Operating Systems(24802)
project #2

소프트웨어학부
2019037129
신민경

목차.

1. 함수 설명

void *check_rows(void *arg):

- 스도쿠의 각 행 별로 1부터 9까지의 숫자가 한 번씩 등장하는지 확인
- i번째 행이 올바른 구성이라면 valid[0][i]에 1 저장

void *check_columns(void *arg):

- 스도쿠의 각 열 별로 1~9까지의 숫자가 한 번씩 등장하는지 확인
- i번째 열이 올바른 구성이라면 valid[1][i]에 1 저장

void *check_subgrid(void *arg):

- 스도쿠의 3x3 서브그리드에서 1~9까지의 숫자가 한 번씩 등장하는지 확인
- 총 9개의 서브그리드 id를 인자로 받아 올바른 구성이라면 valid[2][id]에 1 저장

void check_sudoku(void):

- 스도쿠 퍼즐이 올바르게 구성되어 있는지 검증하는 함수.
- 11개의 스레드를 생성하여 행, 열, 3x3 서브그리드의 검증을 동시에 진행.
- 11개의 스레드 ID를 저장하기 위해 tid배열을 생성해주고, check_subgrid

함수에 인자를 넘겨주기 위해서 subgrid_id 배열을 만들어 몇 번째 서브그리드인지 저장한다. 여기서 배열을 만드는 이유는 스레드는 동시에 작업을 진행하기 때문에 하나의 변수를 여러 개의 스레드가 공유할 경우 값이 바뀌어 원하는 값이 제대로 전달되지 않기 때문이다. 따라서 스레드 별로 사용할 변수를 만들어주기 위해서 배열을 만들어 서브그리드 ID를 저장해 넘겨준다.

2. 프로그램 소스코드

```
/*
 * 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}};

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

/*
 * 스도쿠 퍼즐의 각 행이 올바른지 검사한다.
 * 행 번호는 0부터 시작하며, i번 행이 올바르면 valid[0][i]에 1을 기록한다.
 */
void *check_rows(void *arg)
{
    for(int i=0; i<9; i++){
        int cnt[10] = {0,};
        valid[0][i] = 1;

        for(int j=0; j<9; j++){ // columns
```

```
            cnt[sudoku[i][j]]++; // 각 숫자가 몇 번 나왔는지 카운트
        }

        for(int j=1; j<=9; j++){
            if(cnt[j] != 1){ // 1부터 9까지의 숫자가 한 번씩 등장한게 아니라면
                valid[0][i] = 0; // 비정상으로 체크하고 종료
                break;
            }
        }
    }
    pthread_exit(NULL); // 스레드 종료
}

/*
 * 스도쿠 퍼즐의 각 열이 올바른지 검사한다.
 * 열 번호는 0부터 시작하며, j번 열이 올바르면 valid[1][j]에 1을 기록한다.
 */
void *check_columns(void *arg)
{
    for(int i=0; i<9; i++){
        int cnt[10] = {0,};
        valid[1][i] = 1;

        for(int j=0; j<9; j++){ // rows
            cnt[sudoku[j][i]]++; // 각 숫자가 몇 번 나왔는지 카운트
        }
        for(int j=1; j<=9; j++){
            if(cnt[j] != 1){ // 1부터 9까지의 숫자가 한 번씩 등장한게 아니라면
                valid[1][i] = 0; // 비정상으로 체크하고 종료
                break;
            }
        }
    }
    pthread_exit(NULL); // 스레드 종료
}

/*
 * 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)
{
    int id = *(int *)arg; // subgrid ID로 0~8 숫자

    int i = 3*(id/3); // subgrid의 시작 행 번호
    int j = 3*(id%3); // subgrid의 시작 열 번호
    int cnt[10] = {0,};

    for(int di=0; di<3; di++){
```

```
            for(int dj=0; dj<3; dj++){
                cnt[sudoku[i+di][j+dj]]++; // subgrid의 숫자 등장 횟수 카운트
            }
        }

        valid[2][id] = 1;
        for(int n=1; n<=9; n++){
            if(cnt[n] != 1){ // 1~9까지의 숫자가 한 번씩 등장한게 아니라면
                valid[2][id] = 0; // 비정상으로 체크하고 종료
                break;
            }
        }

        pthread_exit(NULL); // 스레드 종료
}

/*
 * 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;

    /*
     * 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.
     */
    pthread_t tid[11];

    if (pthread_create(&tid[9], NULL, check_rows, NULL) != 0) {
        fprintf(stderr, "pthread_create error: check_rows\n");
        exit(-1);
    }

    /*
     * 스레드를 생성하여 각 열을 검사하는 check_columns() 함수를 실행한다.
     */
    if (pthread_create(&tid[10], 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.
     */
    int subgrid_id[9];
    for(int i=0; i<9; i++){
        subgrid_id[i] = i;
        if (pthread create(&tid[i], NULL, check subgrid, &subgrid id[i]) != 0)
        {
            fprintf(stderr, "pthread_create error: check_subgrid %d\n",i);
            exit(-1);
        }
    }
    /*
     * 11개의 스레드가 종료할 때까지 기다린다.
     */
    for(int i=0; i<11; i++)
        pthread_join(tid[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 섞
는다.
     */
    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);
}
```

