Faculty of Engineering & Applied Science

CRN:74026
Operating Systems

Lab 3
Group#: 5

Due Date: March 27th, 2021

First Name	Last Name	Student Number
Philip	Jasionowski	100751888
William	Robinson	100751756

Our 3 major components are the Main function, the sudokuChecker and the sudokuSolver

SudokuChecker

```c
int checkSudoku ()
{
    int returnVal = 0;
    int *res = 0;
    //coor squares[9];
    pthread_t thy;
    pthread_t thx;
    pthread_t th[9];


    //creates one instance of the x and y threads
    if (pthread_create(&thy, NULL, &checkThreadY, NULL)≠0){
        perror("thread creation failed");
    }
    if (pthread_create(&thx, NULL, &checkThreadX, NULL)≠0){
        perror("thread creation failed");
    }
    //loops and creates 9 squareChecker threads

    for (int i = 0; i < 9; i++)
    {
        int *a = malloc(sizeof(int));
        *a = i;
        if (pthread_create(&th[i], NULL,checkThreadSquare, a));
    }

    //joins x and y threads
    if (pthread_join(thy, (void**) &res)≠0){
        perror("thread join failed");
    }
    //sums res* (the return value) to returnVal
    returnVal += *res;
    if (pthread_join(thx, (void**) &res)≠0){
        perror("thread join failed");
    }
    returnVal += *res;
    for (int i = 0; i < 9; i++)
    {
        if (pthread_join(th[i], (void**) &res)≠0){
            perror("thread join failed");
        };
        //sums res* (the return value) to returnVal
        returnVal += *res;
    }

//returns returnVal, if the number is greater than 0 then the solution is invalid
    return returnVal;
}

```

This is the main checkSudoku function, it creates a total of 11 threads and checks if the solution presented is valid. It creates a y checking thread and an x checking thread. Then it loops 9 times and creates 9 square checker threads

Figure 1 checkSudoku function

```c
void *checkThreadY(void *args)
{
    //debugging code
    //allocates an int for returning
    int *result = malloc(sizeof(int));
    *result = 0;
    //Loops through array to check if each collumn sums to 45
    for (int x = 0; x<9; x++)
    {
        int temp = 0;
        for (int y = 0; y<9; y++)
        {
            temp += puzzle[y][x];
        }
        if (temp != 45)

//if any column does not equal 45 the result int will get switched to 1 and return 1
        {
            *result = 1;
        }
    }
    //returns the result casted as a void pointer
    return ((void *) result);
}
```

Figure 2 checkY thread

Here is the checkY thread, the checkX thread is functionally similar so I will ignore that. The thread allocates memory for result so that it will not be destroyed when the thread finishes. The thread then iterates through each column (in the x thread its each row) and sums up the total, in a standard sudoku game the sum for a valid row, column or square will be 45 (1+2+3+4+5+6+7+8+9 = 45)

If the result is not 45 it sets result to 1. After it finishes iterating through it returns 1 if it is invalid and 0 if it is not.

```c
void *checkThreadSquare(void *args)
{
    int index = *(int *)args;
    int *result = malloc(sizeof(int));
    *result = 0;
    int x1 = 0;
    int y1 = 0;
    //gets coordinates from a single int here
    x1 = ((int)floor(index / 3)) * 3;
    y1 = (index % 3) * 3;

    int temp = 0;
    for (int x = 0; x < 3; x++)
    {
        for (int y = 0; y < 3; y++)
        {
            temp += puzzle[y+y1][x+x1];
        }
    }
    if (temp != 45)

//if any column does not equal 45 the result int will get switched to 1 and return 1
        {
            *result = 1;
        }
    //returns the result casted as a void pointer
    //frees the argument (we allocated memory for it in the loop)
    free(args);
    return ((void *) result);
}
```

Here is the checkThreadSquare thread. It takes an in argument of 0-8. 0 being the first square (top left) and 8 being the last one (bottom right) it then converts this to a coordinate using this math snippet.

```c
    x1 = ((int)floor(index / 3)) * 3;
    y1 = (index % 3) * 3;

```

Afterwards it iterates in the square and sums. If the sum is not 45 it returns a 1.

SudokuSolve function

```c
int solve(int sudoku[9][9])
{
    int row;
    int column;
//if sudoku puzzle is solved (every cell is full) return 1
    if(!find_empty_cell(sudoku, &row, &column)) return 1;
    for (int guess = 1; guess < 10; guess++)
//cycles through numbers 1 to 9 to check for a valid guess at each part
    {
        if (valid(sudoku, row, column, guess))
        {
            sudoku[row][column] = guess;
//iterates deeper
            if(solve(sudoku)) return 1;
//if that guess does not yield a solution, set the square equal to 0 and iterate further
            sudoku[row][column] = 0;
        }
    }
  return 0;
}
```

The solve function is a depth first recursive algorithm. It scans for an empty cell and then guesses the numbers 1 through 9. After each guess it calls the solve function one deeper, if the guess does not work it works back up 1 level and iterates to the next guess.

```c
int valid(int sudoku[9][9], int row, int column, int guess)
{
    int corner_x = row / 3 * 3;
    int corner_y = column / 3 * 3;

    for (int x = 0; x < 9; ++x)
    {
//effeciant code to increment through each square, row and column of the guess
//if the number is found, return a 0, if the number is not found it returns a 1
        if (sudoku[row][x] == guess) return 0;
        if (sudoku[x][column] == guess) return 0;
        if (sudoku[corner_x + (x % 3)][corner_y + (x / 3)] == guess) return 0;
    }
    return 1;
}
```

Main Function

```c
int main(int argc, char const *argv[])
{
    //opening file code
    FILE *FILE1;
    FILE1 = fopen("puzzle.txt", "r");
    if(FILE1 == NULL){
        //throw error
        printf("Error opening file.\n");
        exit(1);
    }

    for(int i = 0; i < 9; i++){
        for(int j = 0; j < 9; j++){
            fscanf(FILE1, "%d", &puzzle[i][j]);
        }
    }
    //closes file upon reading
    fclose(FILE1);
    printf("Your sudoku puzzle is: \n");
    print(puzzle);
    printf("\n");

    printf("\nSolving the given puzzle......\n");

    solve(puzzle);

    print(puzzle);
    printf("\n");

    int *res = 0;

//if checkSudoku() returns more than 1 we know it was not a valid solution
    if (checkSudoku() >= 1)
    {
        printf("Sudoku Solution is invalid \n");
    }
    else
    {
        printf("Sudoku Solution is valid \n");
        //outputs file here
        FILE *FILE2;

        FILE2= fopen("solution.txt","w+");
//open of create file named solution.txt
        if(FILE2 == NULL){ // chek for error
            printf("Error opening/creating file.\n");
            exit(1);
        }

        for(int i = 0; i < 9; i++){ //for loop to add text to file
            for(int j = 0; j < 9; j++){
                fprintf(FILE2,"%d ", puzzle[i][j]);
            }
            fprintf(FILE2,"\n");
        }
        //closes file from output
        fclose(FILE2);
        printf("Solution printed to solution.txt\n");
    }
}
```

Here is the main function in its entirety. It opens the puzzle.txt file, scans it and adds the contents to an array called puzzle. Then it prints it and solves it. Afterwards it checks the solution, if it is valid it will then output the solution to a text file.