OS PROJECT

I200649 Muhammad Mujtaba   I201884 Ahmed Baig

Sudoku Solver

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

PSEUDO CODE OF PHASE 1:

```
//Checking within a matrix block
void * checkSingleMatrix(void * in)
   for (int i = starting_row; i < starting_row + 3; ++i){
      for (int j = starting_column; j < starting_column + 3; ++j){
         int num = grid[i][j];
                       if(num<0 || num>9){
                          printf("\nThe invalid number is: %d", num);
         }
         if(value[num] != 0){
                          invalid+=1;
            check=1;
                          printf("\nthread id of invalid thread: ", pthread_self());
                          printf("\nInvalid Square At: %d", starting_row);
                          printf(" %d", starting_column);
         }
         else{
            value[num] = 1;
          if(check == 1){
                    return 0;
   }
   return (void *) 1;
}
```

```
//Checkig the rows
void * checkRows(void * in) {
  for(int i = starting_row; i < 9; ++i){
    int row[10] = {0};
    for(int j = starting_column; j < 9; ++j){
      int num = grid[i][j];
      if(num<0 || num>9){
                      printf("\nThe invalid number is: %d", num);
      }
      if(row[num] != 0){
                      invalid+=1;
        check=1;
                      printf("\nthread id of invalid thread: ", pthread_self());
        printf("\nInvalid row:%d ", j);
        printf( "\nAt column:%d ", i);
                      return (void *) 0;
      }
      else{
        row[num] = 1;
      }
    }
  if(check == 1){
          return 0;
  }

  return (void *) 1;
```

```
}

//Check the columns
void * checkColumns(void * in) {
   for(int i = starting_column; i < 9; ++i){
      int col[10] = {0};
      for(int j = starting_row; j < 9; ++j){
         int num = grid[j][i];
         if(num<0 || num>9){
                        printf("\nThe invalid number is: %d", num);
         }
         if(col[num] != 0){
                           invalid+=1;
            check=1;
                           printf("\nthread id of invalid thread: ", pthread_self());
            printf("\nInvalid column:%d ", i);
            printf( "\nAt Row:%d ", j);
            return (void *) 0;
         }
         else{
            col[num] = 1;
         }
   if(check == 1){
             return 0;
   }
          return (void *) 1;
}
```

PESUDO CODE FOR PHASE 2:

```
//These are the functions(prototypes) that we have created  :
void* valid(void*);
int solve();
int find_empty_cell(int *, int *);

typedef struct
{
  int row;
  int column;
  int guess;
}val;

int invalid=0;
{
  int row = 0;
  int column = 0;

  if (solve())
  {
    printf("INVALID TRIES: %d", invalid);
    printf("\n\n\n correct solution: \n");
    for (int x = 0; x < 9; ++x)
    {
      for (int y = 0; y < 9; ++y)
      {
        printf(" %d", puzzle[x][y]);
      }
      printf("\n");

      if (x % 3 == 2)
      {
        printf("\n");
      }
    }
  }
  else
  {
    printf("\n\nSOLUTION NOT FOUND\n\n");
    printf("\n+-----+-----+-----+\n");

    for (int x = 0; x < 9; ++x)
    {
      for (int y = 0; y < 9; ++y)
      {
        printf("%d", puzzle[x][y]);
      }
      printf("\n");
```

```
      if (x % 3 == 2)
      {
        printf("\n");
      }
    }
  }
  return 0;
}

void* valid(void* param)
{
            val* v   = (val*)param;

  int corner_x = v->row / 3 * 3;
  int corner_y = v->column / 3 * 3;

  for (int x = 0; x < 9; ++x)
  {
    if (puzzle[v->row][x] == v->guess)
    {
      return (void*)0;
    }

    if (puzzle[x][v->column] == v->guess)
    {
      return (void*)0;
    }
    if (puzzle[corner_x + (x % 3)][corner_y + (x / 3)] == v->guess)
    {
      return (void*)0;
    }
  }

  return (void*)1;
}

int find_empty_cell(int *row, int *column)
{
  for (int x = 0; x < 9; x++)
  {
    for (int y = 0; y < 9; y++)
    {
      if (!puzzle[x][y])
      {
        *row = x;
        *column = y;

        return 1;
      }
    }
  }
  return 0;
}
```

```
pthread_t tid[10];

void* results[10];


//solving function

//THE GUESSING PART IS IMPLIMENTED USING THREADS TO CHECK IN PARALLEL WHETHER THE
GUESS WAS CORRECT OR NOT

int solve()

{

 int row;

 int column;


 if(!find_empty_cell(&row, &column))

 {

  return 1;

 }


 val* data = (val*) malloc(sizeof(val));


 for (int guess = 1; guess < 10; guess++)

 {

	data->row = row;

	data->column = column;

	data->guess = guess;


	pthread_create(&tid[guess-1],NULL, valid, (void*) data);

  pthread_join(tid[guess-1],&results[guess-1]);
```

```
    if ((int)results[guess-1])
    {
      puzzle[row][column] = guess;

      if(solve())
      {
        return 1;
      }

      puzzle[row][column] = 0;
            }

    printf("----------------------\n");
    printf("Thread ID: ", pthread_self());
    printf("Row:%d",data->row);
    printf("Col:%d",data->column);
    printf("This is invalid position for Guess:%d",data->guess);
    printf("\n");

    invalid++;
            }

  return 0;
}
```

