Graduate Systems (CSE638) — PA01: Processes and Threads

Student Name: Rahul Pardasani
Roll No.: MT25035
Deadline: 23 Jan 2026
GitHub Repo URL: https://github.com/rahul25035/GRS_PA01

AI Usage Declaration (Mandatory)

I used AI assistance for the following components:

1. IO functions in part B, part C and part D c codes.
2. Awk commands to calculate the measurement stats inside the loop of the shell scripts in parts C and D.
3. Output formats of the stats.
4. The plotting script of part D.
5. Report and readme file.
6. Used LLMs for debugging purposes

---

1. Problem Statement

This assignment compares **process-based parallelism (fork)** and **thread-based parallelism (pthread)** by running three worker functions (**cpu**, **mem**, **io**) and measuring CPU usage, memory impact, I/O activity, and execution time.

---

2. System & Experimental Setup

2.1 Machine Details

- **OS:** Linux
- **No. of CPU cores:** 2
- **RAM:** 8 GB

2.2 Tools Used

- gcc (compilation)

- make
- top (CPU%)
- taskset (CPU pinning)
- iostat (disk stats)
- time (execution time)

2.3 Fixed Parameters Used

- **Last digit of roll number:** 5
- **Loop count (N):** (last digit × 10^3) = 5000
  (If last digit is 0, used 9 → N = 9000)

---

3. Part A : Program Implementations

3.1 Program A (Processes using fork())

File: MT25035_PartA_A.c

Goal: Create **2 child processes** (parent not counted) using fork().

Implementation Summary:

- The parent process calls fork() twice.
- Each child prints its PID and exits.
- The parent waits for both child processes using wait().

Observed Output:

```
None
Child 1: pid=<pid>

Child 2: pid=<pid>
```

3.2 Program B (Threads using pthread)

File: MT25035_PartA_B.c

Goal: Create **2 threads** (main thread not counted) using pthread.

Implementation Summary:

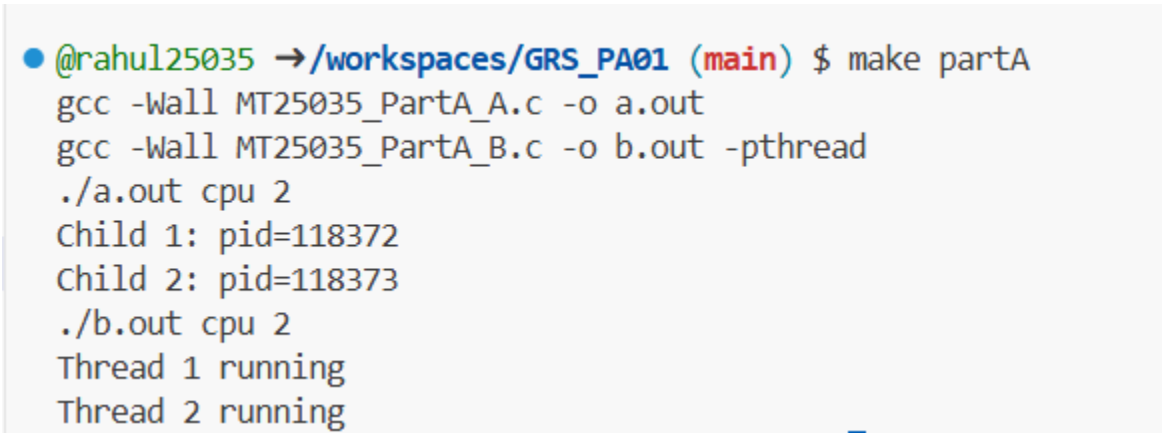
- Two threads are created using pthread_create().
- Each thread prints its thread ID.
- The main thread waits using pthread_join().

Observed Output:

```
None
Thread 1 running

Thread 2 running
```

3.3 Images



---

4. Part B : Worker Functions

Files:

- MT25035_PartB_A.c (process-based workers)
- MT25035_PartB_B.c (thread-based workers)

All worker functions execute a loop with **ITER = 5000**, derived from the last digit of the roll number (5×10^3).

4.1 Worker: cpu (CPU-Intensive)

Work Done:

- Performs deeply nested loops with arithmetic operations.
- No I/O or large memory allocation involved.

Expected Behavior:

- High CPU usage
- Minimal memory and I/O usage

4.2 Worker: mem (Memory-Intensive)

Work Done:

- Allocates a 256 MB buffer using malloc().
- Repeatedly accesses memory pages to stress RAM.

Expected Behavior:

- High memory usage
- Moderate CPU usage

4.3 Worker: io (I/O-Intensive)

Work Done:

- Repeatedly writes 4 KB buffers to a file using write().
- Forces disk writes using fsync().

Expected Behavior:

- High disk I/O activity
- Lower CPU and memory utilization due to I/O wait

5. Part C : Experiments (2 processes/2 threads)

Files:

- Script: MT25035_PartC_main.sh
- Raw Data CSV: MT25035_PartC_results.csv

5.1 Measurement Method

For each variant (A/B + cpu/mem/io):

- Used taskset to pin execution to 0 and 1 CPU cores.
- Sampled CPU% using top at regular intervals.
- Observed disk behavior using iostat.
- Measured elapsed time using time.

