Jerome Anaeki

**Homework 2**

**Description**

I used 3 thread functions. One is used to check columns, one is used to check rows, and the other is to check 3X3 subgrids. I used input/output functions to open the files and save the 9X9 grid in a 2-D array. I created 11 structure variables and were used to pass information when creating worker threads. I created 11 threads and were initialized with thread functions and variables for return values. Then, I called pthread\_join() function for each thread.

**Discussion**

11 children threads:

Advantages – improves the throughput

Disadvantages – increases complexity

3 threads:

Advantages – The program will speed up with less threads

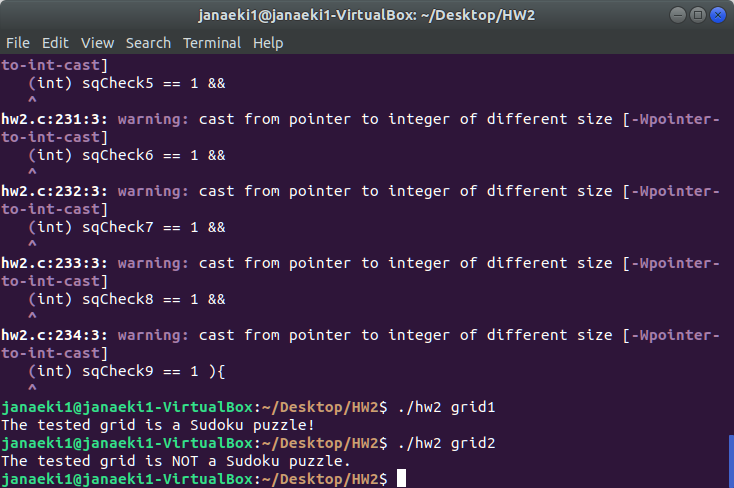
Disadvantages – less synchronization of tasks

1 thread:

Advantages – Bugs can easily be found

Disadvantages – Poor performance

**Screenshot**

****

**Code**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

/\* You will need a structure to store the information to be passed to each thread (see "Passing Parameters to Each Thread" in the textbool's project description)\*/

typedef struct{

int row;

int column;

int (\* grid)[9];

} parameters;

/\* Declare the thread that checks columns \*/

void \* column\_worker(void \* param); // the function parameter is for the structure you defined

/\* You will also need to declare threads that checks rows and 3x3 subgrids \*/

void \* row\_worker(void \* param); //Declaration of the thread that checks rows

void \* subgrid\_worker(void \* param); //Declaration of the thread that checks 3x3 subgrid

/\* You will need to declare an array of integer values that is visible to each thread. The value in the array (0 or 1) indicates whether the worker thread's number is valid (see "Returning Results to the Parent Thread" in the textbook's project description) \*/

int workers[11];

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

{

char \*n;

if (argc != 2) {

fprintf(stderr,"Usage: ./hw2 <filename>\n");

return -1;

}

n = argv[1];

FILE \*fp = fopen(n, "r");

if(fp == NULL){

printf("Error: File cannot be found\n");

}

int i, j, ch, grid[9][9];

for(i = 0; i < 9; i++){

for(j = 0; j < 9; j++){

ch = fgetc(fp);

grid[i][j] = ch;

grid[i][j] = grid[i][j] - 48; //Converts character to integer value

ch = fgetc(fp);

}

}

/\*You need to assign values to the structure variable. Then you can create multiple worker threads by passing the information using the structure variable\*/

//Parameter for columns

parameters \* col\_param = (parameters \*) malloc(sizeof(parameters));

col\_param->row = 0;

col\_param->column = 0;

col\_param->grid = grid;

//Parameter for rows

parameters \* row\_param = (parameters \*) malloc(sizeof(parameters));

row\_param->row = 0;

row\_param->column = 0;

row\_param->grid = grid;

//Parameter for first 3X3 subgrid

parameters \* sq1 = (parameters \*) malloc(sizeof(parameters));

sq1->row = 0;

sq1->column = 0;

sq1->grid = grid;

//Parameter for second 3X3 subgrid

parameters \* sq2 = (parameters \*) malloc(sizeof(parameters));

sq2->row = 0;

sq2->column = 3;

sq2->grid = grid;

//Parameter for third 3X3 subgrid

parameters \* sq3 = (parameters \*) malloc(sizeof(parameters));

sq3->row = 0;

sq3->column = 6;

sq3->grid = grid;

//Parameter for fourth 3X3 subgrid

parameters \* sq4 = (parameters \*) malloc(sizeof(parameters));

sq4->row = 3;

sq4->column = 0;

sq4->grid = grid;

//Parameter for fifth 3X3 subgrid

parameters \* sq5 = (parameters \*) malloc(sizeof(parameters));

sq5->row = 3;

sq5->column = 3;

sq5->grid = grid;

//Parameter for sixth 3X3 subgrid

parameters \* sq6 = (parameters \*) malloc(sizeof(parameters));

sq6->row = 3;

sq6->column = 6;

sq6->grid = grid;

//Parameter for seventh 3X3 subgrid

parameters \* sq7 = (parameters \*) malloc(sizeof(parameters));

sq7->row = 6;

sq7->column = 0;

sq7->grid = grid;

//Parameter for eigth 3X3 subgrid

parameters \* sq8 = (parameters \*) malloc(sizeof(parameters));

sq8->row = 6;

sq8->column = 3;

sq8->grid = grid;

//Parameter for ninth 3X3 subgrid

parameters \* sq9 = (parameters \*) malloc(sizeof(parameters));

sq9->row = 6;

sq9->column = 6;

sq9->grid = grid;

