1/4/2023

Sudoku Project Report

by Usman(20i0937), Ahmed(20i1893)

Contents

Pseudocodes

Phase 1

Column Validation

```
function Column_Validation(void *args)
    chk = 1
    valid_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    j = 0
    num_check = 0
    k = 0
    for i = 0 to 8
        for loop = 0 to 9
            valid_arr[loop] = 0
        for j = 0 to 8
            num_check = sudoku[j][i]
            if num_check < 1 or num_check > 9
                print error message
                exit thread
            else if valid_arr[num_check] = 1
                incol_i_c[k] = i
                incol_i_r[k] = j
                incol_v[k] = num_check
                print error message
                k = k + 1
                chk = 2
                invalid_count = invalid_count + 1
            else
                valid_arr[num_check] = 1
    if chk = 1
        valid[0] = 1
    if chk = 2
        valid[0] = 0
    exit thread
```

Row Validation

```
function Row_Validation(void *args)
    chk = 1
    valid_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    j = 0
    k = 0
    for i = 0 to 8
        for loop = 0 to 9
            valid_arr[loop] = 0
        for j = 0 to 8
            check_num = sudoku[i][j]
            if check_num < 1 or check_num > 9
                print error message
                exit thread
            else if valid_arr[check_num] = 1
                inrow_i_r[k] = i
                inrow_i_c[k] = j
                inrow_v[k] = check_num
                print error message
                k = k + 1
                chk = 2
                invalid_count = invalid_count + 1
            else
                valid_arr[check_num] = 1
    if chk = 1
        valid[1] = 1
    if chk = 2
        valid[1] = 0
    exit thread
```

3x3 Validation

```
function three_Grid_Validation(void *args)
    k = 0
    chk = 1
    valid_arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    value = 0
    para = (parameters*) args
    r = para.row
    c = para.column
    if c > 6 or r > 6 or c % 3 != 0 or r % 3 != 0
        print error message
        exit thread
    for i = r to r + 2
        for loop = 0 to 9
            valid_arr[loop] = 0
        for j = c to c + 2
            value = sudoku[i][j]
            if value < 1 or value > 9
                print error message
            else if valid_arr[value] = 1
                in3x3_i_r[k] = i
                in3x3_i_c[k] = j
                in3x3_v[k] = value
                print error message
                chk = 2
                k = k + 1
                invalid_count = invalid_count + 1
            else
                valid_arr[value] = 1
    if chk = 1
        valid[r + c/3] = 1
    exit thread
```

Phase 2

Sudoku Row Solver

```
function suduko_rowsolver(void *args)
    lock mutex
    para = (data*) args
    t1, t2 = new threads
    rowind = para.row
    colind = para.column
    val = para.value
    findreplaceentry = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    replacemententry = 0

    for i = rowind to rowind, j = 0 to 8
        findreplaceentry[sudoku[i][j]] = 1
    for i = 0 to 9
        if findreplaceentry[i] = 0 and i != 0
            replacemententry = i

    print replacemententry
    if replacemententry >= 1 and replacemententry <= 9
        sudoku[rowind][colind] = replacemententry
        moves = moves + 1
        unlock mutex
        exit thread
```

Sudoku Column Solver

```
function suduko_colsolver(void *args)
    lock mutex
    para = (data*) args
    t1, t2 = new threads
    rowind = para.row
    colind = para.column
    val = para.value
    findreplaceentry = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    replacemententry = 0

    for i = colind to colind, j = 0 to 8
        findreplaceentry[sudoku[j][i]] = 1
    for i = 0 to 9
        if findreplaceentry[i] = 0 and i != 0
            replacemententry = i
    if replacemententry >= 1 and replacemententry <= 9
        print replacemententry
        sudoku[rowind][colind] = replacemententry
        moves = moves + 1
    unlock mutex
```

3x3 Sudoku Solver

```
function suduko_3x3solver(void *args)
    para = (data*) args
    t1, t2 = new threads
    rowind = para.row
    colind = para.column
    val = para.value
    findreplaceentry = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    replacemententry = 0
    faultyrow = 0
    faultycol = 0

    for i = rowind to rowind+2
        for j = colind to colind+2
            if findreplaceentry[sudoku[i][j]] = 1
                faultyrow = i
                faultycol = j
            else
                findreplaceentry[sudoku[i][j]] = 1
    for i = 0 to 9
        if findreplaceentry[i] = 0 and i != 0
            replacemententry = i
    if replacemententry >= 1 and replacemententry <= 9
        print replacemententry
        sudoku[faultyrow][faultycol] = replacemententry
        moves = moves + 1
```

Operating system concepts used in Pseudocodes

Synchronization (using thread):

Threading is used in these pseudocodes to allow for concurrent execution of the different validation functions (Column_Validation, Row_Validation, three_Grid_Validation). Each of these functions is executed in a separate thread, allowing them to run in parallel and potentially finish faster than if they were executed sequentially.

