운영체제론 Project #2

소프트웨어학부 2017012251 윤영훈

Source Code 설명

※ 변경하지 않은 Section에 대해서는 설명하지 않음 ※

[check_rows]

```
32  void *check_rows(void *arg)
33  {
34      // 여기를 완성하세요
35      int chknum = 0;                                     // To find same number in same row
36
37      for (int i = 0; i < 9; i++){
38          for (int k = 0; k < 8; k++){
39              if (chknum != 0)
40                  break;
41              for (int j = k+1; j < 9; j++){
42                  if (sudoku[i][k] == sudoku[i][j]){
43                      chknum++;                           // If chknum != 0, it means find same number in same row
44                      break;
45                  }
46              }
47          }
48          if (chknum == 0)
49              valid[0][i] = 1;
50          else
51              valid[0][i] = 0;
52          chknum = 0;
53      }
54
55      pthread_exit(0);
56  }
```

각각의 row 내부에서 아래와 같은 방식으로 값을 비교한다.

k가 0일 때, j를 1부터 8까지 1씩 더해가며 sudoku[i][k]와 sudoku[i][j] 비교
k가 1일 때, j를 2부터 8까지 1씩 더해가며 sudoku[i][k]와 sudoku[i][j] 비교
...
k가 8일 때, j를 8부터 8까지 1씩 더해가며 sudoku[i][k]와 sudoku[i][j] 비교

이때 중복이 발생한다면 미리 0으로 선언해둔 chknum의 값을 +1.
chknum이 0이 아니므로 k, j에 대한 for문 탈출 후 line 51. valid[0][i] = 0;

k, j에 대한 for문을 모두 돌았음에도 같은 값이 발견되지 않는다면 chknum는 0이므로
line 49. valid[0][i] = 1;

다음 i를 위해 line 52. chknum = 0; 으로 chknum를 초기화한다.

마지막으로 line 55. pthread_exit(0); 을 통해 Thread를 종료한다.

[check_columns]

```
62  void *check_columns(void *arg)
63  {
64      // 여기를 완성하세요
65      int chknum = 0;                                 // To find same number in same column
66
67      for (int i = 0; i < 9; i++){
68          for (int k = 0; k < 8; k++){
69              if (chknum != 0)
70                  break;
71              for (int j = k+1; j < 9; j++){
72                  if (sudoku[k][i] == sudoku[j][i]){
73                      chknum++;                           // If chknum != 0, it means find same number in same column
74                      break;
75                  }
76              }
77          }
78          if (chknum == 0)
79              valid[1][i] = 1;
80          else
81              valid[1][i] = 0;
82          chknum = 0;
83      }
84
85
86      pthread_exit(0);
87  }
```

check_row의 매커니즘과 동일하다.

각각의 column 내부에서 아래와 같은 방식으로 값을 비교한다.

k가 0일 때, j를 1부터 8까지 1씩 더해가며 sudoku[k][i]와 sudoku[j][i] 비교
k가 1일 때, j를 2부터 8까지 1씩 더해가며 sudoku[k][i]와 sudoku[j][i] 비교
...
k가 8일 때, j를 8부터 8까지 1씩 더해가며 sudoku[k][i]와 sudoku[j][i] 비교

이때 중복이 발생한다면 미리 0으로 선언해둔 chknum의 값을 +1.
chknum가 0이 아니므로 k, j에 대한 for문 탈출 후 line 81. valid[1][i] = 0;

k, j에 대한 for문을 모두 돌았음에도 같은 값이 발견되지 않는다면 chknum는 0이므로
line 79. valid[1][i] = 1;

다음 i를 위해 line 82. chknum = 0; 으로 chknum를 초기화한다.

마지막으로 line 86. pthread_exit(0); 을 통해 Thread를 종료한다.

[check_subgrid]

```
94  void *check_subgrid(void *arg)
95  {
96      // 여기를 완성하세요
97      location_t *sub_sudoku = (location_t *) arg;
98      int row = sub_sudoku->row / 3;
99      int column = sub_sudoku->column / 3;
100
101     int chknum = 0;                                  // To find same number in same sub_sudoku
102     int loop_chk = 0;                                //
103     int array[9];                                    // loop_chk & array will use to make 3x3 sub_sudoku -> 1 array
104
105     for (int i = 3 * row; i < 3 * (row+1); i++){
106         for (int k = 3 * column; k < 3 * (column+1); k++){
107             array[loop_chk] = sudoku[i][k];
108             loop_chk++;                              // sub_sudoku[3][3] -> array[9]
109         }
110     }
111
112     for (int i = 0; i < 8; i++){
113         if (chknum != 0)
114             break;
115         for (int k = i+1; k < 9; k++){
116             if (array[i] == array[k]){
117                 chknum++;                            // If chknum != 0, it means find same number in same sub_sudoku
118                 break;
119             }
120         }
121     }
122
123     if (chknum == 0)
124         valid[2][3*row+column] = 1;
125     else
126         valid[2][3*row+column] = 0;
127
128
129
130
131     pthread_exit(0);
132 }
```

각각의 sub_sudoku에 따라 row, column index가 다르므로 미리 선언해둔 location_t 구조체를 사용하여 각 sub_sudoku에 따른 row, column index를 line 97-99를 통해 적용한다.

또한 3x3 sub_sudoku를 분해하여 순서대로 array에 집어넣고 중복을 검사하기 위해 line 101-103을 선언한다.

1	2	3
4	5	6
7	8	9

➔

1	2	...	8	9

3x3 sub_sudoku가 성공적으로 분해되어 array에 입력된다면 이후 앞선 check_rows, check_columns와 동일한 매커니즘으로 중복을 검사한다.

