[check_rows]

```
/*
 * 스도쿠 퍼즐의 각 행이 올바른지 검사한다.
 * 행 번호는 0부터 시작하며, i번 행이 올바르면 valid[0][i]에 true를 기록한다.
 */
void *check_rows(void *arg)
{
    for(int i = 0; i < 9; i++) //행 선택
    {
            bool isRow = true;
            int rowValid = 0; // 행 확인용 flag
            for(int j = 0; j < 9; j++) //행 순회
            {
                    if(rowValid & (1 << sudoku[i][j])) // 구간에 이미 있는 값일 경우 false
                    {
                            isRow = false;
                            break;
                    }
                    rowValid |= (1 << sudoku[i][j]); // 값 flag 추가
            }
            valid[0][i] = isRow;
    }
    return ;
}
```

bool isRow로 해당 열이 올바른지 체크합니다.

rowValid는 비트마스킹을 이용해 flag에 역할을 합니다. 이중 for문을 이용해 행을 선택하고, 순회하면서 해당 행에 숫자를 확인하고 이미 이전에 있던 숫자라면 isRow를 false로 만들고 종료합니다. 만약 이전에 없던 숫자라면 flag에 추가를 한 후 계속 순회합니다.

[check_columns]

```
/*
 * 스도쿠 퍼즐의 각 열이 올바른지 검사한다.
```

```
 * 열 번호는 0부터 시작하며, j번 열이 올바르면 valid[1][j]에 true를 기록한다.
 */
void *check_columns(void *arg)
{
    for(int i = 0; i < 9; i++) // 열 순회
    {
            bool isCol = true;
            int colValid = 0; // 열 확인용 flag
            for(int j = 0; j < 9; j++) //열 선택
            {
                    if(colValid & ( 1 << sudoku[j][i])) // 구간에 이미 있는 값일 경우
                    {
                            isCol = false;
                            break;
                    }
                    colValid |= ( 1 << sudoku[j][i]); // 값 flag 추가
            }
            valid[1][i] = isCol;
    }
    return ;
}
```

이전 코드와 동일하게 작동합니다. 열을 선택한 후, 해당 열을 순회하면서 flag를 체크해 만약에 이미 확인했던 숫자라면 false를, 아니라면 flag에 추가하고 남은 열을 마저 순회합니다.

[check_subgrid]

```
/*
```

```
 * 스도쿠 퍼즐의 각 3x3 서브그리드가 올바른지 검사한다.
 * 3x3 서브그리드 번호는 0부터 시작하며, 왼쪽에서 오른쪽으로, 위에서 아래로 증가한다.
 * k번 서브그리드가 올바르면 valid[2][k]에 true를 기록한다.
 */
void *check_subgrid(void *arg)
{
        bool isSubGrid = true;
        int subGridValid = 0;

        int *ptrCast = (int *)arg; // subGrid 정보
        int rowPos = ptrCast[0]; // subGrid 좌측 상단 행
        int colPos = ptrCast[1]; // subGrid 좌측 상단 열
        int subGridNum = ptrCast[2]; // subGrid 번호

        for(int r = 0; r < 3; r++)
        {
                for(int c = 0; c < 3; c++)
                {
                        if(subGridValid & ( 1 << sudoku[rowPos + r][colPos + c])) // subGrid
순회하며 해당 번호 이미 count했을 시
                        {
                                isSubGrid = false;
                                break;
                        }
                        subGridValid |= ( 1 << sudoku[rowPos + r][colPos + c]); //subGrid
번호 추가
        }
        valid[2][subGridNum] = isSubGrid;
    return ;
}
```

void check_sudoku(void)에서 subgrid에 정보를 받아옵니다. 정보는 총 3가지로, subgrid 좌측 상단 번호에 행, 열정보, 그리고 subgrid번호를 가져옵니다. 그냥 subgrid 번호만 가져온 후 check_subgrid에서 정보를 계산하는게 편하겠지만 한 번 여러 개 테스트용으로 진행했습니다.

