MT25042 — PA01 Report: Processes and Threads

Roll Number: MT25042
Course: Graduate Systems (CSE638)
Date: January 23, 2026

Table of Contents

Part A — Program Implementations

Program A: Process-based (fork)

File: `MT25042_Part_A_Program_A.c`

Description:
Program A creates 2 child processes using `fork()` system call. Each child process executes the specified worker function (cpu, mem, or io). The parent process waits for all children to complete using `waitpid()`.

Key Implementation Details: - Uses `fork()` to create separate address spaces for each process - Each child process runs independently with its own memory - Parent tracks all child PIDs and waits for completion

Screenshot: Program A execution with `top` monitoring

Program B: Thread-based (pthread)

File: `MT25042_Part_A_Program_B.c`

Description:
Program B creates 2 threads using POSIX pthread library. All threads share the same address space and execute the specified worker function concurrently. The main thread joins all worker threads before exiting.

Key Implementation Details: - Uses `pthread_create()` to spawn threads within same process - Threads share memory space (more efficient for shared

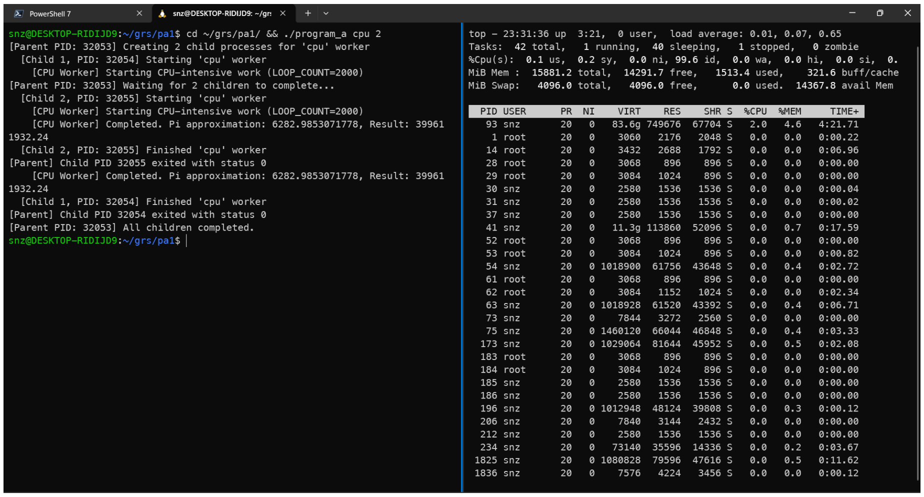

Figure 1: Program A - top output

data) - Uses `pthread_join()` for synchronization

Screenshot: Program B execution with `top` monitoring

Part B — Worker Functions

Files: `MT25042_Part_B_workers.c`, `MT25042_Part_B_workers.h`

LOOP_COUNT Configuration

```
#define LOOP_COUNT 2000  // Last digit of MT25042 is 2, so 2 × 1000 = 2000
```

Worker Function: `cpu` (CPU-intensive)

Implementation: - Computes Pi using Leibniz series (slow convergence, many iterations) - Performs trigonometric calculations (sin, cos, tan) - Executes nested loop multiplications (matrix-like operations)

Expected Behavior: High CPU%, minimal memory usage, negligible I/O

Observed Results (from CSV): | Program | CPU% | Memory (KB) | I/O Write (KB) | |———|——|————-|—————-| | Program_A | 195% | 1,792 | 4 | | Program_B | 195% | 2,304 | 4 |

Analysis: Both programs achieve ~195% CPU utilization (indicating both cores/processes active). Thread-based Program B shows slightly higher memory due to thread stack overhead.

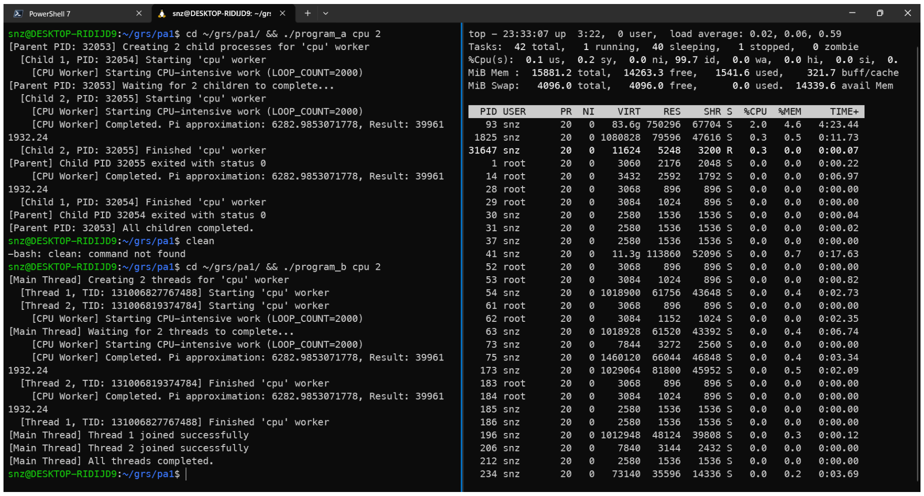

Figure 2: Program B - top output

Worker Function: `mem` (Memory-intensive)

