

Operating Systems Project 2

The Sudoku Puzzle Validator

By HONGGEUN JI [2019042633], Division of Computer Science, HYU ERICA

07.04.21

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

Introduction

This report is dealing with the second assignment given by Dr. HEEKUCK OH. The assignment was to complete the sudoku checking program with the given skeleton source code. By using the thread, we can build the checking system inspecting the part of the sudoku. It is not an efficient way to do this; however, the goal of this project is to implement the actual code that works well with the pthread.

How does it work? - The algorithm of those functions

1. `void check_sudoku(void)`

This function will call every other checking function using multiple threads. It will print out the original sudoku puzzle and check whether the sudoku is valid to the row direction and column direction. Also, it will check every subgrid(9x9) at the end.

2. `void *check_rows(void *arg)`

The starting point of the thread to check every row in the sudoku puzzle. There is an argument when we call this; however, there is nothing to pass for the thread as the thread is required to check all of the rows (no need to specify the location of grid).

3. `void *check_columns(void *arg)`

The starting point of the thread to check every column in the sudoku puzzle. Again, there is an `*arg` parameter, but we do not use this as this thread also needs to check every column.

4. `void *check_subgrid(void *arg)`

The starting point of the thread to check every subgrid in the sudoku puzzle. The number of subgrids will be nine as each of the threads will inspect the given 3x3 subgrids. In this regard, we need nine threads using this function as a starting point.

5. `void *shuffle_sudoku(void *arg)`

This will shuffle the given original sudoku. Nothing to revise and nothing to add. Use this function with a thread to see how it works when we have a not valid sudoku puzzle.

6. `void array_reset(int *arr, int arr_size, int to_what)`

As some part of the 2D array needs to be initialized with the value of `0` or `NULL`, use this function to set every element with the value of `to_what`. Somehow, it is quite similar to the `memset` for a string.

The Codes

```

/*
 * Copyright 2021. Heekuck Oh, all rights reserved
 * 이 프로그램은 한양대학교 ERICA 소프트웨어학부 재학생을 위한 교육용으로 제작되었습니다.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#define TRUE 1
#define FALSE 0

/*
 * 기본 스도쿠 퍼즐
 */
int sudoku[9][9] =
{{6,3,9,8,4,1,2,7,5},{7,2,4,9,5,3,1,6,8},{1,8,5,7,2,6,3,9,4},{2,5,6,1,3,7,4,8,9},{4,9,1,5,8,2,6,3,7},{8,7,3,4,6,9,5,2,1},{5,4,2,3,9,8,7,1,6},{3,1,8,6,7,5,9,4,2},{9,6,7,2,1,4,8,5,3}};

/*
 * valid[0][0], valid[0][1], ..., valid[0][8]: 각 행이 올바른지 1, 아니면 0
 * valid[1][0], valid[1][1], ..., valid[1][8]: 각 열이 올바른지 1, 아니면 0
 * valid[2][0], valid[2][1], ..., valid[2][8]: 각 3x3 그리드가 올바른지 1, 아니면 0
 */
int valid[3][9];

/* structure for passing data to threads
 * this will only be used for subgrid checking
 */
typedef struct {
    int row;
    int col;
    int sudgrid_worker_num;
} location_t;

pthread_mutex_t lock;

// array elements reset to 0
void array_reset(int *arr, int arr_size, int to_what)
{
    int i;
    for(i = 0; i < arr_size; ++i) *(arr+i) = to_what;
}

/*
 * 스도쿠 퍼즐의 각 행이 올바른지 검사한다.

```

```

* 행 번호는 0부터 시작하며, i번 행이 올바르면 valid[0][i]에 1을 기록한다.
*/
void *check_rows(void *arg)
{
    // running indices
    int i, j;

    // total checked elements of the row
    // if the total == 9, it means there is no problem with the given sudoku row
    // this will allow us to check whether the certain row is valid or not in O(1)
    int total = 0;

    // simple hash map to use the value inside of the sudoku row as a index
    int row_hash[9];

    // index for hash (== row value)
    int hash_idx;

    // reset every element to 0
    array_reset(&valid[0][0], 9, 0);
    array_reset(row_hash, 9, 0);

    // check every elements
    for(i = 0; i < 9; ++i) {
        for(j = 0; j < 9; ++j) {
            hash_idx = sudoku[i][j]-1;
            if(!row_hash[hash_idx]) { // if there hasn't showed up the row value,
                row_hash[hash_idx] = 1;
                total++;
            }
            else break; // no need to figure out the others
        }
        array_reset(row_hash, 9, 0);

        if(total == 9) valid[0][i] = 1; // when the internal for loop haven't broken
        total = 0;
    }
}

/*
* 스도쿠 퍼즐의 각 열이 올바른지 검사한다.
* 열 번호는 0부터 시작하며, j번 열이 올바르면 valid[1][j]에 1을 기록한다.
*/
void *check_columns(void *arg)
{
    // running indices
    int i, j;

