

**VNR Vignana Jyothi Institute of Engineering and Technology (Affiliated to**

**J.N.T.U, Hyderabad) Bachupally(v), Hyderabad, Telangana, India.**

**Multithreaded Sudoku Validator**

A course project submitted in complete requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

IN

**COMPUTER SCIENCE AND ENGINEERING - ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

Submitted by

B PRIYANKA 22071A6609

B RAMARAJU 22071A6610

O AJAY 22071A6645

G THARUN 22071A6657

Under the guidance of   
**Mr. E. Guru Mohan   
Assistant Professor**

**Dept. of CSE - AIML & IoT**



**VNR Vignana Jyothi Institute of Engineering and Technology (Affiliated to   
J.N.T.U, Hyderabad)**

**Bachupally(v), Hyderabad, Telangana, India.**

**CERTIFICATE**

This is to certify that 22071A6609, 22071A6610, 22071A6645 and 22071A6657 have completed their course project work at Dept. of CSE-AIML & IoT of VNR VJIET, Hyderabad entitled “**MultiThreaded Sudoku Validator"** in complete fulfilment of the requirements for the award of B.Tech degree during the academic year 2024-2025. This work is carried out under my supervision and has not been submitted to any other University/Institute for award of any degree/diploma.

**Mr. E. Guru Mohan Dr. Y. Sagar**

**Assistant Professor Associate Professor and Head**

CSE-AIML & IoT Department CSE-AIML & IoT Department

VNRVJIET VNRVJIET

**DECLARATION**

This is to certify that our project report titled **“MULTITHREADED SUDOKU VALIDATOR”** submitted to Vallurupalli Nageswara Rao Institute of Engineering and Technology in complete fulfilment of requirement for the award of Bachelor of Technology in Computer Science and Engineering - Artificial Intelligence and Machine Learning is a bona fide report to the work carried out by us under the guidance and supervision of Mr. E. Guru Mohan, Assistant Professor, Department of CSE-AIML & IoT, Vallurupalli Nageswara Rao Institute of Engineering and Technology. To the best of our knowledge, this has not been submitted in any form to other universities or institutions for the award of any degree or diploma.

|  |  |  |  |
| --- | --- | --- | --- |
| **B Priyanka**  22071A6609 | **B Ramaraju**  22071A6610 | **O. Ajay**  22071A6645 | **G Tharun**  22071A6657 |

**ACKNOWLEDGEMENT**

Over a span of two years, VNRVJIET has helped us transform ourselves from mere amateurs in the field of Computer Science into skilled engineers capable of handling any given situation in real time. We are highly indebted to the institute for everything that it has given us. We would like to express our gratitude towards the principal of our institute, **Dr. Challa Dhanunjaya Naidu** and the Head of the Dept. of CSE - AIML & IoT, **Dr. Y. Sagar** for their kind co- operation and encouragement which helped us complete the project in the stipulated time. Although we have spent a lot of time and put in a lot of effort into this project, it would not have been possible without the motivating support and help of our project guide **Mr. E. Guru Mohan** We thank him for his guidance, constant supervision and for providing necessary information to complete this project. Our thanks and appreciations also go to all the faculty members, staff members of VNR VJIET, and all our friends who have helped us put this project together.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **SNo** | **Title** | **Page No.** |
| 1. | Abstract | 6 |
| 2. | Introduction | 7 |
| 3. | Existing Solution | 8 |
| 4. | Proposed Solution | 9 |
| 5. | Implementation | 10 - 14 |
| 6. | Result | 15 - 16 |
| 7. | Conclusion | 17 |
| 8. | References | 18 |

**ABSTRACT**

The Sudoku puzzle, renowned for its logic-based challenge, often demands efficient validation algorithms to confirm the puzzle's correctness. In this study, we propose a multithreaded Sudoku validator designed to expedite the validation process through parallelism. The validator divides the Sudoku grid into discrete sections, assigning each section to an individual thread for simultaneous validation. Leveraging parallel processing capabilities, this approach aims to improve efficiency by distributing the workload across multiple threads.The validator incorporates synchronization mechanisms to ensure coherent and error-free validation results. Through meticulous thread management and allocation strategies, it aims to optimize resource utilization while validating the puzzle's rows, columns,and subgrids concurrently. Additionally, scalability remains a crucial aspect, enabling the validator to accommodate varying grid sizes and complexities efficiently.Our investigation focuses on the validator's ability to detect and report inconsistencies within the Sudoku grid, ensuring accurate identification of invalid configurations or duplicates. Furthermore, we evaluate its performance in terms of speed, scalability, and resource management to ascertain its viability in handling diverse Sudoku puzzles.

