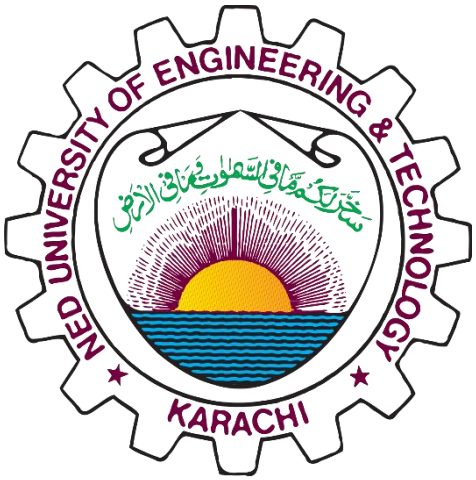
PROJECT REPORT

OPERATING SYSTEMS
(CT-353)



Submitted to:

Mr. Muhammad Mubashir Khan

Group Leader: Yashfa CT-21054

Ayesha Naz CT-21053

Namra Muneer CT-21059

Ruhma Alavi CT-21064

Department of Computer Science & Information Technology
NED University of Engineering & Technology

Problem Statement:

Sudoku is a popular number puzzle game where the objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids that compose the grid contain all of the digits from 1 to 9. Validating a completed Sudoku solution for correctness is a computationally intensive task, especially for larger-sized grids. The challenge is to efficiently validate the solution using parallel processing to improve performance. It should implement both single-threaded and multithreaded approaches to verify the correctness of the solution.

Proposed Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <iostream>
#include <chrono>

#define num_threads 27

using namespace std;
using namespace std::chrono;

/**
 * Structure that holds the parameters passed to a thread.
 * This specifies where the thread should start verifying.
 */
typedef struct
{
   // The starting row.
   int row;
   // The starting column.
   int col;
   // The pointer to the sudoku puzzle.
   int (* board)[9];
```

```
} parameters;

/*
   Initialize the array which worker threads can update to 1 if the
   corresponding region of the sudoku puzzle they were responsible
   for is valid.
*/
int result[num_threads] = {0};

// Prototype for 3x3 square function.
void *check_grid(void *params);

// Prototype for the check_rows function.
void *check_rows(void *params);

// Prototype for the check_cols function.
void *check_cols(void *params);

// Prototype for the single thread sudoku check function.
int sudoku_checker(int sudoku[9][9]);


void display_sudoku(int sudoku[9][9]) {
   cout << "Sudoku Grid:" << endl;
   for (int i = 0; i < 9; ++i) {
      for (int j = 0; j < 9; ++j) {
         cout << sudoku[i][j] << " ";
      }
      cout << endl;
   }
}

void input_sudoku(int sudoku[9][9]) {
   cout << "Enter the Sudoku grid (9x9):" << endl;
   for (int i = 0; i < 9; ++i) {
      cout << "Enter row " << i + 1 << " (separate numbers by space): ";
      for (int j = 0; j < 9; ++j) {
```

```
            cin >> sudoku[i][j];
        }
    }
}

/***
 * ENTRY POINT *
 **/

int main(void) {
    int sudoku[9][9];

    input_sudoku(sudoku);

    display_sudoku(sudoku);

    // Display grid size, content, and number of threads hardcoded
    cout << "Grid Size: 9x9" << endl;
    cout << "Grid Content:" << endl;
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) {
            cout << sudoku[i][j] << " ";
        }
        cout << endl;
    }
    cout << "Number of Threads: " << num_threads << endl;

    // Starting time for single thread execution
    steady_clock::time_point start_time_single_thread = steady_clock::now();

    if (sudoku_checker(sudoku))
        printf("Sudoku solution is invalid\n");
    else
        printf("Sudoku solution is valid\n");

    // Compute and return the elapsed time in milliseconds.
    steady_clock::time_point end_time_single_thread = steady_clock::now();
```

```
    duration<double> elapsed_time_single_thread =
duration_cast<duration<double>>(end_time_single_thread -
start_time_single_thread);

    cout << endl << "Total time using single thread: " <<
elapsed_time_single_thread.count() << " seconds" << endl << endl;