```

```

// total checked elements of the column
// if the total == 9, it means there is no problem with the given sudoku column
// this will allow us to check whether the certain column is valid or not in O(1)
int total = 0;

// simple hash map to use the value inside of the sudoku column as a index
int col_hash[9];

// index for hash (== column value)
int hash_idx;

// reset every element to 0
array_reset(&valid[1][0], 9, 0);
array_reset(col_hash, 9, 0);

// check every elements
for(i = 0; i < 9; ++i) {
    for(j = 0; j < 9; ++j) {
        hash_idx = sudoku[j][i]-1;
        if(!col_hash[hash_idx]) { // if there hasn't showed up the column value,
            col_hash[hash_idx] = 1;
            total++;
        }
        else break; // no need to figure out the others
    }
    array_reset(col_hash, 9, 0);

    if(total == 9) valid[1][i] = 1; // when the internal for loop haven't broken
    total = 0;
}
}

/*
* 스도쿠 퍼즐의 각 3x3 서브그리드가 올바른지 검사한다.
* 3x3 서브그리드 번호는 0부터 시작하며, 왼쪽에서 오른쪽으로, 위에서 아래로 증가한다.
* k번 서브그리드가 올바르면 valid[2][k]에 1을 기록한다.
*/
void *check_subgrid(void *arg)
{
    location_t* data = (location_t*)arg;

    // running indices
    int i, j;

    // total checked elements of the subgrid
    // if the total == 9, it means there is no problem with the given sudoku subgrid
    // this will allow us to check whether the certain subgrid is valid or not in O(1)
    int total = 0;

```

```

// simple hash map to use the value inside of the sudoku subgrid as a index
int subgrid_hash[9];

// index for hash (== column value)
int hash_idx;

// reset every element to 0
array_reset(subgrid_hash, 9, 0);

// check every elements
for(i = data->row; i < data->row + 3; ++i) {
    for(j = data->col; j < data->col + 3; ++j) {
        hash_idx = sudoku[i][j]-1;
        if(!subgrid_hash[hash_idx]) { // if there hasn't showed up the subgrid value,
            subgrid_hash[hash_idx] = 1;
            total++;
        }
        else break; // no need to figure out the others
    }
}
if(total == 9) valid[2][data->sudgrid_worker_num] = 1; // when the internal for loop haven't been broken
// pthread_mutex_unlock(&lock);
}

/*
* 스토쿠 퍼즐이 올바르게 구성되어 있는지 11개의 스레드를 생성하여 검증한다.
* 한 스레드는 각 행이 올바른지 검사하고, 다른 한 스레드는 각 열이 올바른지 검사한다.
* 9개의 3x3 서브그리드에 대한 검증은 9개의 스레드를 생성하여 동시에 검사한다.
*/
void check_sudoku(void)
{
    // running indices
    int i, j;

    int worker_num = 0; // worker #
    pthread_t* workers;

    // for this project, only 11 threads are required
    if((workers = malloc(sizeof(pthread_t)*11)) == NULL) {
        fprintf(stderr, "malloc error: allocation for workers\n");
        exit(-1);
    }

    /*
    * 검증하기 전에 먼저 스토쿠 퍼즐의 값을 출력한다.
    */
    for (i = 0; i < 9; ++i) {
        for (j = 0; j < 9; ++j)
            printf("%2d", sudoku[i][j]);

