Operating Systems–2
Programming Assignment 2: Dynamic Validation of Sudoku
Submission Deadline: 2nd March 2025, 9:00 pm

Goal:- This assignment aims to design a multithreaded program to determine whether a solution to a Sudoku puzzle is valid through a Dynamic multi-threaded mechanism in C++.

Sudoku Validation Details:- This assignment is an extension of Programming Assignment1 which is based on Project 1 given in the 10th edition of the textbook (pdf page 262).

For a given N X N sudoku, where N is a square, there will be N rows, N columns and N grids to be validated as shown below.

Validation Condition:- For a N X N sudoku, the validation condition is:
- Each row contains unique values from 1 - N.
- Each column contains unique values from 1 - N.
- Each of the nXn sub-grids, contains a unique value from 1 - N where, $N = n^2$

For example:- For 4X4 sudoku, you need to check each row, each column and each of the four 2X2 subgrids have values from 1 to 4.

Assignment Details:- This assignment is an extension of the programming assignment 1. Consider a sudoku. The goal of the problem is to find if sudoku is valid or not in parallel using the dynamic mechanism (as discussed in class).

In this programming assignment, you will build upon your previous work measuring the performance of determining whether a sudoku is valid or not in parallel in C++. Previously, you had allocated the rows of the matrix to threads statically (in either mixed or chunk mode).

This assignment is an extension of the previous programming assignment. However, unlike the static allocation of tasks to each thread, each thread chooses a part of the matrix to check dynamically using a **shared counter C** explained as follows:
1. Each thread increments C by **taskInc**.
2. The increment determines the thread's next validation task: number of rows, or columns, or subgrids.
3. After the thread completes the current task, it again increments C as explained in step1. This continues **until all tasks are completed**.

Please note the complication that may arise with step 2 when the number of rows/columns/subgrids remaining to be processed may be less than taskInc. This can happen when N is not a multiple of taskInc. Special care must be taken to handle such cases.

As a part of your assignment, you will have to handle the following:

1. **Synchronization Problems:** Multiple threads will **compete to increment C.** The increment operation requires **safe access** to avoid race conditions. To handle the synchronization issues problem, you must implement different mutual exclusion algorithms to increment C as discussed in the class, which are: (a) **TAS**, (b) **CAS**, (c) **Bounded CAS.** Wrappers for TAS and CAS are provided by the C++ atomic library.

2. **Early Termination:-** Similar to Assignment1, the computation can be terminated, once a thread identifies failure in Sudoku validation. You can choose any technique for early termination such as: (a) Locks & Global Variables (b) Thread Cancellation etc. However, unlike Assignment1, there is no extra credit for implementing this.

Input File:-
The input to the program will be a file, named input.txt
The first line of the input file will contain two values **K**, **N,**and **taskInc,** where K is the number of threads, N is the dimensional value of N X N sudoku and taskInc is how many tasks a thread picks at once.

From the second line, the sudoku of dimension N X N will be present, with each row in a new line. This sudoku can be generated through a python script.

A sample input file will be like this :-
4 4 2
1 3 4 2
2 4 1 3
4 2 3 1
3 1 2 4
In the above example, K = 4, N = 4 (n=2) and taskInc

Output File:-
For ease of understanding, you need to generate an output file in which you will store the details of the execution of each thread and at last it should output whether the given sudoku is valid or invalid. As a part of the output, you will have to record the following events:

1. CS entry event
2. Incrementing the counter C to grab a rows/cols/chunks
3. CS exit event
4. Checking the validity of a rows/cols/chunks

5. Final outcome – validity of the Sudoku
6. Time taken to check the validity of the Sudoku. In case of early exit, the time taken will be less.
7. Average time taken by a thread to enter the CS
8. Average time taken taken by a thread to exit the CS
9. Worst-case time taken by a thread to enter the CS
10. Worst-case time taken taken by a thread to exit the CS

The times taken mentioned in points 6, 7, 8, 9, 10 will be the final events recorded after the validation (or invalidation) of Sudoku. The analysis of the output should show that correctness of the mutual exclusion algorithms implemented.

