Operating Systems (Code: COMP2006)

CURTIN UNIVERSITY
School of Electrical Engineering, Computing, and Mathematical Sciences
Discipline of Computing

Assignment

Semester 1 2024

Sudoku Solution Validator (SSV)

Due Date: **4PM (local time), Monday 6th of May 2024**

Objective

The objective of this programming assignment is to give you some experiences in using multiple threads and inter-thread communications. You will learn how to create threads and solve their critical section problems.

Assignment Description

The Sudoku puzzle considers a (9 x 9) grid, and its correct/valid solution must have the following properties:

- Each column must contain all of digits 1 to 9,
- Each row must contain all of digits 1 to 9, and
- Each of the nine (3 x 3) sub-grids must contain all of digits 1 to 9.

The following shows one possible valid solution of Sudoku.

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

The Sudoku Solution Validator (SSV) determines if a solution to Sudoku puzzle is valid. **One possible way** of multitasking the validator is to create **four threads** that together validate the following Sudoku properties (see the sub-grid numbers below):

- **Thread-1** validates three (3 x 3) sub-grids, i.e., 1, 2 and 3. In addition, **Thread-1** validates three (1 x 9) sub-grids, i.e., row-1, row-2 and row-3.

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

- **Thread-2** validates three (3 x 3) sub-grids, i.e., 4, 5 and 6. In addition, **Thread-2** validates row-4, row-5 and row-6.
- **Thread-3** validates three (3 x 3) sub-grids, i.e., 7, 8 and 9. In addition, **Thread-3** validates row-7, row-8 and row-9.
- **Thread-4** validates each of the nine columns.

In this assignment, you are asked to write a program in C language that implements a Multithreads Sudoku Solution Validator (**mssv**) using the four **threads**. Specifically, given a solution of the Sudoku puzzle, the **parent** thread and the **four child threads** do the following steps (not necessarily in order).

Parent thread

1) The **parent thread** creates **Sol**, **Row**, **Col**, **Sub**, and **Counter**.

   Sol stores the (9 x 9) Sudoku solution from which each of the four threads reads its assigned input. Note that the parent thread fills in **Sol** from a given input file, called **solution**. The following is the content of file **solution** for the given example, where each integer is separated by one space.

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

   Row is an array of integers, where **Row**[1], **Row**[2], …, **Row**[9] is set to a value one each if rows 1, 2, …, 9 each is valid, respectively.

   Col is an array of integers, where **Col**[1], **Col**[2], …, **Col**[9] is set to a value one each if columns 1, 2, …, 9 each is valid, respectively.

   Sub is an array of integers, where **Sub**[1], **Sub**[2], …, **Sub**[9] is set to a value one each if sub-grids 1, 2, …, 9 each is valid, respectively.
   Each element in arrays **Row**, **Col**, and **Sub** is initialized with a value of zero.

Counter is an integer (initialized to 0) that contains the total number of valid rows, columns, and sub-grids.

For the given example:

- Row[1], Row[2], Row[3], Sub[1], Sub[2], and Sub[3] each is set to "1" by **Thread-1** because it finds that the (3x3) sub-grid 1, 2, 3 and rows 1, 2, 3 are all valid.
- Row[4] = 1, Row[5] = 1, Row[6] = 1, Sub[4] = 1, Sub[5] = 1, and Sub[6] = 1 because **Thread-2** finds that the (3x3) sub-grid 4, 5, 6 and rows 4, 5, 6 are all valid.
- Row[7], Row[8], Row[9], Sub[7], Sub[8], and Sub[9] each is set to "1" by **Thread-3** because it finds that the sub-grid 7, 8, 9 and rows 7, 8, 9 are all valid.
- **Thread-4** will set each element in array Col to a value "1" because it finds each of the nine columns is valid.
- **Thread-1**, **Thread-2**, **Thread-3**, and **Thread-4** each will increment the value of Counter by 6, 6, 6, and 9, respectively. Thus, for the given example, we have **Counter** = 27 because nine rows, nine columns, and nine (3 x 3) sub-grids are valid.

2) The **parent** thread creates four **child** threads and assign each child thread a **region** of the Sudoku solution to validate. The region for **Thread-1** is (3 x 3) sub-grids 1, 2, 3, and rows 1, 2, 3. For **Thread-2**, its region is (3 x 3) sub-grids 4, 5, 6, and rows 4, 5, 6 while for **Thread-3**, its region is (3 x 3) sub-grids 7, 8, 9, and rows 7, 8, 9. Finally, the region for **Thread-4** is columns 1 to 9. You need to design a way for the parent thread to assign each child thread with its corresponding region.

3) The parent thread waits for the results (i.e., the contents of **Row**, **Col**, **Sub**, and **Counter**) from the four child threads. The parent thread **blocks** while waiting for the results from all child threads.

