PA01: Processes and Threads

Mudit Kumar (MT25073) | *Graduate Systems (CSE638)*

1. GitHub Repository

The source code and artifacts are hosted at:
https://github.com/Muditkumar123/MT25073-GRS-PA_1/

2. Implementation Overview

This assignment explores the performance differences between Processes (fork) and Threads (pthread) under three workloads:
- CPU Intensive: Complex math calculations stressing the ALU.
- Memory Intensive: 5MB buffer allocation with stride access to trigger TLB misses.
- I/O Intensive: 128KB repeated writes using 'fsync()' to force physical disk I/O.

3. Part C: Baseline Observations (2 Workers)

Baseline performance metrics for 2 concurrent workers pinned to a single core:

Program	Mode	Time(s)	CPU(%)	Read(KB)	Write(KB)
Program_A	cpu	6.3	101.0	2756	180
Program_A	mem	6.4	100.0	0	32
Program_A	io	14.22	26.5	12	983820
Program_B	cpu	6.32	100.0	64	64
Program_B	mem	6.17	100.0	0	28
Program_B	io	15.22	22.0	0	983860

Key Observations:

- I/O Bottleneck: The IO task took significantly longer (~15s) compared to CPU/Mem (~6s), confirming the workload was successfully bound by disk speed.
- Massive Writes: The 'fsync' implementation successfully forced massive writes (~980 MB) to the disk.
- Zero Disk Reads: Despite reading the file back, iostat reported 0 KB Reads. This is because the Linux kernel served the data directly from the Page Cache (RAM).

4. Part D: Scalability Analysis

We scaled Program A (2-5 processes) and Program B (2-8 threads) to analyze performance trends.

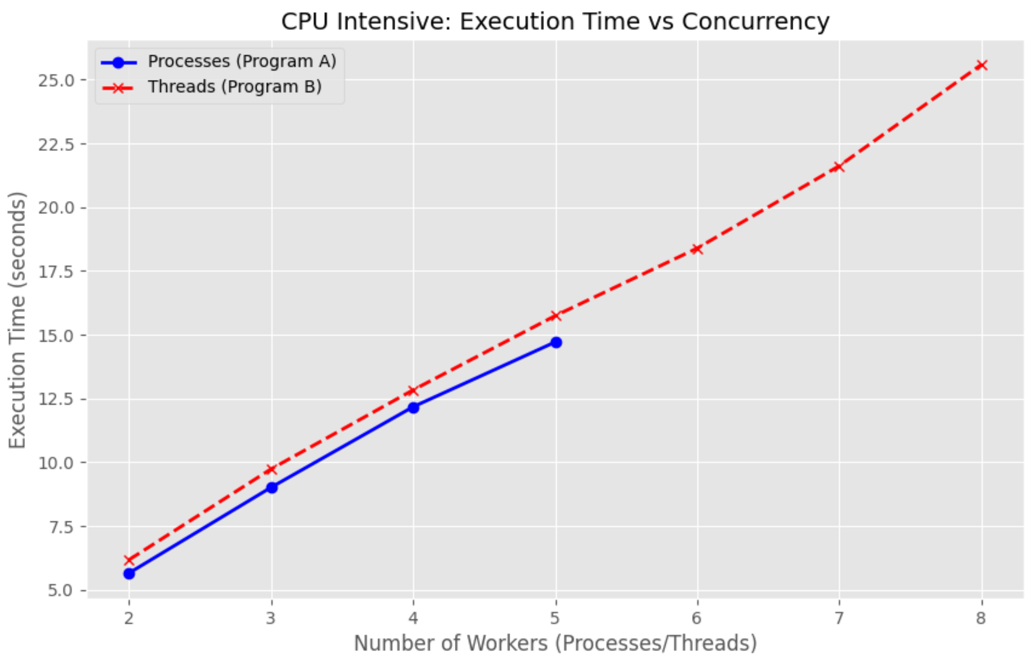

Figure 1: CPU Intensive Scalability

Analysis: Execution time increases linearly. Since workers were pinned to a single core, they executed sequentially (time-slicing).