    // Starting time for execution with 27 threads
    steady_clock::time_point start_time_threads = steady_clock::now();

    pthread_t threads[num_threads];

    int threadIndex = 0;

    // ====== Create the threads ======
    //Create one thread each for the 9 3x3 grid, one thread each for the 9 columns,
and one thread each for the 9 rows. There will be a total of 27 threads created
    //Function call parameters are Thread identifier, thread attributes, function the
thread executes, parameters passed to function
    //Syntax for pthread_create function is pthread_create(pthread_t * thread,
pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            // ====== Declaration of the parameter for the 3X3 grid check threads
=======
            if (i % 3 == 0 && j % 3 == 0)
            {
                parameters *gridData = (parameters *) malloc(sizeof(parameters));
                gridData->row = i;
                gridData->col = j;
                gridData->board = sudoku;
                pthread_create(&threads[threadIndex++], NULL, check_grid,
gridData);
            }
```

```
        // ====== Declaration of the parameter for the row check threads
=======
        if (j == 0)
        {
            parameters *rowData = (parameters *) malloc(sizeof(parameters));
            rowData->row = i;
            rowData->col = j;
            rowData->board = sudoku;
            pthread_create(&threads[threadIndex++], NULL, check_rows,
rowData);
        }

        // ====== Declaration of the parameter for the column check threads
=======
        if (i == 0)
        {
            parameters *columnData = (parameters *)
malloc(sizeof(parameters));
            columnData->row = i;
            columnData->col = j;
            columnData->board = sudoku;
            pthread_create(&threads[threadIndex++], NULL, check_cols,
columnData);
        }
      }
   }

   // ======= Wait for all threads to finish their tasks =======
   //Parameters are Thread identifier and the return value of the function
executed by the thread
   for (int i = 0; i < num_threads; i++)
      pthread_join(threads[i], NULL);

   // If any of the entries in the valid array are 0, then the Sudoku solution is
invalid
   for (int i = 0; i < num_threads; i++)
   {
```

```
        if (result[i] == 0)
        {
            cout << "Sudoku solution is invalid" << endl;

            // Compute and return the elapsed time in milliseconds.
            steady_clock::time_point end_time_threads = steady_clock::now();
            duration<double> elapsed_time_threads =
duration_cast<duration<double>>(end_time_threads - start_time_threads);

            cout << endl << "Total time using 27 threads: " <<
elapsed_time_threads.count() << " seconds" << endl;
            return 1;
        }
    }
    cout << "Sudoku solution is valid" << endl;

    // Compute and return the elapsed time in milliseconds.
    steady_clock::time_point end_time_threads = steady_clock::now();
    duration<double> elapsed_time_threads =
duration_cast<duration<double>>(end_time_threads - start_time_threads);

    cout << endl << "Total time using 27 threads: " <<
elapsed_time_threads.count() << " seconds" << endl;

    return 0;
}

/*
 * Checks if a square of size 3x3 contains all numbers from 1-9.
 * There is an array called validarray[10] initialized to 0.
 * For every value in the square, the corresponding index in validarray[] is
checked for 0 and set to 1.
 * If the value in validarray[] is already 1, then it means that the value is
repeating. So, the solution is invalid.
 *
 * @param   void *      The parameters (pointer).
 */
```

```
void *check_grid(void * params)
{
   parameters *data = (parameters *) params;
   int startRow = data->row;
   int startCol = data->col;
   int validarray[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 (validarray[val] != 0)
            pthread_exit(NULL);
         else
            validarray[val] = 1;
      }
   }

   // If the execution has reached this point, then the 3x3 sub-grid is valid.
   result[startRow + startCol / 3] = 1; // Maps the 3X3 sub-grid to an index in
the first 9 indices of the result array
   pthread_exit(NULL);
}

/**
 * Checks each row if it contains all digits 1-9.
 * There is an array called validarray[10] initialized to 0.
 * For every value in the row, the corresponding index in validarray[] is checked
for 0 and set to 1.
 * If the value in validarray[] is already 1, then it means that the value is
repeating. So, the solution is invalid.
 *
 * @param   void *      The parameters (pointer).
 */
void *check_rows(void *params)
{
   parameters *data = (parameters *) params;
```

