

운영체제론 프로젝트2

소프트웨어학부

컴퓨터전공

2019040519 김태형

1. 함수 설명

- `void *check_rows(void *arg)`

스도쿠 퍼즐의 9개의 행이 올바른지 검사하는 함수.

총 두개의 for loop를 사용하며, outer for loop의 변수 `i`는 스도쿠 퍼즐의 행을 나타내고, inner for loop의 변수 `j`는 스도쿠 퍼즐의 열을 나타낸다. 현재 숫자가 이전에 나왔는지 체크하기 위한 배열 `before`와, for loop를 빠져나와 현재 행의 validity를 판단하는 변수 `is_valid`를 이용해 `i`번째 행의 validity를 판정한다. `i` 값이 증가할 때 마다 초기화하여 모든 행에 대해 사용할 수 있도록 하였다.

현재 숫자가 valid한 경우, 현재 숫자 - 1을 `before` 배열의 index로 사용하여 1을 증가시킨다. 현재 숫자가 valid하지 않은 경우, `is_valid`의 값을 0으로 변경하고, break문으로 inner for loop를 빠져나온다.

이후, `is_valid` 값을 이용해 `i`번째 행이 올바르다면 `valid[0][i]`에 1을 기록하고, 올바르지 않다면 0을 기록한다. 모든 행에 대해 해당 작업들을 반복하고, outer for loop를 빠져나온 뒤, `pthread_exit()` 명령으로 스레드를 종료한다.

```
/*
 * 스도쿠 퍼즐의 각 행이 올바른지 검사한다.
 * 행 번호는 0부터 시작하며, i번 행이 올바르면 valid[0][i]에 1을 기록한다.
 */
void *check_rows(void *arg)
{
    // 여기를 완성하세요
    for (int i = 0; i < 9; i++){
        int before[9] = {0,}; // 현재 숫자가 이전에 나왔는지를 체크하는 배열
        int is_valid = 1; // for loop를 빠져나온 뒤, i번째 행이 올바른지 판단할 때 사용하는 변수
        for (int j = 0; j < 9; j++){
            int num = sudoku[i][j];
            if (num > 9 || num < 1 || before[num - 1] > 0){ // 현재 숫자가 valid 하지 않음
                // is_valid를 0으로 변경 후, for loop 빠져나옴
                is_valid = 0;
                break;
            }
            else {
                before[num - 1]++; // 현재 숫자를 배열의 index로 사용
            }
        }
        if(is_valid == 1) // i번째 행이 올바름
            valid[0][i] = 1;
        else // i번째 행이 올바르지 않음
            valid[0][i] = 0;
    }
    // 모든 행 체크 완료
    pthread_exit(NULL);
}
```

- void *check_columns(void *arg)

스도쿠 퍼즐의 9개의 열이 올바른지 검사하는 함수.

총 두개의 for loop를 사용하며, outer for loop의 변수 j는 스도쿠 퍼즐의 열을 나타내고, inner for loop의 변수 i는 스도쿠 퍼즐의 행을 나타낸다. 현재 숫자가 이전에 나왔는지 체크하기 위한 배열 before와, for loop를 빠져나와 현재 열의 validity를 판단할 때 사용하는 변수 is_valid를 이용해 j번째 열의 validity를 판정한다. j 값이 증가할 때 마다 초기화하여 모든 열에 대해 사용할 수 있도록 하였다.

현재 숫자가 valid한 경우, 현재 숫자 - 1을 before 배열의 index로 사용하여 1을 증가시킨다. 현재 숫자가 valid하지 않은 경우, is_valid의 값을 0으로 변경하고, break문으로 inner for loop를 빠져나온다.

이후, is_valid 값을 이용해 i번째 열이 올바르다면 valid[1][j]에 1을 기록하고, 올바르지 않다면 0을 기록한다. 모든 열에 대해 해당 작업을 반복하고, outer for loop를 빠져나온 뒤 pthread_exit()명령으로 스레드를 종료한다.

```
/*
 * 스도쿠 퍼즐의 각 열이 올바른지 검사한다.
 * 열 번호는 0부터 시작하며, j번 열이 올바르면 valid[1][j]에 1을 기록한다.
 */
void *check_columns(void *arg)
{
    // 여기를 완성하세요
    for (int j = 0; j < 9; j++){
        int before[9] = {0,}; // 현재 숫자가 이전에 나왔는지를 체크하는 배열
        int is_valid = 1; // for loop를 빠져나온 뒤, j번째 열이 올바른지 판단할 때 사용하는 변수
        for (int i = 0; i < 9; i++){
            int num = sudoku[i][j];
            if (num > 9 || num < 1 || before[num - 1] > 0){ // 현재 숫자가 valid 하지 않음
                // is_valid를 0으로 변경 후, for loop 빠져나옴
                is_valid = 0;
                break;
            }
            else {
                before[num - 1]++; // 현재 숫자를 배열의 index로 사용
            }
        }
        if(is_valid == 1) // j번째 열이 올바름
            valid[1][j] = 1;
        else // j번째 열이 올바르지 않음
            valid[1][j] = 0;
    }
    // 모든 열 체크 완료
    pthread_exit(NULL);
}
```

- void *check_subgrid (void *arg)

스도쿠 퍼즐의 k번 3x3 서브그리드가 올바른지 검사하는 함수.