A sample output would be something like this:-

```
Thread 1 requests to enter CS1 at 10:00 hrs
Thread 2 requests to enter CS1 at 10:00 hrs
Thread 3 requests to enter CS1 at 10:00 hrs
Thread 1 enters CS1 at 10:02 hrs
Thread 1 grabs row 1 at 10:03 hrs
Thread 1 leaves CS1 at 10:04 hrs

Thread 2 enters CS1 at 10:05 hrs
Thread 2 grabs row 2  at 10:06 hrs.
Thread 1 completes checking of row 1 at 10:07 hrs and finds it as valid.
Thread 2 leaves CS1 at 10:07 hrs

Thread 3 enters CS1 at 10:08 hrs
Thread 3 grabs row 3  at 10:09 hrs.
Thread 3 leaves CS1 at 10:10 hrs
Thread 2 completes checking of row 2 at 10:10 hrs and finds it as valid.
.
.
.
.
.
Sudoku is valid.

The total time taken is 25.05 microseconds.
Average time taken by a thread to enter the CS is 10 microseconds
Average time taken by a thread to exit the CS is 2 microseconds
Worst-case time taken by a thread to enter the CS is 20 microseconds
Worst-case time taken by a thread to exit the CS is 5 microseconds
```

Report Details:- As part of this assignment, you must prepare a report describing your program's low-level design. Further, you must perform the following experiments described below, measure the performance, and provide the observations you recorded in the tabular format.

As a part of the report, you will perform three experiments. You will show the output of the experiments in the form of plots and tables. Plots are nice and intuitive to visualize. However, if the number of curves are many, then plots can get too cluttered. In that case, tables are better. Both are explained below.

Plots: You will get 4 curves for the three experiments, one for each mutual exclusion method which are as follows:

(a) TAS
(b) CAS
(c) Bounded CAS
(d) Sequential

Note that there is a (minor) difference between sequential and single-threaded execution.

Extra Curves: You can also add the curves from assignment1 to these plots: Chunk and Mixed. This is optional though!

Tables: You will have to develop the following tables for each of the three experiments below. You will have to show the CS Entry and Exit time for each of the curves mentioned above:

Average Case Entry Tables

S. No	X-axis	TAS Avg CS Entry Time	CAS Avg CS Entry Time	Bounded CAS Avg CS Entry Time	TAS Avg CS Exit Time	CAS Avg CS Exit Time	Bounded CAS Avg CS Exit Time	Total Time Taken

Worst Case Entry Tables

S. No	X-axis	TAS Worst-Case CS Entry Time	CAS Worst-Case CS Entry Time	Bounded CAS Worst-Case CS Entry	TAS Worst-Case CS Exit Time	CAS Worst-Case CS Exit Time	Bounded CAS Worst-Case CS Exit	Total Time Taken

				Time			Time	

If you wish, you can combine both the above tables into a single table. It can be seen that CS Entry and Exit times are not valid for static algorithms.

Experiments:

1. Time vs. Sudoku Size (Fixed Threads)

- X-axis: Sudoku size: 20^2, 30^2, 40^2, 50^2, 60^2, ...
  (After that, work in increments of 10 to 100 or follow the system's supported limit, whichever comes first.)
- Y-axis: Time taken
- Fixed: taskInc = 20, Num of threads = 8

2. Time vs. Task Increment (taskInc)

- X-axis: taskInc values: 10 to 50 in the increments of
- Y-axis: Time taken
- Fixed: Sudoku size =90^2, Num of threads = 8

3. Time vs. Number of Threads (Fixed Sudoku Size)

- X-axis: Threads: 1, 2, 4, 8, 16, 32
- Y-axis: Time taken
- Fixed: Sudoku size =90^2, taskInc = 20

To handle temporary outliers, ensure that each point in the above plots is averaged over 5 times. Thus, you will run your experiment 5 times for each point on the x-axis.

Thus, you will have three plots and tables for the above three experiments. You must write the observation you gained based on the plots & tables described above and mention any anomalies, if you observe in the report as well.

Submission Format:- You have to upload:

1. The source code in the following format: Assgn2Src-<RollNo>.c
2. Input file as inp.txt
3. Readme: Assgn2Readme-<RollNo>.txt, which contains the instructions for executing the program
4. Report: Assgn2Report- <RollNo>.pdf.
5. Zip all the above document and name it as Assgn2-<RollNo>.zip

Please follow this naming convention. Otherwise, your assignment will not be graded.

Grading Policy:- The policy for grading this assignment will be -

(1) Design as described in the report and analysis of the results: 50% (2) Execution of

the tasks based on the description in the readme: 40% (3) Code documentation and indentation: 10%.

Please note:

1. All assignments for this course have a late submission policy of a penalty of 10% each day after the deadline of six days. After that, it will not be evaluated.
2. *All submissions are subjected to plagiarism checks. Any case of plagiarism will result in F grade. You can refer to the department's website for more information regarding the anti-plagiarism policy.*