Synchronization (using mutex):

Synchronization, specifically using mutex, is used in these pseudocodes to protect shared resources from being accessed by multiple threads at the same time. For example, the invalid_count variable is incremented whenever an invalid value is found in the sudoku grid. This variable is shared by all threads, so a mutex lock is used to ensure that only one thread can access and modify it at a time, preventing race conditions.

Code illustration

The three functions are all similar in structure, each taking a pointer to void as an argument and returning nothing. They are all checking for errors in the sudoku grid, and printing error messages if they find any. The functions are also updating several global variables in each case, such as chk, valid_arr, k, inrow_i_r, inrow_i_c, inrow_v, and invalid_count.

The Column_Validation function checks for errors in the sudoku grid by iterating through each column and checking if the numbers in the column are valid (i.e., between 1 and 9 inclusive). If an invalid number is found, the function prints an error message and exits the thread. If a duplicate number is found, the function updates the inrow_i_r, inrow_i_c, and inrow_v variables and prints an error message.

The Row_Validation function is similar to the Column_Validation function, but it checks for errors in each row of the sudoku grid instead of each column.

The three_Grid_Validation function checks for errors in each 3x3 subgrid of the sudoku grid. It takes a parameters structure as an argument, which contains the row and column values to start checking from. If the row and column values are out of bounds or not divisible by 3, the function prints an error message and exits the thread. Otherwise, it iterates through the 3x3 subgrid and checks for invalid or duplicate numbers in the same way as the previous two functions.

Implemented Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define threads_number 9

pthread_mutex_t mut1;


int check_duplicate_arr[threads_number+2] = {0};
int invalid_count = 0;
int incol_i_r[9] = {0};
int incol_i_c[9] = {0};
int incol_v[9] = {0};
int inrow_i_r[9] = {0};
int inrow_i_c[9] = {0};
int inrow_v[9] = {0};
int in3x3_i_r[9] = {0};
int in3x3_i_c[9] = {0};
int in3x3_v[9] = {0};
int moves = 0;

typedef struct
{
    int row;
    int column;
} validity_parameters;

typedef struct
{
    int row;
    int column;
    int value;
} solution_parameters;

int sudoku_9x9_grid[9][9] = {
    {4, 2, 6, 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},
    {6, 9, 4, 1, 8, 2, 5, 7, 3},
    {2, 8, 5, 4, 7, 3, 9, 1, 6}
};

void *Column_Validation(void *args)//function to validate each column. It
iterates through each column and returns true if all columns are resolved and
don't contain duplicate entries
{

    int chk = 1;
    int valid_arr[10] = {0};
    int j = 0;
    int num_check = 0;
    int k = 0;
    for(int i = 0; i < 9 ; i++)//row
    {
        for (int loop = 0; loop < 10; loop++)
        {
            valid_arr[loop] = 0;
        }

        for(j = 0 ; j < 9 ; j++)//col
        {
            num_check = sudoku_9x9_grid[j][i];
            if(num_check < 1 || num_check > 9)
            {
                printf("Column or Row entries = %d ,are out of
bound!\n",num_check);
                pthread_exit(NULL);
            }
            else if(valid_arr[num_check] == 1)
            {
                incol_i_c[k] = i;//saving column index of wrong entry
                incol_i_r[k] = j;//saving row index of wrong entry
                incol_v[k] = num_check;//saving value of wrong entry
                printf("Number = %d, has duplicate on Index =
[%d][%d]\n",incol_v[k],incol_i_r[k]+1,incol_i_c[k]+1);
                k++;
                chk = 2;
                invalid_count++;//increasing invalid entry count
            }
            else
            {
                valid_arr[num_check] = 1;
```

