Lab Report #2: MULTITHREADED PROGRAMING

Sarem Shalforoosh

Linh Ngo

CSC 345-01: Operating Systems

The College of New Jersey

Date of Submission: 3/19/2021

Table of Contents

Implementation

Our program takes a sudoku board from a text file named *input.txt,* which includes 81 numbers listed in 9 rows, and offers three modes to verify if the arrangement of numbers is a sudoku solution. The user would execute the program in the command line as so:

```
$ ./main mode
```

Where mode would equal 1, 2, or 3. Depending on the input mode, the program will either run the validator in a single process, multiple threads, or multiple processes. The program includes 3 base functions that checks either a row, column or square of the board. The program will print out the board as shown in the input file, if the board is a valid sudoku solution, and the average run time of the method of execution.

A. Mode 1: Running Sudoku Solution Validator with One Process

In mode 1, one process is used to validate the board. In this process, three functions are called (check_square, check_row, and check_column), and if the solution for the parameter is valid the function returns a 1, else it returns a 0. The function check_square is called 9 times, and each time it checks a different square of the sudoku board. The function check_row checks all 9 rows of the board, and the function check_column checks all 9 columns of the board. If all parameters are valid, the entire board is a solution, else the board is not a solution.

B. Mode 2: Running Sudoku Solution Validator with Multiple Threads

In mode 2, 11 different threads are created and used to validate the board. One thread checks if the rows are valid, one thread checks if the columns are valid, and one thread is created for each of the nine squares with each thread checking if their respective square is valid. A runner function is invoked in the thread calls. Depending on which parameter is being checked (row, column, or square), the runner function will run the appropriate functions (check_square, check_row, and check_column) to validate each parameter. If all parameters are valid, the entire board is a solution, else the board is not a solution.

C. Mode 3: Running Sudoku Solution Validator with Multiple Processes

In mode 3, the program creates 11 different child processes to validate the board. One child process checks if the rows are valid, one child process checks if the columns are valid, and one child process is created for each of the nine squares with each child process checking if their respective square is valid. A shared memory was created between the processes in order to track the results of each process validation. The first 9 child processes used the function check_square to validate the squares, the 10th child process used check_row to validate the rows, and the 11th child process used check_column to validate the columns. Afterwards, the parent process collects the result of each child process to see if the entire board is a solution or not.

Statistical Experiment I: Comparing Mode 1 with Mode 2

Null Hypothesis: There is no statistically significant difference between the two methods.

We ran 50 independent runs of Mode 1 and Mode 2 and compared the average run times of each mode.

A. Mode 1 Average Run Time: 0.00008732

B. Mode 2 Average Run Time: 0.00105132

From our experiment, we can reject the null hypothesis. Mode 1 runs about 12 times as fast as Mode 2. We conclude that running functions on one process takes less time than running a process concurrently on multiple threads. We assume that the run time for multithreading is so much longer because of the operations to create the threads and context switching from one thread to the next. In the single process, the CPU does not have to spend time context switching from each thread, and with 11 different threads the time for context switching is quite substantial.