3. 컴파일 과정

```
hihiroo@hihiroo-VirtualBox:~/Desktop$ gcc -v proj2-1.skeleton.c -lpthread
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1~18.04' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs
--enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-7 --program-prefi
x=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libd
ir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstd
cxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-system-zlib --wi
th-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32
,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build
=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/7/cc1 -quiet -v -imultiarch x86_64-linux-gnu proj2-1.skeleton.c -quiet -dumpbase proj2-1.skeleton.c -mtune=g
eneric -march=x86-64 -auxbase proj2-1.skeleton -version -fstack-protector-strong -Wformat -Wformat-security -o /tmp/cck7eeFX.s
GNU C11 (Ubuntu 7.5.0-3ubuntu1~18.04) version 7.5.0 (x86_64-linux-gnu)
        compiled by GNU C version 7.5.0, GMP version 6.1.2, MPFR version 4.0.1, MPC version 1.1.0, isl version isl-0.19-GMP

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/local/include/x86_64-linux-gnu"
ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/7/../../../../x86_64-linux-gnu/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/x86_64-linux-gnu/7/include
 /usr/local/include
 /usr/lib/gcc/x86_64-linux-gnu/7/include-fixed
 /usr/include/x86_64-linux-gnu
 /usr/include
End of search list.
GNU C11 (Ubuntu 7.5.0-3ubuntu1~18.04) version 7.5.0 (x86_64-linux-gnu)
        compiled by GNU C version 7.5.0, GMP version 6.1.2, MPFR version 4.0.1, MPC version 1.1.0, isl version isl-0.19-GMP

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: b62ed4a2880cd4159476ea8293b72fa8
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64'
 as -v --64 -o /tmp/ccqj2H83.o /tmp/cck7eeFX.s
GNU assembler version 2.30 (x86_64-linux-gnu) using BFD version (GNU Binutils for Ubuntu) 2.30
COMPILER_PATH=/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-gn
u/7/:/usr/lib/gcc/x86_64-linux-gnu/
LIBRARY_PATH=/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-gnu/7/..
/../../../lib/:/lib/x86_64-linux-gnu/:/lib/../lib/:/usr/lib/x86_64-linux-gnu/:/usr/lib/../lib/:/usr/lib/gcc/x86_64-linux-gnu/7/../../../:/l
ib/:/usr/lib/
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/7/collect2 -plugin /usr/lib/gcc/x86_64-linux-gnu/7/liblto_plugin.so -plugin-opt=/usr/lib/gcc/x86_64-linux-gn
u/7/lto-wrapper -plugin-opt=-fresolution=/tmp/cc4CMJDa.res -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s -plugin-opt=-p
ass-through=-lc -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s --build-id --eh-frame-hdr -m elf_x86_64 --hash-style=gnu
--as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie -z now -z relro /usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/Scrt
1.o /usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/7/crtbeginS.o -L/usr/lib/gcc/x86_64-linu
x-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/7/../../../../lib -L/lib/x86_64-linux-g
nu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/7/../../.. /tmp/ccqj2H83.o -lpthread -lgcc -
-push-state --as-needed -lgcc_s --pop-state -lc -lgcc --push-state --as-needed -lgcc_s --pop-state /usr/lib/gcc/x86_64-linux-gnu/7/crtendS.
o /usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/crtn.o
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64'
hihiroo@hihiroo-VirtualBox:~/Desktop$ ./a.out
```

컴파일 화면

4. 실행 결과물 및 설명

```
hihiroo@hihiroo-VirtualBox:~/Desktop$ ./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 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 4 9 8 1 1 2 1 5
 3 2 7 1 5 2 3 4 6
 5 8 1 3 1 2 8 9 7
 3 4 6 9 5 1 7 4 3
 1 1 3 4 6 7 4 1 7
 4 8 8 3 2 8 5 7 1
 5 3 2 5 1 2 2 1 4
 4 8 6 1 5 8 7 9 6
 7 9 1 2 8 2 5 3 8
---
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)
---
hihiroo@hihiroo-VirtualBox:~/Desktop$
```

실행 결과 화면

첫 번째 check_sudoku는 모두 올바른 구성이기 때문에 YES를 출력하고 있다.

두 번째 check_sudoku는 값 두 개를 맞바꿨기 때문에 바뀐 값이 있던 행과 열이 NO를 출력하고 있다.

세 번째 check_sudoku는 shuffle_sudoku를 스레드를 생성하여 실행하고 바로 check_sudoku를 실행했기 때문에 섞이는 도중에 스도쿠 퍼즐을 검증한다. 따라서 shuffle_sudoku를 먼저 호출했음에도 원래의 스도쿠가 그대로 출력이 된 이유는 shuffle_sudoku 스레드보다 check_sudoku가 더 빠르게 진행되었기 때문이다. 실제로 컴퓨터 환경에 따라서 같은 코드를 실행해봤을 때 다른 결과가 나온 것을 확인할 수 있었다.

네 번째 check_sudoku는 shuffle_sudoku 스레드가 종료된 이후에 검증을 했으므로 바뀐 스도쿠와 그에 따른 검증 결과가 출력되고 있다.