매개변수 arg를 통해 현재 검사하려는 서브그리드의 시작 row값, 시작 col값, 서브그리드의 index 값을 전달받는다. 총 두개의 for loop를 사용하며, outer for loop는 row부터 row+2까지 반복하고, inner for loop는 col부터 col+2까지 반복하여, 스도쿠 퍼즐의 k번째 3x3 서브그리드를 순회한다.

스도쿠의 9개의 행에 대해 검사하는 check_rows()와 9개의 열에 대해 check_columns()와는 달리, validity의 판정 범위가 스도쿠 퍼즐의 k번 3x3 서브그리드이므로, before 배열을 outer for loop의 바깥쪽에 선언하여 행이 증가함에 따라 초기화되지 않게끔 하였고, valid하지 않음이 확인된 즉시, valid[2][k]에 0을 기록하고, pthread_exit() 명령으로 스레드를 종료한다.

outer for loop를 정상적으로 빠져나온 경우, 해당 서브그리드가 올바르다는 의미이므로, valid[2][k]에 1을 기록하고, pthread_exit() 명령으로 스레드를 종료한다.

```
// 서브그리드를 식별하기 위한 구조체
typedef struct{
    int idx;
    int row;
    int col;
} subgrid_args;
```

▲ subgrid_args 구조체

```
/*
 * 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)
{
    // 여기를 완성하세요
    subgrid_args *args = (subgrid_args*) arg;
    int row = args -> row;
    int col = args -> col;
    int k = args -> idx; // 서브그리드의 index
    int before[9] = {0,}; // 현재 숫자가 이전에 나왔는지를 체크하는 배열
    for (int i = row; i < row + 3; i++){
        for (int j = col; j < col + 3; j++){
            int num = sudoku[i][j];
            if (num > 9 || num < 1 || before[num - 1] > 0){ // 현재 숫자가 valid 하지 않음
                // k번 서브그리드가 올바르지 않음 -> valid[2][k]에 0을 기록후, 스레드 종료
                valid[2][k] = 0;
                pthread_exit(NULL);
            }
            else {
                before[num - 1]++; // 현재 숫자를 배열의 index로 사용
            }
        }
    }
    // k번 서브그리드가 올바름
    valid[2][k] = 1;
    pthread_exit(NULL);
}
```

- void check_sudoku (void)

스도쿠 퍼즐이 올바르게 구성되어 있는지 총 11개의 스레드를 생성하여 검증하는 함수.

pthread_create(&rowtid, NULL, check_rows, NULL) 명령어를 이용해 스도쿠의 각 행을 검사하는 check_rows 스레드를 실행한다. 스도쿠의 모든 행이 검사 대상이므로, 매개변수는 전달하지 않는다.

pthread_create(&coltid, NULL, check_columns, NULL) 명령어를 이용해 스도쿠의 각 열을 검사하는 check_columns 스레드를 실행한다. 스도쿠의 모든 열이 검사 대상이므로, 매개변수는 전달하지 않는다.