**INTRODUCTION**

A Multithreaded Sudoku Validator is an application used to validate the solution of a Sudoku puzzle.It is a multithreaded application where concept of multithreading is being used to verify the solution.The application should efficiently check whether the given 9 x 9 grid adheres to the rules of Sudoku,ensuring that each row,column,and 3 x 3 subgrid contains all digits from 1 to 9 without repetition.With every rule related to Sudoku it validates the solution using concept multithreading internally and results the output whether the solution the user has got is matched or not.The goal is to create a robust and parallelized solution that optimizesthe verification process for large Sudoku grids,enhancing the performance of Sudoku puzzle validation.Multithreading is a concept in operating systems which is being used in this application refers to the concurrent execution of multiple threads within a single process.A thread is the smallest unit of execution and consists of a program counter, a register set, and a stack space.Unlike processes,threads within the same process share the same resources, such as memory space and file descriptors.

## Motivation

Validating a Sudoku puzzle in a multithreaded environment can be motivated by the opportunity to expedite the process. Multithreading allows for simultaneous checking of different regions of the puzzle, potentially reducing the overall time required for validation. By breaking down the puzzle into sections checked by different threads, you can leverage parallel processing to enhance efficiency and speed up the validation process, making it more responsive and scalable, especially with larger Sudoku grids.

A multithreaded application for validating the solution to a Sudoku puzzle.The application should efficiently check whether the given 9 x 9 grid adheres to the rules of Sudoku,ensuring that each row, column, and 3 x 3 subgrid contains all digits from 1 to 9 without repetition.The goal is to create a robust and parallelized solution that optimizes the verification process for large Sudoku grids,enhancing the performance of Sudoku puzzle validation.

## Objectives

**EXISTING SOLUTIONS**

A single-threaded Sudoku validator is a program designed to check the validity of a Sudoku puzzle using a single thread of execution. This type of validator processes the grid sequentially, ensuring that each row, column, and subgrid adheres to the rules of Sudoku.

### Key Features

1. **Sequential Processing:**
   * The validator operates in a single thread, processing the grid one element at a time.
   * It checks each row, column, and subgrid in a systematic manner.
2. **Validation Rules:**
   * **Rows:** Each row must contain the numbers 1-9 without repetition.
   * **Columns:** Each column must contain the numbers 1-9 without repetition.
   * **Subgrids:** Each 3x3 subgrid must contain the numbers 1-9 without repetition.

### Benefits

* **Simplicity:**
  + The logic is straightforward and easy to implement.
  + Debugging is easier due to the sequential nature of the process.
* **Predictability:**
  + The execution order is fixed, making the behavior of the validator consistent and predictable.

### Limitations

* + While sufficient for standard 9x9 Sudoku puzzles, this approach might not be the most efficient for larger or more complex puzzles. It processes each part of the grid one after the other, which might be slower compared to parallel processing.

**PROPOSED SOLUTION**

Our proposed solution for a multithreaded Sudoku validator leverages multiple threads to validate Sudoku puzzles efficiently and accurately. The system performs parallel checks on rows, columns, and subgrids, significantly improving performance, especially for large-scale Sudoku puzzles. At the core of our solution is a robust validation algorithm that ensures thread-safe operations and efficient synchronization.

**Advantages of Multithreaded Sudoku Validator:**

**1.Improved Performance:**

* Parallel Execution: Reduces overall validation time by leveraging multi-core processors.
* Efficient Resource Utilization: Thread pooling optimizes the use of resources.