System Specifications:

Muhammad Mujtaba:

Device name: LAPTOP-LJUODP4R4

Processor: AMD Ryzen 5 @ 2.3GHz

Graphic Card: AMD Raedon

Installed RAM: 8.0 GB (5.78 GB usable)

System type: 64-bit operating system, x64-based processor

Ahmed Baig

Device name: LAP-727

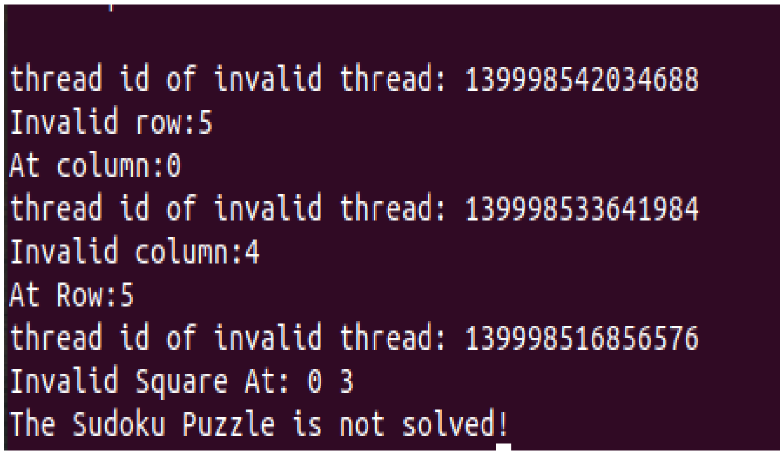
Processor: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz

Installed RAM: 8.00 GB (7.84 GB usable)

