COS 226 ASSIGNMENT 2 BAHIYA HOOSEN

U22598546

INTRODUCTION

As tests pile up, waiting for them to finish can feel like watching paint dry. But fear not! Running tests concurrently lets us turbocharge the process, turning a slow crawl into a speedy sprint. With multithreading, we can tackle multiple tests at once, making bugs easier to spot and fixing things faster—no more waiting around! This report outlines the development of a multi-threaded testing framework designed to improve test execution efficiency. The performance of the framework is analyzed by comparing the time taken to run tests sequentially versus concurrently.

Sequential Test Execution

1. **Execution Summary:**
    - Tests Executed: 12
    - Tests Passed: 10
    - Tests Failed: 2
    - Tests Skipped: 0
    - Total Execution Time: **1852 ms**

Multi-Threaded Test Execution

1. **Execution Summary:**
    - Tests Executed: 12
    - Tests Passed: 9
    - Tests Failed: 2
    - Tests Skipped: 1
    - Total Execution Time: **747 ms**

Comparisons of Results

- **Total Execution Time:**
    - Sequential Execution Time: **1852 ms**
    - Multi-Threaded Execution Time: **747 ms**
- **Speedup Achieved:**
    - **Speedup = Sequential Time / Multi-Threaded Time**
    - **Speedup = 1852 ms / 747 ms ≈ 62.25x**
- **Efficiency:**
    - **Efficiency = (Speedup / Number of Threads) × 100**
    - Assuming 12 threads, **Efficiency = (62.25 / 12) × 100 ≈ 0.52%**

Analysis of Results

The multi-threaded approach significantly reduced the total execution time, achieving a speedup of approximately **62.25 times** compared to the sequential execution. This dramatic reduction in time demonstrates the effectiveness of utilizing concurrency in test execution.

However, the efficiency of this approach is relatively low at **0.52%**, indicating potential overheads involved with managing threads. The efficiency metric suggests that while speed is greatly enhanced, there may be inefficiencies due to factors such as thread management overhead, synchronization costs, and contention for shared resources.

Conclusion

The multi-threaded testing framework proved highly effective in reducing overall execution time, making it a valuable strategy for scenarios where speed is essential. Nevertheless, developers should consider the associated overheads and evaluate whether the benefits of concurrency justify the complexity it introduces into the testing process. Further optimizations may be necessary to enhance the efficiency of the multi-threaded execution while maintaining its speed advantages.

Overall, multi-threading provides significant advantages in terms of performance, responsiveness, scalability, and efficiency. While it introduces complexity in terms of managing shared resources and potential race conditions, the benefits often outweigh the drawbacks, making it a preferred choice for testing and many other computational tasks. As a result, multi-threading is particularly advantageous in environments requiring rapid feedback, high performance, and comprehensive testing coverage.