이중 for loop를 돌며, subgrid_args 타입 구조체 arg에 k번 서브그리드의 index, row 시작 값, col 시작 값을 저장하고, pthread_create(&subgridtid[k++], NULL, check_subgrid, arg) 명령어를 통해 k번 서브그리드의 정보를 담은 구조체를 check_subgrid 스레드의 매개변수로 전달한다. 총 9개의 check_subgrid 스레드가 실행된다.

```
void check_sudoku(void)
{
    int i, j;
    int gid[9];
    pthread_t rowtid, coltid, subgridtid[9];

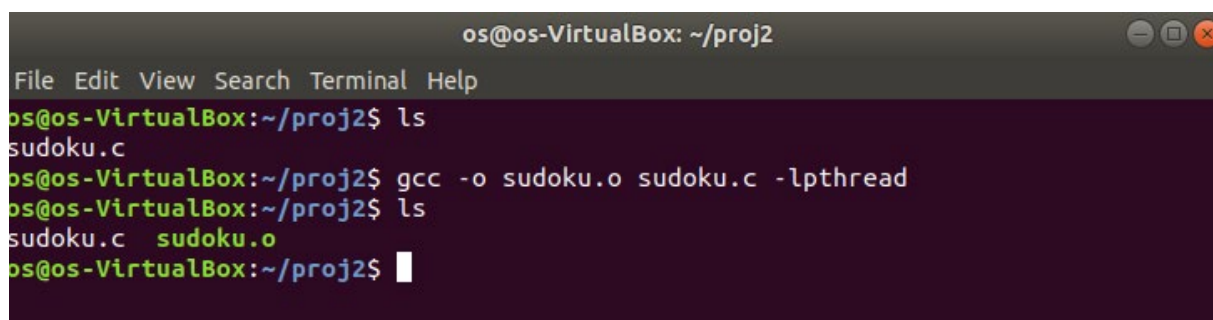
    /*
     * 검증하기 전에 먼저 스도쿠 퍼즐의 값을 출력한다.
     */
    for (i = 0; i < 9; ++i) {
        for (j = 0; j < 9; ++j)
            printf("%2d", sudoku[i][j]);
        printf("\n");
    }
    printf("---\n");
    /*
     * 스레드를 생성하여 각 행을 검사하는 check_rows() 함수를 실행한다.
     */
    // 여기를 완성하세요
    pthread_create(&rowtid, NULL, check_rows, NULL);
    /*
     * 스레드를 생성하여 각 열을 검사하는 check_columns() 함수를 실행한다.
     */
    // 여기를 완성하세요
    pthread_create(&coltid, 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.
     */
    // 여기를 완성하세요
    int k = 0; // 서브그리드의 index를 나타내는 변수
    for (i = 0; i <= 6; i = i + 3){
        for (j = 0; j <= 6; j = j + 3){
            subgrid_args *arg = (subgrid_args *)malloc(sizeof(subgrid_args));
            arg -> idx = k;
            arg -> row = i;
            arg -> col = j;
            pthread_create(&subgridtid[k++], NULL, check_subgrid, arg);
        }
    }
}
```

이후, pthread_join(rowtid, NULL) 명령어로 check_rows 스레드의 종료를, pthread_join(coltid, NULL) 명령어로 check_columns 스레드의 종료를, for loop를 돌며 pthread_join(subgrid[k], NULL) 명령어로 9개의 check_subgrid 스레드의 종료를 기다린다.

```
/*
 * 11개의 스레드가 종료할 때까지 기다린다.
 */
// 여기를 완성하세요
pthread_join(rowtid, NULL); // check_rows 스레드의 종료를 기다림
pthread_join(coltid, NULL); // check_columns 스레드의 종료를 기다림
for (k = 0; k < 9; k++)
    pthread_join(subgridtid[k], NULL); // 9개의 check_subgrid 스레드의 종료를 기다림

/*
 * 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");
}
```

2. 컴파일 과정



```
os@os-VirtualBox: ~/proj2
File Edit View Search Terminal Help
os@os-VirtualBox:~/proj2$ ls
sudoku.c
os@os-VirtualBox:~/proj2$ gcc -o sudoku.o sudoku.c -lpthread
os@os-VirtualBox:~/proj2$ ls
sudoku.c  sudoku.o
os@os-VirtualBox:~/proj2$
```

-lpthread 옵션을 주어 sudoku.c 파일을 컴파일 한 결과, 별다른 오류 메시지 없이 sudoku.o 파일이 생성된 것을 확인할 수 있다.

3. 실행 결과물

- 첫번째 Basic Test

```
os@os-VirtualBox:~/proj2$ ./sudoku.o
***** BASIC TEST *****
 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)
---
```

▲ 첫번째 Basic Test 출력 결과

```
샘플출력.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
***** BASIC TEST *****
 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)
---
```

▲ 첫번째 Basic Test의 샘플 출력 결과

첫번째 Basic Test의 경우, 샘플 출력과 동일하게 모든 행과 열, 3x3 서브그리드가 스도쿠의 조건을 만족한다는 것을 정상적으로 검증하였다.

- 두번째 Basic Test

```

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)
---
```

- ▲ 두번째 Basic Test의 출력 결과

```

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)
---
```

- ▲ 두번째 Basic Test의 샘플 출력 결과

두번째 Basic Test의 경우, 샘플 출력과 동일하게 5번째 행과 6번째 행, 2번째 열과 3번째 열, 4번째 3x3 서브그리드와 6번째 3x3 서브그리드가 스도쿠의 조건을 만족하지 않는다는 것을 정상적으로 검증하였다.

