[운영체제] 프로젝트2 : 스도쿠 체크

2017012642 강동진

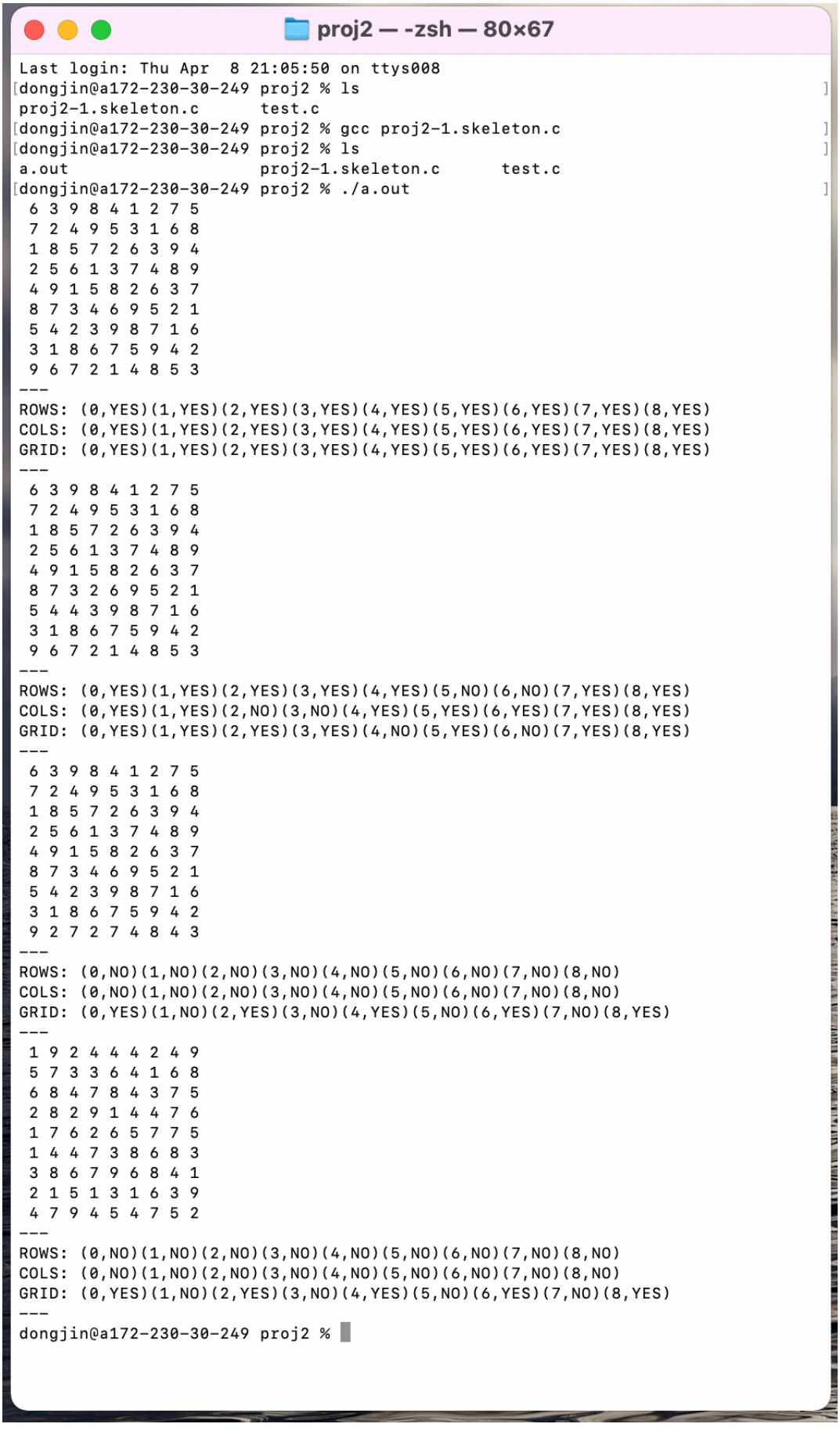
알고리즘

1. `void *check_rows(void *arg)`
    1. 이중 반복문을 통해 row를 한 줄씩 검사한다.
    2. > 1. 각 row의 값을 검사하고, 그 row가 valid한지, invalid한지 판단하는 check_stack[10]을 생성한 후 0으로 초기화 해준다.
    3. > 1. 한 row의 각 값이 1 ~ 9 내에 존재하는지 검사한다.
       > 2. 1 ~ 9 내에 존재하는 경우, 그 값을 인덱스로 하는 check_stack 에 1을 더해준다.
       > 3. 한 row 내에서 값이 2 이상인 check_stack 인덱스가 존재할 경우 invalid라고 판단하여 check_stack[0] 에 1을 저장한다.
       > 4. 위의 과정을 9번 반복한다.
    4. > 2. check_stack[0] 값을 검사하여 valid에 해당하는 값을 넣어준다.
       > 3. 위의 과정을 9번 반복한다.
    5. 스레드를 종료한다.
2. `void *check_columns(void *arg)`
    1. 이중 반복문을 통해 column을 한 줄씩 검사한다.
    2. > 1. 각 column의 값을 검사하고, 그 column이 valid한지, invalid한지 판단하는 check_stack[10]을 생성한 후 0으로 초기화 해준다.
    3. > 1. 한 column의 각 값이 1 ~ 9 내에 존재하는지 검사한다.
       > 2. 1 ~ 9 내에 존재하는 경우, 그 값을 인덱스로 하는 check_stack 에 1을 더해준다.
       > 3. 한 column 내에서 값이 2 이상인 check_stack 인덱스가 존재할 경우 invalid라고 판단하여 check_stack[0] 에 1을 저장한다.
       > 4. 위의 과정을 9번 반복한다.
    4. > 2. check_stack[0] 값을 검사하여 valid에 해당하는 값을 넣어준다.
       > 3. 위의 과정을 9번 반복한다.
    5. 스레드를 종료한다.
3. `void *check_subgrid(void *arg)`
    1. arg를 통해 subgrid의 인덱스를 받아온다.
    2. subgrid가 3x3이고, 전체 스도쿠 판이 9x9 임을 이용해서 해당 subgrid의 검사할 범위를 구한다.
    3. 구한 범위를 이중 반복문을 통해 검사한다.
    4. 검사 방법은 `check_rows` 나 `check_columns` 와 동일하다.

4. `void check_sudoku(void)`
    1. 11개의 스레드를 생성해야 하므로 `thread_t workers[11]` 을 선언해준다.
    2. 스도쿠 판을 출력한다.
    3. `workers[0]` 스레드로 `check_rows` 를 실행하며, arg는 주지 않는다.
    4. `workers[1]` 스레드로 `check_columns` 를 실행하며, arg는 주지 않는다.
