MSU Denver, M&CS CS 3600: Operating Systems, Summer 2020 Dr. Weiying Zhu

Homework 3 (100 points), Due Date*: 11:59pm 6/17/2019, Cutoff Date: 11:59pm 06/19/2019**
***A warning but NO late penalty for programming assignments; **Submission will NOT be accepted after the cutoff deadline**
Submission: (1) upload your .c file(s) and .txt input files on Blackboard (NO email submission please!), (2) see Task II for the required submission on cs3600a.

Part I. Review Questions for reading the textbook (No submission; however, the relevant contents will be included in Exams as True/False, Multiple-Choices, and/or Filling-in-the-Blanks questions.)

- **Textbook, Chapter 4, Page 197 ~ : 4.3, 4.6, 4.10, 4.13, 4.15, 4.17, 4.19**

Part II. Task I (90%): Write a C program for Project 1 - Sudoku Solution Validator (NO other language may be used**)**

An option of peer programming: You may choose to work on this assignment individually or in a team of two students. If you choose to work in a ***team of two students***, you **must** 1) **add** both team members' first and last names as the **comments** on Blackboard when submitting the .c files (both team members are required to submit the same .c files on Blackboard), and 2) put a ***team.txt*** file including both team members' names under your HW03/ on cs3600a (both team members are required to complete Task II under both home directories on cs3600a). For grading, I will randomly pick the submission in one of the two *home directories* of the team members on cs3600a, and then give both team members the same grade. If the team information is *missing* on Blackboard or cs3600a, two or more submissions with similar source codes with just variable name/comment changes will be considered to be a violation of the Integrity described in the course policies and both or all will be graded as 0.

Grading: Your programs will be graded via testing and points are associated with how much task it can complete.
1. ALL the ***output messages*** that are REQUIRED to be ***displayed*** by your program in Task I are the only way to demonstrate how much task your program can complete. So, whether your program can display those messages correctly with correct information is critical for your program to earn points for the work it may complete.
2. It is also extremely important for your program to be able to correctly READ and TAKE the information from the ***input file(s)*** that follow the format REQUIRED in this programming assignment, instead of some different format created by yourself. Otherwise, your program may crash or get incorrect information from the input file(s) that follow the required format.
3. Grading is NOT based on reading/reviewing your source code although the source code will be used for plagiarism check. A program that cannot be compiled or crashes while running will receive up to 5% of the total points. A submission of .c files that are similar to any online C program with only variable name changes will receive 0% of the total points.

A. Project Description: This programming project is the **Project 1** defined in **pages P-27 through P-29 at the end of Chapter 4 (Threads & Concurrency) in Textbook (10/E)**. Please read the textbook for details.

B. Program Design and Additional or Modified Requirements: these requirements must be followed in addition to what is defined in the textbook. If it is ***not the same*** as what is described in the textbook, please follow the description/requirements in ***this document***.

1. Inputs and Data structures:

1.1 This **C program** must read a 9 x 9 Sudoku puzzle from an input text file named **"SudokuPuzzle.txt"** into a 9 x 9 matrix (2 dimension int array), e.g., **sudokuPuzzle[9][9]**. This file must consists of 9 rows. The *i*-th row in the file **"SudokuPuzzle.txt"** consists of the 9 integers in the *i*-th row of a given Sudoku puzzle, where **tab ('\t')** MUST be used to **separate** those numbers in each row of the file. For example, the first two rows in this file for the puzzle shown in Figure 4.26 should be

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

This C program must print the matrix sudokuPuzzle[ ][ ] after reading it from the input file
(Hint: `fscanf()` can be used to read formatted data from a text file. %d is the format for reading a decimal integer.)

1.2 The following **shared data structures** must be declared as **global variables** such that they are **shared by all threads**

1.2.1 **sudokuPuzzle[9][9]**, a two-dimension integer array recording all numbers in the given Sudoku puzzle.

1.2.2 **a 9-element Boolean array for 9 columns**, where **elements [0] through [8]** are true or false if **columns 1 through 9** in the Sudoku puzzle are valid or invalid, respectively. (Hint: you might have to use "`typedef int bool;`" and then use `#define` to define your own `TRUE` and `FALSE`.)

1.2.3 **a 9-element Boolean array for 9 rows**, where **elements [0] through [8]** are true or false if **rows 1 through 9** in the Sudoku puzzle are valid or invalid, respectively.

