Homework 2 Report
CSC 4320 Operating Systems
Spring 2018
Name: Jimmy Tran
Email: jtran25@student.gsu.edu

1) Source Code

```
/**

 * A skeleton for checking a Sudoku puzzle through multithread



 */



#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>




#define NUMBER_OF_THREADS   9 // we use 9 threads in this project

#define PUZZLE_SIZE         9  // it is a 9x9 puzzle




/* example puzzle */



// this is a sample table, the values will be replaced by those read from the input file, except -1

int puzzle[PUZZLE_SIZE+1][PUZZLE_SIZE+1] = {

                        {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},

                        {-1,5,3,4,6,7,8,9,1,2},
                        {-1,6,7,2,1,9,5,3,4,8},
                        {-1,1,9,8,3,4,2,5,6,7},
                        {-1,8,5,9,7,6,1,4,2,3},
                        {-1,4,2,6,8,5,3,7,9,1},
                        {-1,7,1,3,9,2,4,8,5,6},
                        {-1,9,6,1,5,3,7,2,8,4},
                        {-1,2,8,7,4,1,9,6,3,5},
                        {-1,3,4,5,2,8,6,1,7,9}
                };


int status_map[NUMBER_OF_THREADS] = {0};

/* data structure for passing data to threads */
typedef struct
{
        int thread_number;
        int x;
        int y;
} parameters;

//row-checking thread code
void * row(void * params){
        parameters * data=(parameters *) params;

        int t_num=data->thread_number;

        int row=data->x;

        int col=data->y;

        int good=1;


        for(int j=col;j<col+3;j++){

                int nums[9]={0};

                for(int i=row;i<10;++i){

                        //check row nums

                        int val=puzzle[j][i];

                        if(nums[val-1]==0){nums[val-1]=1;}

                        else{printf("This is not a valid puzzle. %d is a duplicate.\n",val);good=0;}

                }

        }

        if(good==1){

                status_map[t_num]=1;

                printf("row set %d is good.\n",t_num);

        }

        return (void *)1;

};

//column-checking thread code

void * col(void * params){
	parameters * data=(parameters *) params;
	int t_num=data->thread_number;
	int row=data->x;
	int col=data->y;
	int good=1;

	for(int j=col;j<col+3;j++){
		int nums[9]={0};
		for(int i=row;i<10;++i){
			//check col nums
			int val=puzzle[i][j];
			if(nums[val-1]==0){nums[val-1]=1;}
			else{printf("This is not a valid puzzle. %d is a duplicate.\n",val);good=0;}
		}
	}
	if(good==1){
		status_map[t_num]=1;
		printf("col set %d is good.\n",t_num-3);
	}
	return (void *)1;
};
//subgrid-checking thread code
void * sub(void * params){
	parameters * data=(parameters *) params;

	int t_num=data->thread_number;

	int row=data->x;

	int col=data->y;

	int good=1;


	for(int i=0;i<3;i++){

		int nums[9]={0};

		for(int j=row;j<row+3;j++){

			for(int k=col;k<col+3;++k){

				int val=puzzle[j][k];

				if(nums[val-1]==0){nums[val-1]=1;}

				else{printf("This is not a valid puzzle. %d is a
duplicate.\n",val);good=0;}

			}

		}

		col+=3;

	}


	if(good==1){

		status_map[t_num]=1;

		printf("subgrid set %d is good.\n",t_num-6);

	}
```

```
            return (void *)1;

};


int main(int argc, char *argv[])

{

            // get a puzzle from a txt file, which needs to be input from command line

            char filename[100];

            printf("Please enter your filename:\n");

            scanf("%s",filename);


  FILE *myFile;

  myFile = fopen(filename, "r");


   //read file into array

   int i,j;


   if (myFile == NULL)

   {

       printf("Error Reading File\n");

       exit (0);

   }

   for (i = 1; i < PUZZLE_SIZE + 1; i++)

   {
```