이후 중복이 발생했다면 chknum != 0 이므로 line 126. valid[2][3*row+column] = 0;
중복이 발생하지 않았다면 chknum == 0 이므로 line 124. valid[2][3*row+column] = 1;

마지막으로 line 131. pthread_exit(0); 을 통해 thread를 종료한다.

[check_sudoku]

```c
void check_sudoku(void)
{
    printf("\nCHECK_SUDOKU START\n");
    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_row;
    pthread_attr_t attr_row;

    pthread_attr_init(&attr_row);
    pthread_create(&tid_row, &attr_row, check_rows, NULL);

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

    pthread_t tid_column;
    pthread_attr_t attr_column;

    pthread_attr_init(&attr_column);
    pthread_create(&tid_column, &attr_column, check_columns, NULL);
```

```c
    /*
     * 9개의 스레드를 생성하여 각 3x3 서브그리드를 검사하는 check_subgrid() 함수를 실행한다.
     * 3x3 서브그리드의 위치를 식별할 수 있는 값을 함수의 인자로 넘긴다.
     */

    pthread_t tid_subgrid[9];
    pthread_attr_t attr_subgrid[9];
    location_t *chk_subgrid[9];

    for (i = 0; i < 3; i++){
        for (j = 0; j < 3; j++){
            pthread_attr_init(&attr_subgrid[3*i+j]);

            chk_subgrid[3*i+j] = (location_t *)malloc(sizeof(location_t));
            chk_subgrid[3*i+j]->row = 3*i;
            chk_subgrid[3*i+j]->column = 3*j;

            pthread_create(&tid_subgrid[3*i+j], &attr_subgrid[3*i+j], check_subgrid, chk_subgrid[3*i+j]);
        }
    }

    /*
     * 11개의 스레드가 종료할 때까지 기다린다.
     */
    pthread_join(tid_row, NULL);                    // wait for check_row thread
    pthread_join(tid_column, NULL);                 // wait for check_column thread
    for (i = 0; i < 9; i++){
        pthread_join(tid_subgrid[i], NULL);         // wait for each check_subgird threads
    }
```

shuffle_sudoku와 check_sudoku가 어떤 순서로 실행되는지 확인하기 위해
line 141. printf("\nCHECK_SUDOKU START\n");를 적용.
(이후 CHECK_SUDOKU END, SHUFFLE_SUDOKU START, SHUFFLE_SUDOKU END 모두 출력)

line 158 – 162 : check_rows() 함수 실행을 위한 thread 관련 변수 선언 및 thread 생성
line 168 – 172 : check_columns() 함수 실행을 위한 thread 관련 변수 선언 및 thread 생성
line 179 – 193 : check_subgrid() 함수 실행을 위한 thread 관련 변수 선언 및 thread 생성

check_subgrid()의 경우 9개의 sub_sudoku가 생기므로 각 thread 관련 변수를 배열로 선언하고, 각 sub_sudoku마다 row, column index가 다르므로 location_t 구조체에 저장하여 arg 인자로 전달한다.

모든 thread를 생성한 뒤 line 198 – 202를 통해 11개의 thread가 모두 종료할 때까지 기다린다. 이때 subgrid의 경우 9개 각각의 thread 종료를 위해 for 문을 활용했다.

Terminal 실행

```
xion@xion-VirtualBox:~/eclipse-workspace/osproj2/src$ gcc -pthread -o proj2 proj2-1.skeleton.c
xion@xion-VirtualBox:~/eclipse-workspace/osproj2/src$ ./proj2

CHECK_SUDOKU START
 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)
---
CHECK_SUDOKU END


CHECK_SUDOKU START
 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)
---
CHECK_SUDOKU END
```

```
CHECK_SUDOKU START
 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)
---
CHECK_SUDOKU END

SHUFFLE_SUDOKU START
SHUFFLE_SUDOKU END

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

기본적으로 -pthread 옵션을 통해 thread 컴파일을 진행한다.

첫 번째 결과에서는 모든 row, column, grid가 스도쿠 규칙을 만족하므로 YES가 출력된다.

두 번째 결과에서는 (6,2) / (5,3)의 값 2 / 4의 위치를 서로 변경했으므로 아래의 값이 NO로 변경되었다.

ROW	5, 6
COLUMN	2, 3
GRID	4, 6

세 번째 결과에서는 바뀐 값을 다시 되돌리고, shuffle_sudoku를 통해 sudoku를 섞는다.

섞는 중간에 check_sudoku를 통해 스도쿠를 검증하지만, 거의 대부분의 상황에서는 shuffle_sudoku thread가 실행되기 전 check_sudoku thread가 먼저 실행되어버린다. 그로 인해 모든 row, column, grid가 shuffle이 적용되기 전인 첫 번째 결과와 동일한 결과를 출력한다. (Linux OS가 원인으로 예상)

※ 100번이 넘는 컴파일 + 실행을 통해 검증해보아도 섞는 도중에 check_sudoku가 실행되는 상황을 확인할 수 없었다 ※

마지막 네 번째 결과에서는 shuffle_sudoku를 통해 스도쿠가 섞였으므로 ROW와 COLUMN 전체가 NO를 출력한다. GRID의 경우는 shuffle_sudoku에서 짝수 index의 subgrid는 서로 값을 바꿔주었고, 홀수 index의 subgrid에만 rand()를 통해 임의의 값을 집어넣었으므로 짝수 index의 subgrid에서는 YES가, 홀수 index의 subgrid에서는 NO가 출력된다.