National University of computer and emerging sciences

SUDOKU SOLUTION VALIDATOR USING PTHREADS LIBRARY

PROJECT REPORT

OPERATING SYSTEMS

K213355 Talha Shahid

K214950 Muhammad Ali Raza

K214539 Muhammad Hamood Siddique

May 2023

Introduction

The Sudoku solution validator project aims to develop a software tool that can validate the correctness of a given Sudoku solution. Sudoku is a popular logic-based number puzzle game that requires players to fill a 9x9 grid with digits in such a way that each row, column, and 3x3 sub grid contains all of the digits from 1 to 9 without any repetition. The proposed solution validator will verify if the given solution adheres to these rules, ensuring its validity. This project will employ programming techniques and algorithms to efficiently validate the solution, which can help improve the overall Sudoku-solving experience for enthusiasts and professionals alike. The solution validator can also serve as a valuable tool for Sudoku puzzle creators to ensure the validity of their puzzles.

Project Specifications

Software Specs:

Main function deals with Threads creations and calling methods/Threads.

ValidateRowThread deals with checking and validating some conditions for rows.

ValidateColThread deals with checking and validating some conditions for columns.

ValidateGridThread deals with validating subgrids of 3*3.

The check_line function deals with checking and validating entries of the sudoku grid like those entries are in the (1-9) range and not equal to (0).

Tools, and Technologies:

Programming Language: C++ language

Tools: Visual Studio Code

Platform: Ubuntu 22.04

Problem Analysis

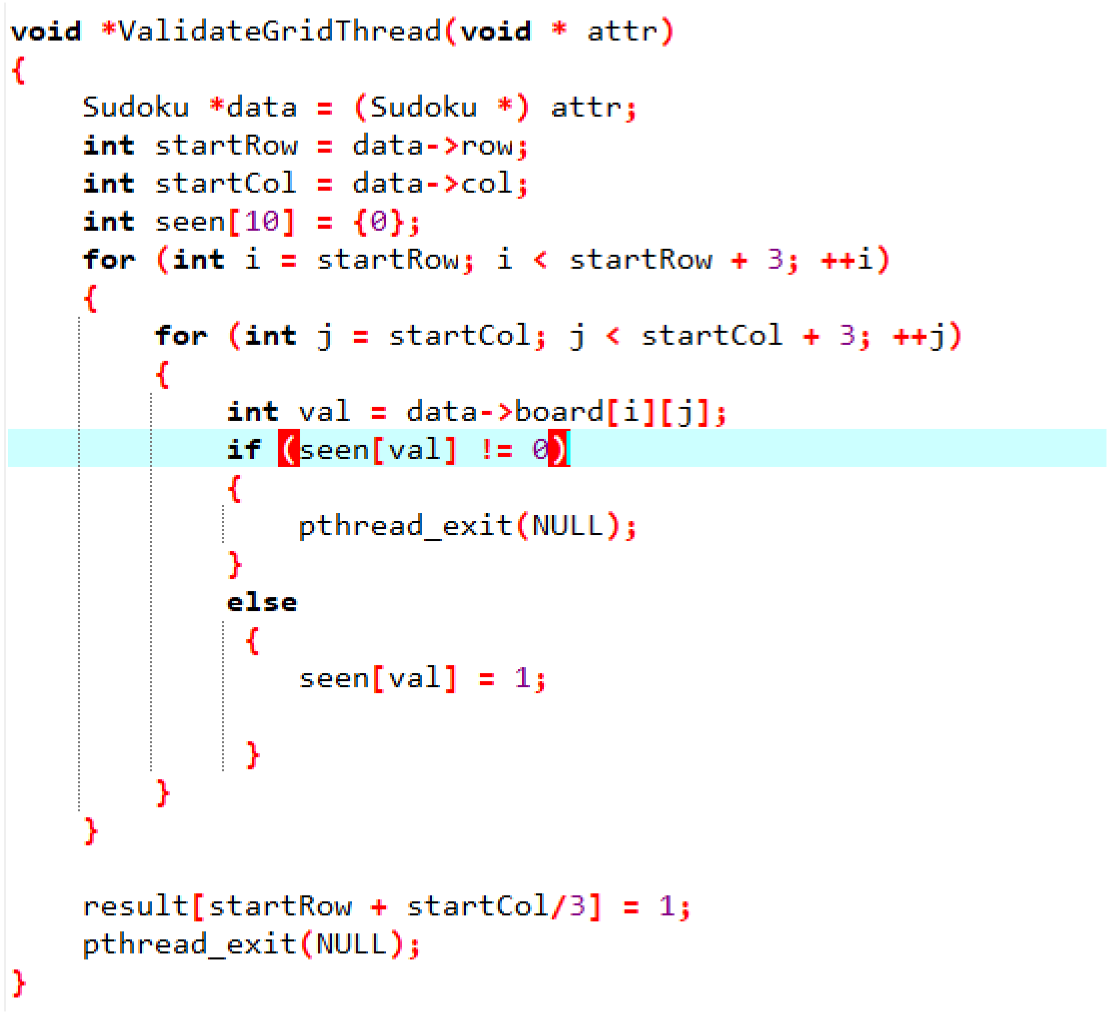
The main problem to solve in a sudoku solution validator project is to determine whether a given solution follows the rules of the game and is valid. This requires analyzing the input solution to check for duplicate numbers within rows, columns, and boxes, as well as ensuring that all spaces are filled with valid numbers.