Implementation: - Allocates two 16MB arrays (larger than typical L3 cache) - Performs `memcpy` operations (memory bandwidth test) - Random access patterns to cause cache misses - Sequential access with modifications

Expected Behavior: Moderate CPU%, high memory usage, minimal I/O

Observed Results (from CSV): | Program | CPU% | Memory (KB) | I/O Write (KB) | |———|——|————-|—————-| | Program_A | 199% | 33,920 | 4 | | Program_B | 199% | 67,200 | 4 |

Analysis: Program B (threads) shows ~2x memory because threads share address space but each allocates its own arrays. Program A processes have copy-on-write initially but diverge.

Worker Function: `io` (I/O-intensive)

Implementation: - Creates temporary files per process/thread - Writes 1MB blocks to disk repeatedly - Uses `fsync()` to force disk writes (bypass OS cache) - Reads data back from disk

Expected Behavior: Low CPU%, moderate memory, very high I/O

Observed Results (from CSV): | Program | CPU% | Memory (KB) | I/O Write (KB) | Time (s) | |———|——|————-|—————-|———-| | Program_A | 42% | 2,048 | 4,096,004 | 24.92 | | Program_B | 41% | 3,712 | 4,096,004 | 24.97 |

Analysis: Low CPU% confirms I/O bound nature. High system time indicates kernel involvement in I/O operations. Both programs show similar I/O patterns.

Part C — Measurements and Analysis

Raw Data: `MT25042_Part_C_CSV.csv`

Measurement Methodology

- **CPU Affinity:** Programs pinned to CPUs 0,1 using `taskset`
- **Metrics Collection:** `/usr/bin/time -v` for comprehensive statistics
- **Metrics Captured:** CPU%, Max RSS (memory), File system I/O, Execution time

Summary Table: All Six Combinations

Program	Worker	CPU%	Memory (KB)	I/O Write (KB)	Real Time (s)	User Time (s)	Sys Time (s)
Program_A	cpu	195	1,792	4	0.25	0.48	0.00
Program_B	cpu	195	2,304	4	0.32	0.62	0.00
Program_A	mem	199	33,920	4	36.15	72.12	0.12
Program_B	mem	199	67,200	4	42.58	84.79	0.09
Program_A	io	42	2,048	4,096,004	24.92	0.08	10.40
Program_B	io	41	3,712	4,096,004	24.97	0.05	10.29

Screenshot: `top` during CPU worker execution

Screenshot: `iostat` during I/O worker execution

Analysis and Discussion

CPU Worker Observations: - Both programs achieve near-maximum CPU utilization (~195% = 2 cores fully used) - Negligible system time indicates pure user-space computation - Program B (threads) slightly slower due to thread management overhead

Memory Worker Observations: - Program B shows 2x memory compared to Program A - This is because threads share address space but allocate separate arrays - Execution time difference reflects memory contention patterns

I/O Worker Observations: - Very low CPU% (~41-42%) confirms I/O-bound behavior - High system time (10+ seconds) indicates kernel I/O processing -

