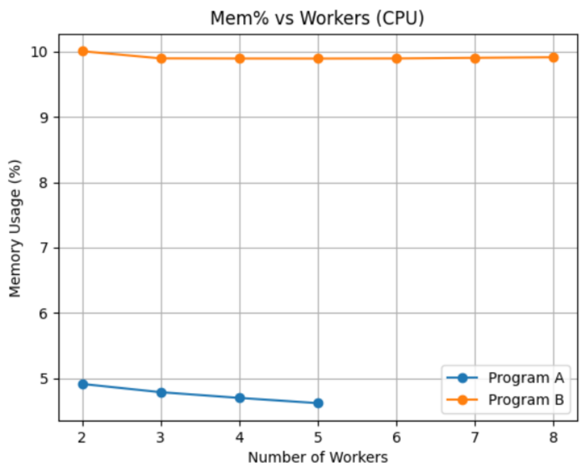


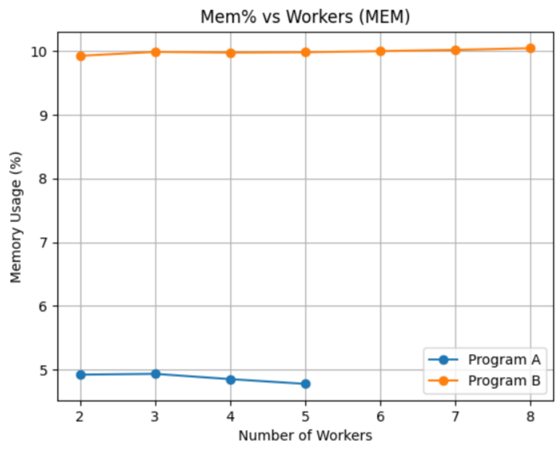
Observations:

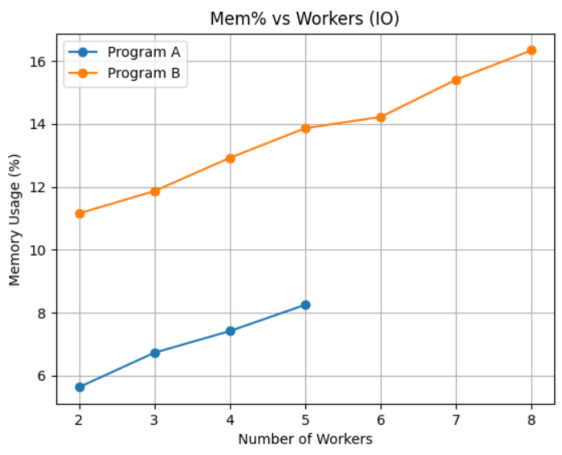
- Program A's CPU% drops from 48.8% (2 workers) to 46.0% (5 workers)
- Program B maintains 48% to 50% CPU utilization across all worker counts (2-8 workers)

As we are running in a single core, increase in number of process will increase the competition for same core. The context switching increase, thus less CPU% more context switching happens.

2. Memory Utilization





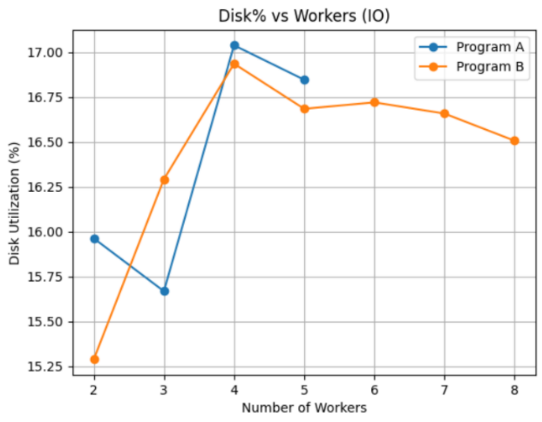


Observations:

- Program A grows from 5.6% (2 workers) to 8.3% (5 workers)
- Program B's memory rises from 11.1% (2 workers) to 16.3% (8 workers)
- Program B is using more memory than A. Almost 2x A.

Program B uses threads and they have their own stack space. So with more workers, more space consumed. Program A uses processes but still due to some underlying mechanism it doesn't consume much space.

3. IO Utilization

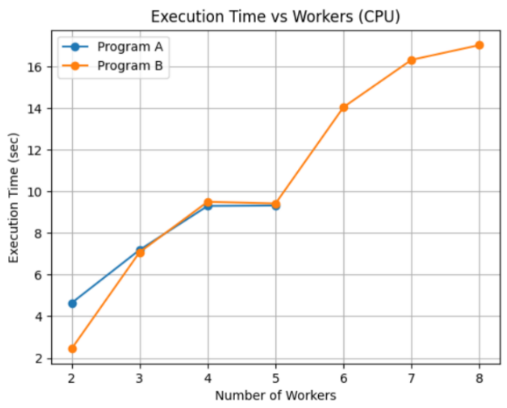


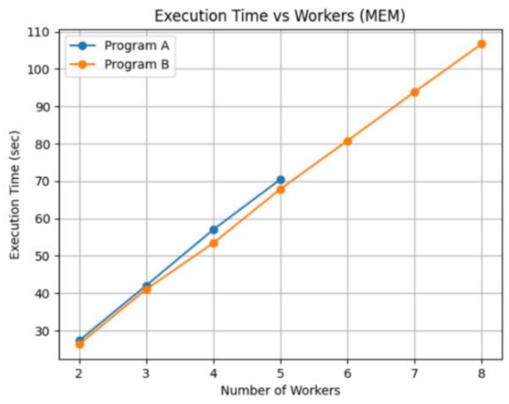
Observations:

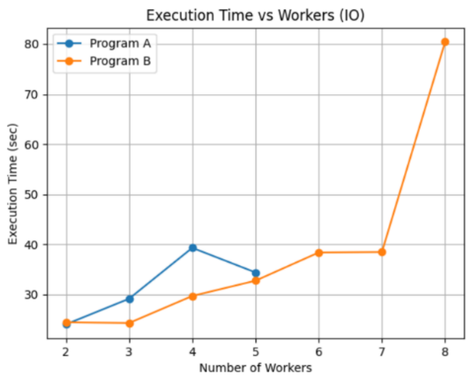
- Both Program A and B show almost similar behaviour with disk utilization reaching a peak and falling down as number of workers increase. The peak for A is 16.82% and 16.72% for B.

I think for more number of workers, the concurrent writing causes delays. I don't have much idea. They both reach an optimal place with 4 workers.

4. Execution time







Observations:

- CPU intensive work shows a linear increase in time with more workers on single core.
- It feels like Memory-intensive work increases the execution time linearly regardless of workers.
- Io Intensive work sees an increase in time for A and sudden fall. For B, it keeps on increasing as with increase in number of workers, queue for disk increases.

AI Declaration

I have used AI tools, most specifically ChatGPT 5.2 for generating the codes for almost every part except part A and B. I have made the necessary changes to the code according to the assignment. All code is run and verified by me and I have tried my level best to adhere to the assignment rules.

Github Repo: https://github.com/Reehan09Sarmah/GRS-PA01-MT25036