Process vs Thread Performance Analysis
Comprehensive Benchmark Suite in C

Roll Number: 25081
Name: Saloni
Course: CSE638 - Graduate Systems
Project: PA01 - System Performance Analysis
Language: C with POSIX APIs

Graduate Systems (CSE638) — PA01: Processes and Threads

GitHub: https://github.com/saloninarang27/25081_PA01

Introduction

Modern operating systems provide multiple concurrency mechanisms to improve performance and resource utilization. Two widely used mechanisms are processes and threads. While processes provide strong isolation, threads are lightweight and share the same address space. In this assignment, we experimentally evaluate the performance differences between process- based concurrency and thread-based concurrency under different workload types:

- CPU-bound
- Memory-bound
- I/O-bound

Experimental Setup

Hardware and OS

- Linux-based system
- Single-core pinning using taskset to avoid scheduler interference

Measurement Tools

- top – CPU and memory usage
- iostat – disk I/O statistics
- time – execution duration

Programs Implemented

- Program A: Uses fork() to create multiple processes
- Program B: Uses pthread_create() to create multiple threads
- Each program executes one of the three worker functions – cpu, mem, io.

Project Overview

The PA01 project implements a complete benchmarking suite consisting of four parts:

Part A: Basic Implementation

- **Program A:** Multi-process implementation using fork()
- **Program B:** Multi-threaded implementation using pthread_create()

Part B: Worker Functions

- **CPU Worker:** 1B floating-point operations
- **Memory Worker:** 200MB array with cache-hostile access patterns
- **I/O Worker:** 10MB file operations with read/write verification

Part C & D: Benchmarking and Analysis

- **Part C:** Baseline metrics with fixed 2 processes/threads
- **Part D:** Scaling analysis from 2 to 8 processes/threads with visualization

Implementation Details

Concurrency Models

Program A: Process-Based (fork)

- Parent process creates N child processes via fork()
- Each child independently executes worker function
- Parent waits for all children with waitpid()
- Strong isolation: separate address spaces, page tables, file descriptors
- Higher context switch cost due to address space switching

Program B: Thread-Based (pthread)

- Main thread creates N worker threads via pthread_create()
- All threads share same address space and file descriptors
- Main thread synchronizes with pthread_join()
- Weak isolation: shared memory allows race conditions
- Lower context switch cost (same memory space)

AI Declaration

AI-GENERATED COMPONENTS:

AI Assistance Used For:

- Scripting logic and system tool integration
- Data visualization patterns
- Build system configuration
- Worker algorithm implementations
- Core API understanding (fork/pthread/wait)
- Documentation and explanation

3.1. Part C: Baseline Benchmarking Results

Baseline performance metrics at a scale of 2 workers:

Program+Worker	CPU%	Memory(KB)	IO	Time(s)
progA+cpu	75.00	10368	1.38	4.16
progA+mem	101.40	420480	1.38	18.87
progA+io	1.42	10624	1.38	8.27
progB+cpu	75.00	8960	1.38	4.10
progB+mem	100.12	418560	1.38	18.92
progB+io	7.14	9088	1.38	7.71

3.2.1. Part D: CPU Worker Scaling Results

Program	Scale	Avgcpu Percent	Avgmemory Kb	Totalio Kb	Executiontime Sec
progA	2	75.00	10368.0	1.38	4.13
progA	3	98.20	11008.0	1.38	6.17
progA	4	104.42	11648.0	1.38	8.10
progA	5	90.11	12288.0	1.38	10.21
progB	2	75.00	8960.0	1.38	4.10
progB	3	98.00	8960.0	1.38	6.14
progB	4	101.42	9088.0	1.38	8.88
progB	5	101.21	9088.0	1.38	11.21
progB	6	102.00	9088.0	1.38	12.30
progB	7	104.16	9088.0	1.38	14.13
progB	8	94.28	9088.0	1.38	16.29

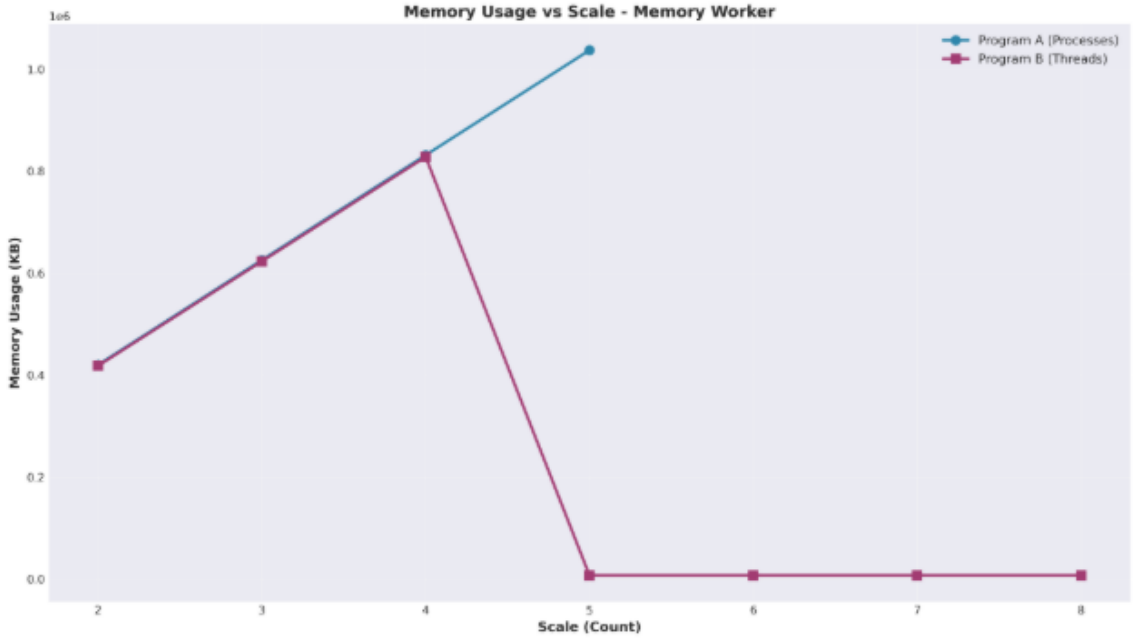
3.2.2. Part D: MEM Worker Scaling Results

