

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	5.43	4.15
progA+mem	97.60	420480	5.43	17.98
progA+io	4.15	10624	17.43	7.75
progB+cpu	77.50	8960	17.43	4.15
progB+mem	98.18	418560	5.42	17.90
progB+io	4.28	9088	25.41	7.71

3.2.1. Part D: CPU Worker Scaling Results

Program	Scale	Avgcpu Percent	Avgmemory Kb	Totalio Kb	Executiontime Sec
progA	2	72.75	10368	5.74	4.24
progA	3	97.84	11008	17.73	6.39
progA	4	95.98	11520	5.7	8.44
progA	5	101.11	12288	5.7	10.69
progB	2	75.00	8960	9.66	4.25
progB	3	100.00	8960	5.66	6.40
progB	4	100.00	9088	5.63	9.30
progB	5	97.97	9088	377.61	10.59
progB	6	90.90	9088	5.62	12.71
progB	7	91.67	9088	5.59	15.19
progB	8	99.41	9088	5.56	17.23

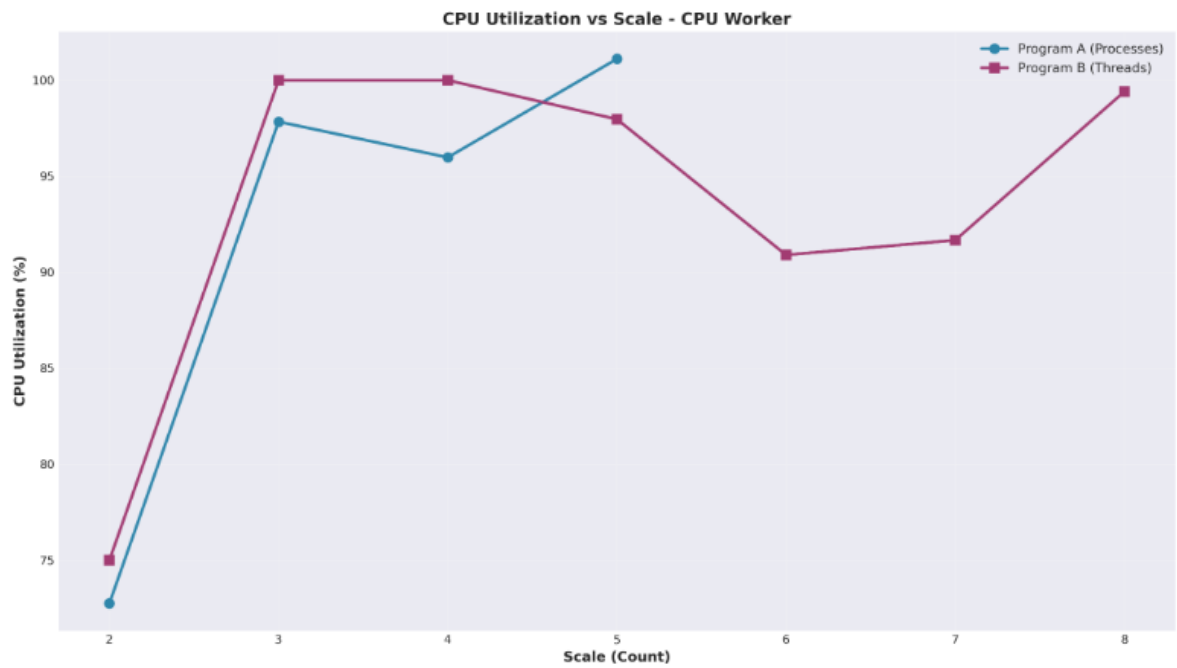
3.2.2. Part D: MEM Worker Scaling Results