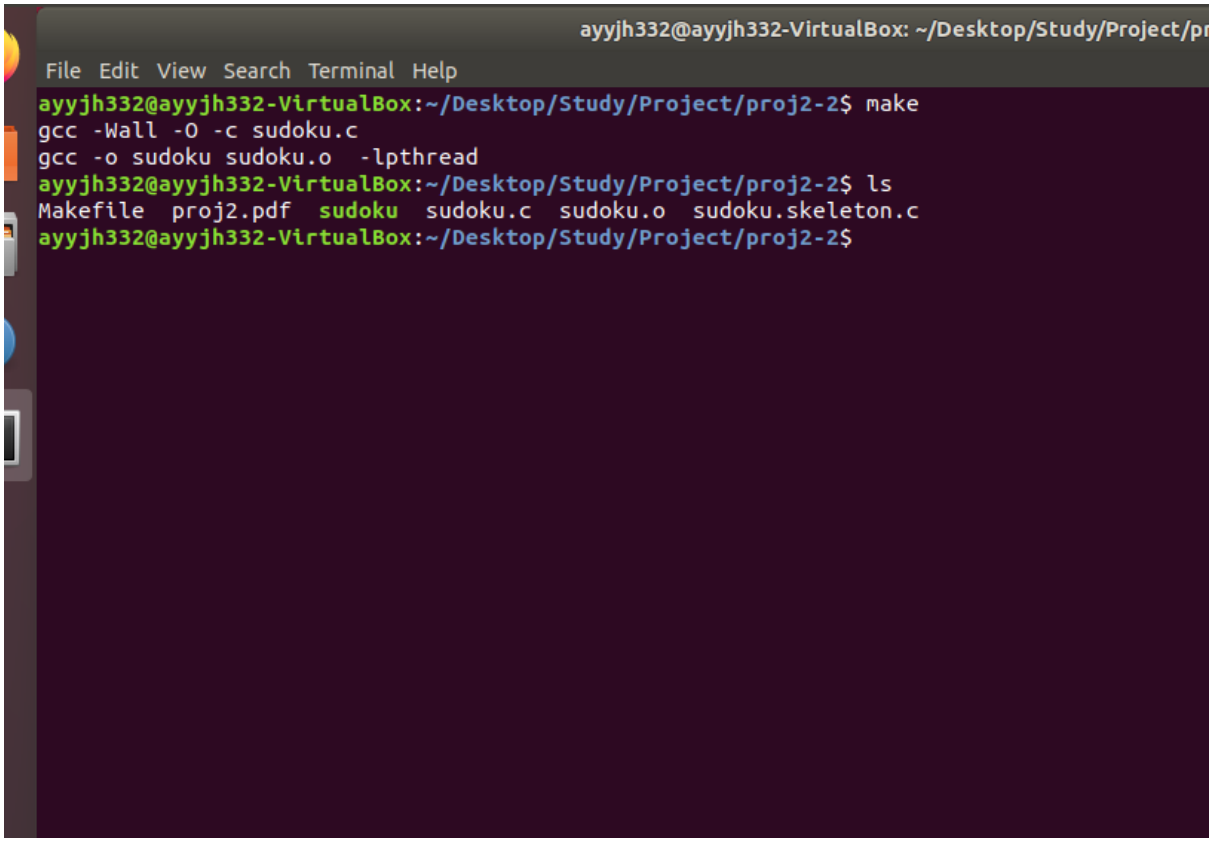
해당 정보들을 가지고 subgrid를 순회하며 해당 번호가 이미 bit flag내에 있다면 false, 없다면 flag에 추가한 후 마저 순회합니다.

[check_sudoku]