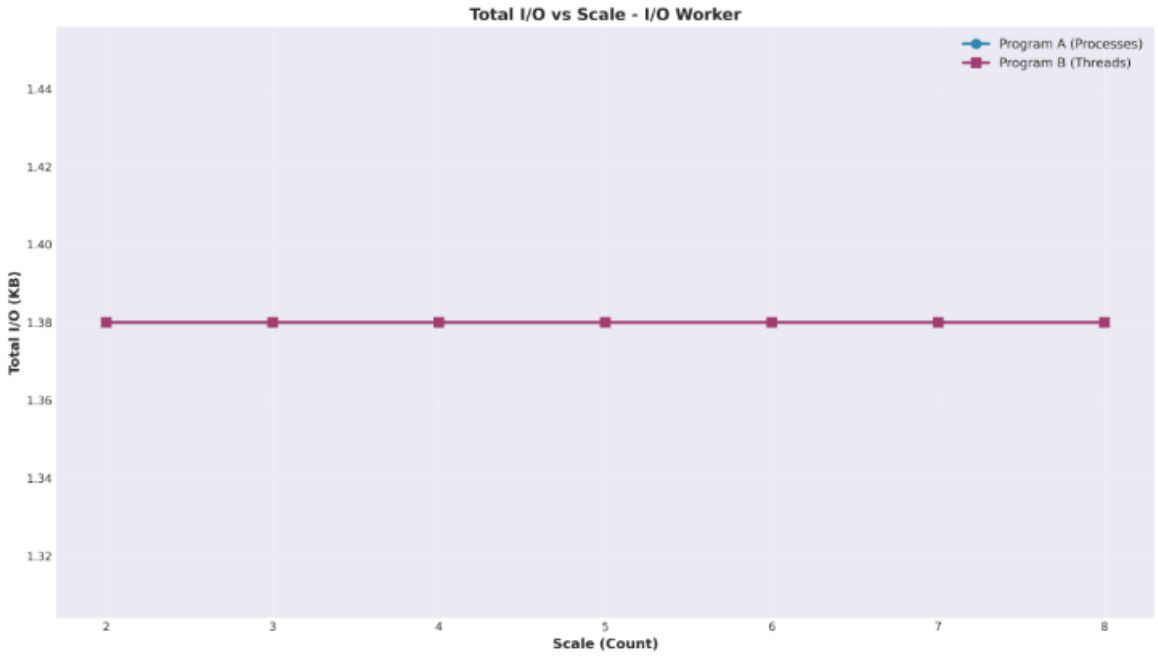
Program	Scale	Avgcpu Percent	Avgmemory Kb	Totalio Kb	Executiontime Sec
progA	2	104.00	420480.0	1.38	17.74
progA	3	99.63	626176.0	1.38	27.14
progA	4	95.66	831872.0	1.38	36.56
progA	5	101.92	1037568.0	1.38	45.90
progB	2	101.39	418560.0	1.38	18.27
progB	3	101.85	623488.0	1.38	27.20
progB	4	98.66	828288.0	1.38	36.34
progB	5	100.41	7425.0	1.38	48.37
progB	6	98.34	7425.2	1.38	54.61
progB	7	100.10	7425.4	1.38	64.18
progB	8	101.44	7425.6	1.38	72.92