```
    int row = data->row;

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

    // If the execution has reached this point, then the row is valid.
    result[9 + row] = 1; // Maps the row to an index in the second set of 9 indices of the result array
    pthread_exit(NULL);
}

/**
 * Checks each column if it contains all digits 1-9.
 * There is an array called validarray[10] initialized to 0.
 * For every value in the row, the corresponding index in validarray[] is checked for 0 and set to 1.
 * If the value in validarray[] is already 1, then it means that the value is repeating. So, the solution is invalid.
 *
 * @param   void *      The parameters (pointer).
 */
void *check_cols(void *params)
{
    parameters *data = (parameters *) params;
    //int startRow = data->row;
    int col = data->col;

    int validarray[10] = {0};
    for (int i = 0; i < 9; i++)
    {
```

```
        int val = data->board[i][col];
        if (validarray[val] != 0)
            pthread_exit(NULL);
        else
            validarray[val] = 1;
    }

    // If the execution has reached this point, then the column is valid.
    result[18 + col] = 1; // Maps the column to an index in the third set of 9
indices of the result array
    pthread_exit(NULL);
}

/**
 * Checks each column/row if it contains all digits 1-9.
 * There is an array called validarray[10] initialized to 0.
 * For every value in the row/column, the corresponding index in validarray[] is
checked for 0 and set to 1.
 * If the value in validarray[] is already 1, then it means that the value is
repeating. So, the solution is invalid.
 *
 * @param   int      the row/column to be checked.
 */
int check_line(int input[9])
{
    int validarray[10] = {0};
    for (int i = 0; i < 9; i++)
    {
        int val = input[i];
        if (validarray[val] != 0)
            return 1;
        else
            validarray[val] = 1;
    }
    return 0;
}
```

```
/**
 * Checks each 3*3 grid if it contains all digits 1-9.
 * There is an array called validarray[10] initialized to 0.
 * For every value in the row/column, the corresponding index in validarray[] is checked for 0 and set to 1.
 * If the value in validarray[] is already 1, then it means that the value is repeating. So, the solution is invalid.
 *
 * @param   void *      The parameters (pointer).
 */
int check_grid(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 validarray[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 (validarray[val] != 0)
                  return 1;
               else
                  validarray[val] = 1;
            }
         }
      }
   }
   return 0;
}
```

```
/**
 * Checks if the sudoku solution is valid or not without using the PThreads function.
 *
 *
 * @param   int      The sudoku solution.
 */
int sudoku_checker(int sudoku[9][9])
{
   for (int i=0; i<9; i++)
   {
      /* check row */
      if(check_line(sudoku[i]))
         return 1;

      int check_col[9];
      for (int j=0; j<9; j++)
         check_col[j] = sudoku[i][j];

      /* check column */
      if(check_line(check_col))
         return 1;

