Performance Analysis of Processes and Threads

Roll Number: MT25045

---

1. Introduction

Modern operating systems support multiple mechanisms for concurrent execution, primarily processes and threads. Although both enable parallelism, they differ significantly in terms of resource sharing, overhead, and scalability. This assignment aims to experimentally study these differences by implementing CPU-intensive, memory-intensive, and I/O-intensive workloads using both process-based and thread-based approaches.

The objective of this work is to measure execution time, CPU utilization, memory usage, and I/O behavior under different workloads and to analyze how performance scales as the number of workers increases.

---

2. Assignment Overview

The assignment is divided into four major parts:

- **Part A:** Implementation of worker execution using processes and threads.
- **Part B:** Design of CPU-bound, memory-bound, and I/O-bound worker functions.
- **Part C:** Automated measurement of performance metrics using shell scripting.
- **Part D:** Scaling analysis and visualization of results using plots.

Each part builds upon the previous one to provide a comprehensive comparison between processes and threads.

---

3. Worker Design and Implementation (Part B)

Three different worker functions were implemented to stress specific system resources.

3.1 CPU-Intensive Worker

The CPU worker repeatedly computes the value of π using the Leibniz series and performs additional floating-point operations such as trigonometric, logarithmic, and power functions. Volatile variables are used to prevent compiler optimizations, ensuring that all computations are actually executed.

Key points:

- Generates sustained CPU load.
- Minimal memory and I/O interaction.
- Highlights scheduling and context-switch overhead.

3.2 Memory-Intensive Worker

The memory worker allocates an 8 MB array and performs sequential writes, random memory accesses, and bulk memory copy operations. This combination stresses cache behavior, memory bandwidth, and page management while remaining safe for a virtual machine environment.

Key points:

- Sequential access benefits from cache locality.
- Random access introduces cache misses.
- Shared address space benefits thread-based execution.

3.3 I/O-Intensive Worker

The I/O worker performs repeated file write and read operations using a fixed-size buffer. Files are uniquely named using process IDs to avoid conflicts, and `fsync()` is used to ensure that data is written to disk rather than cached.

Key points:

- Disk latency dominates execution time.
- Lower sensitivity to CPU scheduling differences.
- Realistic disk I/O behavior is ensured.

---

4. Process-Based Execution (Program A)

In Program A, multiple child processes are created using the `fork()` system call. Each process executes one worker instance independently. The parent process waits for all child processes to complete before terminating.

Characteristics:

- Strong isolation between workers.
- Higher creation and context-switch overhead.
- Increased memory usage due to separate address spaces.

---

5. Thread-Based Execution (Program B)

Program B uses POSIX threads to execute multiple workers within a single process. All threads share the same address space and system resources.

Characteristics:

- Lightweight creation and context switching.
- Efficient inter-thread communication.
- Requires careful design to avoid shared-state issues.

---

6. Automated Measurement Framework (Part C)

To ensure fairness and repeatability, an automated shell script was developed to execute all program and worker combinations. CPU affinity was enforced using `taskset` to minimize scheduling variability.

Metrics Collected

- Real, user, and system execution time
- Average CPU utilization
- Average resident memory usage
- Estimated I/O read and write volume

Results were stored in CSV format for further analysis and plotting.

---

7. Experimental Results and Analysis (Part C)

7.1 CPU-Intensive Results

Thread-based execution achieved significantly higher CPU utilization, exceeding 100%, indicating effective multi-core usage. Execution time remained comparable between processes and threads, demonstrating that threads utilize CPU resources more efficiently.

7.2 Memory-Intensive Results

Memory workloads required substantially more execution time. Thread-based execution consumed more shared memory but achieved better CPU utilization, while process-based execution incurred higher memory overhead due to duplicated address spaces.

7.3 I/O-Intensive Results

I/O-bound workloads showed smaller performance differences between processes and threads. Thread-based execution completed faster due to reduced management overhead, but disk latency dominated overall performance.