1.2.4 **a 9-element Boolean array for 9 3 x 3 subgrids**, where **elements [0] through [8]** are true or false if **3 x 3 subgrids 1 through 9** in the Sudoku puzzle are valid or invalid, respectively. Subgrids are numbered from left to right, top to down. The **1st 3 x 3 subgrid** is the intersection of **rows 1 through 3** and **columns 1 through 3**, the **2nd 3 x 3 subgrid** is the intersection of

rows 1 through 3 and **columns 4 through 6**, …, the **4th 3 x 3 subgrid** is the intersection of **rows 4 through 6** and **columns 1 through 3**, …, the **9th 3 x 3 subgrid** is the intersection of **rows 7 through 9** and **columns 7 through 9**.

1.3 typedef a **struct**ure to represent the index (row and column) range of the elements in a Sudoku puzzle to be processed by each *worker thread* created by the *parent thread*. This structure must include the following four integers.

- **`int topRow`**; //index (0, 1, …, or 8) of top row to be checked by a worker thread
- **`int bottomRow`**; //index (0, 1, …, or 8) of bottom row to be checked by a worker thread
- **`int leftColumn`**; //index (0, 1, …, or 8) of left column to be checked by a worker thread
- **`int rightColumn`**; //index (0, 1, …, or 8) of right column to be checked by a worker thread

For example, in the format of *(<topRow>, <bottomRow>, <leftColumn>, <rightColumn>)*, the *1st column* is represented by *(0, 8, 0, 0)*, the *3rd column* is represented by *(0, 8, 2, 2)*, the 8th row is represented by *(7, 7, 0, 8)*, the *2nd subgrid* is represented by *(0, 2, 3, 5)*, the *7th subgrid* is represented by *(6, 8, 0, 2)*, etc.

2. Worker threads: in this program, the parent thread must create **27 worker threads**, **one for each column, row, or subgrid.**

2.1 The *parent thread* first initializes **9 structures** in an array of structures **type-defined in 1.3** to represent **9 columns**, where for the ***k-th column***, **`topRow`** is **0**, **`bottomRow`** is **8**, and both **`leftColumn`** and **`rightColumn`** are (***k* – 1**).

Next, the parent thread creates **9 worker threads**, one for **each column**, and records the *thread id* of each worker thread in a **`tid_column[]`** array. To the **worker thread** for the ***k-th*** **column**, the parent thread passes the **address of the structure** initialized earlier for the ***k-th*** **column** as a parameter.

2.2 The *parent thread* first initializes **9 structures** in an array of structures **type-defined in 1.3** to represent **9 rows**, where for the ***k-th row***, both **`topRow`** and **`bottomRow`** are (***k* – 1**), **`leftColumn`** is **0**, and **`rightColumn`** is **8**.

Next, the parent thread creates **9 worker threads**, one for **each row**, and records the *thread id* of each worker thread in a **`tid_row[]`** array. To the **worker thread** for the ***k-th*** **row**, the parent thread passes the **address of the structure** initialized earlier for the ***k-th*** **row** as a parameter.

2.3 The *parent thread* first initializes **9 structures** in an array of structures **type-defined in 1.3** to represent **9 subgrids**, where for the **1st, 2nd, and 3rd subgrids**, **`topRow`** is **0** and **`bottomRow`** is **2**, for the **4th, 5th, and 6th subgrids**, **`topRow`** is **3** and **`bottomRow`** is **5**, for the **7th, 8th, and 9th subgrids**, **`topRow`** is **6** and **`bottomRow`** is **8**, for the **1st, 4th, and 7th subgrids**, **`leftColumn`** is **0** and **`rightColumn`** is **2**, for the **2nd, 5th, and 8th subgrids**, **`leftColumn`** is **3** and **`rightColumn`** is **5**, and for the **3rd, 6th, and 9th subgrids**, **`leftColumn`** is **6** and **`rightColumn`** is **8**.

Next, the parent thread creates **9 worker threads**, one for **each subgrid**, and records the *thread id* of each worker thread in a **`tid_subgrid[]`** array. To the **worker thread** for the ***k-th*** **subgrid**, the parent thread passes the **address of the structure** initialized earlier for the ***k-th*** **subgrid** as a parameter.