      /* check grid */
      if(check_grid(sudoku))
         return 1;
   }
   return 0;
}
```

List of Functionalities:

The given code is a multi-threaded and single-threaded Sudoku solver implemented in C++. It provides the following functionalities:

1. **Input Sudoku Grid:** Allows the user to input a 9x9 Sudoku grid.

- void input_sudoku(int sudoku[9][9]): Takes user input for each cell in the Sudoku grid.

2. **Display Sudoku Grid:** Displays the current state of the Sudoku grid.
- void display_sudoku(int sudoku[9][9]): Displays the Sudoku grid.

3. **Single-Threaded Sudoku Checker:**
- int sudoku_checker(int sudoku[9][9]): Checks if the provided Sudoku grid is a valid solution using a single thread.

4. **Multi-Threaded Sudoku Checker:**
- Uses pthreads to create 27 threads (9 for 3x3 grids, 9 for rows, and 9 for columns).
- Each thread checks a specific region (3x3 grid, row, or column) of the Sudoku grid for validity.
- The result is stored in the result array, indicating whether each region is valid or not.
- Threads are joined to wait for their completion.

5. **Check 3x3 Grid Validity:**
- void *check_grid(void *params): Checks if a 3x3 sub-grid of the Sudoku puzzle is valid.

6. **Check Row Validity:**
- void *check_rows(void *params): Checks if a row of the Sudoku puzzle is valid.

7. **Check Column Validity:**
- void *check_cols(void *params): Checks if a column of the Sudoku puzzle is valid.

8. **Check Line Validity:**
- int check_line(int input[9]): Checks if a row or column of the Sudoku puzzle is valid.

9. **Check 3x3 Grid Validity (Single-Threaded):**
- int check_grid(int sudoku[9][9]): Checks if all 3x3 sub-grids in the

Sudoku puzzle are valid.

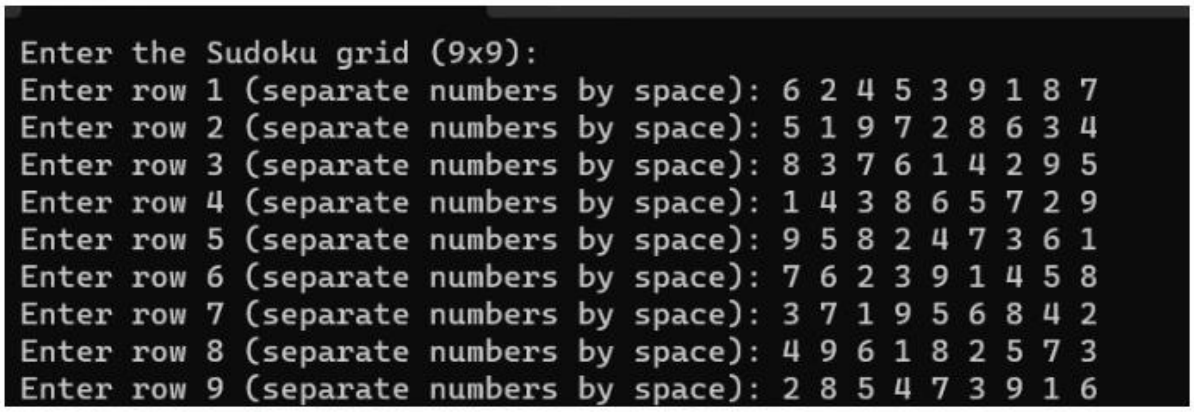
10. **Execution Time:**

- The code includes timing measurements to compare the execution time of the single-threaded and multi-threaded approaches. The duration class from the <chrono> library is used to measure and display the elapsed time.

11. **Main Functionality:**

- Takes user input for the Sudoku grid.
- Displays the initial Sudoku grid.
- Executes the single-threaded Sudoku checker and measures the time taken.
- Executes the multi-threaded Sudoku checker and measures the time taken.
- Displays whether the Sudoku solution is valid or invalid based on the results obtained from threads.

Screenshots Of Each Functionality:



```
Sudoku Grid:
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
Grid Size: 9x9
Grid Content:
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
Number of Threads: 27
Sudoku solution is valid

Total time using single thread: 9.22e-005 seconds

Sudoku solution is valid

Total time using 27 threads: 0.0036071 seconds
```

Tools used in project:

- C++ Programming Language
  Used as the primary language for coding the Sudoku validation logic.

- IDE (Integrated Development Environment)
  Dev C++ was used as our main IDE where we compiled and ran our code.

- Standard C++ Libraries

iostream (`#include <iostream>`):
- Used for input and output operations in C++ (`std::cout` and `std::cin`).
- It allows interaction with the user through the command-line interface.

chrono (`#include <chrono>`):
- Offers functionality to measure time durations and time points in C++.
- Utilized here for measuring the elapsed time for single-threaded and multi-threaded operations.

stdio.h (`#include <stdio.h>`):
- A C standard library for standard input and output operations.
- Although not explicitly used in this code snippet, it's included traditionally for compatibility with C code.

stdlib.h (`#include <stdlib.h>`):
- Another C standard library used for general-purpose functions like memory allocation and type conversions.
- Similar to `stdio.h`, included for compatibility reasons.

pthread.h (`#include <pthread.h>`):
- Header file for POSIX Threads (pthreads) library, primarily used in C for creating and managing threads.
- However, in this code, `std::thread` from the `<thread>` library is utilized for multi-threading instead of the pthreads library.