```
            }
        }

    }
   if(chk == 1)
   check_duplicate_arr[0] = 1;//1 in validityarray indicates that all columns are
valid
   if(chk == 2)
   check_duplicate_arr[0] = 0;
   pthread_exit(NULL);
}

void  *Row_Validation(void *args)//function to validate each row. It iterates
through each row and returns true if all rows are resolved and don't contain
duplicate entries
{
        int chk = 1;
        int valid_arr[10] = {0};
        int j = 0;
        int k = 0;

        for (int i = 0; i < 9; i++) // row
        {
            for (int loop = 0; loop < 10; loop++)
            {
            valid_arr[loop] = 0;
            }


            for (j = 0 ; j < 9 ; j++) // col
            {
                int check_num = sudoku_9x9_grid[i][j];
                if(check_num < 1 || check_num > 9)
                {
                    printf("Column and Row entries are out of bound!\n");
                    pthread_exit(NULL);
                }
                else if(valid_arr[check_num] == 1)
                {
                    inrow_i_r[k] = i;//saving row index of wrong entry
                    inrow_i_c[k] = j;//saving column index of wrong entry
                    inrow_v[k] = check_num;//saving value of wrong entry
                    printf("Number = %d, has duplicate on Index =
[%d][%d]\n",inrow_v[k],inrow_i_r[k]+1,inrow_i_c[k]+1);
                    k++;
```

```
                    chk = 2;
                    invalid_count++;//increasing invalid entry count
                }
                else
                {
                    valid_arr[check_num] = 1;
                }
            }
        }
        if(chk == 1)
        check_duplicate_arr[1] = 1;//1 in validityarray indicates that all rows
are valid
        if(chk == 2)
        check_duplicate_arr[1] = 0;
        pthread_exit(NULL);
}

void *three_Grid_Validation(void *args)
{
    int k = 0;
    int chk = 1;
    int valid_arr[10] = {0};
    int value = 0;
    validity_parameters * para = (validity_parameters*) args;
    int r = para->row;
    int c = para->column;
    if(c > 6 || r > 6 || c % 3 != 0 || r % 3 != 0)
    {
       printf("Given entries for Row and Columns are out of bound!\n");
       pthread_exit(NULL);
    }
    for (int i = r ; i < r + 3 ; i++)
    {

        for (int loop = 0; loop < 10; loop++)
        {
            valid_arr[loop] = 0;
        }

        for (int j = c ; j < c + 3 ; j++)
        {
            value = sudoku_9x9_grid[i][j];
            if(value < 1 || value > 9)
            {
                printf("The values in sudoku table are out of bound!\n");
```

```
            }
            else if(valid_arr[value] == 1)
            {
                in3x3_i_r[k] = i;//saving row index of wrong entry
                in3x3_i_c[k] = j;//saving column index of wrong entry
                in3x3_v[k] = value;//saving value of wrong entry
                printf("Number = %d ,has duplicate on Index =
[%d][%d]\n",in3x3_v[k],in3x3_i_r[k]+1,in3x3_i_c[k]+1);
                chk = 2;
                k++;
                invalid_count++;//increasing invalid entry count
            }
            else
            {
                valid_arr[value] = 1;
            }
        }
    }
    if (chk == 1)
    check_duplicate_arr[r + c/3] = 1;//1 in validityarray indicates that this 3x3
grid is valid

    pthread_exit(NULL);

}

void *suduko_rowsolver(void *args)
{
    pthread_mutex_lock(&mut1);//synchronization technique used
    solution_parameters * para = (solution_parameters*) args;
    pthread_t t1,t2;
    int rowind = para->row;
    int colind = para->column;
    int val = para->value;
    int findreplaceentry[10] = {0};
    int replacemententry;

    for(int i = rowind,j = 0; j < 9; j++)
    {
        findreplaceentry[sudoku_9x9_grid[i][j]] = 1;//1 at a specific index of
findreplaceentry indicates that this value is found in row
    }
    for(int i = 0; i <= 9; i++)
    {
        if(findreplaceentry[i] == 0 && i != 0)
```

```
            replacemententry = i;
    }

    printf("Entry to be placed = %d\n",replacemententry);

    if(replacemententry >= 1 && replacemententry <= 9)
    {
        sudoku_9x9_grid[rowind][colind] = replacemententry;//swaping the row
entry in sudoku 2D matrix
        moves++;
    }
    pthread_mutex_unlock(&mut1);
    pthread_exit(NULL);
}

void *suduko_colsolver(void *args)
{
    pthread_mutex_lock(&mut1);//synchronization technique used
    solution_parameters * para = (solution_parameters*) args;
    pthread_t t1,t2;
    int rowind = para->row;
    int colind = para->column;
    int val = para->value;
    int findreplaceentry[10] = {0};
    int replacemententry;

    for(int i = colind,j = 0; j < 9; j++)
    {
        findreplaceentry[sudoku_9x9_grid[j][i]] = 1;//1 at a specific index of
findreplaceentry indicates that this value is found in row
    }
    printf("\n");
    for(int i = 0; i <= 9; i++)
    {
        if(findreplaceentry[i] == 0 && i != 0)
            replacemententry = i;
    }

    if(replacemententry >= 1 && replacemententry <= 9)
    {
        printf("Entry to be placed = %d\n",replacemententry);
        sudoku_9x9_grid[rowind][colind] = replacemententry;//swaping the column
entry in sudoku 2D matrix
        moves++;
    }
```