```

```

        printf("\n");
    }
    printf("---\n");

    /*
    * 스레드를 생성하여 각 행을 검사하는 check_rows() 함수를 실행한다.
    */
    if (pthread_create(&workers[worker_num++], NULL, check_rows, NULL) != 0) { // worker0 checks the rows
        fprintf(stderr, "pthread_create error: check_rows\n");
        exit(-1);
    }

    /*
    * 스레드를 생성하여 각 열을 검사하는 check_columns() 함수를 실행한다.
    */
    // worker1 checks the columns
    if (pthread_create(&workers[worker_num++], NULL, check_columns, NULL) != 0) {
        fprintf(stderr, "pthread_create error: check_columns\n");
        exit(-1);
    }

    /*
    * 9개의 스레드를 생성하여 각 3x3 서브그리드를 검사하는 check_subgrid() 함수를 실행한다.
    * 3x3 서브그리드의 위치를 식별할 수 있는 값을 함수의 인자로 넘긴다.
    */
    array_reset(&valid[2][0], 9, 0);
    for (i = 0; i < 9; i += 3) {
        for (j = 0; j < 9; j += 3) {
            // malloc a new location_t structure everytime to avoid using a same structure variable
            location_t *data = (location_t*)malloc(sizeof(location_t));
            data->row = i;
            data->col = j;
            data->sudgrid_worker_num = i+(j/3);

            // pass the data saving the row and col to check
            if(pthread_create(&workers[worker_num++], NULL, check_subgrid, data) != 0) {
                fprintf(stderr, "pthread_create error: check_subgrid\n");
                exit(-1);
            }
        }
    }

    /*
    * 11개의 스레드가 종료할 때까지 기다린다.
    */
    for (i = 0; i < 11; ++i) pthread_join(workers[i], NULL);

```

```

/*
 * 각 행에 대한 검증 결과를 출력한다.
 */
printf("ROWS: ");
for (i = 0; i < 9; ++i)
    printf(valid[0][i] == 1 ? "(%d,YES)" : "(%d,NO)", i);
printf("\n");

/*
 * 각 열에 대한 검증 결과를 출력한다.
 */
printf("COLS: ");
for (i = 0; i < 9; ++i)
    printf(valid[1][i] == 1 ? "(%d,YES)" : "(%d,NO)", i);
printf("\n");

/*
 * 각 3x3 서브그리드에 대한 검증 결과를 출력한다.
 */
printf("GRID: ");
for (i = 0; i < 9; ++i)
    printf(valid[2][i] == 1 ? "(%d,YES)" : "(%d,NO)", i);
printf("\n---\n");
}

/*
 * 스도쿠 퍼즐의 값을 3x3 서브그리드 내에서 무작위로 섞는 함수이다.
 */
void *shuffle_sudoku(void *arg)
{
    int i, tmp;
    int grid;
    int row1, row2;
    int col1, col2;

    srand(time(NULL));
    for (i = 0; i < 100; ++i) {
        /*
         * 0부터 8번 사이의 서브그리드 하나를 무작위로 선택한다.
         */
        grid = rand() % 9;
        /*
         * 해당 서브그리드의 좌측 상단 행열 좌표를 계산한다.
         */
        row1 = row2 = (grid/3)*3;
        col1 = col2 = (grid%3)*3;
        /*
         * 해당 서브그리드 내에 있는 임의의 두 위치를 무작위로 선택한다.
         */
    }
}