Solution Design

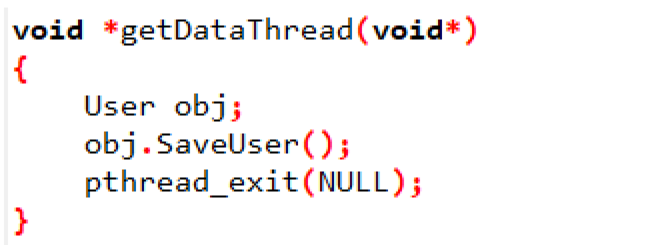
The solution for a sudoku solution validator project involves creating an algorithm that checks the validity of the inputted solution. This algorithm will consist of multithreading to store and compare numbers, as well as loops to iterate through the rows, columns, and boxes of the sudoku grid to ensure that each number is unique and valid. We have also created an algorithm using a single thread.

Snippets

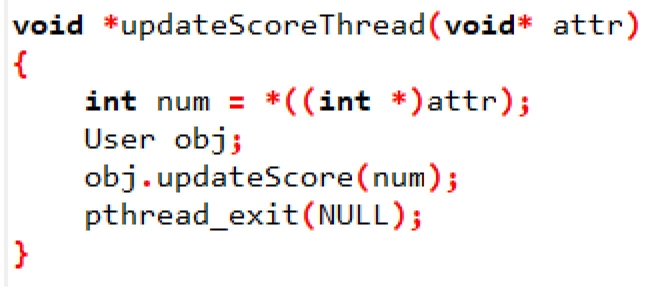
Here below are attached some highlights of our code:



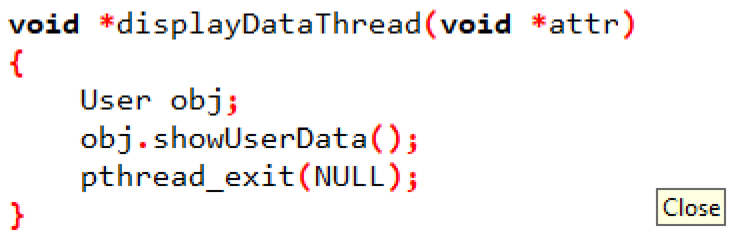
```
void *ValidateGridThread(void * attr)
{
    Sudoku *data = (Sudoku *) attr;
    int startRow = data->row;
    int startCol = data->col;
    int seen[10] = {0};
    for (int i = startRow; i < startRow + 3; ++i)
    {
        for (int j = startCol; j < startCol + 3; ++j)
        {
            int val = data->board[i][j];
            if (seen[val] != 0)
            {
                pthread_exit(NULL);
            }
            else
             {
                seen[val] = 1;

             }
        }
    }

    result[startRow + startCol/3] = 1;
    pthread_exit(NULL);
}
```