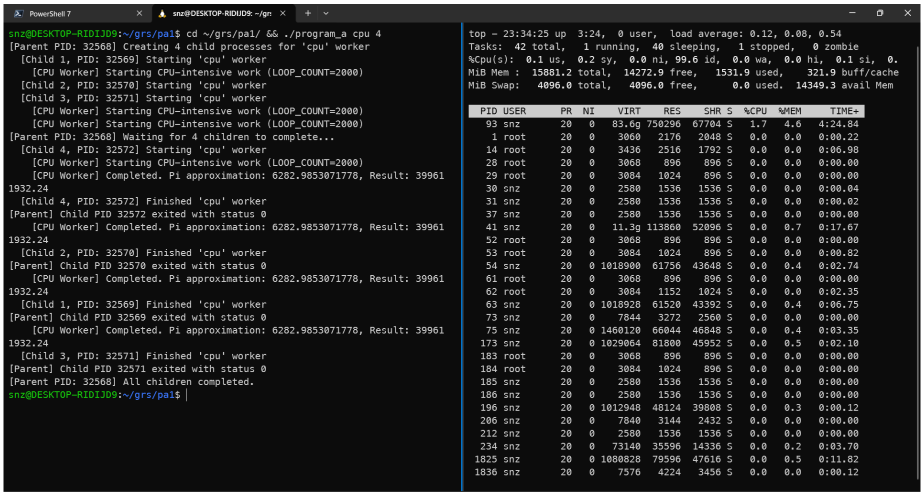

Figure 3: top output - CPU worker

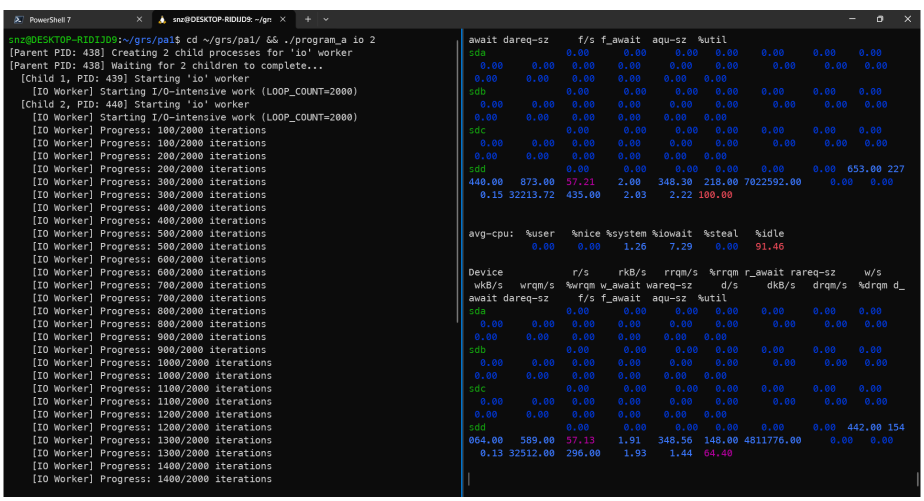

Figure 4: iostat output - IO worker

Both programs show identical I/O write volumes (as expected from same workload)

Part D — Scalability Analysis

Raw Data: `MT25042_Part_D_CSV.csv`

Test Configuration

- **Program A (Processes):** 2, 3, 4, 5 processes
- **Program B (Threads):** 2, 3, 4, 5, 6, 7, 8 threads

Plot 1: CPU Usage vs Process/Thread Count

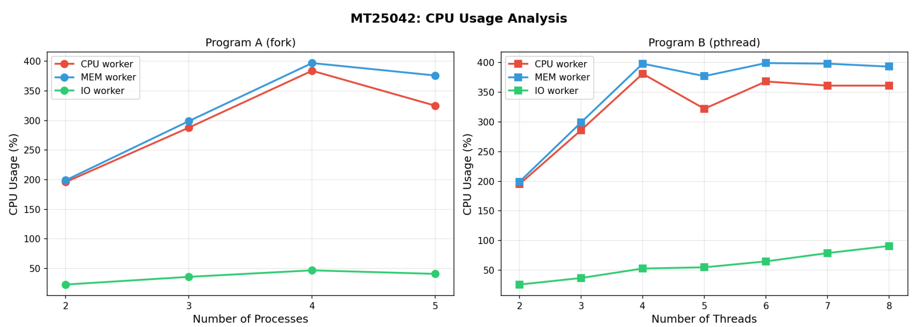

Figure 5: CPU Usage Analysis

Observations: - CPU worker: Linear scaling until hitting physical core limit - Memory worker: High CPU% maintained across counts - I/O worker: CPU% increases with count but remains low (I/O bound)

Plot 2: Execution Time vs Process/Thread Count

Observations: - CPU worker: Time remains low (sub-second) as work is distributed - Memory worker: Time increases with count due to memory bandwidth saturation - I/O worker: Time increases due to disk contention

---

Plot 3: Memory Usage vs Process/Thread Count

Observations: - Program A (processes): Memory relatively constant (each process has own space) - Program B (threads): Memory scales linearly with