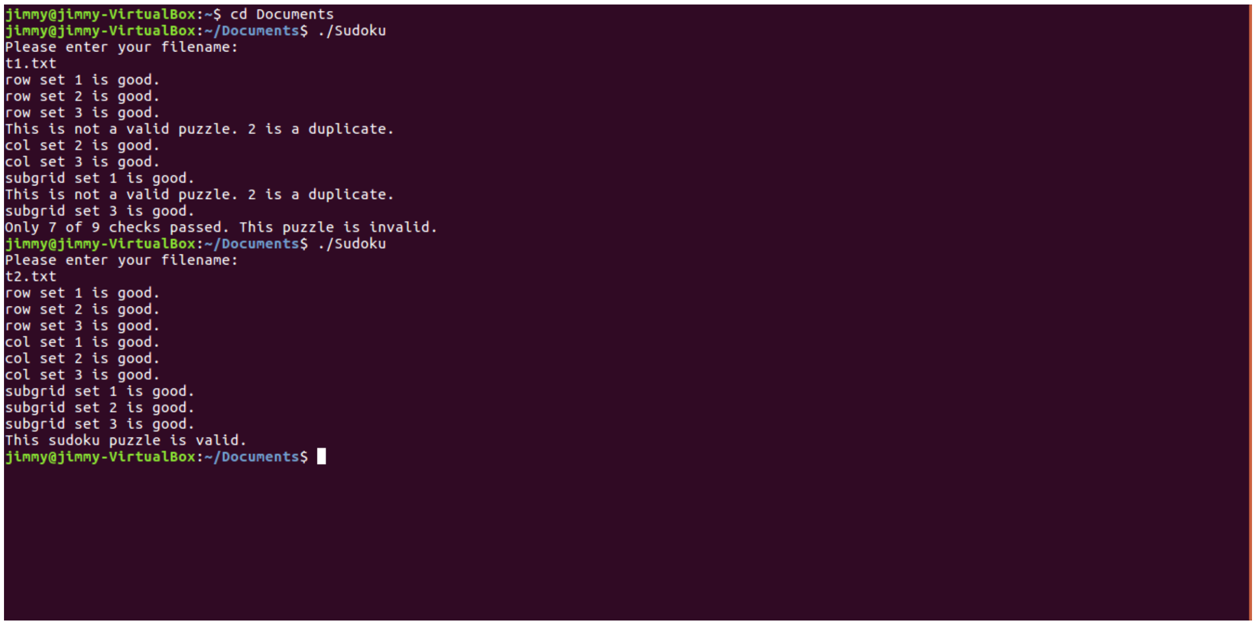
```
                for (j =1; j < PUZZLE_SIZE + 1; j++)

                {

                        fscanf(myFile, "%d,", &puzzle[i][j] );

                }


}


        //pthread_t workers[NUMBER_OF_THREADS];





 //create the 3 threads for row check

        //rows 1-3

        pthread_t thread_row1;

        void * row1;

        parameters *r1data=(parameters *) malloc(sizeof(parameters));

        r1data->thread_number=1;

        r1data->x=1;

        r1data->y=1;

        pthread_create(&thread_row1,NULL,row,(void *)r1data);

        pthread_join(thread_row1,&row1);


        //rows 4-6
	pthread_t thread_row2;

	void * row2;

	parameters *r2data=(parameters *) malloc(sizeof(parameters));

	r2data->thread_number=2;

	r2data->x=4;

	r2data->y=1;

	pthread_create(&thread_row2,NULL,row,(void *)r2data);

	pthread_join(thread_row2,&row2);


	//rows 7-9

	pthread_t thread_row3;

	void * row3;

	parameters *r3data=(parameters *) malloc(sizeof(parameters));

	r3data->thread_number=3;

	r3data->x=7;

	r3data->y=1;

	pthread_create(&thread_row3,NULL,row,(void *)r3data);

	pthread_join(thread_row3,&row3);


//create the 3 threads for column check

	//cols 1-3

	pthread_t thread_col1;

	void * col1;
```

```
parameters *c1data=(parameters *) malloc(sizeof(parameters));

c1data->thread_number=4;

c1data->x=1;

c1data->y=1;

pthread_create(&thread_col1,NULL,col,(void *)c1data);

pthread_join(thread_col1,&col1);


//cols 4-6

pthread_t thread_col2;

void * col2;

parameters *c2data=(parameters *) malloc(sizeof(parameters));

c2data->thread_number=5;

c2data->x=1;

c2data->y=4;

pthread_create(&thread_col2,NULL,col,(void *)c2data);

pthread_join(thread_col2,&col2);


//cols 7-9

pthread_t thread_col3;

void * col3;

parameters *c3data=(parameters *) malloc(sizeof(parameters));

c3data->thread_number=6;

c3data->x=1;
```

```
        c3data->y=7;

        pthread_create(&thread_col3,NULL,col,(void *)c3data);

        pthread_join(thread_col3,&col3);


    //create the 9 threads for the 9 subfields

        //subgrids 1-3

        pthread_t thread_sub1;

        void * sub1;

        parameters *s1data=(parameters *) malloc(sizeof(parameters));

        s1data->thread_number=7;

        s1data->x=1;

        s1data->y=1;

        pthread_create(&thread_sub1,NULL,sub,(void *)s1data);

        pthread_join(thread_sub1,&sub1);


        //subgrids 4-6

        pthread_t thread_sub2;

        void * sub2;

        parameters *s2data=(parameters *) malloc(sizeof(parameters));

        s2data->thread_number=8;

        s2data->x=4;

        s2data->y=1;

        pthread_create(&thread_sub2,NULL,sub,(void *)s2data);
```

```
	pthread_join(thread_sub2,&sub2);


	//subgrids 7-9

	pthread_t thread_sub3;

	void * sub3;

	parameters *s3data=(parameters *) malloc(sizeof(parameters));

	s3data->thread_number=9;

	s3data->x=7;

	s3data->y=1;

	pthread_create(&thread_sub3,NULL,sub,(void *)s3data);

	pthread_join(thread_sub3,&sub3);


	//wait for the threads to exit and check the result and print

	int check=0;

	for(int i=0;i<10;i++){

	if(status_map[i]==1){check++;}

	}

	if(check==9){printf("This sudoku puzzle is valid.\n");}

	else{printf("Only %d of 9 checks passed. This puzzle is invalid.\n",check);}

	pthread_exit(NULL);

	return 0;

}
```

