SJSU CS 149 HW2 SPRING 2020

REMINDER: Each homework is individual. "Every single byte must come from you." Cut&paste from others is not allowed. Keep your answer and source code to yourself only - never post or share them to any site in any way.

[Type your answer. Hand-written answer is not acceptable.]
[Replace YourName and L3SID with your name and last three digit of your student ID, respectively.]

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

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

the remaining code (not shown) does not call `fork()` nor call `pthread_create()`. Assume each invocation to `fork()` or `pthread_create()` was successful.
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.

Use a software to <u>draw one single space-time diagram</u> (similar to the ones for fork2 and fork3 on slides, add a new notation of your choice to represent threads) to justify your answers for both a & b. Hand-drawing receives no credit.

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

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

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.

In this multithreaded application, create parallel threads that check the following criteria:

- A worker thread to check that each column contains the digits 1 through 9 without duplication
- A worker thread to check that each row contains the digits 1 through 9 without duplication
- Nine worker 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 worker threads for validating a Sudoku puzzle in parallel. However, you are welcome to create even more (but not less) worker 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 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 two-dimensional 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 must deallocate each of the memory blocks by using `free()` after a thread completes.

In addition to configuration parameters in the `struct`, one can include additional parameters in the `struct` such as a pointer to the puzzle, etc. to each worker thread. If you are not comfortable with passing puzzle pointers in the `struct`, you can copy the necessary content of a puzzle to the `struct` and then passes the `struct` to each worker thread.

Identifying location and reason for an invalid puzzle

Any problematic cell in a puzzle causes at least one or more of the following three errors: value not between 1 and 9, duplicated value, and missing value. When a row/column/subgrid thread identifies any error (so that the puzzle is not valid), the thread must print out the problematic row/column/puzzle location <u>and</u> the specific error to stdout (to be included in the screenshots). For example, there are three error types,

```
puzzle[1][2] = 3, duplicated value
puzzle[4][5] = 10, not between 1 and 9
row[0], missing value 7
column[8], missing value 9
```

Location is 0-based. Each thread must perform all necessary checks – it cannot identify a single error and then skip the remaining checks. For example, a row thread that checks 9 rows must identify all errors in each row, a column thread that checks 9 columns must identify all errors in each column, and 9 subgrid threads, each of which checks one 3x3 subgrid, must identify all errors in each subgrid.

In fact, any given problematic cell in a puzzle results in multiple errors identified by multiple corresponding threads. For the problematic cell "puzzle[3][7] = 0", how many errors your program should identify (hint: more than 1). Part of your job is to identify for each of these 3 error types, which other errors can occur at the same time?

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. Instead of using an integer array to pass result back to the parent, you can pass the result via another member in the `parameters` struct, or you can have worker thread pass the result via `pthread_exit`, etc.

Validate two puzzles

In your program, hard-code the following two 9x9 grids (say in two variables `puzzle1` and `puzzle2`), and determine if each solution is valid. While each grid should be validated by multiple parallel threads, you can validate `puzzle1` and then `puzzle2` in sequential order (as illustrated) by a single invocation of your program. Note you are not allowed to invoke the program twice, one for each puzzle.

puzzle 1	puzzle 2
7,3,9,8,4,6,1,2,5, 4,1,2,9,7,5,8,6,3, 8,6,5,2,3,1,9,4,7, 5,4,7,6,0,3,2,1,8, 3,9,1,7,8,2,4,5,6, 2,8,6,5,1,4,3,7,9, 9,2,8,4,6,7,5,3,1, 1,7,4,3,5,9,1,8,2, 6,5,3,1,2,8,7,9,4	4,1,9,3,7,6,5,8,2, 3,6,7,8,5,2,1,4,9, 2,8,5,1,9,4,3,7,6, 6,4,1,9,3,7,8,2,5, 9,3,2,4,8,5,6,1,7, 5,7,8,2,6,1,4,9,3, 8,5,6,7,4,9,2,3,1, 1,9,4,5,2,3,7,6,8, 7,2,3,6,1,8,9,5,4

```
void sudoku(…)
{ …
}
int main()
{  …
  sudoku(puzzle1);
   …
  sudoku(puzzle2);
  return 0;
}
```

What you need to do in the report

a. (5 pts) Use puzzle1 and puzzle2 as example, when a certain error type (say X) happens in a puzzle, other error type (say Y) can happen at the same time. Identify all unique pair(s) of error types (the order within a pair does not matter, i.e., {X, Y} is the same as {Y, X}).

b. (15 pts) For each puzzle, is it valid? If not, what is the total number of errors (and break them down to total number of errors for each error type). Fill out the following matrix with amswers.

Puzzle	Valid? (Y/N)	Total number of errors	Number of errors for each error type
puzzle1			
puzzle2			

Include screenshots of the program execution, which includes, for each puzzle, (1) validation result of the puzzle, and (2) specific problems of the puzzle, if any.

c. (5 pts) Include code snippet and describe your exact algorithm (in worker thread) to validate the values are in the range of 1 to 9 (inclusively), no duplication, and all 9 values have shown up.

d. (5 pts) Include code snippet and describe how you pass the results from worker threads back to the main program.

e. (60 pts) source code as separate file.

Reminders

1. Each invocation the program always prints out "CS149 Sudoku from FirstName LastName" only once. For each grid, the program prints the 9x9 grid to stdout, any identified problems of the puzzle if any, and then it prints the final result (valid Sudoku or not) to stdout. Take screenshots of the program execution (including "CS149 Sudoku from …").

2. Any given problematic cell in a puzzle results in multiple errors identified by multiple corresponding threads. Your code should print out all errors in each row, in each column, and in each subgrid. Any missing error is subject to deduction.

3. Any API in a multithreaded application must be thread-safe and reentrant (e.g., on Linux `rand()` vs `rand_r()`). Invoking any API that is not thread-safe or not reentrant is subject to deduction.

4. You must utilize array for various data structure 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: perform a, b, and c as specified.
    - 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` 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:

https://guides.instructure.com/m/4212/l/352349-how-do-i-view-annotation-feedback-comments-from-my-instructor-directly-in-my-assignment-submission

Hint How to pass 1-d int array and 2-d int array to a function?

#define ROW 3
#define COL 4
int a[ROW] = {…};
int b[ROW][COL] = {…};
int main()
{
  chk1d((int *)a);
  chk2d((int *)b);
};
	void chk1d(int *p)
	{
	// specify p[i] or *(p + i) to access to a[i]
	}
		void chk2d(int *q)
		{
		// specify *(q + i * ROW + j) to access to b[i][j]
		}