SOFE 3950: Operating Systems

Lab 3

Deliverables

1. Complete all the /*WRITE YOUR OWN CODE*/ parts in the given source files. Search for these keywords, in case you miss some parts.
2. Write your own Makefile. It is required that your program should be able to compiled by using “make” command.
3. Make sure you code can be successfully compiled. This is a must. Your program may be missing some of the functionalities, but it needs to be compilable. I will unzip the files and simply type “make” to compile your code.
4. Submit the source code and the Makefile in one zip file.
5. The given code is only for your reference, you can also write your own program from scratch.

Sudoku Solution Validator

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. The Figure presents an example of a valid Sudoku puzzle.

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

This lab consists of designing 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 lab. For example, rather than creating one thread that checks all nine columns, you could create nine separate threads and have each of them check one column.

Passing Parameters to Each Thread

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;
```

Pthreads programs will create worker threads using a strategy similar to that shown below:

```
parameters *data = (parameters *) malloc(sizeof(parameters));
data->row = 1;
data->column = 1;
/* Now create the thread passing it data as a parameter */
```

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

Returning Results to the Parent Thread

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 would indicate 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.