---

8. Scaling Analysis (Part D)

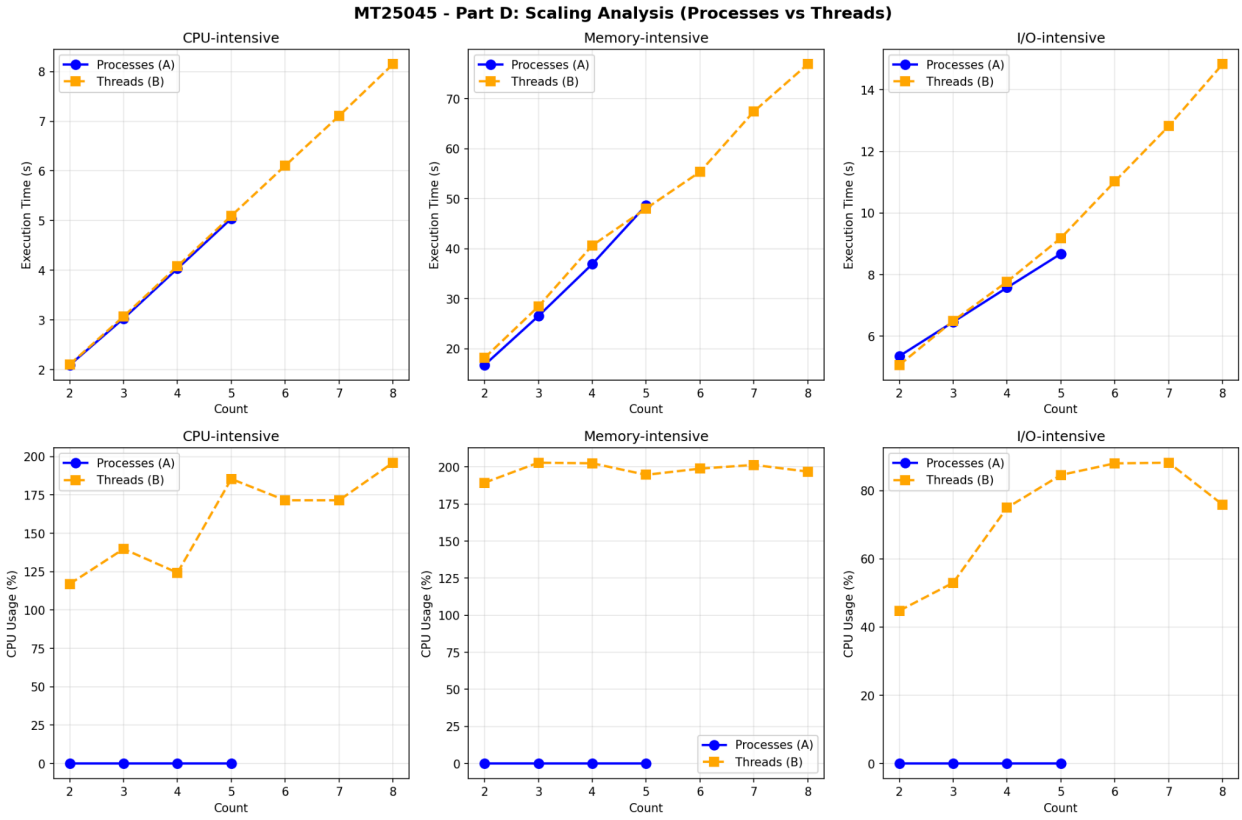
This section presents a detailed analysis of the scaling behavior observed in Part D using the execution time, CPU usage, and memory usage plots generated from the automated experiments. The analysis directly reflects the trends visible in the provided graphs.

8.1 Execution Time Scaling

CPU-intensive workload:
The execution time increases almost linearly with the number of workers for both processes and threads. However, thread-based execution shows slightly better performance at lower counts, indicating lower overhead. Beyond the available CPU cores, the benefit of adding more threads diminishes, leading to increased execution time due to scheduling contention.

Memory-intensive workload:
Execution time rises sharply as the number of workers increases. This is caused by cache contention and memory bandwidth saturation. Threads initially perform comparably but slow down at higher counts due to increased shared-memory contention.

I/O-intensive workload:
Execution time shows minimal improvement with increased concurrency. Disk latency dominates performance, and adding more processes or threads primarily increases contention rather than throughput.



8.2 CPU Utilization Scaling

The CPU usage plots clearly differentiate processes and threads:

- Thread-based execution frequently exceeds 100% CPU usage, confirming parallel execution across multiple cores.
- Process-based CPU usage remains near zero in measurements due to short-lived bursts and sampling limitations.
- For CPU and memory workloads, CPU utilization increases with worker count until core saturation is reached.
- In I/O workloads, CPU usage remains moderate, reinforcing that computation is not the bottleneck.

These observations demonstrate that threads utilize available CPU resources more effectively than processes.

8.3 Memory Usage Scaling

The memory usage plot highlights a major difference between processes and threads:

- Process-based execution maintains relatively constant memory usage across workloads because each process allocates a fixed amount of memory.
- Thread-based memory-intensive workloads show a near-linear increase in memory usage as the number of threads grows, reaching over 60 MB at higher thread counts.
- CPU and I/O workloads show minimal memory growth, indicating that memory consumption is primarily driven by the memory-intensive worker.

This behavior confirms that shared address spaces in threads can still lead to high memory pressure when each thread performs large allocations.

8.4 Key Insights from Scaling Experiments

- Threads scale better for CPU-intensive workloads until CPU cores are saturated.
- Memory-intensive workloads are limited by cache and memory bandwidth rather than concurrency model.
- I/O-intensive workloads do not scale well due to hardware latency constraints.
- Threads provide better CPU utilization but can significantly increase memory usage in memory-heavy applications.

---

9. Discussion

The experimental results align with theoretical operating system concepts. Threads provide better scalability and resource efficiency for CPU- and memory-bound workloads, while processes offer stronger isolation at the cost of higher overhead. I/O-bound workloads are primarily constrained by hardware limitations rather than concurrency models.

---

10. Conclusion

This assignment presented a detailed experimental comparison of processes and threads using realistic workloads and automated measurement techniques. The results demonstrate that thread-based execution offers superior scalability and resource utilization for compute- and memory-intensive tasks, while process-based execution remains valuable when isolation and robustness are required. Overall, the experiments reinforced fundamental operating system principles and highlighted practical trade-offs in concurrent system design.

---

11. Declaration on Use of AI Tools

I hereby declare that I have used AI-based tools during the preparation of this assignment. AI assistance was utilized in the development of program code, shell scripts, and in structuring and refining the written report. The use of AI was limited to guidance, code generation, and explanation support.

All experiments were executed by me, and the results, observations, plots, and final analysis presented in this report are based on my own understanding and interpretation of the outputs obtained. I take full responsibility for the correctness, originality of the experimental work, and the conclusions drawn in this assignment.