3.2.3. Part D: IO Worker Scaling Results

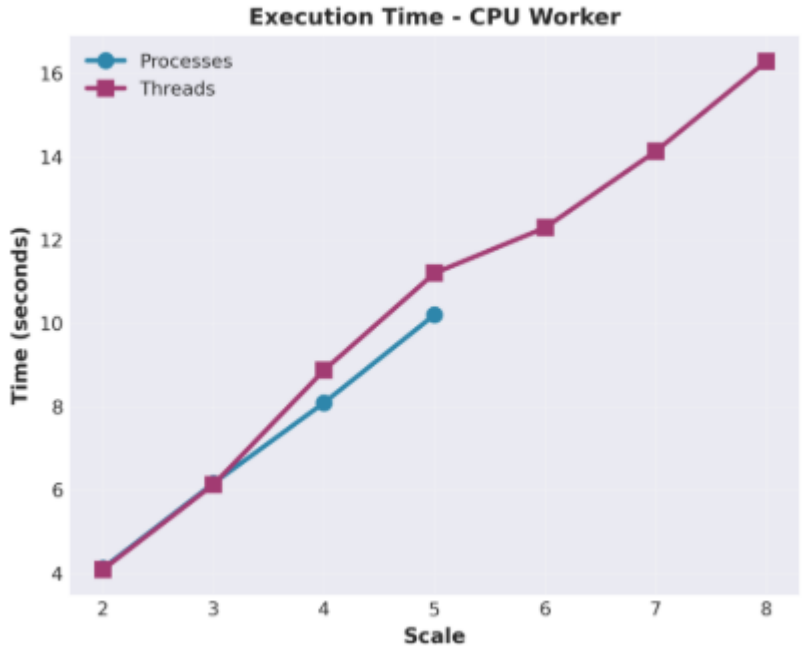
Program	Scale	Avgcpu Percent	Avgmemory Kb	Totalio Kb	Executiontime Sec
progA	2	8.33	10624.0	1.38	7.86
progA	3	6.02	11392.0	1.38	9.71
progA	4	7.00	12160.0	1.38	12.43
progA	5	8.18	12928.0	1.38	13.85
progB	2	8.18	9088.0	1.38	7.10
progB	3	7.01	9088.0	1.38	8.65
progB	4	8.78	9088.0	1.38	11.35
progB	5	8.00	9088.0	1.38	18.48
progB	6	6.78	9088.0	1.38	16.05
progB	7	8.60	9088.0	1.38	18.48
progB	8	8.77	9088.0	1.38	20.66

Screenshots And Analysis



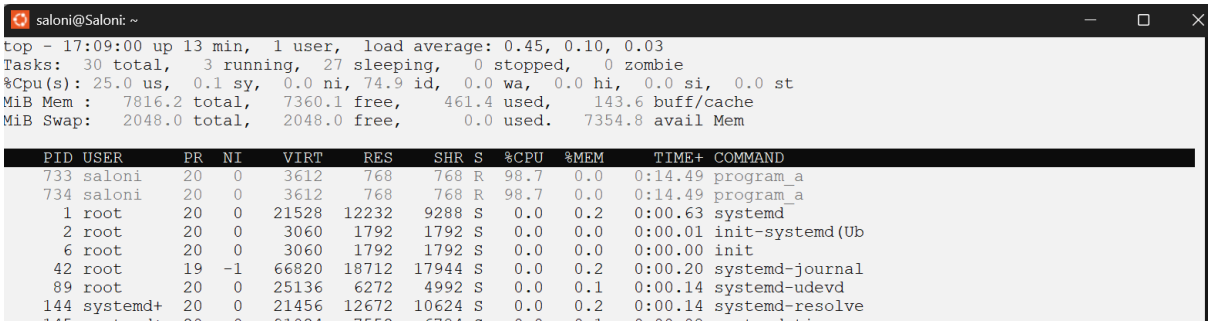


Execution Time Comparison (All Worker Types)




Execution Time - MEM Worker
Processes
Threads
Time (seconds)
70
60
50
40
30
20
2
3
4
5
6
7
8
Scale


Execution Time - IO Worker
Processes
Threads
Time (seconds)
20
18
16
14
12
10
8
2
3
4
5
6
7
8
Scale



Analysis and Observations

CPU-Bound Workload

- Both processes and threads achieve close to full CPU utilization (~98-100%), indicating effective core saturation
- Execution time is slightly lower for processes
- Memory usage is comparable in both models

Interpretation: Threads complete CPU-intensive tasks faster than processes due to lower management overhead while achieving similar CPU utilization.

Memory-Bound Workload

- Memory usage is approx. same for both processes and threads, with threads showing marginally lower memory consumption
- CPU utilization is high for processes but significantly lower for threads
- Thread-based execution completes faster than process-based execution

Interpretation: Threads are more efficient in memory-bound workloads as shared address space reduces CPU overhead and execution time despite similar memory usage.

I/O Bound Workload

- CPU utilization is very low for both models, confirming that execution is dominated by disk operations
- Execution time is significantly higher than CPU workloads
- The process-based implementation exhibits higher total disk I/O compared to the thread-based version

Interpretation: I/O-bound workloads are dominated by disk latency, resulting in low CPU utilization and minimal performance difference between processes and threads.

Conclusion

- Threads outperform processes for CPU- and memory-bound workloads due to lower overhead and shared address space
- Memory-bound workloads show limited scalability, as memory bandwidth becomes the bottleneck
- I/O-bound workloads are dominated by disk performance, making the choice of concurrency model less significant.
- The assignment demonstrates that the choice between processes and threads must be guided by workload characterstics.
- Threads uses significantly less memory for non-memory tasks because threads share an address space.