운영체제론 PROJ2 보고서

소프트웨어학부 2019007329 이지현

1. 함수설명

- check_rows

스도쿠 퍼즐의 각 행이 올바른지 검사하는 함수이다.

총 9개의 행을 조사해야 하기 때문에 9번 반복하는 for문을 작성하였다. row 배열은 i번째 행의 숫자들 중 중복되는 숫자가 있는지 판별하기 위한 배열이다. 9개의 행을 검사할 때마다 row 배열을 초기화 시켜야 하므로 for문 안에서 정의하였다. 쉽게 검사 코드를 작성하기 위해서 i행의 valid값을 미리 1로 지정하였고, 이후 탐색 과정에서 중복을 발견할 경우 0으로 다시 할당한다. 이중 for문을 이용하여 i번째 행의 9개 숫자들을 탐색한다. {스도쿠 숫자}의 중복을 판별하기 위해 {sudoku[i][j]}-1을 index로 설정하고, 그 row[index]의 값에 1을 더해주었다. row[특정 index] 값이 0이 아닌 1이라면, 숫자 {index+1}가 이미 등장했다는 뜻이다. 만약 2 이상이라면 특정 숫자가 두 번 이상 등장했다는 뜻이다. 조건문을 사용하여 row[index]가 2 이상일 경우, valid[0][i행] 값으로 0을 할당한다. i행이 잘못되었다면 더이상 그 행의 숫자를 검사할 필요가 없기 때문에 break로 for문을 빠져나온다. 모든 행을 다 검사했다면 이중 for문을 빠져나오고 phread_exit(NULL)에 도달한다.

- check_columns

스도쿠 퍼즐의 각 열이 올바른지 검사하는 함수이다.

총 9개의 열을 조사해야 하기 때문에 9번 반복하는 for문을 작성하였다. col 배열은 i번째 열의 숫자들 중 중복되는 숫자가 있는지 판별하기 위한 배열이다. 9개의 열을 검사할 때마다 col 배열을 초기화 시켜야 하므로 for문 안에서 정의하였다. 쉽게 검사 코드를 작성하기 위해서 i열의 valid값을 미리 1로 지정하였고, 이후 탐색 과정에서 중복을 발견할 경우 0으로 다시 할당한다. 이중 for문을 이용하여 i번째 열의 9개 숫자들을 탐색하였다. {스도쿠 숫자}의 중복을 판별하기 위해 {sudoku[j][i]}-1을 index로 설정하고, 그 col[index]의 값에 1을 더해주었다. col[특정 index] 값이 0이 아닌 1이라면, 숫자 {index+1}가 이미 등장했다는 뜻이다. 만약 2 이상이라면 특정 숫자가 두 번 이상 등장했다는 뜻이다. 조건문을 사용하여 col[index]가 2 이상일 경우, valid[1][i열] 값으로 0을 할당한다. i열이 잘못되었다면 더이상 그 열의 숫자를 검사할 필요가 없기 때문에 break로 for문을 빠져나온다. 모든 열을 다 검사했다면 이중 for문을 빠져나오고 phread_exit(NULL)에 도달한

다.

행 대신 열을 검사한다는 부분을 제외하고 check_rows와 같은 알고리즘이다.

- check_subgrid

스도쿠 퍼즐의 각 3x3 서브그리드가 올바른지 검사하는 함수이다.

이 함수는 9개의 서브그리드를 한 번에 검사하지 않고 하나의 서브그리드만 검사하는 함수이다. 그렇기 때문에 서브그리드 숫자들 중 중복되는 숫자가 있는지 판별하기 위한 sub 배열을 for문 밖에서 정의하였다. 열과 행 모두 탐색하므로 k번째 서브그리드가 sudoku 배열의 어떤 range 안에서 탐색되어야 하는지 판단하기 위해서 행, 열의 검사 시작 index를 i_set, j_set 변수에 저장하였다. k번째 서브그리드의 검사 시작 행 index는 (k / 3) * 3이다. 검사 시작 열 index는 (k % 3) * 3이다. sudoku 배열의 행은 i_set부터 i_set+3까지, 열은 j_set부터 j_set+3까지 검사된다. 서브그리드 valid 값도 미리 1로 설정해두고 검사 과정에서 중복이 발생하면 0을 할당해준다. 서브그리드를 검사하기 위한 이중 for문 안으로 들어오면 check_row, check_columns 함수와 마찬가지로 특정 숫자가 중복되어 나타났는지 index 변수로 판단하고, 만약 sub[index]가 2 이상이라면 valid[2][k]에 0을 할당한다. 검사하는 서브스레드가 하나이기 때문에 중복 숫자가 나타날 경우 이중 for문을 모두 빠져나와야 한다. goto를 사용하여 한 번에 이중 for문을 빠져나오고, phtread_exit(NULL)을 실행한다.