System type: 64-bit operating system, x64-based processor

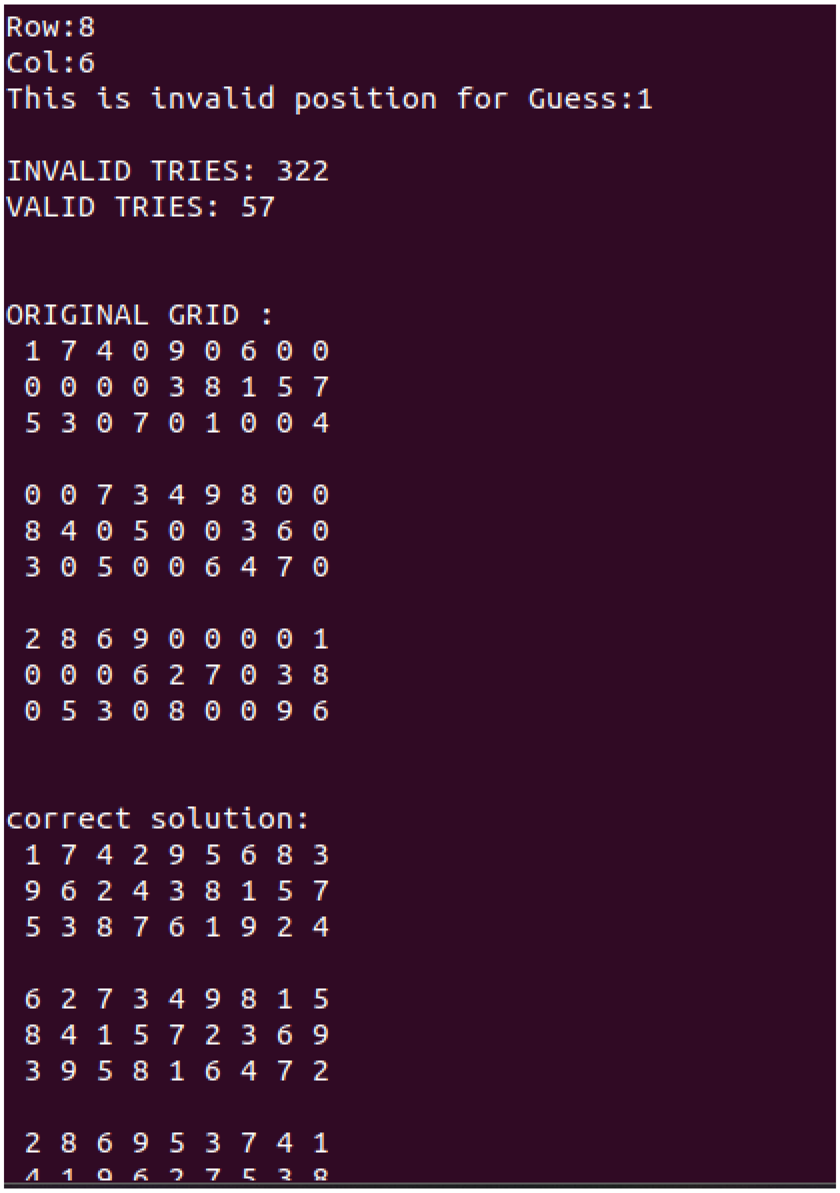
Output snippet

Phase 1



```
thread id of invalid thread: 139998542034688
Invalid row:5
At column:0
thread id of invalid thread: 139998533641984
Invalid column:4
At Row:5
thread id of invalid thread: 139998516856576
Invalid Square At: 0 3
The Sudoku Puzzle is not solved!
```

Phase 2



```
Row:8
Col:6
This is invalid position for Guess:1

INVALID TRIES: 322
VALID TRIES: 57


ORIGINAL GRID :
 1 7 4 0 9 0 6 0 0
 0 0 0 0 3 8 1 5 7
 5 3 0 7 0 1 0 0 4

 0 0 7 3 4 9 8 0 0
 8 4 0 5 0 0 3 6 0
 3 0 5 0 0 6 4 7 0

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


correct solution:
 1 7 4 2 9 5 6 8 3
 9 6 2 4 3 8 1 5 7
 5 3 8 7 6 1 9 2 4

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

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

CONCEPTS USED:

I have made 3 functions,

The first function is to check the rows of the matrix. It uses 2 for loops to check each element of the matrix, the first loop is updating the row number of the element and the second is updating the column number of the element. I have added an array and it compares each element with the array. If the row values are 1-9 the check is true and the row is valid. If not, the row is not valid.

The second function is to check the column of the matrix. It uses 2 for loops to check each element of the matrix, the first loop is updating the row number of the element and the second is updating the column number of the element. I have added an array and it compares each element with the array. If the column values are 1-9 the check is true and the column is valid. If not, the column is not valid.

The third function is to check the 3x3 matrix box of the matrix. It uses 2 for loops to check each element of the matrix, the first loop is updating the row number of the element (and it only runs for the 3 consecutive rows) and the second is updating the column number of the element (and it only runs for the 3 consecutive columns). I have added an array and it compares each element with the array. If the box values are 1-9 the check is true and the box is valid. If not, the box is not valid.

We have used joinable threads in this case to implement the **SUDOKO SOLVER AND VALIDATOR.** So the threads work in **Parallel** which decreases computation time and system resources are used more efficiently. All the threads are joined with the main driver thread which receives the resources released by each thread for further processing. We have used **mutex lock in the Solver** to avoid the board to be read while the change is being updated.

OTHER SENARIO:

We can use threading in “**FILE SEACRCH ENGINE”** with each file being searched in a separate thread. If we must replace something (overwrite it) we can use **semaphores to limit** the threads performing read-write operations**.**

Same concepts can used to make game like Ludo or Chess. Will store the whole board and will create each thread of each box and so on. Will store the location of each piece in each thread id for a smooth execution.