- Random Test

```
***** RANDOM TEST *****
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)
---
```

모든 행과 열, 서브그리드들에 대해 정상적인 검증 결과를 출력한 것을 확인할 수 있다.

```
---
4 7 5 2 6 5 6 5 8
9 4 1 7 8 5 2 1 6
8 9 3 7 9 1 3 4 7
9 3 5 9 1 9 5 9 8
4 1 9 8 2 9 7 2 3
2 7 8 1 5 6 6 1 4
9 5 3 8 1 5 7 8 2
6 4 8 6 4 9 3 5 9
2 5 1 1 3 7 4 7 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,YES)(2,YES)(3,YES)(4,YES)(5,YES)(6,YES)(7,YES)(8,YES)
---
```

모든 행과 열에 대해서는 올바른 검증 결과를 출력하지만,
서브그리드의 경우, 스도쿠의 조건을 만족하지 않음에도 스레드들이 동일한 스도쿠 배열을 공유하면서 생기는 race condition에 의해 잘못된 검증 결과를 출력하는 것을 확인할 수 있다.

```
---
8 2 9 6 4 1 8 7 1
9 7 6 1 9 6 6 3 5
8 2 4 8 2 3 2 4 9
7 9 5 5 1 8 7 4 5
3 1 4 4 2 3 9 6 3
2 8 6 7 6 9 8 2 1
6 1 7 7 2 6 9 1 3
3 2 8 3 4 8 7 6 8
9 5 4 5 9 1 2 4 5
---
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,YES)(2,YES)(3,YES)(4,YES)(5,YES)(6,YES)(7,YES)(8,YES)
---
```

모든 행과 열에 대해서는 올바른 검증 결과를 출력하지만,
마찬가지로 서브그리드에 대해서는 잘못된 검증 결과를 출력하는 것을 확인할 수 있다.

```

---
 9 2 1 3 4 8 7 2 8
 5 7 2 5 3 2 2 3 5
 1 8 4 4 7 1 7 9 8
 6 8 4 6 3 9 1 2 6
 3 9 7 1 7 2 3 7 8
 2 5 1 5 4 8 5 4 9
 7 2 9 1 8 5 9 1 2
 5 6 3 2 6 9 6 5 7
 1 4 8 7 3 4 3 8 4
---
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,YES)(2,YES)(3,YES)(4,YES)(5,NO)(6,YES)(7,YES)(8,YES)
---
```

모든 행과 열에 대해서는 올바른 검증 결과를,
서브그리드에 대해서는 잘못된 검증 결과를 출력하는 것을 확인할 수 있다.

```

---
 5 2 6 1 9 6 9 4 7
 8 3 1 8 4 3 2 7 1
 4 7 9 2 5 7 4 5 8
 6 8 3 6 2 7 9 2 8
 1 7 9 3 8 4 6 5 4
 2 5 4 1 5 9 7 3 1
 1 7 5 8 7 3 2 6 3
 8 9 2 1 4 9 5 7 4
 3 6 4 2 6 5 8 9 1
---
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,YES)(2,YES)(3,YES)(4,NO)(5,YES)(6,YES)(7,YES)(8,YES)
---
```

모든 행과 열에 대해서는 올바른 검증 결과를,
서브그리드에 대해서는 잘못된 검증 결과를 출력하는 것을 확인할 수 있다.

```

---
 5 6 1 4 7 8 7 2 9
 2 3 7 6 1 2 4 8 5
 8 4 9 3 5 9 6 1 3
 5 1 3 3 1 9 9 5 4
 6 2 4 8 2 5 2 7 8
 8 9 7 4 6 7 3 6 1
 9 2 3 3 8 7 9 6 8
 4 6 5 2 4 9 5 4 7
 7 8 1 6 1 5 2 3 1
---
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,YES)(2,YES)(3,YES)(4,YES)(5,YES)(6,YES)(7,YES)(8,YES)
---
```

shuffle_sudoku의 스레드 종료 후 다시 한번 스도쿠 퍼즐을 검증한 것이기에,
모든 행과 열, 서브그리드에 대해 올바른 검증 결과를 출력하는 것을 확인할 수 있다.

4. 과제를 수행하면서 경험한 문제점과 느낀점

Random Test의 출력 과정에서 스레드들이 하나의 스도쿠 배열을 공유하면서 race condition이 발생하고, 이에 따라 잘못된 검증 결과가 나와야 하는데, 예상과는 다르게 나오질 않았었다. 교수님께서 가상 환경설정에서 코어가 하나로 잘못 설정되어 있는 경우, 현실에서의 결과값과 다르게 순차적으로 실행된다고 말씀해 주셔서 가상환경에 cpu를 2개 할당했더니, 정상적으로 잘못된 검증 결과를 출력하는 것을 확인할 수 있었다. 이번 과제를 수행하면서 부모가 자식 스레드를 생성하고 종료를 기다리고, 생성된 자식 스레드는 명령을 수행하고, 종료하여 부모에게 결과를 반환하는 일련의 과정들과 여러 스레드들이 동일한 자원을 공유하면서 생기는 문제에 대해 이해할 수 있었다.