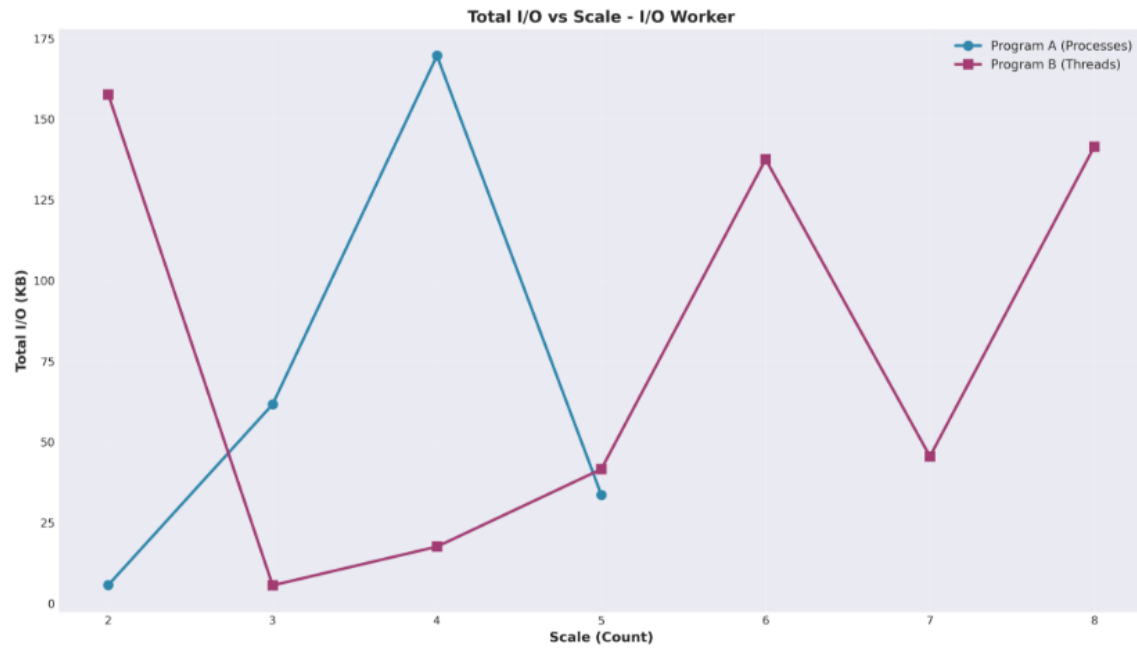
Program	Scale	Avgcpu Percent	Avgmemory Kb	Totalio Kb	Executiontime Sec
progA	2	91.49	420480	177.73	18.85
progA	3	96.55	626176	13.73	27.05
progA	4	102.31	831872	161.7	36.64
progA	5	99.34	1037568	181.69	46.68
progB	2	92.61	418560	25.66	18.89
progB	3	96.69	623488	69.66	27.10
progB	4	95.89	828288	169.63	37.21
progB	5	98.74	7425	81.65	45.22
progB	6	99.85	7425.2	205.61	54.63
progB	7	99.04	9404	357.58	66.52
progB	8	97.12	7425.6	245.55	73.75

