SOFE 3950U / CSCI 3020U:
Operating Systems

Lab #3: Sudoku Solution Validator

Objectives

- Gain experience doing multithreaded programming in C
- Experience with locking and synchronization concepts

Important Notes

- Work in groups of **four** students
- All reports must be submitted as a PDF on blackboard, if source code is included submit everything as an archive (e.g. zip, tar.gz)
- Save the submission as <lab_number>_<first student's id> (e.g. lab3_100123456.pdf)
- If you cannot submit the document on blackboard then please contact the TA with your submission at patrick.smuk@uoit.net **or Pat Smuk on Slack**

Lab Details

Notice

It is recommended for this lab activity and others that you save/bookmark the following resources as they are very useful for C programming.

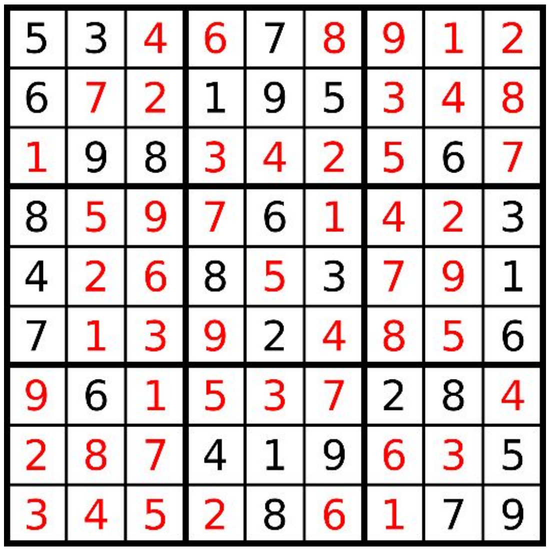
- http://en.cppreference.com/w/c
- http://www.cplusplus.com/reference/clibrary/
- http://users.ece.utexas.edu/~adnan/c-refcard.pdf
- http://gribblelab.org/CBootcamp

The following resources are helpful as you will need to use pthreads in order to make your program multithreaded.

- https://computing.llnl.gov/tutorials/pthreads/
- http://randu.org/tutorials/threads/
- http://pages.cs.wisc.edu/~travitch/pthreads_primer.html

Lab Activity

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

You can pass arbitrary data to a thread by using the **void pointer** (`void *`) argument to **`pthread_create`**. This pointer can point to any data and is passed along to the thread function as the first parameter.

For example, if you want to pass an integer to a thread, you can do something like:

```
int *x = malloc(sizeof(int));
*x = 1;
pthread_t thread;
if (pthread_create(&thread, NULL, my_thread_main, x) != 0) {
    // Handle pthread_create error...
}
```

Your thread would then be able to read the value and free the memory.

```
void *my_thread_main(void *x) {
    int value = *((int*)x);
    free(x); x = NULL;
    // Do something with value.
}
```

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.

You can alternatively make use of the void pointer returned by your thread to return the result. This pointer is given to your main thread via the second parameter of `pthread_join`.

Reading the Puzzle

Your Sudoku puzzle solver must read in a file called **puzzle.txt** which contains the Sudoku puzzle to validate. Entries that are 0 are empty squares and should be ignored.

Your Sudoku validator must print either “valid” or “invalid” for a given puzzle. If any digit appears in the same row, column, or subgrid twice, your program must print “invalid”. Otherwise it must print “valid”.

An example of a valid grid is as follows:

```
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

Deliverables

Notice

Please complete the deliverables and include whatever screenshots and other work is necessary to demonstrate that you have completed the deliverables in your lab submission on Blackboard. All lab report submissions are due on Blackboard prior to the start of the next lab.

1. All source files for your implementation, include a makefile so that the code can be compiled. The solution **must** make use of threading in order to receive marks.

2. Your program **MUST** read an input file **puzzle.txt** containing the Sudoku puzzle to solve as described above and print “valid” if the puzzle is valid or “invalid” if it is not.