**2.Scalability:**

* Handling Large Puzzles: Can efficiently manage larger puzzles.
* Adaptability: Scales with the number of available CPU cores.

**3.Modularity:**

* Task Management: Divides validation into manageable tasks, making the code modular and maintainable.
* Flexibility: Easy to update or extend specific parts of the validation process.

**4.Reliability:**

* Thread-Safe Operations: Synchronization mechanisms ensure data integrity.
* Comprehensive Validation: Concurrent validation of rows, columns, and subgrids guarantees accuracy.

**IMPLEMENTATION**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

int sudoku[9][9];

int i,j,k;

bool verification[11];

int auxSquares = 2;

typedef struct {

int row;

int column;

} parameter;

pthread\_t columns, rows, first\_square, second\_square, third\_square,

fourth\_square, fifth\_square, sixth\_square, seventh\_square, eighth\_square, ninth\_square;

void \*traverseColumn(void \*parameters) {

parameter \*column = (parameter \*) parameters;

int initialColumn = column->column;

int initialRow = column->row;

printf("Thread columns is running. Starting at column %d, row %d.\n", initialColumn + 1, initialRow + 1);

for ( i = initialColumn; i < 9; i++) {

int col[9] = {0};

for ( j = initialRow; j < 9; j++) {

int aux = sudoku[i][j];

if (col[aux - 1] == 0 && aux > 0) {

col[aux - 1] = aux;

} else if (aux <= 0) {

verification[0] = 0;

printf("Thread columns found an invalid number in column [%d], row [%d].\n", i + 1, j + 1);

pthread\_exit(0);

} else {

verification[0] = 0;

printf("Thread columns found a repeated number in column [%d], row [%d].\n", i + 1, j + 1);

pthread\_exit(0);

}

}

}

verification[0] = 1;

printf("Thread columns finished successfully.\n");

pthread\_exit(0);

}

void \*traverseRows(void \*parameters) {

parameter \*row = (parameter \*) parameters;

int initialColumn = row->column;

int initialRow = row->row;

printf("Thread rows is running. Starting at row %d, column %d.\n", initialRow + 1, initialColumn + 1);

for ( i = initialColumn; i < 9; i++) {

int rowArray[9] = {0};

for ( j = initialRow; j < 9; j++) {

int aux = sudoku[i][j];

if (rowArray[aux - 1] == 0 && aux > 0) {

rowArray[aux - 1] = aux;

} else if (aux <= 0) {

verification[0] = 0;

printf("Thread rows found an invalid number in row [%d], column [%d].\n", i + 1, j + 1);

pthread\_exit(0);

} else {

verification[0] = 0;

printf("Thread rows found a repeated number in row [%d], column [%d].\n", i + 1, j + 1);

pthread\_exit(0);

}

}

}

verification[1] = 1;

printf("Thread rows finished successfully.\n");

pthread\_exit(0);

}

void \*traverseSquare(void \*parameters) {

parameter \*squareParam = (parameter \*) parameters;

int initialRow = squareParam->row;

int initialColumn = squareParam->column;

int square[9] = {0};

printf("Thread square is running. Starting at row %d, column %d.\n", initialRow + 1, initialColumn + 1);

for ( i = initialRow; i < initialRow + 3; ++i) {

for ( j = initialColumn; j < initialColumn + 3; ++j) {

int aux = sudoku[i][j];

if (square[aux - 1] == 0 && aux > 0) {

square[aux - 1] = aux;

} else {

verification[auxSquares] = 0;

auxSquares++;

printf("Thread square found a repeated number in row [%d], column [%d].\n", i + 1, j + 1);

pthread\_exit(0);

}

}

}

verification[auxSquares] = 1;

auxSquares++;

printf("Thread square finished successfully started at row %d, column %d.\n", initialRow + 1, initialColumn + 1);

pthread\_exit(0);

}

int main() {

printf("Enter sudoku matrix\n");

for( i=0;i<9;i++){

for( j=0;j<9;j++){

scanf("%d",&sudoku[i][j]);

