Graduate Systems (CSE638)

PA02: Multithreaded Program Analysis using “perf” tool

[40+16 = 56 points]

Deadline: Part A: February 13, 2025; Part B: March 2, 2025

Objective

The goal of this assignment is to:

1. Implement multithreaded program variants to explore CPU-bound, memory-bound, I/O-bound, and mixed workloads.
2. Analyze the performance of these programs using the `perf` tool.
3. Develop a producer-consumer pipeline program with threads and evaluate its performance using `perf`.

Instructions

Part A: Multithreaded Program Variants [40 points]

1. **Task**: **[4/prog; 4*4 = 16 points]**
    - Write **FOUR** separate multithreaded programs (using `pthread` library), each representing a different workload type:
        - **CPU-bound**: Perform intensive mathematical calculations (e.g., matrix multiplication, prime number computation).
        - **Memory-bound**: Perform operations that involve large memory access patterns (e.g., traversing large arrays repeatedly).
        - **I/O-bound**: Perform operations involving frequent file read/write (or network communication).
        - **Mixed workload**: Combine operations from CPU, memory, and I/O-bound tasks in a single program. Use your creativity here.
2. **Requirements**:
    - Use the `pthread` library to implement threads in all programs.
    - Measure and print execution time for each program variant.
3. **Performance Analysis of FOUR programs with `perf`**: **[4*2 + 4 = 12 points]**
    - Use the `perf` tool to profile and analyze each program.

- Analyze metrics such as CPU usage, memory bandwidth (memory reads/writes per sec), cache references, cache misses, and I/O wait time.
- Compare the performance of different variants (i.e., four programs in part 1) **(use plots),** and analyze, and summarize observations.

4. **Performance Analysis (Scalability test)**: **[4*2 + 4 = 12 points]**
   - Increase the number of concurrent threads for each of the FOUR programs mentioned in **Part A.1** to test system scalability.
     - Note: you can increase the number of threads based on what your underlying system supports. For example, you can do this for 2, 4, 8, 10, 50, 100 threads. You stopped at 100 threads because you found that the performance degraded after that.
   - Use the `perf` tool to profile and analyze each variant. Analyze metrics such as CPU usage, memory bandwidth (memory reads/writes per sec), cache references, cache misses, branch misses, context switches, CPU migrations, and I/O wait time.
   - Compare the performance of the same program across a number of concurrent threads to analyze program scalability **(use plots)** and summarize observations.

Part B: Pipeline Program with Producer-Consumer Threads [16 points]

1. **Task**:
   - Implement a multithreaded program in which: **[2+2+2+2 = 8 points]**
     - **Producer threads** generate data and place it into a shared buffer (e.g., populate 100 random numbers to the shared array).
     - **Consumer threads** retrieve data from the shared buffer and process it (e.g., compute averages or perform transformations).
     - Each Producer-Consumer thread pair has its own shared buffer. I.e., You have “N” buffers for “N” producer-consumer thread pairs.
   - Use appropriate synchronization primitives (e.g., mutexes and condition variables) to avoid race conditions and ensure correct behavior.
2. **Requirements**: **[2 points]**
   - Ensure threads operate efficiently without deadlocks or busy-waiting.
   - Measure the program's execution time and throughput (e.g., data items processed per second).
3. **Performance Analysis with `perf`**: **[2+2+2 = 6 points]**
   - Use the `perf` tool to analyze the performance of the producer-consumer pipeline.
   - Focus on metrics like context switches and cache behavior.

- Experiment by varying the numbers of producer and consumer threads. Plot, analyze, and document the results.

Deliverables

1. Source code files for all programs.
2. A detailed report including:
    - Workload explanation for each program in PART A & B (separately for the corresponding submission).
    - `perf` analysis for each program:
        - Key performance metrics observed.
        - Graphs or tables comparing metrics across different variants.
        - Screenshots of `perf` tool outputs for each program.
    - Discussion of results, including bottlenecks and optimization opportunities.
3. Add all of these in a folder named <roll_num>_PA02, and upload the compressed file, <roll_num>_PA02.zip

---

Expected Outcome

This assignment helps students develop a deeper understanding of multithreaded programming, performance profiling, and the use of tools like `perf` to optimize and analyze real-world workloads.