//Declaration of variables for threads

pthread\_t colThr, rowThr, sqThr1, sqThr2, sqThr3, sqThr4, sqThr5, sqThr6, sqThr7, sqThr8, sqThr9;

//Return values for threads

void \* colCheck;

void \* rowCheck;

void \* sqCheck1;

void \* sqCheck2;

void \* sqCheck3;

void \* sqCheck4;

void \* sqCheck5;

void \* sqCheck6;

void \* sqCheck7;

void \* sqCheck8;

void \* sqCheck9;

//Initialization of threads

pthread\_create(&colThr, NULL, column\_worker, (void \*) col\_param);

pthread\_create(&rowThr, NULL, row\_worker, (void \*) row\_param);

pthread\_create(&sqThr1, NULL, subgrid\_worker, (void \*) sq1);

pthread\_create(&sqThr2, NULL, subgrid\_worker, (void \*) sq2);

pthread\_create(&sqThr3, NULL, subgrid\_worker, (void \*) sq3);

pthread\_create(&sqThr4, NULL, subgrid\_worker, (void \*) sq4);

pthread\_create(&sqThr5, NULL, subgrid\_worker, (void \*) sq5);

pthread\_create(&sqThr6, NULL, subgrid\_worker, (void \*) sq6);

pthread\_create(&sqThr7, NULL, subgrid\_worker, (void \*) sq7);

pthread\_create(&sqThr8, NULL, subgrid\_worker, (void \*) sq8);

pthread\_create(&sqThr9, NULL, subgrid\_worker, (void \*) sq9);

/\*You need to call pthread\_join() for each childer thread so that the parent will wait\*/

//Waiting for all threads to finish

pthread\_join(colThr, &colCheck);

pthread\_join(rowThr, &rowCheck);

pthread\_join(sqThr1, &sqCheck1);

pthread\_join(sqThr2, &sqCheck2);

pthread\_join(sqThr3, &sqCheck3);

pthread\_join(sqThr4, &sqCheck4);

pthread\_join(sqThr5, &sqCheck5);

pthread\_join(sqThr6, &sqCheck6);

pthread\_join(sqThr7, &sqCheck7);

pthread\_join(sqThr8, &sqCheck8);

pthread\_join(sqThr9, &sqCheck9);

/\* Finally, after all children returns, you can check the status array that is visible to everyone and see if it is valid. You then print out the final checking result\*/

if((int) colCheck == 1){

workers[0] = 1;

}

if((int) rowCheck == 1){

workers[1] = 1;

}

if((int) sqCheck1 == 1){

workers[2] = 1;

}

if((int) sqCheck2 == 1){

workers[3] = 1;

}

if((int) sqCheck3 == 1){

workers[4] = 1;

}

if((int) sqCheck4 == 1){

workers[5] = 1;

}

if((int) sqCheck5 == 1){

workers[6] = 1;

}

if((int) sqCheck6 == 1){

workers[7] = 1;

}

if((int) sqCheck7 == 1){

workers[8] = 1;

}

if((int) sqCheck8 == 1){

workers[9] = 1;

}

if((int) sqCheck9 == 1){

workers[10] = 1;

}

int a, valid;

for(a=0; a>11; a++){

printf("%d ", workers[a]);

if(workers[a] == 1){

valid += 1;

}

}

if( valid == 11 ||

(int) colCheck == 1 &&

(int) rowCheck == 1 &&

(int) sqCheck1 == 1 &&

(int) sqCheck2 == 1 &&

(int) sqCheck3 == 1 &&

(int) sqCheck4 == 1 &&

(int) sqCheck5 == 1 &&

(int) sqCheck6 == 1 &&

(int) sqCheck7 == 1 &&

(int) sqCheck8 == 1 &&

(int) sqCheck9 == 1 ){

printf("The tested grid is a Sudoku puzzle!\n");

}

else {

printf("The tested grid is NOT a Sudoku puzzle.\n");

}

fclose(fp);

return 0;

}

//Checks each column for numbers from 1-9

void \* column\_worker(void \* param) {

parameters \* data = (parameters \*) param;

int i, j;

int rowStart = data->row;

int colStart = data->column;

for (i = colStart; i < 9; i++) {

int arr[10] = {0};

for (j = rowStart; j < 9; j++) {

int value = data->grid[j][i];

if (arr[value] != 0) {

return (void \*) 0;

}

else{

arr[value] = 1;

}

}

}

return (void \*) 1;

pthread\_exit(0);

}

//Checks each row for numbers from 1-9

void \* row\_worker(void \* param) {

parameters \* data = (parameters \*) param;

int i, j;

int rowStart = data->row;

int colStart = data->column;

for (i = rowStart; i < 9; i++) {

int arr[10] = {0};

for (j = colStart; j < 9; j++) {

int value = data->grid[i][j];

if (arr[value] != 0) {

return (void \*) 0;

}

else{

arr[value] = 1;

}

}

}

return (void \*) 1;

pthread\_exit(0);

}

//Checks each 3X3 subgrid for numbers from 1-9

void \* subgrid\_worker(void \* param) {

parameters \* data = (parameters \*) param;

int i, j;

int rowStart = data->row;

int colStart = data->column;

int arr[10] = {0};

for (i = rowStart; i < rowStart + 3; i++) {

for (j = colStart; j < colStart + 3; j++) {

int value = data->grid[i][j];

if (arr[value] != 0) {

return (void \*) 0;

}

else{

arr[value] = 1;

}

}

}

return (void \*) 1;

pthread\_exit(0);

}