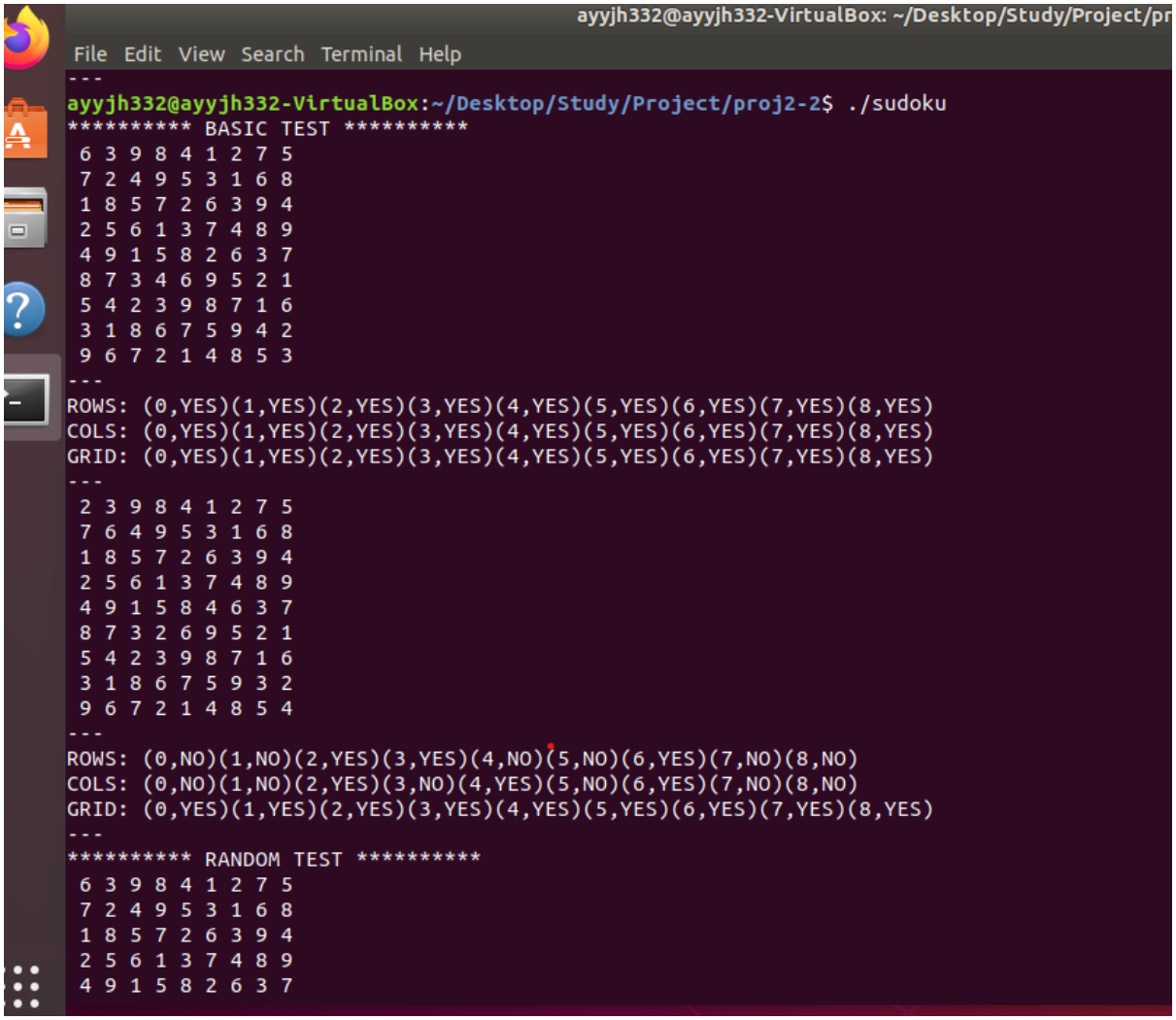
```
/*
 * 스도쿠 퍼즐이 올바르게 구성되어 있는지 11개의 스레드를 생성하여 검증한다.
 * 한 스레드는 각 행이 올바른지 검사하고, 다른 한 스레드는 각 열이 올바른지 검사한다.
 * 9개의 3x3 서브그리드에 대한 검증은 9개의 스레드를 생성하여 동시에 검사한다.
 */
void check_sudoku(void)
{
    // ... 이전 코드
       /*
     * 스레드를 생성하여 각 행을 검사하는 check_rows() 함수를 실행한다.
     */
    pthread_t row_thread;
    pthread_create(&row_thread, NULL, check_rows, NULL);
    /*
     * 스레드를 생성하여 각 열을 검사하는 check_columns() 함수를 실행한다.
     */
    pthread_t col_thread;
    pthread_create(&col_thread, NULL, check_columns, NULL);
    /*
     * 9개의 스레드를 생성하여 각 3x3 서브그리드를 검사하는 check_subgrid() 함수를 실행한다.
     * 3x3 서브그리드의 위치를 식별할 수 있는 값을 함수의 인자로 넘긴다.
     */
    pthread_t sub_grid_thread[9];
    for(int i = 0; i < 9; i++) // subGrid 번호
    {
            int *pos = malloc(sizeof(int) *3);
            pos[0] = (i/3)*3; // 행 위치
            pos[1] = (i%3)*3; // 열 위치
            pos[2] = i; //subGrid 번호
            pthread_create(&sub_grid_thread[i], NULL, check_subgrid, (void *) pos); // 9개의
스레드 생성
    }
    /*
     * 11개의 스레드가 종료할 때까지 기다린다.
     */
    pthread_join(row_thread, NULL);
    pthread_join(col_thread, NULL);
    for(int i = 0; i < 9; i++)
    {
            if(pthread_join(sub_grid_thread[i], NULL) != 0)
            {
                    fprintf(stderr, "Failed to join subgrid");
                    exit(1);
            }
    }
// ... 이후코드
}
```

