# Sudoku Solution Validator (SSV) Report

**Full Name:** Minn Thit Kyaw
**Student ID:** 21456464

## About

This program is written in C, following the standards of C89 syntax. The program is intended to validate the Sudoku grid in the following ways: each row is valid, each column is valid, each subgrid (3x3) is valid. For these checks, we will utilize mulit-threading in C with the library named `pthread`. We will also design our program to efficiently handle inter-thread communications, critical sections and unsignaling issues. In this program, we will create one main(parent) thread and four children threads.

---

## Handbook

### How to run the program

I have created a `Makefile` that will compile the source code `mssv.c` into an executable `mssv`. This `Makefile` will compile the program using C89 by strictly adding `--std=c89` compilation flag.

Make sure your device already has `make` cli installed.

```
make
```

This will output an executable named `mssv`. You can run this program by adding two command line arguments: Sudoku Grid file( `solution` ) and the delay in seconds ( `1-9` )

```
./mssv solution 1
```

To clean all the executables produced by `Makefile`, run this command

```
make clean
```

# Method of Procedures

## How synchronization is achieved

In my MSSV code, synchronization is achieved using pthread mutexes (`pthread_mutex_t`) and condition variables (`pthread_cond_t`). Synchronization is crucial in multi-threaded programming to ensure that multiple threads can safely access and update shared resources without causing data inconsistencies or race conditions.

### Synchronization Mechanisms

1. **Mutex Initialization**

   ```
   pthread_mutex_t lock;
   pthread_mutex_init(&lock, NULL);
   ```

   In this snippet, `lock` is initialized as a mutex. This mutex will be used to ensure that only one thread can access critical sections of code that manipulate shared variables(`ROWS, COLS,..`) at the same time.
2. **Condition Variable Initialization**

   ```
   pthread_cond_t cond;
   pthread_cond_init(&cond, NULL);
   ```

   `cond` is initialized as a condition variable. These variables in my code are used for thread synchronization where one thread signals another thread about the occurrence of a particular condition.
3. **Mutex Locking and Unlocking**

```
pthread_mutex_lock(&lock);
pthread_mutex_unlock(&lock);
```

- `pthread_mutex_lock(&lock)` will create a mutex lock before accessing shared resources. This will ensure only a specific thread can access the shared resouces
- `pthread_mutex_unlock(&lock)` this will release the mutex lock after using shared resources.

### Threads and Shared Resources

1. **Thread Function for Validating Sub-grids and Columns**

```
   void* validateSubgridsAndColumns(void* param) {
       int threadId = *(int*)param;

       for (i = 0; i < 3; i++) {
           pthread_mutex_lock(&lock);
           Sub[startSubgrid + i] = validSub;
           if (validSub) {
               Counter++;
           } else {
               printf("Thread ID-%d: Sub-grid %d is invalid\n",
threadId, startSubgrid + i + 1);
           }
           pthread_mutex_unlock(&lock);
       }
```

- **Accessed Shared Resources:** `Sub[]` (for sub-grid validation) and `Counter`.
- **Mutex Usage:** Used to lock before updating `Sub[]` and `Counter`.

2. **Thread Function for Validating Rows:**

```
        void* validateRows(void* param) {
       int threadId = *(int*)param;

       for (i = 0; i < SIZE; i++) {3
           pthread_mutex_lock(&lock);
           Row[i] = validRow;      /* Update row validation status */
           if (validRow) {
               Counter++;    /* Increment counter if row is valid */
           } else {
               printf("Thread ID-%d: Row %d is invalid\n", threadId, i
+ 1);
           }
           pthread_mutex_unlock(&lock);
       }

   }
```

- **Accessed Shared Resource:** `Row[]` (for row validation) and `Counter`.
- **Mutex Usage:** Used to lock before updating `Row[]` and `Counter`.

In simple words:

- **Mutex (`lock`):** Ensures exclusive access to shared resources (`Row[]`, `Col[]`, `Sub[]`, `Counter`) to prevent data races.
- **Condition Variable (`cond`):** Used to signal waiting threads when a thread completes its task (`completed_threads`, `last_thread_id`).

This will ensure that each thread can safely update shared variables without interference from other threads and maintaining data integrity and synchronization in the multithreaded environments.

---

## Functions not working

I don't have such function in my final code `mssv.c`. One issue I faced was when I first wrote the code. I didn't use condition signaling. This caused a lot of issues for me. I can execute the program one or two time but after that, the program run time is inconsistent. Sometimes, the program ran perfectly but sometimes there is a similar thing happening as infinite loops. There was no infinite loop but the parent thread doesn't get signal from child threads after they finished executing. To solve this, I used something called `pthread_cond()` conditional signaling and then joining all threads after they finished executing with `pthread_join()`. These fixes ensure that all threads in my program coordinate properly and that the main program can safely proceed based on completed tasks.

---

## Sample I/O

I never played Sudoku but read some docs to understand the grid structure.

### Valid Input

This is the valid Sudoku grid as provided in the specifications sheet

```
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
```

- Each row and column must contain 1-9 only oncev (no replication)

We will run with a second of delay for this.

```
minn@MyWindows:/mnt/d/CUD_UG-3/OperatingSystems/Assignment$ ./mssv solution 1
Thread ID-1 is validating sub-grids and columns
Thread ID-2 is validating sub-grids and columns
Thread ID-3 is validating sub-grids and columns
Thread ID-4 is validating rows
Thread ID-2 is valid
Thread ID-3 is valid
Thread ID-1 is valid
Thread ID-4 is valid
Validation Results:
Valid threads:
Thread ID-1 is valid
Thread ID-2 is valid
Thread ID-3 is valid
Thread ID-4 is valid
```

## Invalid Input

The grid will be invalid if there are: duplicates, not 9x9 grid size, etc.

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

- In this grid, there are two 6s in row 1 and two 7s in row 6.

Running with this grid will show it's invalid.

```
minn@MyWindows:/mnt/d/CUD_UG-3/OperatingSystems/Assignment$ ./mssv invalid-solution 1
Thread ID-1 is validating sub-grids and columns
Thread ID-2 is validating sub-grids and columns
Thread ID-3 is validating sub-grids and columns
Thread ID-4 is validating rows
Thread ID-4: Row 1 is invalid
Thread ID-1: Sub-grid 3 is invalid
Thread ID-2: Sub-grid 6 is invalid
Thread ID-3: Column 9 is invalid
Thread ID-4: Row 6 is invalid
Validation Results:
Row 1 is invalid
Sub-grid 3 is invalid
Row 6 is invalid
Sub-grid 6 is invalid
Column 9 is invalid
Valid threads:
Thread ID-2 is valid
Thread ID-4 is valid
```

## Error handling

- User directly runs the program without cli arguments

```
minn@MyWindows:/mnt/d/CUD_UG-3/OperatingSystems/Assignment$ ./mssv
Usage: ./mssv <solution_file> <delay>
```

- User input file is not a 9x9 grid

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

```
minn@MyWindows:/mnt/d/CUD_UG-3/OperatingSystems/Assignment$ ./mssv invalid-solution2 1
Error reading Sudoku grid from file
The file is invalid. It must be a 9x9 grid.
```

- User delay seconds is greater than 10 (It should be between 1 to 10)

```
minn@MyWindows:/mnt/d/CUD_UG-3/OperatingSystems/Assignment$ ./mssv solution 11
The delay should be between 1 to 9 seconds
```