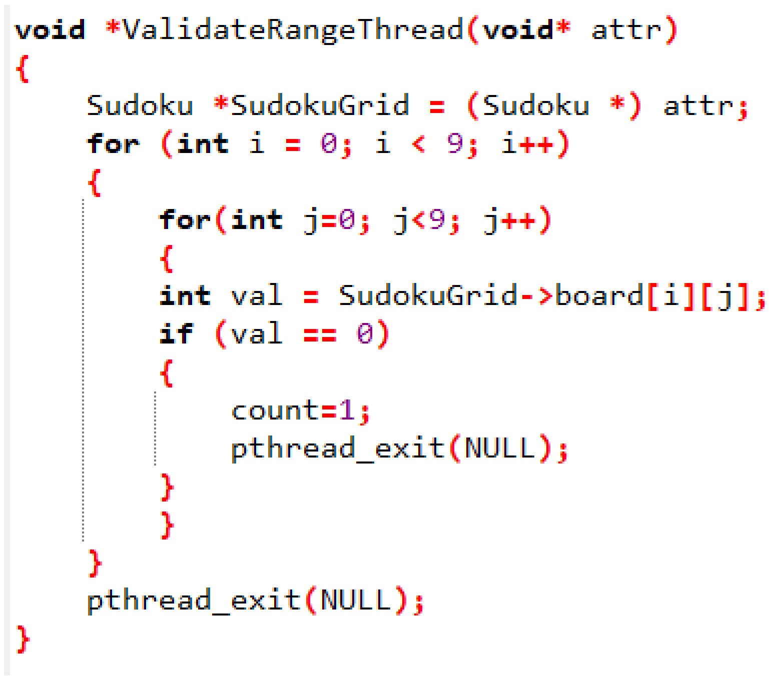
```
void *getDataThread(void*)
{
    User obj;
    obj.SaveUser();
    pthread_exit(NULL);
}
```



```
void *updateScoreThread(void* attr)
{
    int num = *((int *)attr);
    User obj;
    obj.updateScore(num);
    pthread_exit(NULL);
}
```



```
void *displayDataThread(void *attr)
{
    User obj;
    obj.showUserData();
    pthread_exit(NULL);
}
```



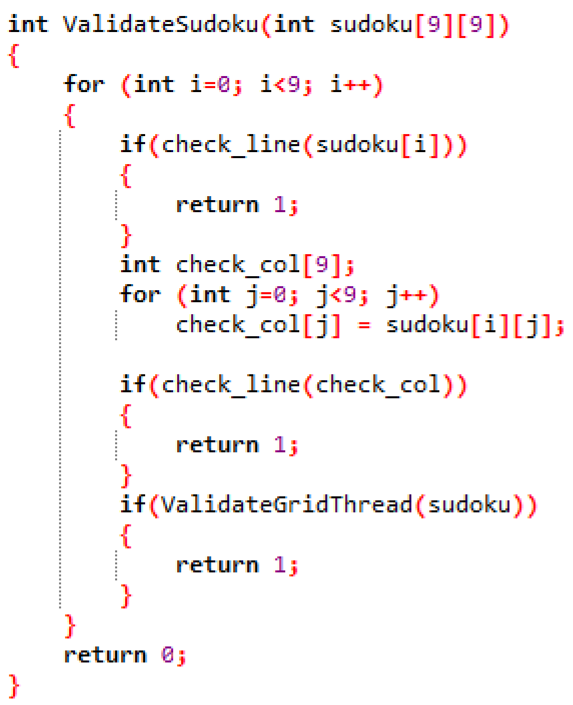
```
void *ValidateRangeThread(void* attr)
{
    Sudoku *SudokuGrid = (Sudoku *) attr;
    for (int i = 0; i < 9; i++)
    {
        for(int j=0; j<9; j++)
        {
        int val = SudokuGrid->board[i][j];
        if (val == 0)
        {
            count=1;
            pthread_exit(NULL);
        }
        }
    }
    pthread_exit(NULL);
}
```

```
void *ValidateColsThread(void *attr)
{
    Sudoku *data = (Sudoku *) attr;
    int col = data->col;

    int seen[10] = {0};
    for (int i = 0; i < 9; i++)
    {
        int val = data->board[i][col];
        if (seen[val] != 0)
        {
            pthread_exit(NULL);
        }
        else
        {
            seen[val] = 1;
        }
    }


    result[18 + col] = 1;
    pthread_exit(NULL);
 }
```