4) When all child threads have completed their validation, the parent thread prints all **invalid** row numbers, invalid column numbers, invalid sub-grid numbers, and the total number of "valid" rows, columns, and sub-grids to the screen.

   For the given example, where the solution is valid, the parent thread prints the following on the screen, where ID-*i* refers to the ID of thread-*i*.

   Thread ID-1: valid
   Thread ID-2: valid
   Thread ID-3: valid
   Thread ID-4: valid

   There are 27 valid sub-grids, and thus the solution is valid.

   However, if row 1, row 2, sub-grid 7, and column 5 are **invalid**, the parent thread will print the following on the screen, where ID-*i* refers to the ID of thread-*i*.

   Thread ID-1: row 1, row 2 are invalid
   Thread ID-2: valid
   Thread ID-3: sub-grid 7 is invalid
   Thread ID-4: column 5 is invalid

There are in total 23 valid rows, columns, and sub-grids, and the solution is invalid.

5) Remove all created resources, e.g., child threads, etc.

6) Terminate.

Child thread

Each child thread-*i* performs the following steps after receiving its assigned region.

1) Validate all sub-grids in its assigned region. For example, **Thread-1** will validate rows 1 to 3, and sub-grids 1 to 3. You must include a *sleep* (**delay**) after a thread finishes **each** validation. This delay may be needed to observe any synchronization issue that can generate inconsistent results. Note that you must provide the value of **delay** when you start running your program; its suggested value is 1, i.e., a delay of 1 second.

2) Each child thread sets Row**[***a***]**, Col[*b*], or Sub[*c*] to "1" if it finds that row *a*, column *b* or sub-grid *c* is valid, respectively.

3) Each child thread updates the value of **Counter**, i.e., **Counter** = **Counter** + "total number of rows, columns and/or subgrids that it found to be valid".

   For example, the thread that validates nine columns, i.e., Thread-4 should increase **Counter** by 9 (or 7) if it found 9 (or 7) valid columns.

4) The **last thread** that updates **Counter** must **wake up** the **parent thread**. This last thread must print its ID on the screen, e.g., if the last thread has ID-4, it prints the following.

   Thread ID-4 is the last thread.

5) Each child thread terminates when it has completed its task.

Notice that **Sol**, **Row**, **Col**, **Sub**, and **Counter** are *shared* by all threads. Furthermore, access to **Row, Col, Sub** and **Counter** can be considered as a **producer-consumer problem**, i.e., the **parent** thread is the **consumer** while all **child** threads are the **producers**. Thus, your solution must address their possible synchronization issues, e.g., by using pthread_mutex_lock(), pthread_mutex_unlock(), pthread_cond_wait(), and pthread_cond_signal().

Let us call your executable program **mssv**. Your program should be run as:

mssv solution delay

where **solution** is the file that contains the Sudoku Solution to be validated, and **delay** is an integer with a value between 1 and 10.

Instruction for submission

1. Assignment submission is **compulsory**. As stated in the unit outline, the penalty for late submission is calculated as follows:
   - *For assessment items submitted within the first 24 hours after the due date/time, students will be penalised by a deduction of 5% of the total marks allocated for the assessment task,*
   - *For each additional 24 hour period commenced an additional penalty of 10% of the total marks allocated for the assessment item will be deducted, and*
   - *Assessment items submitted more than 168 hours late (7 calendar days) will receive a mark of zero.*

2. Please read and follow the **Academic Integrity (including plagiarism and cheating)** section in the unit outline to prevent any possible academic misconduct.

3. You must (i) submit the soft copy of the report to the unit Blackboard (**in one zip file**), and (ii) put your program files, i.e., **mssv.c**, and other files, e.g., makefile, test input and output, in your home directory named **OS/assignment**.

4. Your assignment **report** should include:

   - A signed **Declaration of Originality** form (available in the unit Blackboard). Please read the form carefully before you sign it. Your name should be that as recorded in the student database. By signing the form, among others, you agree on the following two statements:
     1. *The work I am submitting is entirely my own, except where clearly indicated otherwise and correctly referenced.*
     2. *Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.*

   - Software solution of your assignment that includes (i) all source code for the programs with proper in-line and header documentation. Use proper indentation so that your code can be easily read. Make sure that you use meaningful variable names and delete all unnecessary comments that you created while debugging your program; and (ii) readme file that, among others, explains **how to compile your program and how to run the program**.

   - Detailed discussion on how synchronization is achieved when accessing shared resources / variables and which threads access the shared resources.

   - Description of any cases for which your program is not working correctly or how you test your program that makes you believe it works perfectly.

   - Sample inputs and outputs from your running programs. Explain if the outputs are correct / incorrect.

Your report will be assessed (worth 20% of the overall assignment mark).

5. You might be required to demonstrate your program. The demo time will be announced later. Your program must run on a computer in the school.

Failure to meet these requirements may result in the assignment not being marked.