2) Screenshot:



```
jimmy@jimmy-VirtualBox:~$ cd Documents
jimmy@jimmy-VirtualBox:~/Documents$ ./Sudoku
Please enter your filename:
t1.txt
row set 1 is good.
row set 2 is good.
row set 3 is good.
This is not a valid puzzle. 2 is a duplicate.
col set 2 is good.
col set 3 is good.
subgrid set 1 is good.
This is not a valid puzzle. 2 is a duplicate.
subgrid set 3 is good.
Only 7 of 9 checks passed. This puzzle is invalid.
jimmy@jimmy-VirtualBox:~/Documents$ ./Sudoku
Please enter your filename:
t2.txt
row set 1 is good.
row set 2 is good.
row set 3 is good.
col set 1 is good.
col set 2 is good.
col set 3 is good.
subgrid set 1 is good.
subgrid set 2 is good.
subgrid set 3 is good.
This sudoku puzzle is valid.
jimmy@jimmy-VirtualBox:~/Documents$ 
```

3) For my version of the Sudoku program, the main function asks for a text file and then checks if the filename is valid. Then it reads the file and replaces the values in a default 10x10 2D array that stores the sudoku puzzle values.

Once that's done, it creates nine threads using the parameters struct that call three other functions I programmed in: void * row(), void * col(), and void * sub().

Each are called by three threads for a total of nine threads. No parent thread was needed with this implementation. The functions take parameters from the thread that called them and uses those to find where in the sudoku puzzle they're supposed to check.

For example, the first thread calls void * row() which checks the first three rows.

All of these functions use an array of size 9 to track which values have appeared while in the for loops that iterate through the relevant puzzle section.

The functions do not check if input is actually valid or not (barring duplicates which trigger a printf alert); however, at the end of each function there is a check of the arrays to ensure all nine

values have been found. If any are missing, the status map array, which tracks which threads have valid puzzle sections, is not changed for the respective thread.
At the end of the main function after all threads have terminated, status_map is checked to see if all its values are 1 or not. If they're all one, the puzzle is valid. Otherwise, it is invalid and an appropriate message will state so.

4) A single thread would almost be the same as not using a thread in the first place. The program will be slower than its three and nine thread counterparts. Three threads are more efficient than one but requires more processing power. It is also slower than using nine. Nine threads will be the most efficient at processing but will use the most processing power in turn. In general, more threads equals more processing speed but at the cost of more processing power and possible complexity in programming it.