CSC 345-01 Project #2: Multithreaded Programming

Name: Chris Toala & Eric Rivera
Section: 01 w/ Dr. Yoon

1. Introduction

This project explores multithreaded programming by implementing a Sudoku validator using three different approaches: single-threaded, multi-threaded, and multi-process.

2. Implementation Details

The single-threaded approach, Option 1, processes the Sudoku board sequentially. It reads the board from an input file and verifies all rows, columns, and 3x3 subgrids one by one using a loop. This method has minimal overhead and is straightforward to implement.

The multi-threaded approach, Option 2, utilizes POSIX threads (pthreads) to divide the validation process into multiple concurrent tasks. We implemented one using 27 threads (where each thread validates a single row, column, or subgrid). Each thread performs its assigned validation task and stores the result in a shared results array.

The multi-process approach, Option 3, uses fork() to create three separate child processes, each responsible for validating rows, columns, and subgrids, respectively. Instead of sharing memory through threads, we use shm_open to allocate shared memory where child processes write their results. The parent process waits for all child processes to complete using waitpid, then determines the final Sudoku validity based on their outputs.

3. Experimental Results

To compare the performance of these approaches, we conducted 50 independent trials for each method. We note that each trial was conducted using the same Sudoku puzzle. The average execution times were as follows for a particular trial:

- Option 1 (Single-Threaded): 0.00000260 seconds
- Option 2 (Multi-Threaded): 0.00100382 seconds
- Option 3 (Multi-Process): 0.00015852 seconds

Although multi-threading is typically expected to improve performance, our results indicate that for a small 9x9 Sudoku grid, the overhead of creating and managing multiple threads outweighs the benefits of parallel execution. The multi-process approach, which avoids thread

synchronization but incurs process creation overhead, also does not significantly outperform the single-threaded method. This suggests that a simple sequential approach can be more efficient for small-scale problems like this.

4. Statistical Analysis

To determine whether the observed differences in execution time are statistically significant, we will perform a hypothesis test using a t-test. Our null hypothesis (H_0) states that "there is no statistically significant difference in mean runtime between any two given methods." We use the tolerance level of $\alpha = 0.05$. Our alternative hypothesis is that "there is no statistically significant difference in mean runtime between any two given methods."

For the comparison of options 1 and 2, we found a t of

Paired: P-value less than 0.0001 (we reject the null)

^note unpaired is also 0.0001

For options 2 and 3 comparison, we found a t of

Paired: P-value less than 0.0001 (we reject the null)

^note unpaired is also 0.0001

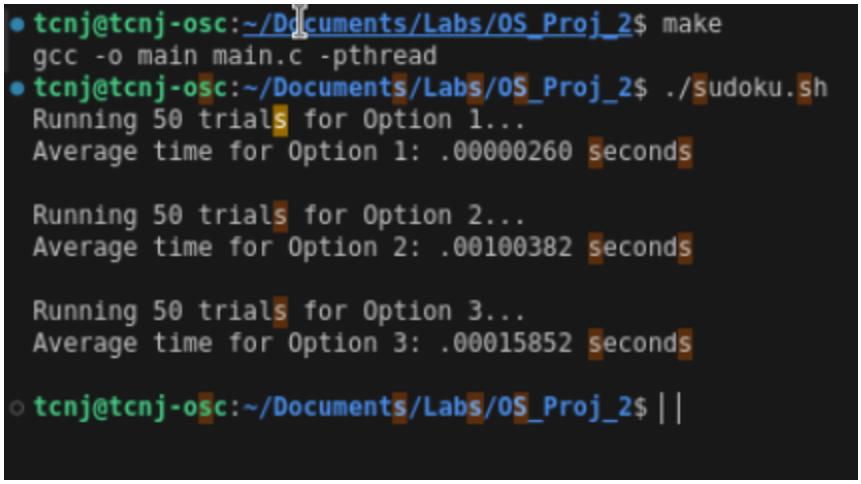
In both comparisons, we reject the null hypothesis.

5. Conclusion

In this project, we implemented and compared three different Sudoku validation methods: single-threaded, multi-threaded, and multi-process. Our results show that for a small 9x9 Sudoku puzzle, the overhead of thread creation and management, such as context switching, outweighs the benefits of parallel execution. The single-threaded approach remains the most efficient.

An accompanying statistical analysis is used to reinforce these claims.

Results obtained:



Trial 3:

0.000143
0.000154
0.000143
0.000140
0.000174
0.000143
0.000143
0.000146
0.000152
0.000144
0.000139
0.000141
0.000168
0.000160
0.000153
0.000147
0.000138
0.000147
0.000184
0.000172
0.000156
0.000161
0.000177
0.000144
0.000172

0.000187
0.000149
0.000160
0.000180
0.000237
0.000226
0.000160
0.000204
0.000147
0.000148
0.000199
0.000136
0.000126
0.000135
0.000140
0.000137
0.000183
0.000140
0.000147
0.000146
0.000188
0.000138
0.000181
0.000145
0.000146

Trial 2:
0.002368
0.000721
0.000924
0.000771
0.000815
0.001021
0.001126
0.000917
0.000932
0.001077
0.001247
0.000793

0.000924
0.001237
0.000967
0.001168
0.000838
0.000863
0.000970
0.001200
0.000858
0.001241
0.000803
0.000789
0.001131
0.000743
0.000865
0.000970
0.001053
0.000864
0.000843
0.000988
0.001135
0.000897
0.001169
0.001061
0.001152
0.001287
0.001161
0.000674
0.000909
0.000663
0.001010
0.000716
0.000932
0.001114
0.001218
0.001163
0.000975
0.000928

Trial 1:
0.000003
0.000002
0.000002
0.000003
0.000002
0.000003
0.000002
0.000002
0.000003
0.000003
0.000002
0.000002
0.000003
0.000002
0.000003
0.000003
0.000002
0.000002
0.000003
0.000002
0.000003
0.000003
0.000003
0.000003
0.000002
0.000002
0.000003
0.000004
0.000003
0.000002
0.000003
0.000002
0.000002
0.000003
0.000003
0.000003
0.000002
0.000003
0.000003

0.000003
0.000003
0.000002
0.000003
0.000003
0.000002
0.000002
0.000003
0.000003
0.000003
0.000002