PA01: Processes and Threads

Course: Graduate Systems (CSE638)
Assignment: PA01 – Processes and Threads
Roll Number: MT25010
Name: Abhinay Prakash
GitHub Repository: https://github.com/IIITD-Abhinay/GRS_PA01

---

1. Introduction

Modern operating systems provide multiple mechanisms for achieving concurrency, primarily through **processes** and **threads**. While processes provide isolation through separate address spaces, threads enable lightweight concurrency by sharing memory within the same process.

The objective of this assignment is to experimentally study and compare the behavior of **process-based** and **thread-based** execution under different workload characteristics. Specifically, we evaluate CPU-intensive, memory-intensive, and I/O-intensive workloads and observe how system performance varies with increasing concurrency.

The experiments focus on measuring **CPU utilization**, **execution time**, and **I/O behavior**, using standard Linux system monitoring tools.

---

2. Implementation Details

2.1 Program A – Process-Based Execution

Program A creates multiple child processes using the fork() system call. Each child process independently executes one of the worker functions (cpu, mem, or io). The parent process waits for all child processes to complete using wait().

Each child process has its own address space, which introduces overhead in terms of memory usage and context switching.

2.2 Program B – Thread-Based Execution

Program B creates multiple threads using the POSIX threads (pthread) library. Each thread executes one of the worker functions. All threads share the same address space, allowing efficient memory sharing and lower creation overhead compared to processes.

Threads are synchronized using pthread_join() to ensure proper completion.

2.3 Worker Functions

Three worker functions were implemented to simulate different workload types:

- **CPU-intensive (cpu)**:
  Performs repeated mathematical computations in a tight loop, keeping the CPU busy with minimal memory or I/O operations.

- **Memory-intensive (mem)**:
  Allocates large memory buffers and repeatedly accesses them, stressing the memory subsystem and CPU cache.
- **I/O-intensive (io)**:
  Performs repeated file write operations, causing the program to spend significant time waiting for disk I/O.

To ensure measurable execution times, the loop count was scaled proportionally based on the roll number, allowing reliable system-level observations.

---

3. Experimental Setup

All experiments were conducted on a Linux-based system (Ubuntu) using a virtualized environment.

- taskset – to pin execution to a specific CPU core
- top – to observe CPU utilization
- iostat – to observe disk I/O behavior
- time – to measure execution time
- Bash scripts – to automate execution and data collection

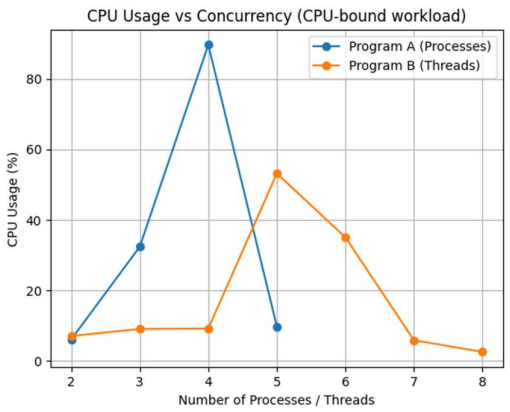
CPU pinning was used to reduce scheduling variability and improve consistency across runs. All measurements were collected automatically and stored in CSV format.

---

4. Results

4.1 CPU Utilization

Figure 1 shows CPU utilization as a function of the number of processes or threads for the CPU-intensive workload.

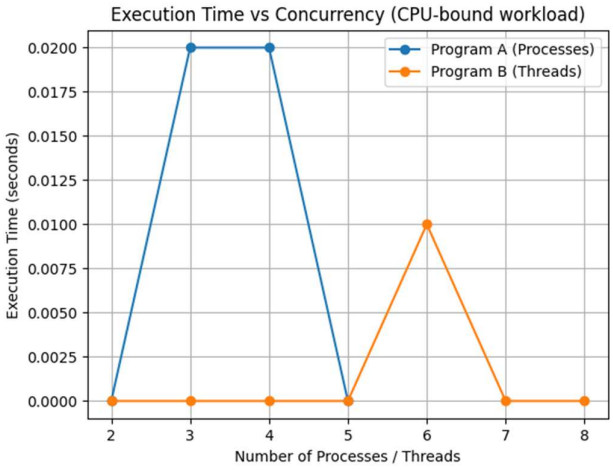


For process-based execution, CPU utilization increases initially but exhibits variability at higher process counts due to increased context switching and process management overhead. Thread-based execution demonstrates better scaling at lower concurrency levels, owing to lower overhead and shared memory usage.

Figure 1: CPU usage vs number of processes/threads

4.2 Execution Time

Figure 2 shows execution time versus the number of processes or threads for the CPU-intensive workload.



Execution time generally increases with higher concurrency levels. While threads initially provide better performance, both approaches experience diminishing returns as CPU cores become saturated.

Figure 2: Execution time vs number of processes/threads

---

5. Discussion

The experiments highlight fundamental differences between processes and threads:

- **Processes** incur higher overhead due to separate address spaces and heavier context switching.
- **Threads** scale more efficiently at lower concurrency levels but eventually suffer from contention and scheduling overhead.
- **CPU-bound workloads** benefit most from parallelism until core saturation.
- **Memory-bound workloads** are limited by cache and memory bandwidth.
- **I/O-bound workloads** show minimal improvement with increased concurrency, as performance is dominated by disk latency rather than CPU availability.

Minor variability in CPU utilization measurements is expected due to short execution times and operating system scheduling effects. Aggregate CPU utilization measured via top provides a reasonable comparison of relative trends across workloads.

---

6. AI Usage Declaration

AI tools (ChatGPT) were used for guidance in understanding system concepts, debugging code, and structuring automation scripts. All code, experiments, and results were reviewed, understood, and validated by the author.

---

7. Conclusion

This assignment demonstrates the practical differences between process-based and thread-based concurrency. While threads offer lower overhead and better scalability under certain conditions, processes provide stronger isolation at the cost of performance. The choice between processes and threads depends on workload characteristics and system constraints.

The experimental results reinforce theoretical expectations regarding CPU, memory, and I/O behavior in concurrent systems.

---

8. References

1. Silberschatz, Galvin, and Gagne, *Operating System Concepts*
2. Linux manual pages: fork(2), pthread_create(3), top(1), iostat(1)