    5. `workers[2]` ~ `workers[10]` 까지 반복문을 통해 `check_subgrid` 를 실행하며, arg는 subgrid의 인덱스를 나타내는 값을 넣어준다.
    6. `pthread_join` 과 반복을을 이용해 11개의 스레드가 모두 종료될 때 까지 기다린다.
    7. 검증 결과를 출력한다.

컴파일 과정 및 결과 출력

1. 스도쿠 판 출력 및 검사 결과 정확하게 출력되었다.
2. 제일 좌측 상단을 (0, 0) 이라고 했을 때, (5, 3) = 4 과 (6, 2) = 2 를 교환해 주었다.
    1. 스도쿠 판 출력 결과 성공적으로 교환된 것을 확인 할 수 있다.
    2. 검사 결과도 마찬가지로, ROWS는 5, 6에서 문제가 발생하고, COLS는 3, 2 에서 문제 발생, GRID 또한 각 숫자가 있던, 4, 6에서 문제가 발생한 것을 볼 수 있다.
3. `shuffle_sudoku` 를 통해 스도쿠 판을 섞는 중에 검사해본 결과
    1. 스도쿠 판을 출력할 때, 8번 subgrid에서 4가 2개가 출력된 것을 확인 할 수 있다.
    2. 스도쿠 판을 출력하는 동안, `shuffle_sudoku` 을 모두 수행하므로, 스도쿠 판을 섞은 후에 한 검사와 같은 결과가 출력되는 것을 볼 수 있다.
4. `shuffle_sudoku` 를 통해 스도쿠 판을 섞은 후 검사해본 결과
    1. 짝수 subgrid의 경우 내부 두 숫자의 위치만 교환했기에 subgrid 검사를 통과한 것을 볼 수 있다.
    2. 홀수 subgrid의 경우 내부의 두 숫자의 위치에 무작위 새로운 수를 채워넣었기 때문에 subgrid 검사를 통과하지 못한 것을 확인 할 수 있다.


proj2 — -zsh — 80×67
Last login: Thu Apr 8 21:05:50 on ttys008
[dongjin@a172-230-30-249 proj2 % ls
proj2-1.skeleton.c test.c
[dongjin@a172-230-30-249 proj2 % gcc proj2-1.skeleton.c
[dongjin@a172-230-30-249 proj2 % ls
a.out proj2-1.skeleton.c test.c
[dongjin@a172-230-30-249 proj2 % ./a.out
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 2 7 2 7 4 8 4 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)
---
1 9 2 4 4 4 2 4 9
5 7 3 3 6 4 1 6 8
6 8 4 7 8 4 3 7 5
2 8 2 9 1 4 4 7 6
1 7 6 2 6 5 7 7 5
1 4 4 7 3 8 6 8 3
3 8 6 7 9 6 8 4 1
2 1 5 1 3 1 6 3 9
4 7 9 4 5 4 7 5 2

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)
---
dongjin@a172-230-30-249 proj2 %