행/열에 경우 스레드를 1개 생성 후 함수를 실행해줍니다. subgrid의 경우 스레드 9개를 생성한 후, 각 스레드에 subgrid에 시작위치를 포함하여 함수를 실행시킵니다. 이후 총 11개의 스레드가 종료될 때 까지 대기합니다.

-컴파일 과정



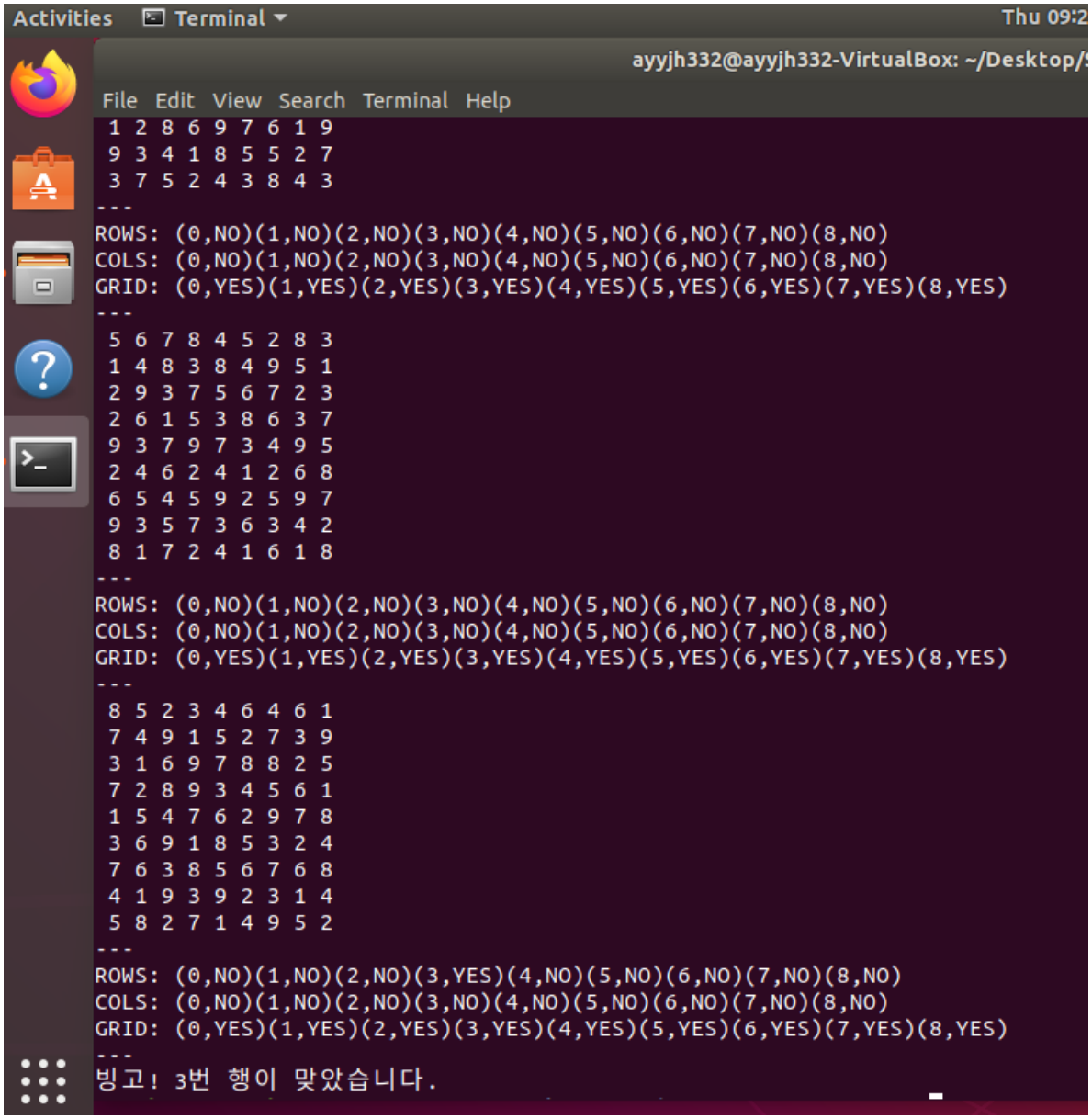
2. 실행 결과물 상세 설명 및 캡쳐



Basic Test의 경우 이미 완성된 스도쿠를 테스트하는 부분이며 정상 작동됩니다.

Basic Test의 2번째에 경우 몇몇 서브그리드 내에서 스왑하여 테스트를 진행합니다.

이후 스도쿠를 계속 셔플하는 중에 스도쿠를 체크하는 RANDOM TEST로 넘어갑니다.



