Michael Ralea & Jordan Sinoway
CSC345-02
March 19, 2021

Project 2: Multithreaded Programming

Implemented Requirements

- Sudoku Validation for Threads and Processes
    - The program correctly analyzes a sudoku board for options 1, 2, and 3. Option 1 uses 11 threads (9 for checking the 3x3 grids, 1 for all rows, and 1 for all columns) while Option 2 uses 27 threads (9 for the 3x3 grids, 9 for the rows, and 9 for the columns). Option 3 uses 11 processes as an alternative method to Option 1 (9 processes for checking 3x3 grids, 1 for rows, 1 for columns).The program also tracks the time elapsed during the computational period of the program and returns it to the user. Timing begins after the text file is read and input arguments are processed; timing only tracks the part of the program that identifies the option the user chooses, creates the corresponding threads, and joins all threads. Each function automatically exits if the global variable “invalid” is true. This requires the use of 8 total functions:
        - areRowsValid
            - Takes in no data and simply iterates through the sudoku table stored as a global variable. Stores each row in a temporary array and checks for repeat numbers.
        - areRowsValid_process
            - Acts in the same manner as as the areRowsValid function, except for using process based exits instead of thread based.
        - isRowValid
            - Takes in row and column index, column index should always be 0. Checks that one row.
        - areColsValid
            - Takes in no data and simply iterates through the sudoku table stored as a global variable. Stores each column in a temporary array and checks for repeat numbers.
        - areColsValid_process
            - Acts in the same manner as as the areColsValid function, except for using process based exits instead of thread based.
        - isColValid
            - Takes in row and column index, row index should always be 0. Checks that one column.
        - is3x3Valid
            - Takes in row and column index when the mod of those indexes by 3 equals 0. Check for duplicate elements.

- is3x3Valid_process
  - Acts in the same manner as as the areRowsValid function, except for using process based exits instead of thread based.

Option 1 vs Option 2 vs Option 3 (*11 vs 27 Threads vs 11 Processes*)
Number of individual runs: 30 per option

Valid Sudoku Solution

	Average Runtime (s)
Option 1 (11 Threads)	0.00094
Option 2 (27 Threads)	0.00151
Option 3 (11 Processes)	0.00175

Invalid Sudoku Solution

	Average Runtime (s)
Option 1 (11 Threads)	0.00062
Option 2 (27 Threads)	0.00144
Option 3 (11 Processes)	0.00118

The tables above show the average runtimes for each option in our program, which were calculated across 30 runs per option. We completed tests for both a valid given solution and an invalid solution, to see if there were major differences in the program's completion time. All the values from the runs can be seen in the table below:

Sudoku Board is Valid					Sudoku Board is Invalid			
Run #	Option 1	Option 2	Option 3		Run #	Option 1	Option 2	Option 3
1	0.00138	0.002015	0.002436		1	0.001208	0.001946	0.001565
2	0.001603	0.002128	0.001912		2	0.000578	0.0015	0.001383
3	0.001577	0.001586	0.001716		3	0.000658	0.001521	0.001071
4	0.00115	0.001773	0.001562		4	0.000636	0.001718	0.001393
5	0.001488	0.001566	0.001992		5	0.000583	0.002011	0.001079
6	0.001092	0.001582	0.001579		6	0.000704	0.001749	0.001126
7	0.000987	0.001662	0.002432		7	0.000653	0.001936	0.000969

8	0.001383	0.001799	0.002229		8	0.000662	0.001482	0.001215
9	0.001255	0.001315	0.001232		9	0.000579	0.0013	0.001229
10	0.001207	0.001479	0.001106		10	0.000541	0.001271	0.001346
11	0.001915	0.001416	0.001129		11	0.000614	0.002099	0.001261
12	0.001376	0.001618	0.001299		12	0.000608	0.00154	0.001064
13	0.000581	0.001416	0.001388		13	0.000656	0.001447	0.001128
14	0.000712	0.001388	0.001315		14	0.000603	0.001313	0.000954
15	0.000736	0.001246	0.001085		15	0.000579	0.001372	0.001017
16	0.000649	0.001396	0.001535		16	0.000541	0.00142	0.001557
17	0.000774	0.0015	0.001679		17	0.000614	0.001459	0.001221
18	0.000686	0.001728	0.001642		18	0.000608	0.001242	0.000957
19	0.000663	0.001609	0.001635		19	0.000603	0.001151	0.001019
20	0.00066	0.001403	0.001538		20	0.000718	0.001235	0.000972
21	0.000624	0.001522	0.001433		21	0.000674	0.001263	0.001095
22	0.000635	0.001514	0.001198		22	0.000563	0.001263	0.001066
23	0.000692	0.001396	0.001225		23	0.000485	0.001241	0.001126
24	0.00058	0.001242	0.001429		24	0.000567	0.001315	0.000911
25	0.000616	0.001291	0.001112		25	0.000582	0.001208	0.001021
26	0.0006	0.001338	0.001337		26	0.000674	0.001233	0.001458
27	0.000641	0.001264	0.001666		27	0.000563	0.001186	0.001053
28	0.000687	0.001254	0.001206		28	0.000485	0.00118	0.001412
29	0.000685	0.001425	0.002288		29	0.000567	0.001158	0.001645
30	0.000687	0.001331	0.001316		30	0.000582	0.001381	0.001187
Avg Time	0.000944	0.001507	0.001748		Avg Time	0.000623	0.001438	0.001183

These results are also summarized on the last page in the form of graphs. The results from the averages and tables are fairly clear. They show that for processing a valid or invalid sudoku solution, 11 threads is the quickest and most efficient method of checking. The option utilizing 11 threads was shown to be 60.64% faster on average when compared to option 2 (27 threads), and 86.17% faster on average than option 3 (11 processes). Even though 27 threads could solve a more complex problem faster than 11 threads, for the complexity of this problem, the overhead necessary for 27 threads ends up costing more time than it takes for 11 threads.
The results also show that the threads are more efficient for this task than processes, as both option 1 (11 threads) option 2 (27 threads) outperformed option 3 with its processes in most cases, except if the solution was invalid, where option 3 beat option 2.

Therefore, the null hypothesis “there is no statistically significant difference between the two methods” has been rejected for all options, as there has shown to be a statistically significant difference between all methods in a variety of tests.