2.4 Each *worker thread* must check **all the nine cells** in the Sudoku puzzle that are assigned to this *worker thread* (hint: index range in sudokuPuzzle[][] is **`topRow`** <= *row index* <= **`bottomRow`** **and** **`leftColumn`** <= *column index* <= **`rightColumn`)**

- If those nine cells contain each of 9 digits (1 through 9) exactly once,
  1) set the corresponding entry in the *Boolean array* declared in 1.2.2 for *columns*, 1.2.3 for *rows*, or 1.2.4 for *subgrids* as **`TRUE`**
  2) Display the following message in a new line to stdout, where each < … > must be replaced by the corresponding value

```
<worker thread ID in HEX> TRow: <topRow>, BRow: <bottonRow>, LCol:<leftColumn>, RCol: <rightColumn> valid!
```

- Otherwise, if those nine cells do NOT contain each of 9 digits (1 through 9) exactly once,
  1) set the corresponding entry in the *Boolean array* in 1.2.2 for *columns*, 1.2.3 for *rows*, or 1.2.4 for *subgrids* as **`FALSE`**
  2) Display the following message in a new line to stdout, where each < … > must be replaced by the corresponding value

```
<worker thread ID in HEX> TRow: <topRow>, BRow: <bottonRow>, LCol:<leftColumn>, RCol: <rightColumn> invalid!
```

2.5 The *parent thread* must wait for all 27 worker threads to complete, then the *parent thread*

1) Display the contents of **`tid_column[]`** array and the *Boolean array* declared in 1.2.2 in **nine lines** to stdout, where each < … > must be replaced by the corresponding value

```
Column: <TID in HEX>  <"valid" or "invalid" depending on its entry in the Boolean array>
```

2) Repeat 1) for rows by using the info in the **`tid_row[]`** array and the *Boolean array* declared in 1.2.3

```
Row: <TID in HEX>  <"valid" or "invalid" depending on its entry in the Boolean array>
```

3) Repeat 1) for subgrids by using the info in the **`tid_subgrid[]`** array and the *Boolean array* declared in 1.2.4

```
Subgrid: <TID in HEX>  <"valid" or "invalid" depending on its entry in the Boolean array>
```

4) Display the final conclusion "*valid*" ("TRUE" in all entries of all three Boolean arrays) or "*invalid*"

```
   Sudoku Puzzle: <"valid" or "invalid" depending on all values in three Boolean arrays>
```

Task II (10%): **Test your program on the virtual server cs3600a.msudenver.edu**

Warning: to complete this part, especially when you work at home, you must first (1) **connect to GlobalProtect** using your NetID account (please read "**how to connect to GlobalProtect …**" at https://www.msudenver.edu/technology/remoteaccess/); then (2) **connect to the virtual servers cs3600a** using ***sftp*** and ***ssh*** command on MAC/Linux or ***PUTTY*** and ***PSFTP*** on Windows. For details, you may refer to **Lab 1.**

ITS only supports GlobalProtect on MAC and Windows machines. If your home computer has a different OS, it is your responsibility to figure out how to connect to cs3600a for programming assignments and submit your work by the cutoff deadline. Such issues cannot be used as an excuse to request any extension.

1. MAKE a directory "**HW03**" under your home directory on **cs3600a.msdenver.edu**.

2. UPLOAD, COMPILE, and TEST your program under "**HW03**" on **cs3600a.msudenver.edu**.

3. SAVE ***ALL your .txt input files*** and a ***file named testResults.txt*** under "**HW03**" on cs3600a.msudenver.edu, which captures the outputs of your program when you test it. You can use the following commands to redirect the standard output (stdout) to a **file** on UNIX, Linux, or Mac, and view the contents of the file

```
gcc prog_name.c -o prog_name -lpthread   //compile prog_name.c into an executable program
                        //may include -std=c99 if using c99 standard in the gcc command
./prog_name args | tee tstResults.txt     //run the executable program
                                          //while copying stdout to the given .txt file
cat tstResults.txt      //display the contents of the file tstResults.txt
```

4. If you work in a *team of two* students, you must put a ***team.txt*** file including *both* team members' names under your HW03/ on cs3600a (*both* team members are required to complete Task II under their own home directories on cs3600a). For grading, I will randomly pick the submission in one of the two *home directories* of the team members on cs3600a, and then give *both* team members the same grade.