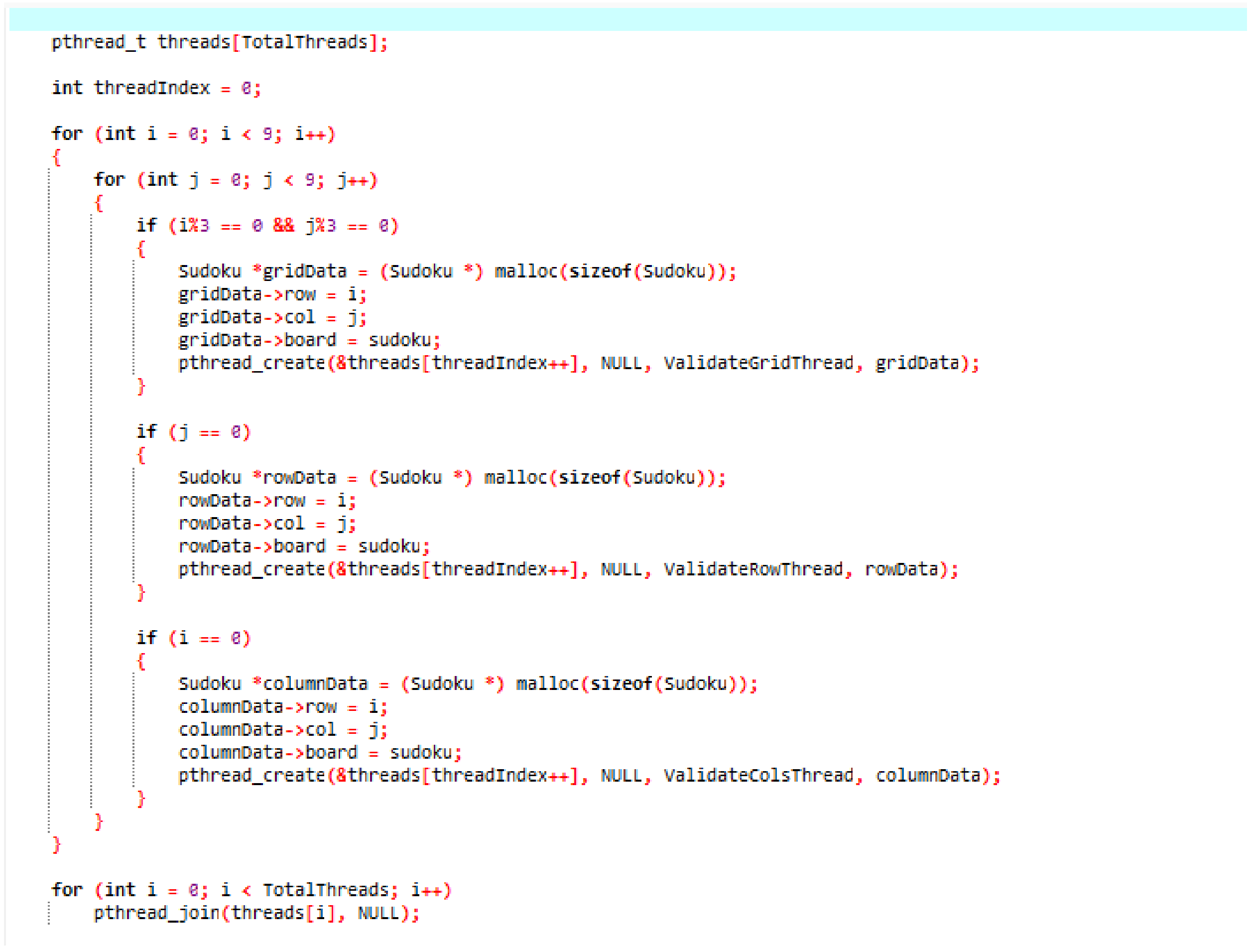
```c
void *ValidateRowThread(void *attr)
{
    Sudoku *data = (Sudoku *) attr;
    int row = data->row;

    int seen[10] = {0};
    for (int j = 0; j < 9; j++)
    {
        int val = data->board[row][j];
        if (seen[val] != 0)
        {
            pthread_exit(NULL);
        }
        else
        {
            seen[val] = 1;
        }
    }


    result[9 + row] = 1;
    pthread_exit(NULL);
}
```

```
int check_line(int input[9])
{
    int seen[10] = {0};
    for (int i = 0; i < 9; i++)
    {
        int val = input[i];
        if (seen[val] != 0)
        {
            return 1;
        }
        else
         {
            seen[val] = 1;
        }
    }
    return 0;
}
```

```
int ValidateGridThread(int sudoku[9][9])
{
    int temp_row, temp_col;

    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            temp_row = 3 * i;
            temp_col = 3 * j;
            int seen[10] = {0};

            for(int p=temp_row; p < temp_row+3; p++)
            {
                for(int q=temp_col; q < temp_col+3; q++)
                {
                    int val = sudoku[p][q];
                    if (seen[val] != 0)
                    {
                        return 1;
                    }
                    else
                     {
                        seen[val] = 1;
                    }
                }
            }
        }
    }
    return 0;
}
```



```c
int ValidateSudoku(int sudoku[9][9])
{
    for (int i=0; i<9; i++)
    {
        if(check_line(sudoku[i]))
        {
            return 1;
        }
        int check_col[9];
        for (int j=0; j<9; j++)
            check_col[j] = sudoku[i][j];

        if(check_line(check_col))
        {
            return 1;
        }
        if(ValidateGridThread(sudoku))