}

}

parameter \*checkRows = (parameter \*) malloc(sizeof(parameter));

checkRows->row = 0;

checkRows->column = 0;

parameter \*checkColumns = (parameter \*) malloc(sizeof(parameter));

checkColumns->row = 0;

checkColumns->column = 0;

parameter \*square1 = (parameter \*) malloc(sizeof(parameter));

square1->row = 0;

square1->column = 0;

parameter \*square2 = (parameter \*) malloc(sizeof(parameter));

square2->row = 0;

square2->column = 3;

parameter \*square3 = (parameter \*) malloc(sizeof(parameter));

square3->row = 0;

square3->column = 6;

parameter \*square4 = (parameter \*) malloc(sizeof(parameter));

square4->row = 3;

square4->column = 0;

parameter \*square5 = (parameter \*) malloc(sizeof(parameter));

square5->row = 3;

square5->column = 3;

parameter \*square6 = (parameter \*) malloc(sizeof(parameter));

square6->row = 3;

square6->column = 6;

parameter \*square7 = (parameter \*) malloc(sizeof(parameter));

square7->row = 6;

square7->column = 0;

parameter \*square8 = (parameter \*) malloc(sizeof(parameter));

square8->row = 6;

square8->column = 3;

parameter \*square9 = (parameter \*) malloc(sizeof(parameter));

square9->row = 6;

square9->column = 6;

pthread\_create(&columns, NULL, traverseColumn, (void \*) checkColumns);

pthread\_create(&rows, NULL, traverseRows, (void \*) checkRows);

pthread\_create(&first\_square, NULL, traverseSquare, (void \*) square1);

pthread\_create(&second\_square, NULL, traverseSquare, (void \*) square2);

pthread\_create(&third\_square, NULL, traverseSquare, (void \*) square3);

pthread\_create(&fourth\_square, NULL, traverseSquare, (void \*) square4);

pthread\_create(&fifth\_square, NULL, traverseSquare, (void \*) square5);

pthread\_create(&sixth\_square, NULL, traverseSquare, (void \*) square6);

pthread\_create(&seventh\_square, NULL, traverseSquare, (void \*) square7);

pthread\_create(&eighth\_square, NULL, traverseSquare, (void \*) square8);

pthread\_create(&ninth\_square, NULL, traverseSquare, (void \*) square9);

pthread\_join(columns, NULL);

pthread\_join(rows, NULL);

pthread\_join(first\_square, NULL);

pthread\_join(second\_square, NULL);

pthread\_join(third\_square, NULL);

pthread\_join(fourth\_square, NULL);

pthread\_join(fifth\_square, NULL);

pthread\_join(sixth\_square, NULL);

pthread\_join(seventh\_square, NULL);

pthread\_join(eighth\_square, NULL);

pthread\_join(ninth\_square, NULL);

for ( k = 0; k < 11; ++k) {

if (!verification[k]) {

printf("\nOops... The Sudoku was NOT solved.\n");

exit(0);