5.2 Results Table (Part C)

The following results summarize the automated measurements recorded in MT25035_PartC_results.csv. CPU usage is reported relative to the two CPU cores to which the program was pinned using taskset.

Program + Function	CPU Usage	Memory Usage	I/O Activity
A + cpu	Very High (~185%)	Low (~1.2 MB)	Low
A + mem	High (~136%)	High (~411 MB)	Low
A + io	Low (~7%)	Low (~1.7 MB)	High (~10,290 KB/s)
B + cpu	High (~155%)	Low (~1.4 MB)	Low
B + mem	Very High (~165%)	High (~513 MB)	Low

B + io	Low (~7%)	Low (~1.4 MB)	Very High (~40,379 KB/s)

5.3 Screenshots

```
@rahul25035 →/workspaces/GRS_PA01 (main) $ make partC
chmod +x MT25035_PartC_main.sh
./MT25035_PartC_main.sh
Compiling programs...
Compilation done
================================

components=2
Prog       CPU%     Mem        IO       Time(s)
-------------------------------------------------------------
A+cpu      175.84   1.82MB     97.53    7.00
B+cpu      184.30   1.40MB     29.60    6.91
A+mem      169.92   411.31MB   40.00    6.84
B+mem      145.98   410.80MB   19.06    6.60
A+io       9.12     1.41MB     8237.00  5.15
B+io       9.10     0.94MB     26642.00 5.29

Results saved to MT25035_PartC_results.csv
```

5.4 Analysis (Part C)

- **CPU Workload:** Both variants saturate the two pinned cores (~155-185%), confirming the task is computation-heavy.
- **Memory Workload:** High memory usage (~411-513 MB) causes a slight CPU dip compared to the CPU task due to memory access latency.
- **I/O Workload:** CPU usage drops to minimum (~7%) as workers spend most time in I/O wait for disk writes.
- **Process vs. Thread:** Thread-based (B) I/O shows significantly higher throughput (~40k vs ~10k KB/s) in this sample, suggesting lower overhead.

---

6. Part D : Scaling Experiments (vary processes/threads)

Files:

- MT25035_PartD_A.c (process-based scaling)
- MT25035_PartD_B.c (thread-based scaling)
- Raw Data CSV: MT25035_PartD_results.csv

6.1 Experiment Plan

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

All experiments use **ITER = 5000** and the same worker logic as Part C.

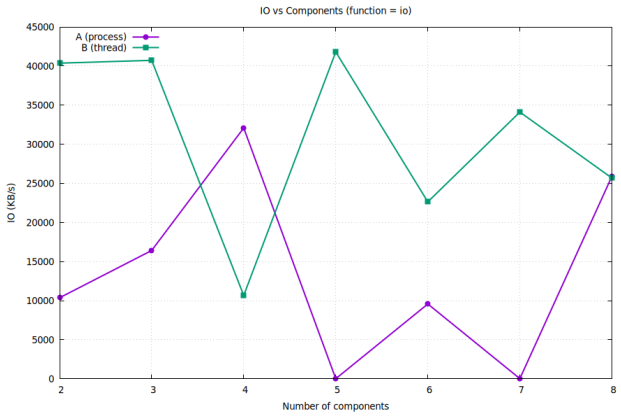
6.2 Collected Metrics

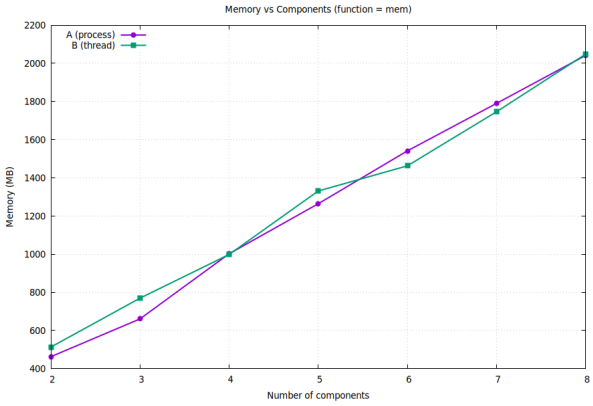
For each run, the following were recorded using the automated script:

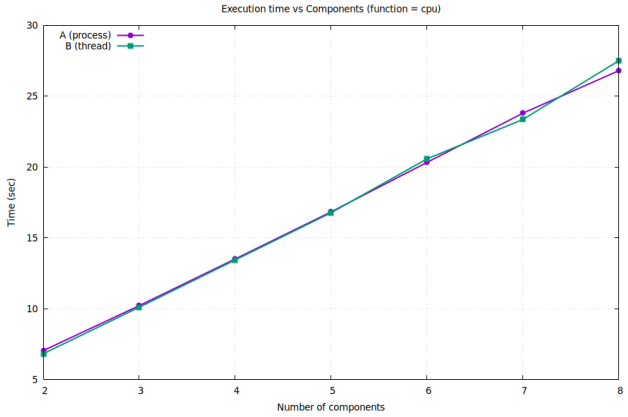
- Average CPU utilization (top)
- Execution time (time)
- Memory usage (RSS from top)
- Disk I/O activity (iostat, write rate / utilization)