앞서 말한 것과 같이 Basic Test 이후 계속 셔플이 진행되는 중 5번 스도쿠를 체크한 이후, 셔플이 끝나고 마지막 체크를 진행합니다.

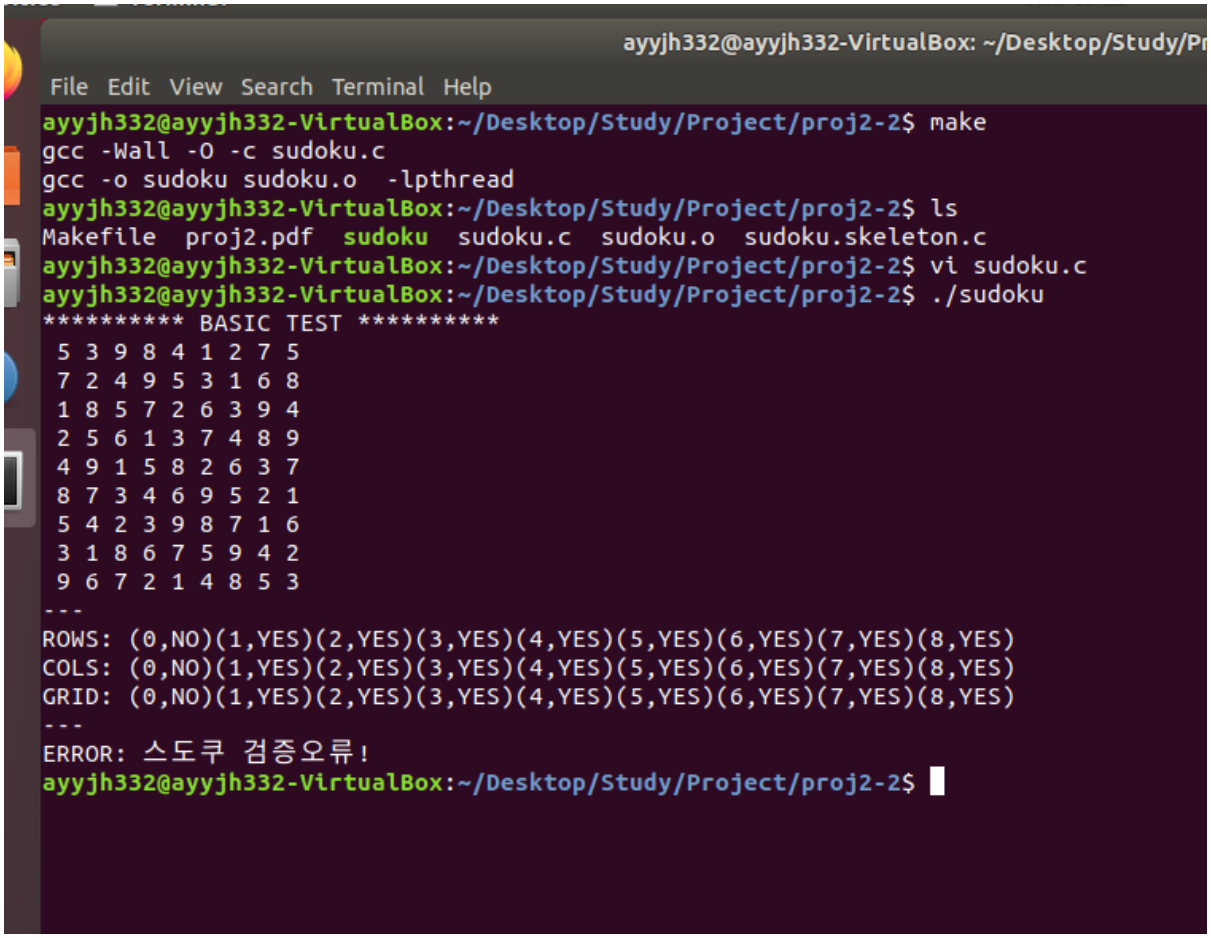
이 때, 셔플 중 발생하는 스도쿠 체크는 공유자원이 동기화된 상태가 아니기에 불일치가 있을 수 있습니다. 셔플이 진행중에 스도쿠 체크에서 출력을 하기에 같은 서브그리드 내에서 같은 숫자가 2번 출력되는 것이 확인되는데, 2번째 스도쿠 체크에 2번째 서브그리드의 경우 8과 4, 5가 각각 2개씩 들어있는 것을 볼 수 있습니다. 하지만 check_subgrid 함수에 의해서는 모든 서브그리드가 옳다는 출력이 나왔습니다.