- check_sudoku

스도쿠 퍼즐이 올바르게 구성되어 있는지 11개의 스레드를 생성하여 검증하는 함수이다.

pthread_t tid[11]로 총 11개의 tid를 생성한다. check_rows에 스레드 한개(tid[0]), check_columns에 스레드 한개(tid[1]), check_subgrid에 스레드 9개(tid[2]~tid[10])를 사용한다. pthread_create로 스레드를 생성한 뒤 check_rows 함수를 실행하고, 스레드를 하나 더 생성하여 check_columns 함수도 실행시킨다. 서브그리드의 경우 9개의 스레드를 생성해야 하므로 for문을 사용하여 tid[2]부터 tid[10]까지 사용할 수 있도록 for문 i의 range를 2부터 11전(10)까지로 지정하였다. for문 안에서 pthread_create로 총 9개의 스레드를 생성하고 check_subgrid 함수를 실행한다. 이 때 서브그리드의 위치를 식별할 수 있는 값을 함수의 인자로 넘기기 위해 temp 배열을 이용하였다. 총 0~8번째 서브그리드를 식별하기 위해 temp의 index는 for문 range에서 -2를 뺀 i-2를 사용하였고, temp[i-2]에 i-2를 할당해 주었다. 각 함수에 대한 스레드를 전부 생성한 뒤에 모든 스레드가 종료할 때까지 기다리기 위해 각 pid에 대한 phread_join을 11번 반복해주었다. 이후 스레드가 모두 종료되었다면 각 결과에 대한 검증 결과가 출력된다.

2. 컴파일 과정

```
os@os-VirtualBox:~/Documents/theory_assignment/proj2-2$ make
gcc -Wall -O -c sudoku.c
gcc -o sudoku sudoku.o  -lpthread
```

3. 실행 결과물 설명

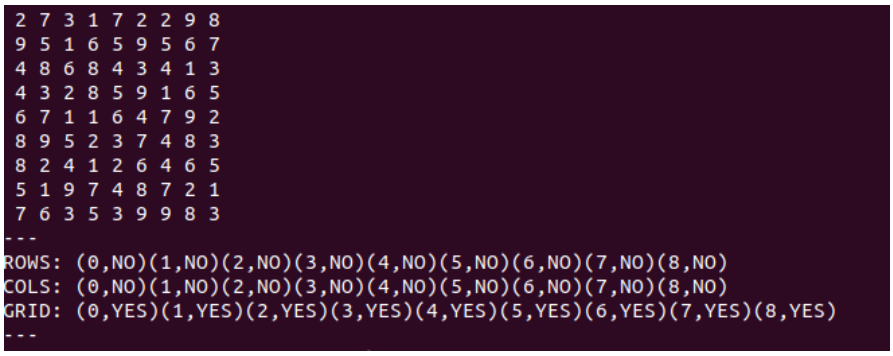
```
os@os-VirtualBox:~/Documents/theory_assignment/proj2-2$ ./sudoku
********** BASIC TEST **********
 6 3 9 8 4 1 2 7 5
 7 2 4 9 5 3 1 6 8
 1 8 5 7 2 6 3 9 4
 2 5 6 1 3 7 4 8 9
 4 9 1 5 8 2 6 3 7
 8 7 3 4 6 9 5 2 1
 5 4 2 3 9 8 7 1 6
 3 1 8 6 7 5 9 4 2
 9 6 7 2 1 4 8 5 3
---
ROWS: (0,YES)(1,YES)(2,YES)(3,YES)(4,YES)(5,YES)(6,YES)(7,YES)(8,YES)
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)
---
 2 3 9 8 4 1 2 7 5
 7 6 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 4 6 3 7
 8 7 3 2 6 9 5 2 1
 5 4 2 3 9 8 7 1 6
 3 1 8 6 7 5 9 3 2
 9 6 7 2 1 4 8 5 4
---
ROWS: (0,NO)(1,NO)(2,YES)(3,YES)(4,NO)(5,NO)(6,YES)(7,NO)(8,NO)
COLS: (0,NO)(1,NO)(2,YES)(3,NO)(4,YES)(5,NO)(6,YES)(7,NO)(8,NO)
GRID: (0,YES)(1,YES)(2,YES)(3,YES)(4,YES)(5,YES)(6,YES)(7,YES)(8,YES)
---
```