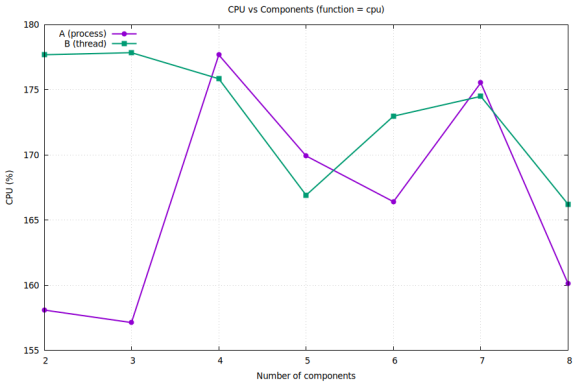
All raw values are available in MT25035_PartD_results.csv.

6.3 Plots and Images (Part D)










```
@rahul25035 ➜/workspaces/GRS_PA01 (main) $ make partD
```

components=2

Prog	CPU%	Mem	IO	Time(s)
A+cpu	158.10	1.88MB	34.37	7.050
B+cpu	177.68	1.00MB	50.40	6.830
A+mem	173.14	463.05MB	8.80	7.090
B+mem	171.76	513.51MB	67.79	7.100
A+io	9.10	1.59MB	10377.00	5.230
B+io	9.55	1.03MB	40374.00	5.230

components=3

Prog	CPU%	Mem	IO	Time(s)
A+cpu	157.14	1.96MB	17.64	10.220
B+cpu	177.84	1.27MB	36.57	10.110
A+mem	141.47	661.00MB	61.84	9.670
B+mem	183.46	769.64MB	23.43	10.340
A+io	16.18	2.50MB	16419.00	5.940
B+io	5.46	1.10MB	40731.20	6.370

components=4

Prog	CPU%	Mem	IO	Time(s)
A+cpu	177.67	2.38MB	2877.67	13.510
B+cpu	175.84	1.38MB	78.64	13.440
A+mem	178.17	1002.28MB	55.56	15.200
B+mem	188.72	998.55MB	37.60	15.300
A+io	15.10	2.88MB	32086.40	6.830
B+io	15.46	1.20MB	10668.00	6.750

components=5

Prog	CPU%	Mem	IO	Time(s)
A+cpu	169.92	2.60MB	9314.18	16.830
B+cpu	166.91	1.38MB	90.46	16.770
A+mem	181.48	1264.40MB	21.53	20.110
B+mem	192.39	1331.20MB	87.61	20.250
A+io	11.90	2.19MB	21.33	7.930
B+io	18.80	1.15MB	41805.33	8.040

components=6

Prog	CPU%	Mem	IO	Time(s)
A+cpu	166.41	3.60MB	10514.77	20.320
B+cpu	172.97	1.51MB	58.15	20.570
A+mem	179.59	1541.27MB	43.70	25.870
B+mem	175.21	1462.97MB	51.37	24.490
A+io	19.73	3.62MB	9578.00	8.780
B+io	11.46	1.50MB	22636.00	7.720

components=7

Prog	CPU%	Mem	IO	Time(s)
A+cpu	175.53	4.00MB	5457.05	23.790
B+cpu	174.49	1.39MB	36.49	23.350
A+mem	180.10	1790.11MB	22.28	30.720
B+mem	177.32	1746.19MB	68.82	29.670
A+io	15.62	3.33MB	32.67	8.180
B+io	16.30	1.50MB	34113.33	8.370

```
components=8
Prog   CPU%     Mem       IO         Time(s)
-----------------------------------------------------
A+cpu  160.12   3.18MB    4862.41    26.790
B+cpu  166.21   1.38MB    47.89      27.490
A+mem  179.61   2041.34MB 62.11      34.190
B+mem  169.43   2048.00MB 69.80      34.460
A+io   18.20    4.00MB    25892.67   9.150
B+io   15.35    1.50MB    25678.67   8.700

Results saved to MT25035_PartD_results.csv
Making Plots...
Plots created: cpu_vs_components.png, mem_vs_components.png, io_vs_components.png, time_vs_components.png
```

6.4 Observations & Discussion (Part D)

Linear Scaling (Time): Execution time for CPU and Memory tasks increases linearly with the number of components (2 to 8).

Linear Scaling (Memory): Memory usage grows by ~256 MB per component, reaching ~2 GB at 8 components.

CPU Saturation: CPU usage remains consistently high (~160-180%) regardless of component count because the 2 pinned cores stay fully saturated.

I/O Noise: I/O throughput is highly fluctuating ("noisy") due to OS disk buffering and write caching.

Resource Efficiency: Program B (threads) uses significantly less memory for non-memory tasks (e.g., 1.38 MB vs 3.18 MB for CPU task) because threads share an address space.