해당 이유는 제가 생각하기로는 출력을 하는 와중에도 셔플 스레드는 계속 셔플을 돌리고 있기에

2번째 스도쿠의 경우 2번째 서브그리드에 가장 처음인 8이 출력된 후, 다음 행을 출력할 때, 8이 스왑되어 다시 출력된 것으로 보입니다.

여기서 의문이 생겼습니다. 그렇다면, 어째서 check_sudoku로 인하여 스도쿠를 확인할 때 check_sudoku에서 subgrid는 모두 True가 된 것인가? 로직 적으로는 당연히 True가 나오는 것이 맞으나, 공유자원이 동기화가 되어있지 않으니 중간에 스왑이 되면서 서브그리드가 깨져야하는 것 아닌가 라는 생각이 들었습니다.

먼저 첫번째 가능성으로 비트 플래그로 체크하는 것이 문제가 있는지였습니다.



기본 스도쿠에 첫번째 숫자를 5로 바꾸고 테스트해본 결과 바로 오류가 나왔으며, 몇 번 더 테스트를 진행하면서 스도쿠를 체크하는 함수는 문제가 없다 생각했습니다.

두 번째 가능성으로 단순히 check_subgrid가 출력보다 빠르기에, 출력은 셔플된 늦은 결과를 받아오지만, check_subgrid는 셔플과 비슷하거나 더 빨라 섞이기 전에 결과 도출이 되는 것으로 생각했습니다.

마지막 최종 결과는 각 서브그리드는 서브그리드 내에서만 스왑이 됐기에 모두 True이며, 반복 노

력에 끝에 3번 행의 빙고를 찾았습니다.

3. 과제를 수행하면서 경험한 문제점과 느낀점

과제 진행을 하면서 겪은 문제점은 왜 출력은 중복 숫자가 나오는데, 서브그리드가 모두 YES로 뜨는지 였습니다. 최종적인 결론은 출력이 속도가 더 느리기 때문에로 생각을 마무리했으나 이는 추측일 뿐 아쉬움이 남는 것 같습니다. 추후 다른 학우와 얘기도 해보고 시간을 들여 조금 더 테스트를 해보면서 확인이 필요할 것 같습니다.

느낀점에서는 멀티 스레드를 사용할 때 공유 자원이 동기화가 안되어 있다면 큰 문제가 있을 것 같았습니다. 지금 스도쿠의 경우와 같이 공유 자원을 단순히 읽어오기만 하는 것 뿐만이 아니라, 각 스레드가 공유 자원을 수정까지 할 수 있다면 그것에 대한 위상 관계가 매우 중요할 것 같습니다.

참고자료 : https://github.com/cfperea/multithreaded-sudoku/blob/master/main.c