```



```

    row1 += rand() % 3; col1 += rand() % 3;
    row2 += rand() % 3; col2 += rand() % 3;
    /*
     * 홀수 서브그리드이면 두 위치에 무작위 수로 채우고,
     */
    if (grid & 1) {
        sudoku[row1][col1] = rand() % 8 + 1;
        sudoku[row2][col2] = rand() % 8 + 1;
    }
    /*
     * 짝수 서브그리드이면 두 위치에 있는 값을 맞바꾼다.
     */
    else {
        tmp = sudoku[row1][col1];
        sudoku[row1][col1] = sudoku[row2][col2];
        sudoku[row2][col2] = tmp;
    }
}
pthread_exit(NULL);
}

/*
 * 메인 함수는 위에서 작성한 함수가 올바르게 동작하는지 검사하기 위한 것으로 수정하면 안 된다.
 */
int main(void)
{
    int tmp;
    pthread_t tid;

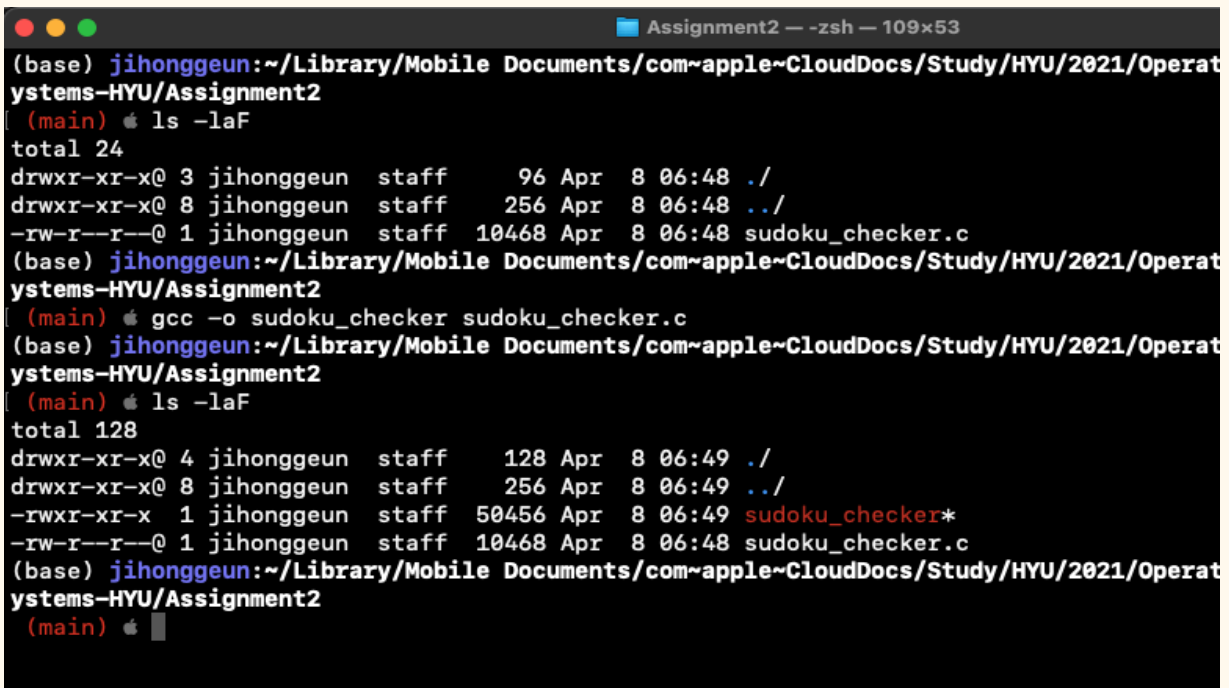
    /*
     * 기본 스도쿠 퍼즐을 출력하고 검증한다.
     */
    check_sudoku();
    /*
     * 기본 퍼즐에서 값 두개를 맞바꾸고 검증해본다.
     */
    tmp = sudoku[5][3]; sudoku[5][3] = sudoku[6][2]; sudoku[6][2] = tmp;
    check_sudoku();
    /*
     * 기본 스도쿠 퍼즐로 다시 바꾼 다음, shuffle_sudoku 스레드를 생성하여 퍼즐을 섞는다.
     */
    tmp = sudoku[5][3]; sudoku[5][3] = sudoku[6][2]; sudoku[6][2] = tmp;
    if (pthread_create(&tid, NULL, shuffle_sudoku, NULL) != 0) {
        fprintf(stderr, "pthread_create error: shuffle_sudoku\n");
        exit(-1);
    }
    /*
     * 무작위로 섞는 중인 스도쿠 퍼즐을 검증해본다.
     */

```

```
check_sudoku();
/*
 * shuffle_sudoku 스레드가 종료될 때까지 기다린다.
 */
pthread_join(tid, NULL);
/*
 * shuffle_sudoku 스레드 종료 후 다시 한 번 스도쿠 퍼즐을 검증해본다.
 */
check_sudoku();
exit(0);
}
```

How to compile and run?

On Mac, you may compile this program without any gcc options.