```
    pthread_mutex_unlock(&mut1);
    pthread_exit(NULL);
}

void *suduko_3x3solver(void *args)
{
    pthread_mutex_lock(&mut1);//synchronization technique used
    solution_parameters * para = (solution_parameters*) args;
    pthread_t t1,t2;
    int rowind = para->row;
    int colind = para->column;
    int val = para->value;
    int findreplaceentry[10] = {0};
    int replacemententry;
    int faultyrow,faultycol;

    for(int i = rowind; i < rowind+3; i++)
    {
        for(int j = colind; j < colind+3; j++)
        {
            if(findreplaceentry[sudoku_9x9_grid[i][j]] == 1)//If 1 at a specific
index already than it indicates that duplication has occured
            {
                faultyrow = i;
                faultycol = j;
            }
            else
            {
                findreplaceentry[sudoku_9x9_grid[i][j]] = 1;//1 at a specific
index of findreplaceentry indicates that this value is found in row
            }
        }
    }
    printf("\n");
    for(int i = 0; i <= 9; i++)
    {
        if(findreplaceentry[i] == 0 && i != 0)
            replacemententry = i;
    }
    if(replacemententry >= 1 && replacemententry <= 9)
    {
        printf("Entry to be placed = %d\n",replacemententry);
        sudoku_9x9_grid[faultyrow][faultycol] = replacemententry;//swaping the
entries of 3x3 grid in sudoku 2D matrix
        moves++;
```

```
    }
    pthread_mutex_unlock(&mut1);
    pthread_exit(NULL);
}

int main()
{
    pthread_t t1,t2,threads[threads_number];
    pthread_mutex_init(&mut1,NULL);//initializing mutex tobe used later in
functions
    int ind = 0;

    printf("Matrix before solution\n");//printing sudoku 9x9 grid
    for(int i = 0; i < 9; i++)
    {
        for(int j = 0; j < 9; j++)
        {
            printf("%d ",sudoku_9x9_grid[i][j]);
        }
        printf("\n");
    }

    printf("\nColumn Validation\n");
    pthread_create(&t1,NULL,Column_Validation,NULL);//thread for column
validation
    printf("Thread ID for column validation = %d\n",t1);
    pthread_join(t1,NULL);

    printf("\nRow Validation\n");
    pthread_create(&t2,NULL,Row_Validation,NULL);//thread for row validation
    printf("Thread ID for row validation = %d\n",t2);
    pthread_join(t2,NULL);
    printf("\n3x3 Validation\n");
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            if(i % 3 == 0 && j % 3 == 0)
            {
            validity_parameters *para = (validity_parameters*)
malloc(sizeof(validity_parameters));
            para->row = i;
            para->column = j;
            pthread_create(&threads[ind],NULL,three_Grid_Validation,para);//threa
ds for 3x3 grid validation
```

```
            printf("Thread ID for 3x3 validation = %d, Index =
%d\n",threads[ind],ind);
            pthread_join(threads[ind], NULL);
            ind++;
            }
        }
    }

    while(1)//loop until all duplications are resolved
    {
        printf("\nTotal invalid entries = %d\n",invalid_count);

        int counter1 = 0,counter2 = 0,counter3 = 0;

        for (int  i = 0; i < 9; i++)
        {
            if (in3x3_v[i] != 0 || in3x3_v[i] != NULL)
            {
                counter3++;//counter for invalid entries in 3x3 grid
            }
            if (incol_v[i] != 0 || incol_v[i] != NULL)
            {
                counter2++;//counter for invalid entries in columns
            }
            if (inrow_v[i] != 0 || inrow_v[i] != NULL)
            {
                counter1++;//counter for invalid entries in rows
            }
        }

        printf("\nrow issues = %d\n",counter1);
        printf("col issues = %d\n",counter2);
        printf("3x3 issues = %d\n",counter3);
        pthread_t t3[counter1];//threads against invalid entries in rows
        pthread_t t4[counter2];//threads against invalid entries in columns
        pthread_t t5[counter3];//threads against invalid entries in 3x3 grids


        for (int  i = 0; i < counter1; i++)
        {
            solution_parameters *d = (solution_parameters*)
malloc(sizeof(solution_parameters));
            d->value = inrow_v[i];
            d->row = inrow_i_r[i];
            d->column = inrow_i_c[i];
```