소스코드

```c
/*
 * 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>

/*
 * 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}};
int valid[3][9];

void *check_rows(void *arg)
{
    //sudoku[i]; i번째 행(가로)
    for (int i = 0; i < 9; i++)
    {
        // 각 수가 1번만 나왔는지 체크하는 배열.
        // 0번 인덱스는 valid/invalid를 구분하는 flag
        int check_stack[10] = {0};

        for (int j = 0; j < 9; j++)
        {
            // 값이 1 ~ 9 를 벗어난 경우.
            if (sudoku[i][j] <= 0 || sudoku[i][j] > 9)
            {
                // invalid
                check_stack[0] = 1;
                break;
            }

            // 값이 1 ~ 9 내인 경우.
            check_stack[sudoku[i][j]]++;

            // 같은 수가 2번 이상 나오는 경우.
            if (check_stack[sudoku[i][j]] > 1)
            {
                // invalid
                check_stack[0] = 1;
                break;
            }
        }
```

```
        // 해당 row가 valid한지 판단.
        if (check_stack[0])
        {
            // invalid
            valid[0][i] = 0;
        }
        else
        {
            // valid
            valid[0][i] = 1;
        }
    }
    // 종료
    pthread_exit(0);
}

void *check_columns(void *arg)
{
    // sudoku[row][col]
    for (int col = 0; col < 9; col++)
    {
        // 각 수가 1번만 나왔는지 체크하는 배열.
        // 0번 인덱스는 valid/invalid를 구분하는 flag
        int check_stack[10] = {0};

        for (int row = 0; row < 9; row++)
        {
            // 값이 1 ~ 9 를 벗어난 경우.
            if (sudoku[row][col] <= 0 || sudoku[row][col] > 9)
            {
                // invalid
                check_stack[0] = 1;
                break;
            }

            // 값이 1 ~ 9 내인 경우.
            check_stack[sudoku[row][col]]++;

            // 같은 수가 2번 이상 나오는 경우.
            if (check_stack[sudoku[row][col]] > 1)
            {
                check_stack[0] = 1;
                break;
            }
        }
        // 해당 column이 valid한지 판단.
        if (check_stack[0])
        {
            // invalid
```

```
            valid[1][col] = 0;
        }
        else
        {
            // valid
            valid[1][col] = 1;
        }
    }
    // 종료
    pthread_exit(0);
}

void *check_subgrid(void *arg)
{
    // void * 타입의 arg를 int * 타입으로 캐스팅
    int *anker = (int *)arg;

    // args를 통해 subgrid 특정.
    int row_num = *anker / 3;
    int col_num = *anker % 3;
    int row_section = row_num * 3;
    int col_sectioin = col_num * 3;

    // 각 수가 1번만 나왔는지 체크하는 배열.
    // 0번 인덱스는 valid/invalid를 구분하는 flag
    int check_stack[10] = {0};

    // 범위는 row_section ~ row_section+2 까지
    // 범위는 col_section ~ col_section+2 까지 가 check한다.
    for (int i = row_section; i < row_section + 3; i++)
    {
        for (int j = col_sectioin; j < col_sectioin + 3; j++)
        {
            // 값이 ! ~ 9 를 벗어난 경우.
            if (sudoku[i][j] <= 0 || sudoku[i][j] > 9)
            {
                // invalid
                check_stack[0] = 1;
                break;
            }

            // 값이 1 ~ 9 내인 경우.
            check_stack[sudoku[i][j]]++;

            // 같은 수가 2번 이상 나오는 경우.
            if (check_stack[sudoku[i][j]] > 1)
            {
                check_stack[0] = 1;
                break;
```

```
            }
        }
    }
    if (check_stack[0])
    {
        valid[2][*anker] = 0;
    }
    else
    {
        valid[2][*anker] = 1;
    }
    // 종료
    pthread_exit(0);
}

/*
 * 스도쿠 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)
{
    int i, j;

    // 11개의 쓰레드 생성
    pthread_t workers[11];

    /*
     * 검증하기 전에 먼저 스도쿠 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[0], NULL, check_rows, NULL) != 0)
    {
        perror("thread error 0");
    }

    /*
     * 스레드를 생성하여 각 열을 검사하는 check_columns() 함수를 실행한다.
     */
```

```
    if (pthread_create(&workers[1], NULL, check_columns, NULL) != 0)
    {
        perror("thread error 1");
    }

    /*
     * 9개의 스레드를 생성하여 각 3x3 서브그리드를 검사하는 check_subgrid() 함수를 실행한다.
     * 3x3 서브그리드의 위치를 식별할 수 있는 값을 함수의 인자로 넘긴다.
     */
    int thargs[9];
    for (int i = 0; i < 9; i++)
    {
        // 포인터를 사용하여 인자를 전달하기 때문에, 따로 배열을 만들어주었다.
        thargs[i] = i;

        // 몇 번째 subgrid인지에 대한 정보를 인자로 넘겨준다.
        if(pthread_create(&workers[i + 2], NULL, check_subgrid, &thargs[i]) !=
0) {
          perror("thread error subgrid");
        }
    }

    /*
     * 11개의 스레드가 종료할 때까지 기다린다.
     */
    for (int 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 기다란다.
     */
    pthread_join(tid, NULL);
    /*
     * shuffle_sudoku 스레드 종료 후 다시 한 번 스도쿠 퍼즐을 검증해본다.
     */
    check_sudoku();
    exit(0);
}
```