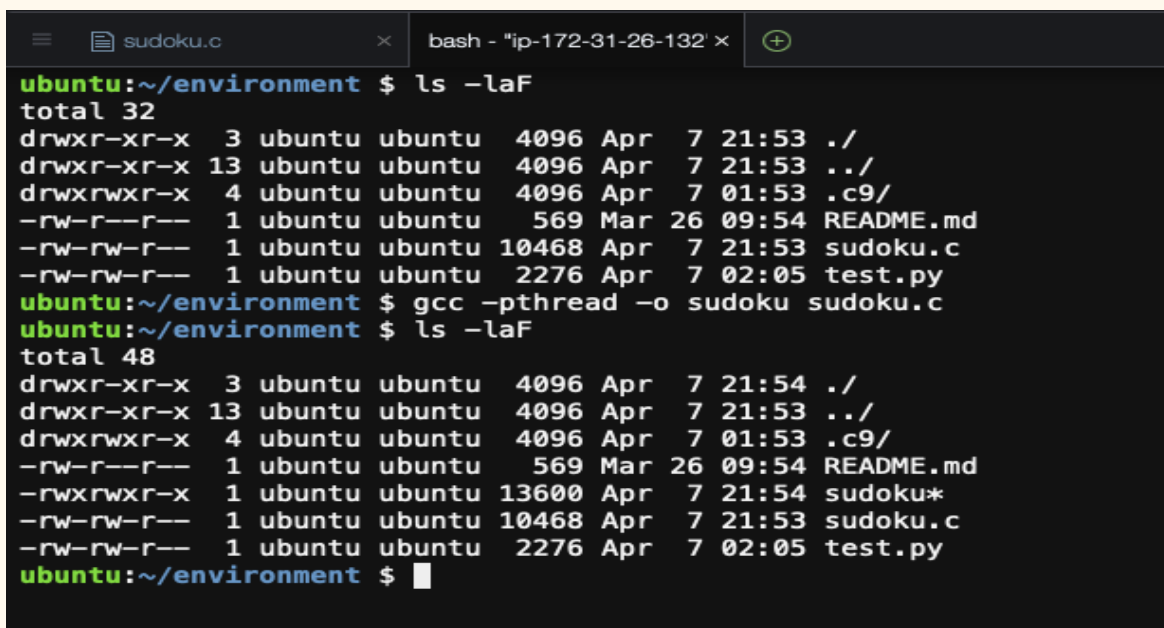


```

Assignment2 - zsh - 109x53
(base) jihonggeun:~/Library/Mobile Documents/com~apple~CloudDocs/Study/HYU/2021/Operat
ystems-HYU/Assignment2
(main) % ls -laF
total 24
drwxr-xr-x@ 3 jihonggeun  staff    96 Apr  8 06:48 ./
drwxr-xr-x@ 8 jihonggeun  staff   256 Apr  8 06:48 ../
-rw-r--r--@ 1 jihonggeun  staff 10468 Apr  8 06:48 sudoku_checker.c
(base) jihonggeun:~/Library/Mobile Documents/com~apple~CloudDocs/Study/HYU/2021/Operat
ystems-HYU/Assignment2
(main) % gcc -o sudoku_checker sudoku_checker.c
(base) jihonggeun:~/Library/Mobile Documents/com~apple~CloudDocs/Study/HYU/2021/Operat
ystems-HYU/Assignment2
(main) % ls -laF
total 128
drwxr-xr-x@ 4 jihonggeun  staff   128 Apr  8 06:49 ./
drwxr-xr-x@ 8 jihonggeun  staff   256 Apr  8 06:49 ../
-rwxr-xr-x  1 jihonggeun  staff 50456 Apr  8 06:49 sudoku_checker*
-rw-r--r--@ 1 jihonggeun  staff 10468 Apr  8 06:48 sudoku_checker.c
(base) jihonggeun:~/Library/Mobile Documents/com~apple~CloudDocs/Study/HYU/2021/Operat
ystems-HYU/Assignment2
(main) %

```

On Linux, you should use *-pthread* option.



```

ubuntu:~/environment $ ls -laF
total 32
drwxr-xr-x  3 ubuntu ubuntu 4096 Apr  7 21:53 ./
drwxr-xr-x 13 ubuntu ubuntu 4096 Apr  7 21:53 ../
drwxrwxr-x  4 ubuntu ubuntu 4096 Apr  7 01:53 .c9/
-rw-r--r--  1 ubuntu ubuntu  569 Mar 26 09:54 README.md
-rw-rw-r--  1 ubuntu ubuntu 10468 Apr  7 21:53 sudoku.c
-rw-rw-r--  1 ubuntu ubuntu  2276 Apr  7 02:05 test.py
ubuntu:~/environment $ gcc -pthread -o sudoku sudoku.c
ubuntu:~/environment $ ls -laF
total 48
drwxr-xr-x  3 ubuntu ubuntu 4096 Apr  7 21:54 ./
drwxr-xr-x 13 ubuntu ubuntu 4096 Apr  7 21:53 ../
drwxrwxr-x  4 ubuntu ubuntu 4096 Apr  7 01:53 .c9/
-rw-r--r--  1 ubuntu ubuntu  569 Mar 26 09:54 README.md
-rwxrwxr-x  1 ubuntu ubuntu 13600 Apr  7 21:54 sudoku*
-rw-rw-r--  1 ubuntu ubuntu 10468 Apr  7 21:53 sudoku.c
-rw-rw-r--  1 ubuntu ubuntu  2276 Apr  7 02:05 test.py
ubuntu:~/environment $