basic test 첫번째 결과는 원본 스도쿠이기 때문에 모든 행렬에서 YES 결과가 나왔다.

두번째 결과는 sudoku[0][0] <-> sudoku[1][1], sudoku[5][3] <-> sudoku[4][5], sudoku[7][7] <-> sudoku[8][8]로 기본 퍼즐에서 세 개를 맞바꾸었기 때문에 바꾼 index의 행과 열에 NO 결과가 나왔다.

다음으로 random test이다.

기본 스도쿠 퍼즐에서 shuffle_sudoku 함수를 이용해 퍼즐을 섞는데, 무작위로 퍼즐를 섞는 동시에 검증 함수를 5번 실행하였다. 섞는 중에 검증을 시행했기 때문에 테스트 결과가 올바르지 않을 수 있다. 첫번째 실행 결과를 보면, 스도쿠의 0번째 열은 중복이 없으나 검증 결과는 NO로 출력되었다. 이렇게 스도쿠와 valid 결과가 다르게 검증된 테스트 결과가 많다.



마지막으로 suffle_sudoku 스레드가 종료될 때까지 기다린 뒤 검증한 테스트 결과이다.

섞는 과정이 끝난 후 검증하였기 때문에 퍼즐의 출력과 검증 결과가 일치한다.

4. 문제점과 느낀점

스레드를 단순히 생성하는 것에서 끝나는 것이 아니라, 스레드가 끝날때까지 기다리는 과정이 필요하다는 것을 알 수 있었다. pthread_join을 사용하지 않고 스레드를 여러개 생성하였다면 어떤 부분에서 문제가 발생했는지 알기 어려웠을 것이다. 사실 proj1의 파이프 생성 코드를 작성하면서도 굳이 쓰지 않는 파이프를 close하는 과정이 필요할까 의문이 들곤 했다. 쓰지 않는 파이프를 close했던 이유도 이와 비슷하다고 생각한다. 위험 상황을 모두 고려하고 예외를 처리하는 과정은 분야를 가리지 않고 중요하기 때문이다.

스레드의 중요성 또한 알 수 있었다. 이번 과제는 많은 계산 과정이 필요하지는 않았지만, 계산량이 많은 프로그램을 스레드 없이 실행한다면 굉장히 비효율적일 것이다. 프로그램의 계산 복잡도에 맞게 스레드를 생성하여 실행하는 방법은 프로그램 실행 시간을 줄일 수 있는 좋은 방법 중 하나라 생각한다.

이번 프로젝트는 스레드를 pthread_create로 직접 생성해주었지만, 수업시간에 배운 OpenMP를 이용한다면 더욱 쉽게 스레드를 생성하고 join할 수 있다. 실습 시간에 OpenMP를 이용하여 코드를 작성해 보았기 때문에 pthread_create에 비해 스레드 관리 측면에서 더 효율적일 것이라 생각한다. 또한 동기화 문제가 발생한 스레드를 관찰하는 것이 이번 과제의 주제였지만, 6장에서 다루는 동기화 문제 해결 방법을 배워 이 프로젝트에서 발생하는 다양한 동기화 문제를 해결해 본다면 스레드와 동기화를 이해하는데 큰 도움이 될 것 같다.