SJSU CS/SE 149 HW2 Spring 2018

REMINDER: Each homework (including programming question) is individual. "Every single byte must come from you." Cut&paste from others is not allowed.

1. (10 pts) Consider the following code segment:

```
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
      fork();
      pthread_create( ... );
}
fork();
```

the remaining code (not shown) does not call fork nor call pthread_create.
a. How many unique processes are created (including the first main or root process)? Justify your answer.
b. How many unique threads are created by `pthread_create()`? Justify your answer.

2. [programming question] (90 pts) A ***Sudoku*** puzzle uses a 9 × 9 grid in which each column and row, as well as each of the nine 3 × 3 subgrids (separated by darker vertical and horizontal lines), must contain all of the digits 1 to 9.

6	5	3	1	2	8	7	9	4
1	7	4	3	5	9	6	8	2
9	2	8	4	6	7	5	3	1
2	8	6	5	1	4	3	7	9
3	9	1	7	8	2	4	5	6
5	4	7	6	9	3	2	1	8
8	6	5	2	3	1	9	4	7
4	1	2	9	7	5	8	6	3
7	3	9	8	4	6	1	2	5

Please design a multithreaded application in C with Pthreads - it determines whether the solution to a Sudoku puzzle is valid. Q2 can be completed on Linux or Linux VM.

There are several different ways of multithreading this application. One suggested strategy is to create parallel threads that check the following criteria:

- A thread to check that each column contains the digits 1 through 9 without duplication
- A thread to check that each row contains the digits 1 through 9 without duplication
- Nine threads to check that each of the 3 × 3 subgrids contains the digits 1 through 9 without duplication

This would result in a total of eleven separate threads for validating a Sudoku puzzle in parallel. However, you are welcome to create even more (but not less) threads for this project. For example, rather than creating one thread that checks all nine columns, you could create nine separate threads and have each of them check one column.

The main program must invoke `pthread_join` to wait for the termination of each worker thread. Since the idea is to run multiple worker threads in parallel, the main program must invoke all `pthread_create` before invoking any `pthread_join`.

Representing the 9 x 9 Puzzle

One may presentment the 9 x 9 grid with a two dimensional array as follows:

```
int grid[GRID_SIZE][GRID_SIZE];
```

In HW2, you can hard-code initial values to the array. For example, the following code initializes a 3 x 3 array:

```
int grid[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

where row 0 is {1, 2, 3}, row one is {4, 5, 6}, and row two is {7, 8, 9}. In row 0, column 0, 1, and 2 has values 1, 2, and 3, respectively; in row one, column 0, 1, and 2 has values 4, 5, and 6, respectively; in row two, column 0, 1, and 2 has values 7, 8, and 9, respectively.

Passing Parameters to Each Thread

The parent thread will create the worker threads, passing each worker the location that it must check in the Sudoku grid. This step will require passing several parameters to each thread. The easiest approach is to create a data structure using a `struct`. For example, a structure to pass the row and column where a thread must begin validating would appear as follows:

```
/* structure for passing data to threads */
typedef struct
{
   int row;
   int column;
} parameters;
```

With Pthreads, one can create worker threads using a strategy similar to that shown below:

```
parameters *data = (parameters *) malloc(sizeof(parameters));
data->row = 1;
data->column = 1;
/* Now create the thread passing it data as a parameter */
```

The data pointer will be passed to the `pthread_create()`, which in turn will pass it as a parameter to the function that is to run as a separate thread. You should deallocate each of the memory blocks by using `free()` after a thread completes.

Returning Results to the Parent Thread

Each worker thread is assigned the task of determining the validity of a particular region of the Sudoku puzzle. Once a worker has performed this check, it must pass its results back to the parent. One good way to handle this is to create an array of integer values that is visible to each thread. The *i*th index in this array corresponds to the *i*th worker thread. If a worker sets its corresponding value to 1, it is indicating that its region of the Sudoku puzzle is valid. A value of 0 would indicate otherwise. When all worker threads have completed, the parent thread checks each entry in the result array to determine if the Sudoku puzzle is valid.

In your program, please hard-code the 9x9 grid with the solution as follows and take screenshots of your program execution.

```
6,5,3,1,2,8,7,9,4,
1,7,4,3,5,9,6,8,2,
9,2,8,4,6,7,5,3,1,
2,8,6,5,1,4,3,7,9,
3,9,1,7,8,2,4,5,6,
5,4,7,6,9,3,2,1,8,
8,6,5,2,3,1,9,4,7,
4,1,2,9,7,5,8,6,3,
7,3,9,8,4,6,1,2,5
```

Please note

1. Each invocation the program always prints out “CS149 Sudoku from FirstName LastName” only once. Take screenshots of the program execution (including “CS149 Sudoku from …”). The program can then print out the 9x9 grid to stdout before it prints the final result (valid Sudoku or not) to stdout.

2. Any API in a multi-threaded application must be thread-safe (e.g., on Linux `rand()` vs `rand_r()`). Invoking any non thread-safe API is subject to deduction.

3. You must utilize array for various data structures, and utilize loop to remove repeating code. Any repetitive variable declaration and/or code are subject to deduction.

Compile your program with "`gcc -o sudoku sudoku.c -pthread`". You can execute the program with "`./sudoku`".

====

Submit the following files as individual files (do not zip them together):

- `CS149_HW2_YourName_L3SID` (.pdf, .doc, or .docx), which includes
  - Q1: answers and justifications
  - Q2:
    - in less than one page, high level description of exact steps for each thread.
    - screenshots of the program execution. Screenshots that are not readable, or screenshot without "CS149 Sudoku from …" from the program, will receive 0 point for the entire homework.
- `sudoku_YourName_L3SID.c` Please ident your source code and include comments .

The ISA and/or instructor leave feedback to your homework as comments and/or annotated comment. To access annotated comment, click "view feedback" button. For details, see the following URL:
http://guides.instructure.com/m/4212/l/106690-how-do-i-use-the-submission-details-page-for-an-assignment