```

Show us the RESULT

```

Assignment2 — -zsh — 109x53
(base) jihonggeun:~/Library/Mobile Documents/com~apple~CloudDocs/Study/HYU
systems-HYU/Assignment2
(main) ./sudoku_checker
6 3 9 8 4 1 2 7 5
7 2 4 9 5 3 1 6 8
1 8 5 7 2 6 3 9 4
2 5 6 1 3 7 4 8 9
4 9 1 5 8 2 6 3 7
8 7 3 4 6 9 5 2 1
5 4 2 3 9 8 7 1 6
3 1 8 6 7 5 9 4 2
9 6 7 2 1 4 8 5 3
-----
ROWS: (0, YES) (1, YES) (2, YES) (3, YES) (4, YES) (5, YES) (6, YES) (7, YES) (8, YES)
COLS: (0, YES) (1, YES) (2, YES) (3, YES) (4, YES) (5, YES) (6, YES) (7, YES) (8, YES)
GRID: (0, YES) (1, YES) (2, YES) (3, YES) (4, YES) (5, YES) (6, YES) (7, YES) (8, YES)
-----
6 3 9 8 4 1 2 7 5
7 2 4 9 5 3 1 6 8
1 8 5 7 2 6 3 9 4
2 5 6 1 3 7 4 8 9
4 9 1 5 8 2 6 3 7
8 7 3 2 6 9 5 2 1
5 4 4 3 9 8 7 1 6
3 1 8 6 7 5 9 4 2
9 6 7 2 1 4 8 5 3
-----
ROWS: (0, YES) (1, YES) (2, YES) (3, YES) (4, YES) (5, NO) (6, NO) (7, YES) (8, YES)
COLS: (0, YES) (1, YES) (2, NO) (3, NO) (4, YES) (5, YES) (6, YES) (7, YES) (8, YES)
GRID: (0, YES) (1, YES) (2, YES) (3, YES) (4, NO) (5, YES) (6, NO) (7, YES) (8, YES)
-----
6 3 9 8 4 1 2 7 5
7 2 4 9 5 3 1 6 8
1 8 5 7 2 6 3 9 4
2 5 6 1 3 7 4 8 9
4 9 1 5 8 2 6 3 7
8 7 3 4 6 9 5 2 1
5 4 2 3 9 8 7 1 6
3 1 8 6 7 5 9 4 2
9 6 7 2 1 4 8 5 3
-----
ROWS: (0, NO) (1, NO) (2, NO) (3, NO) (4, NO) (5, NO) (6, NO) (7, NO) (8, NO)
COLS: (0, NO) (1, NO) (2, NO) (3, NO) (4, NO) (5, NO) (6, NO) (7, NO) (8, NO)
GRID: (0, YES) (1, NO) (2, YES) (3, NO) (4, YES) (5, NO) (6, YES) (7, NO) (8, YES)
-----

```

```
-----  
6 2 7 6 4 4 1 2 3  
5 9 4 4 5 2 9 4 5  
8 1 3 8 1 7 6 7 8  
1 3 4 2 7 6 4 4 5  
3 7 2 9 5 8 2 2 5  
8 4 6 4 3 1 4 6 8  
9 3 1 6 3 7 8 5 4  
4 7 8 7 5 6 3 1 7  
5 2 6 2 1 8 2 6 9  
-----  
ROWS: (0,NO) (1,NO) (2,NO) (3,NO) (4,NO) (5,NO) (6,NO) (7,NO) (8,NO)  
COLS: (0,NO) (1,NO) (2,NO) (3,NO) (4,NO) (5,NO) (6,NO) (7,NO) (8,NO)  
GRID: (0,YES) (1,NO) (2,YES) (3,NO) (4,YES) (5,NO) (6,YES) (7,NO) (8,YES)  
-----
```

**<- After pthread_join()
(now it is a valid result)**