        {
            return 1;
        }
    }
    return 0;
}
```



```c
    pthread_t threads[TotalThreads];

    int threadIndex = 0;

    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            if (i%3 == 0 && j%3 == 0)
            {
                Sudoku *gridData = (Sudoku *) malloc(sizeof(Sudoku));
                gridData->row = i;
                gridData->col = j;
                gridData->board = sudoku;
                pthread_create(&threads[threadIndex++], NULL, ValidateGridThread, gridData);
            }

            if (j == 0)
            {
                Sudoku *rowData = (Sudoku *) malloc(sizeof(Sudoku));
                rowData->row = i;
                rowData->col = j;
                rowData->board = sudoku;
                pthread_create(&threads[threadIndex++], NULL, ValidateRowThread, rowData);
            }

            if (i == 0)
            {
                Sudoku *columnData = (Sudoku *) malloc(sizeof(Sudoku));
                columnData->row = i;
                columnData->col = j;
                columnData->board = sudoku;
                pthread_create(&threads[threadIndex++], NULL, ValidateColsThread, columnData);
            }
        }
    }

    for (int i = 0; i < TotalThreads; i++)
        pthread_join(threads[i], NULL);
```

Result

The end result of team efforts is a sudoku solution validator program using the "PTHREADS" library. This essentially checks and validates all test cases which can be applied on a sudoku solution by rows and columns and on sub grids.

Reference

We got the idea for this project from our textbook (Abraham-Silberschatz-Operating-System-Concepts) (Page No# 262). The sample present in the book Helped us to better understand the problem and work on an efficient solution to the problem.