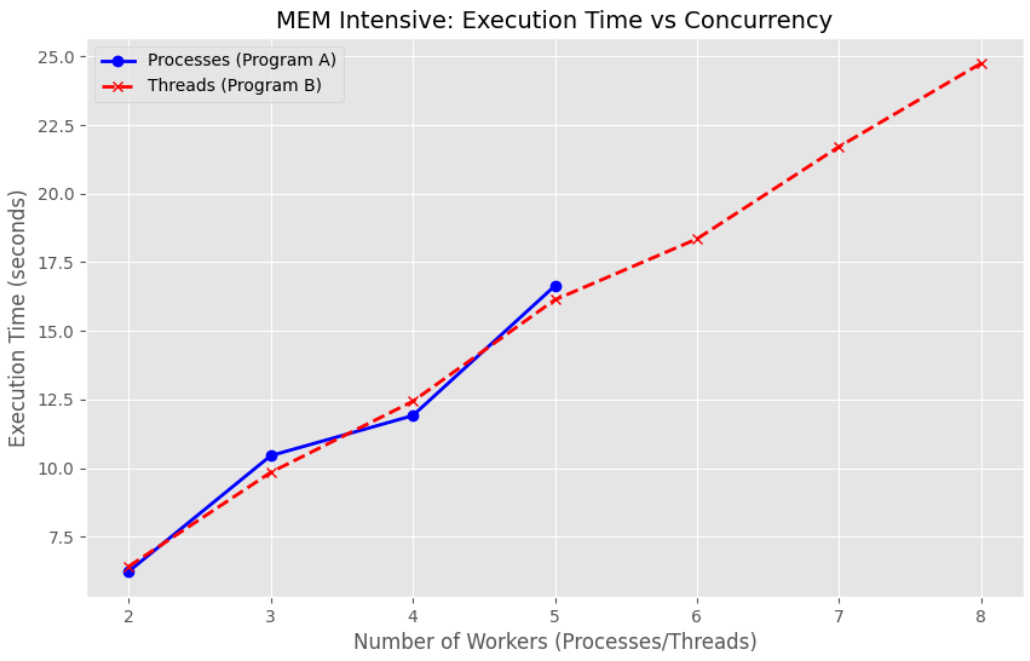

Figure 2: Memory Intensive Scalability

Analysis: Similar linear trend. Threads and Processes showed comparable overhead profiles for memory access.

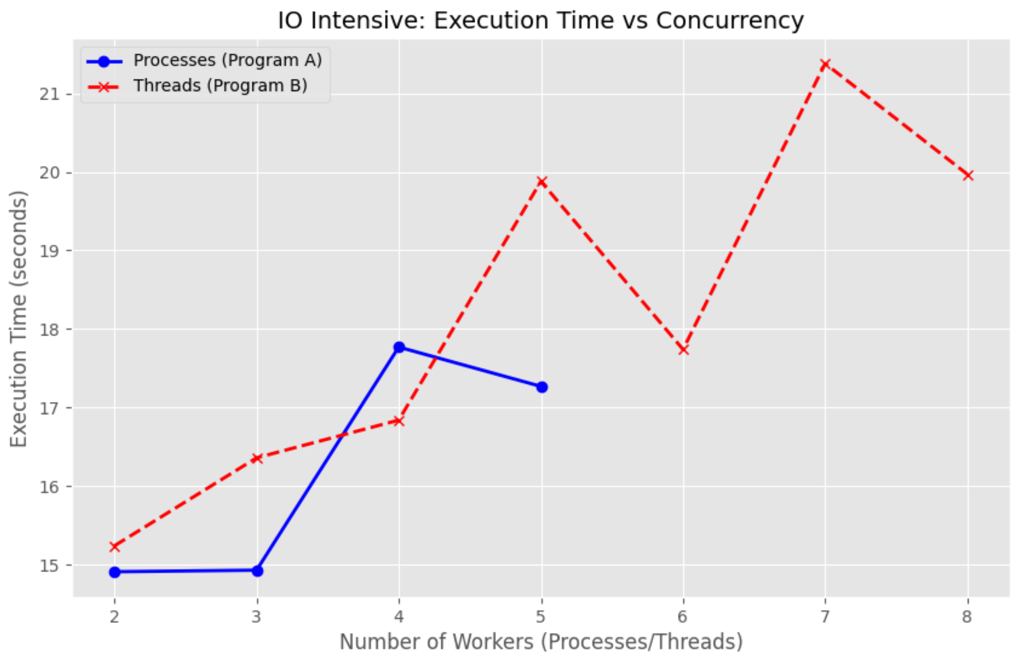

Figure 3: I/O Intensive Scalability

Analysis: The curve is flatter (15s-20s). The disk hit saturation early; adding more threads did not speed up physical disk rotation.

5. AI Usage Declaration

I utilized an AI assistant to support the development of this assignment. My usage was as follows:

1. C Code Development: The AI generated the initial code structure and logic for the worker functions. I manually transcribed and verified this code line-by-line to ensure I fully understood the implementation logic.
2. Scripting Support: The AI provided logic for Bash scripts (seq loops, taskset).
3. Debugging: The AI helped diagnose WSL2-specific issues, such as identifying correct iostat columns.
4. Visualization: The AI provided the Python script to generate plots from CSV data.

All final analysis and conclusions are based on the data I collected.