}

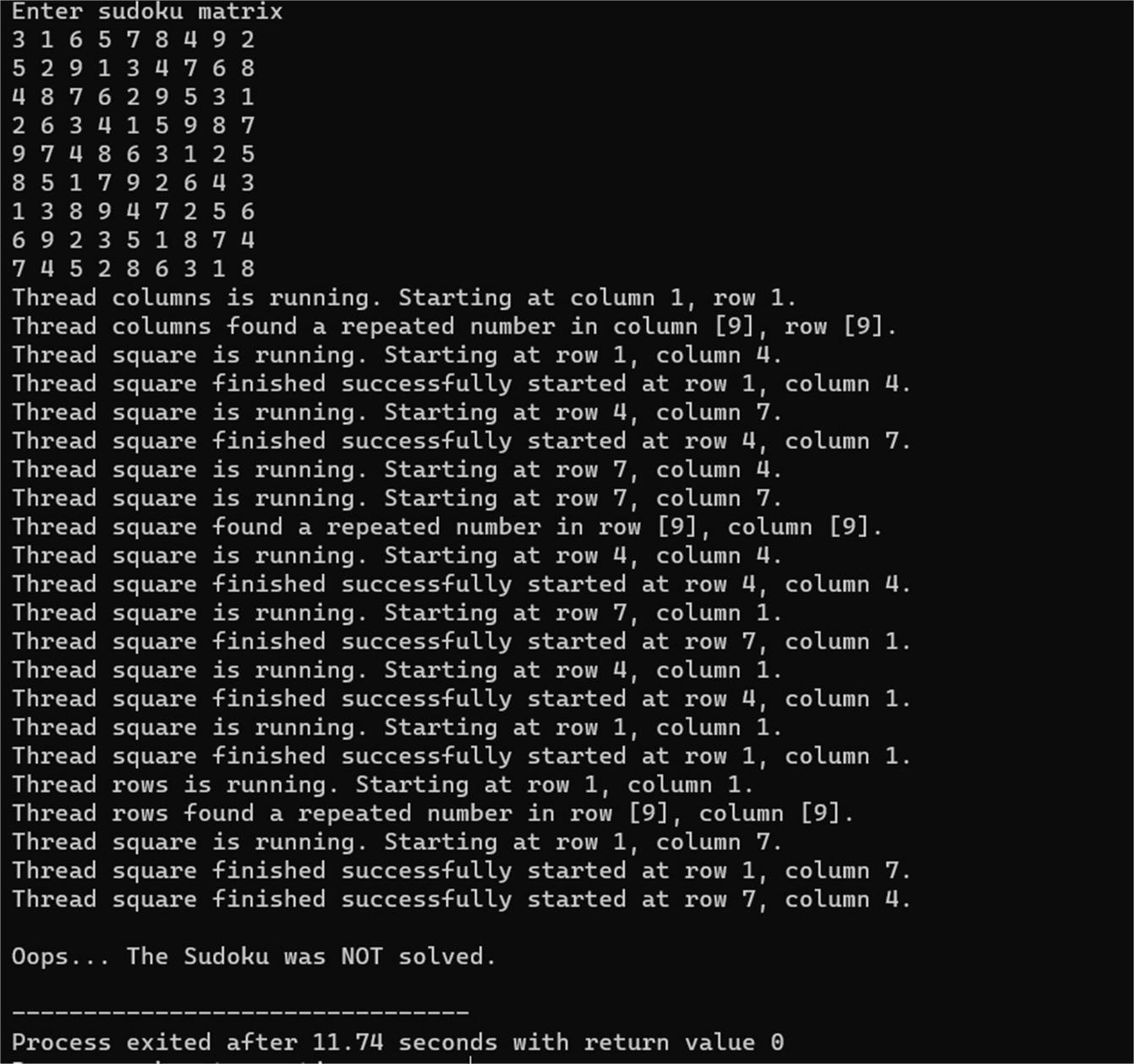
}

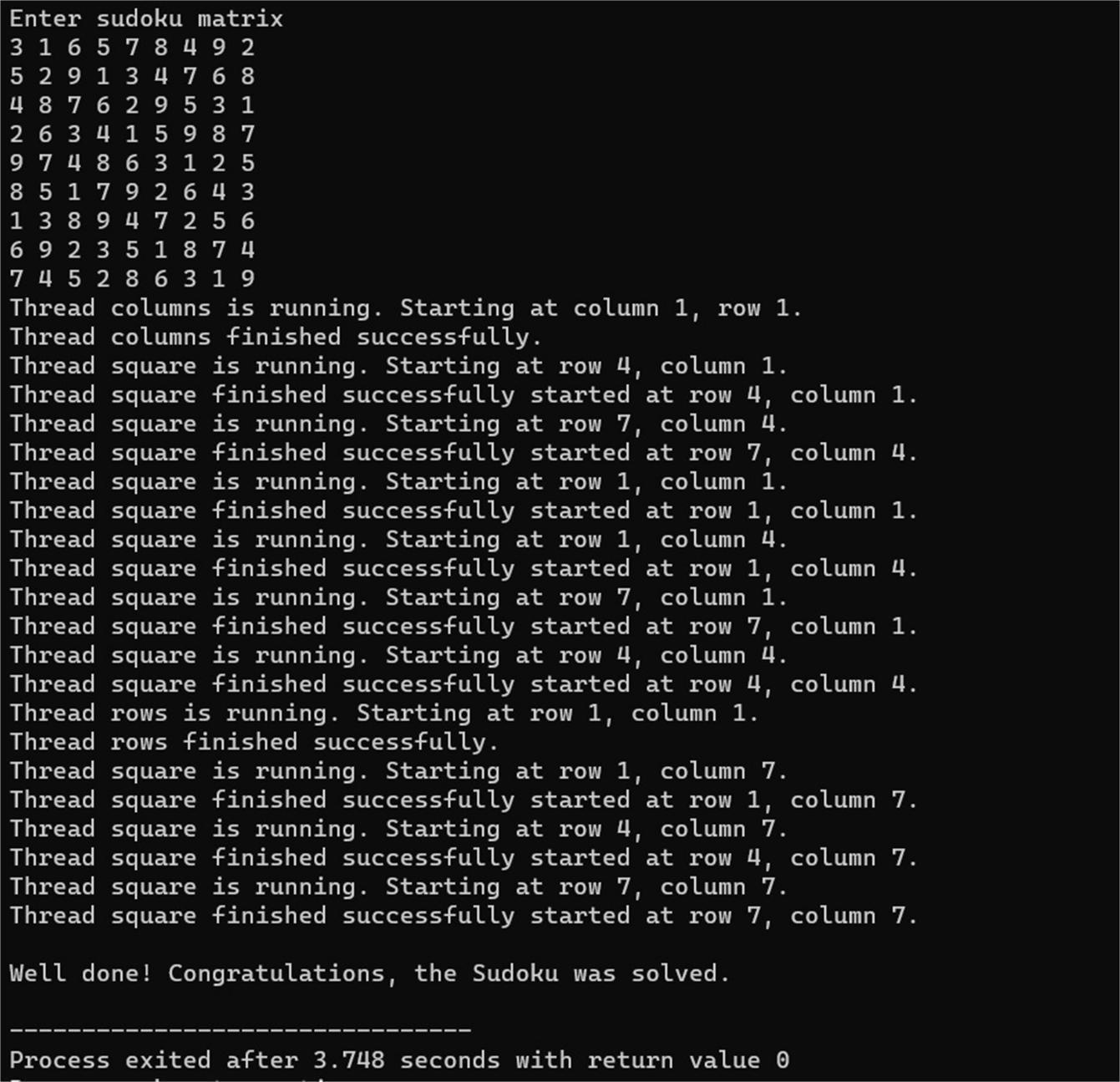
printf("\nWell done! Congratulations, the Sudoku was solved.\n");

return 0;

}

**RESULTS**





**CONCLUSION**

In conclusion, the project Multithreaded Sudoku Validator has validated the solution for the sudoku using concept multithreading where the user provides the input that has got after solving the sudoku and according to the rules of sudoku, this validator helped the user to check whether the solution he/she got.The implementation of this project code concluded the working of multithreading concept in validating the solution.multiple threads works for the row assigned to it and evaluates individually the result.

As we know there are different ways of multithreading this application.we have got to know concepts related to multithreading and it’s application in validating the solution.Concurrency, responsiveness and resource sharing are clearly participated in multithreading in this project and our team members have got an idea of working principle of this project and how threads are assigned to each row to check with the digits 1 to 9. 3x3 subgrids contains all digits from 1-9 without repetition and nine threads are to check these subgrids.Compare to other approaches in validating the solution of Sudoku this multithreading is efficient and time saving in executing as multiple threads works at a time.It provides the result with accuracy and without taking much execution time compared to other approaches,this is the advantage of this multithreading approach.And it is user-friendly if one thread stops working then other thread helps the process to continue.This is the one of the best way of approach we have concluded.

**REFERENCES**

1. GeeksforGeeks - Multithreaded Sudoku Validator in C++
2. Rosetta Code - Sudoku Solver
3. Python concurrent.futures module
4. Multithreaded Sudoku Solver in Python