Lab Exercise 3: Multithreaded Programming

COMP3499 Operating Systems for Engineers
Electrical and Computer Engineering
Wentworth Institute of Technology

Objectives:

- To create a multithreaded program that checks whether a Sudoku puzzle is valid.

A Sudoku puzzle uses a 9 × 9 grid in which each column and row, as well as each of the nine 3 × 3 subgrids, must contain all of the digits 1 ··· 9. Figure L3-1 presents an example of a **valid** Sudoku puzzle.

6	2	4	5	3	9	1	8	7
5	1	9	7	2	8	6	3	4
8	3	7	6	1	4	2	9	5
1	4	3	8	6	5	7	2	9
9	5	8	2	4	7	3	6	1
7	6	2	3	9	1	4	5	8
3	7	1	9	5	6	8	4	2
4	9	6	1	8	2	5	7	3
2	8	5	4	7	3	9	1	6

Figure L3-1

In this lab exercise, you will design a multithreaded application that determines whether the solution to a Sudoku puzzle is valid.

There are several different ways of multithreading this application. One suggested strategy is to create threads that check the following criteria:

- A thread to check that each column contains the digits 1 through 9
- A thread to check that each row contains the digits 1 through 9
- Nine threads to check that each of the 3 × 3 subgrids contains the digits 1 through 9

This would result in a total of eleven separate threads for validating a Sudoku puzzle. However, you are welcome to create even more threads for this project.

The parent thread will create the worker threads, passing each worker the location that it must check in the Sudoku grid. This step will require passing several parameters to each thread. The easiest approach is to create a data structure using a struct. For example, a structure to pass the row and column where a thread must begin validating would appear as follows:

```
//structure for passing data to threads
typedef struct
{
    int row;
    int column;
} parameters;
```

Then create worker threads using a strategy similar to that shown below:

```
parameters *data = (parameters *) malloc(sizeof(parameters));
data->row = 1;
data->column = 1;
//this creates a struct of type parameters called data
//passing in values row = 1 and column = 1 (start values for checking)
```

The data pointer will be passed to `pthread create()`, which in turn will pass it as a parameter to the function that is to run as a separate thread. For example:

```
pthread_create(&workers[0], 0, row_worker, data);
```

creates a thread, calls a `row_worker` function, passing in the `data` struct.

Each worker thread is assigned the task of determining the validity of a particular region of the Sudoku puzzle. Once a worker has performed this check, it must pass its results back to the parent. One good way to handle this is to create an array of integer values that is visible to each thread. The i^{th} index in this array corresponds to the i^{th} worker thread. If a worker sets its corresponding value to `1`, it is indicating that its region of the Sudoku puzzle is valid. A value of `0` indicates otherwise. When all worker threads have completed, the parent thread checks each entry in the result array to determine if the Sudoku puzzle is valid.

Run the code for a valid sudoku puzzle, take a screenshot showing that it detects that it is valid.

```
int puzzle[9][9] = {
                      {5,3,4,6,7,8,9,1,2},
                      {6,7,2,1,9,5,3,4,8},
                      {1,9,8,3,4,2,5,6,7},
                      {8,5,9,7,6,1,4,2,3},
                      {4,2,6,8,5,3,7,9,1},
                      {7,1,3,9,2,4,8,5,6},
                      {9,6,1,5,3,7,2,8,4},
                      {2,8,7,4,1,9,6,3,5},
                      {3,4,5,2,8,6,1,7,9}
                   };
```

Then change the above to show that you can detect an invalid puzzle.

What to submit:

- A PDF on Brightspace with the names of the team members and screenshots requested above. The screenshots should each have appropriate figure numbers and captions.
- .c files requested above. Do not zip the files together.