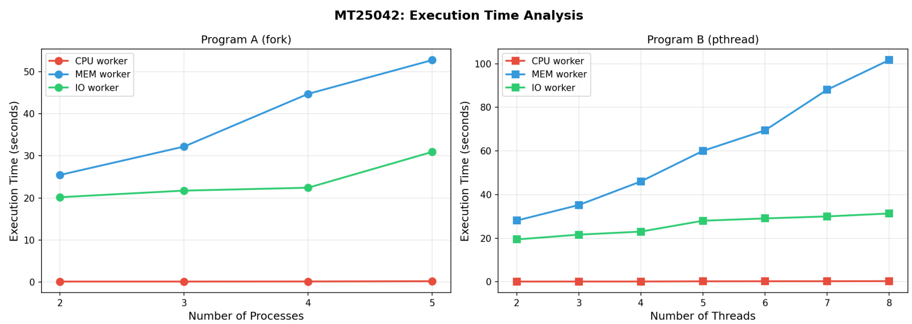

Figure 6: Execution Time Analysis

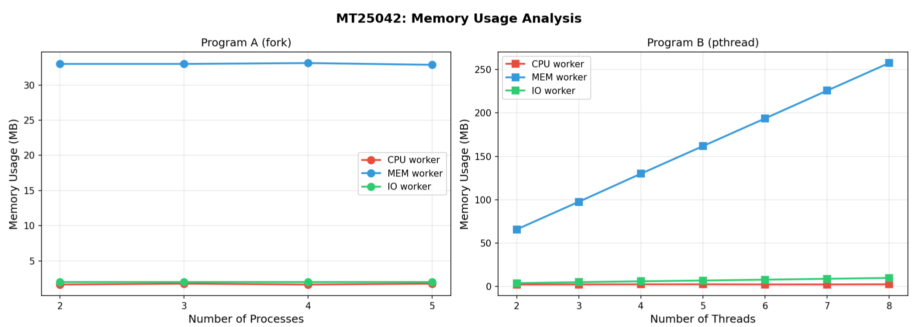

Figure 7: Memory Usage Analysis

thread count for `mem` worker - I/O worker: Minimal memory footprint regardless of count

---

Plot 4: Combined Comparison

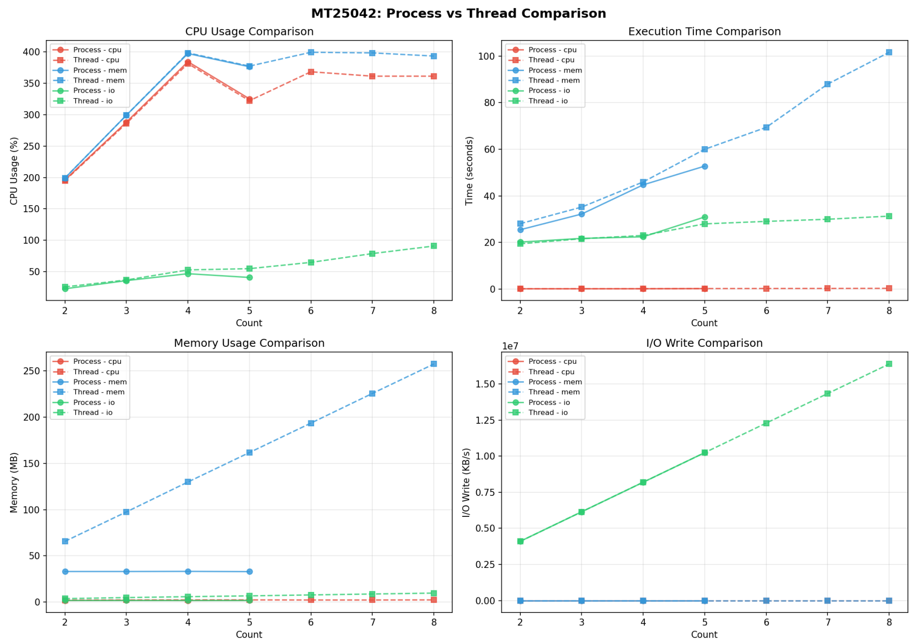

Figure 8: Combined Analysis

Scalability Discussion

Process vs Thread Comparison:

Aspect	Processes (Program A)	Threads (Program B)
Memory Overhead	Higher (separate address spaces)	Lower (shared memory)
Creation Cost	Higher (fork overhead)	Lower (lighter weight)
Isolation	Complete isolation	Shared state risks
Scalability	Limited by system resources	Better for I/O-bound tasks

Key Findings: 1. **CPU-bound tasks:** Both scale similarly until core saturation 2. **Memory-bound tasks:** Threads show higher aggregate memory due to shared accounting 3. **I/O-bound tasks:** Performance limited by disk, not parallelism model

---

AI Usage Declaration

The following components were generated with AI assistance and have been reviewed, understood, and verified:

Component	AI Assistance
`MT25042_Part_A_Program_A.c`	Code structure and fork implementation
`MT25042_Part_A_Program_B.c`	Code structure and pthread implementation
`MT25042_Part_B_workers.c`	Worker function algorithms
`MT25042_Part_B_workers.h`	Header file structure
`MT25042_Part_C_script.sh`	Bash automation script
`MT25042_Part_D_script.sh`	Bash automation script
`MT25042_Part_D_plot.py`	Python plotting script
`Makefile`	Build automation
`README.md`	Documentation

Declaration: I have reviewed and understand every line of code in this submission. I can explain the implementation details, design decisions, and defend the approach during viva examination.

GitHub Repository URL

Repository: https://github.com/shahnawazdev/GRS-Assignments

Assignment Folder: `GRS_PA01`