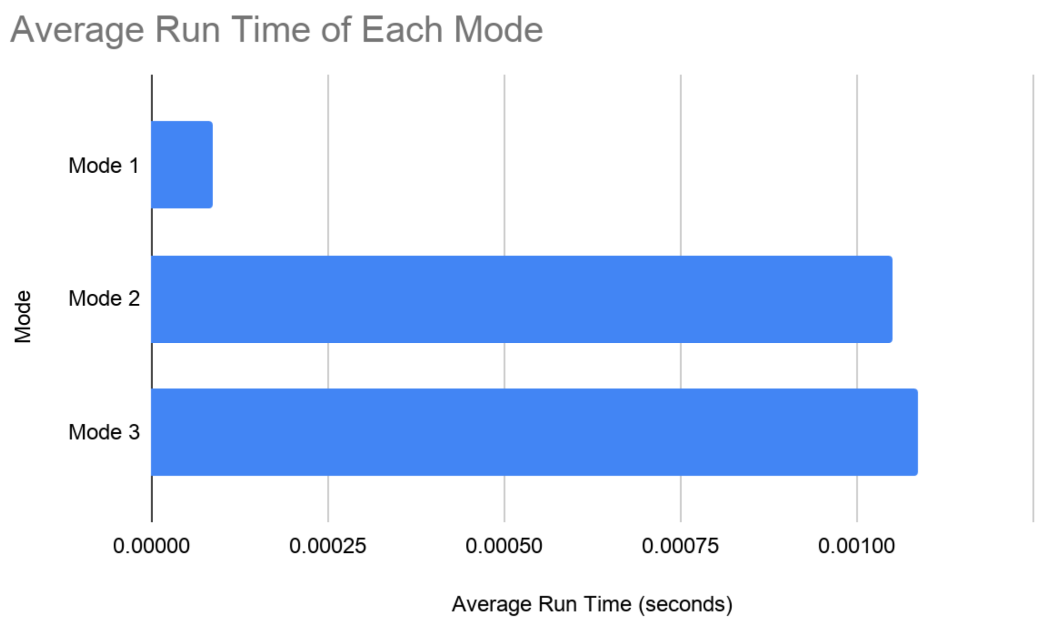

Figure 1: Average Run Time of Each Mode

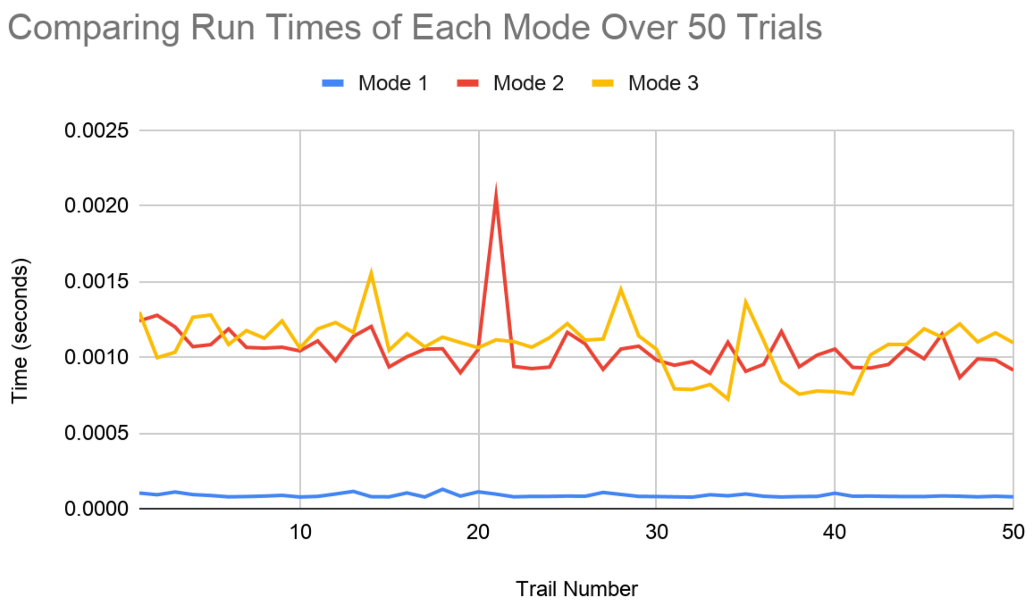

Figure 2: Run Times of Each Mode Over 50 Trials

Statistical Experiment II: Comparing Mode 1 with Mode 3

Null Hypothesis: There is no statistically significant difference between the two methods.

We ran 50 independent runs of Mode 1 and Mode 3 and compared the average run times of each mode.

A.	Mode 1	Average Run Time: 0.00008732
B.	Mode 3	Average Run Time: 0.00108686

From our experiment, we can reject the null hypothesis. Mode 1 runs about 12 times as fast as Mode 3. We conclude that running functions on one process takes less time than running multiple child processes concurrently. We believe that it takes more time, because the child processes have to communicate between each other and write and read from a shared memory, which increases run time. In addition, the forking of processes also increases the run time of the program because it takes up resources and delays the process from starting.