```
            printf("\nValue at trouble = %d, row = %d, column = %d\n",d->value,d-
>row+1,d->column+1);
            pthread_create(&t3[i],NULL,suduko_rowsolver,d);//thread calling
fuction to resolve duplication in rows
            printf("Thread ID for row solution = %d\n",t3);
            pthread_join(t3[i],NULL);
        }
        for (int  i = 0; i < counter2; i++)
        {
            solution_parameters *d = (solution_parameters*)
malloc(sizeof(solution_parameters));
            d->value = incol_v[i];
            d->row = incol_i_r[i];
            d->column = incol_i_c[i];
            printf("Value at trouble = %d, row = %d, column = %d\n",d->value,d-
>row+1,d->column+1);
            pthread_create(&t4[i],NULL,suduko_colsolver,d);//thread calling
fuction to resolve duplication in columns
            printf("Thread ID for column solution = %d\n",t4);
            pthread_join(t4[i],NULL);
        }

        for (int  i = 0; i < counter3; i++)
        {
            solution_parameters *d = (solution_parameters*)
malloc(sizeof(solution_parameters));
            for(int i = 0; i < 9; i++)
            {
                for(int j = 0; j < 9; j++)
                {
                    if(i%3 == 0 && j%3 == 0)
                    {
                        d->value = in3x3_v[i];
                        d->row = i;
                        d->column = j;
                        printf("Value at trouble = %d, row = %d, column =
%d\n",d->value,d->row+1,d->column+1);
                        pthread_create(&t5[0],NULL,suduko_3x3solver,d);//thread
calling fuction to resolve duplication in 3x3 grid
                        printf("Thread ID for 3x3 solution = %d, Index =
%d\n",t5[0],0);
                        pthread_join(t5[0],NULL);
                    }
                }
            }
```

```
        }

        pthread_create(&t1,NULL,Row_Validation,NULL);//again row validation
        printf("Thread ID for column validation = %d\n",t1);
        pthread_join(t1,NULL);
        pthread_create(&t2,NULL,Column_Validation,NULL);//again column validation
        printf("Thread ID for row validation = %d\n",t2);
        pthread_join(t2,NULL);
        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                if(i % 3 == 0 && j % 3 == 0)
                {
                validity_parameters *para = (validity_parameters*)
malloc(sizeof(validity_parameters));
                para->row = i;
                para->column = j;
                pthread_create(&threads[ind],NULL,three_Grid_Validation,para);//a
gain 3x3 grid validation
                printf("Thread ID for 3x3 validation = %d, Index =
%d\n",threads[ind],ind);
                pthread_join(threads[ind], NULL);
                ind++;
                }
            }
        }

        int va0 = check_duplicate_arr[0], va1 = check_duplicate_arr[1];

        if(va0 == 1 && va1 == 1)//breaks while loop if all rows and columns are
valid & resolved
        {
            printf("\nSudoku has become valid!\n");
            break;
        }
    }

    printf("Number of moves = %d\n",moves);

    printf("Matrix after solution\n");//printing matrix after solution
    for(int i = 0; i < 9; i++)
    {
        for(int j = 0; j < 9; j++)
        {
```

```
            printf("%d ",sudoku_9x9_grid[i][j]);
        }
        printf("\n");
    }

    pthread_mutex_destroy(&mut1);
    pthread_exit(NULL);
}
```

Machine Specifications

Asus Vivo Book S14

Brand = Asus

Series = Vivo Book

Model = S433FL

OS Version = Windows 11

Processor = Core i7 10^{th} generation

Processor Speed = 10510U Core i7 1.8 GHz up to 4.9 GHz

Ram = 16GB DDR4

Rom = 1TB SSD

Dell Vostro 3468

Brand = Dell

Series = Vostro

Model = Vostro 3468

OS Version = Windows 10

Processor = Core i7 7^{th} generation

Processor Speed = 7500U Core i7 2.7Ghz up to 3.5Ghz

Ram = 8GB DDR4

Rom = 256GB SSD + 500GB Hard Disk

OS concepts in another scenario

One scenario where threading and mutex locks could be used is in the development of a multiplayer online game. In such a game, it would be necessary to ensure that actions taken by one player do not interfere with or disrupt the gameplay of other players. Mutex locks could be used to synchronize access to shared game state data, while threading could be used to allow multiple players to interact with the game simultaneously. This would ensure a smooth and fair gaming experience for all players.

Group detail

Our group consists of two members.

- **Usman(20i-0937)**

  Worked on phase 1, that includes row, column and 3x3 grid validation of Sudoku 2D matrix.

- **Ahmed(20i-1893)**

  Worked on phase 2, that includes row, column and 3x3 grid duplicate entry solution.