3.2.3. Part D: IO Worker Scaling Results

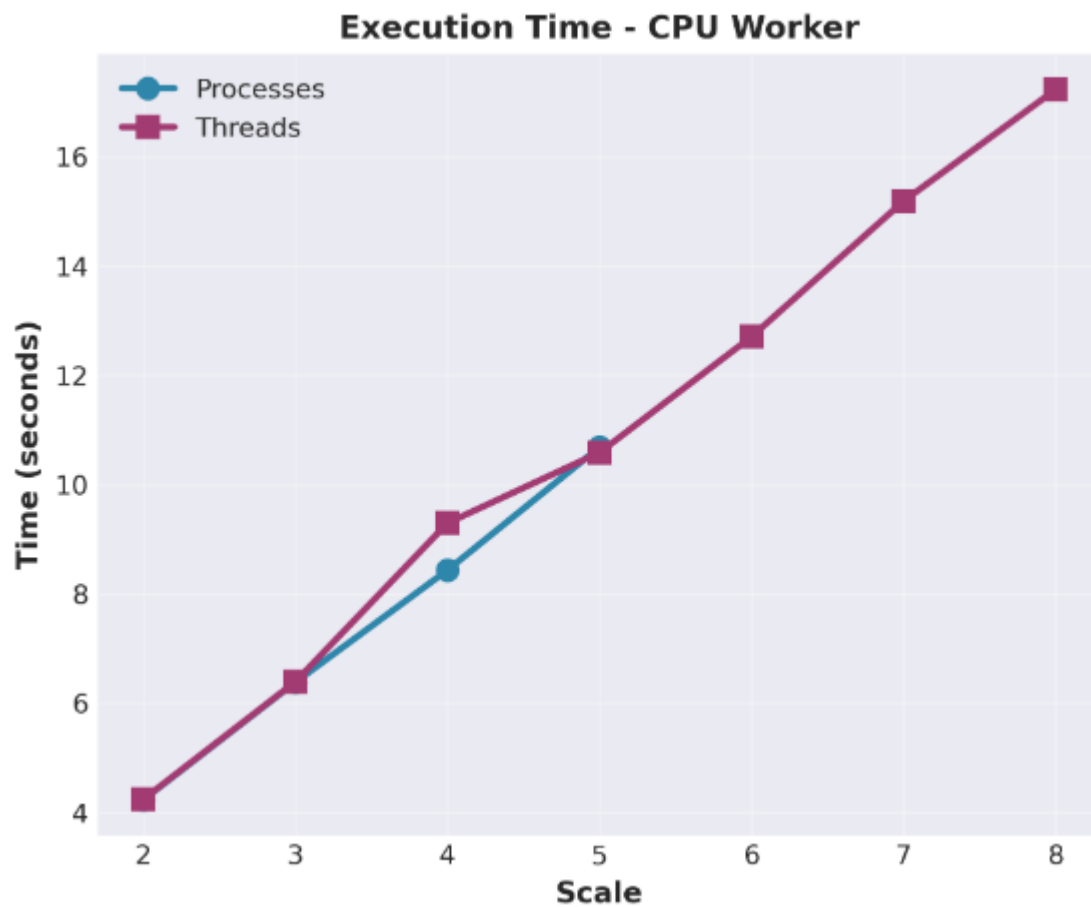
Program	Scale	Avgcpu Percent	Avgmemory Kb	Totalio Kb	Executiontime Sec
progA	2	6.00	10624	5.74	11.60
progA	3	7.00	11392	61.7	11.91
progA	4	9.23	12160	169.69	15.87
progA	5	5.00	12928	33.68	19.82
progB	2	3.23	9088	157.65	10.93
progB	3	7.00	9088	5.64	11.84
progB	4	5.00	9088	17.62	16.40
progB	5	6.36	9088	41.62	16.86
progB	6	9.88	9088	137.59	19.91
progB	7	8.18	9088	45.58	27.53
progB	8	9.42	9088	141.53	27.76

Screenshots And Analysis





Execution Time Comparison (All Worker Types)





saloni@Saloni: ~										
top - 17:09:00 up 13 min, 1 user, load average: 0.45, 0.10, 0.03										
Tasks: 30 total, 3 running, 27 sleeping, 0 stopped, 0 zombie										
%Cpu(s): 25.0 us, 0.1 sy, 0.0 ni, 74.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st										
MiB Mem : 7816.2 total, 7360.1 free, 461.4 used, 143.6 buff/cache										
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used, 7354.8 avail Mem										
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
733	saloni	20	0	3612	768	768	R	98.7	0.0	0:14.49 program_a
734	saloni	20	0	3612	768	768	R	98.7	0.0	0:14.49 program_a
1	root	20	0	21528	12232	9288	S	0.0	0.2	0:00.63 systemd
2	root	20	0	3060	1792	1792	S	0.0	0.0	0:00.01 init-systemd(Ub
6	root	20	0	3060	1792	1792	S	0.0	0.0	0:00.00 init
42	root	19	-1	66820	18712	17944	S	0.0	0.2	0:00.20 systemd-journal
89	root	20	0	25136	6272	4992	S	0.0	0.1	0:00.14 systemd-udev
144	systemd+	20	0	21456	12672	10624	S	0.0	0.2	0:00.14 systemd-resolve

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

Analysis: 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

Analysis: 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

Analysis: 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 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 characteristics.
- Threads use significantly less memory for non-memory tasks because threads share an address space.