

Sudoku Solution Validator

CMPE 180C Operating System Design

Spring 2020

Submitted by:

Team 12

Akhil Cherukuri 014525420 akhil.cherukuri@sjsu.edu

Dishant Shah 014615614 dishant.shah@sjsu.edu

Course Instructor

Dr. Hungwen Li

Table of Contents:

Chapter 1: Introduction

Chapter 2: Background and Objectives

Chapter 3: Approach/Methodology/Algorithm

Chapter 4: Results, Finding and Analysis

Chapter 5: Conclusion and Recommendation

References

Appendix: code/program

Chapter 1: Introduction

Sudoku is a puzzle game consisting of a 9x9 square grid consisting of randomly filled numbers that are between 1-9 (1). The goal of the game is to fill the empty squares with a number from 1-9 such that each row and column has every number from 1-9 without repeating. Each of the 9 3x3 squares also needs to be filled with 1-9 as well. There can be multiple solutions to Sudoku puzzles especially the easier ones with fewer randomly filled squares. Normally, to check a solution, the puzzle is compared to the officially provided solution. If they do not match, manually checking the puzzling by hand can take time and effort. It would be more efficient to develop a program that can validate any solution by checking if the board fulfills the three requirements. Developing a program utilizing threads to simultaneously validate the solution of the problem.

Multi-threading is useful in any situation where a single thread has to wait for a resource, and you can run another thread in the meantime. This includes a thread waiting for an I/O request or database access while another thread continues with CPU work. In the sudoku solution validator, if we use only one thread to solve the problem (brute force method), it would take more time than it would if we divide the possible algorithm into several sub-algorithms which are as independent as possible and yet combine to give the solution. You are allowing the CPU to make several calculations simultaneously to solve the problem faster. This would allow for optimal utilization of resources thus reducing the overall time required.

In the sudoku solution validator, we will utilize threads and implement multi-threading to help solve and develop a tool for determining the validity of the solution efficiently. we will be using 1 thread for row calculations, 1 thread for column calculations, and 9 threads for each 3x3 grid calculations for a total of 11 threads which will be synchronized using the fork-join strategy. Each of the parent thread will return a Boolean value depending on whether the column, row or square is valid. The parent thread will wait until the validation of the parts is completed. It will determine if the overall solution is valid or not by determining that all Boolean values are true. In this project, we will implement multi-threading as well as the fork-join synchronization strategy to develop an efficient tool for sudoku solution validation.

Chapter 2: Background

Sudoku is a mathematical puzzle game that was first published in 1979 in Dell Magazines (2). There have been predecessors on other number-based grid games in the past, but this was the first modern implementation of the game. Sudoku has since grown in popularity with a dedicated fan base. Its popularity is so significant that a large number of newspapers feature a sudoku puzzle in their word/ number puzzles section. As previously stated, sudoku consists of a 9x9 array randomly filled with numbers in the 1-9 range which needs to be filled to complete the game. 3 conditions that need to be fulfilled for the solution to the given puzzle are valid.

Row: Each row needs to contain every number from 1-9 without repeats.

Column. Each column needs to contain every number from 1-9 without repeat.

Square: there are 9 3x3 squares in the 9x9 grid of the puzzle. Each of them must contain every number from 1-9 without repeating as well.

Any filled sudoku puzzle that does not match the above requirements cannot be considered a valid solution. However, there can be multiple solutions to the puzzle, and it is possible to stumble across a solution to the puzzle that is not the official one provided in the puzzle book or the newspaper. During such circumstances, having a program that can determine the validity of the solution would be beneficial to save time and effort.

Thread is an execution unit that consists of its program counter, a stack, and a set of registers. Threads are also known as lightweight processes. Creating multiple threads is more economical than creating multiple processes as threads take fewer resources and as well as share resources such as code, data, and files between each thread under that process which is known as Inter-thread communication. Another advantage is that context switches between threads are faster than between processes. Threads are a popular way to improve the application through parallelism. The Central Processing Unit switches rapidly back and forth among the threads giving the illusion that the threads are running in parallel. This allows for threads to be an ideal way to access the computation resources for the needs of this project.

Threads are visible only from within the process, where they share all process resources like address space, open files, and so on. Because threads share data, a change on one thread can be visible by another thread in the operating system. Apart from data, each thread has its unique properties or private variables like Thread ID, Register state, Stack, Priority, Thread-Private Storage.

A system can be utilized either to use a single thread or multi-threads. In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution. In a multithreaded process in a shared-memory multiprocessor environment, each thread in the process can run on a separate processor at the same time, resulting in parallel execution. An application can have hundreds of threads and still not consume many kernel resources. The amount of kernel resources the application uses is largely determined by the application requirements. Implementation of multi-threading will allow for algorithm for validation to be split into smaller subsections which can be run in a more efficient manner

The user threads must be mapped to kernel threads either by Many to One Model, One to One Model, or Many to Many Models. There are 3 types of threads i.e., POSIX Pitheads (Linux, UNIX), Win32 threads (windows), Java threads. There are two types of threads: User threads are threads that do not need kernel support. These threads are used inside an application program. Kernel threads are threads that are used inside an Operating System. These threads support the kernel to perform multiple tasks at the same time or to serve multiple kernel system calls simultaneously.

Multithreading has the possibility of resulting in improper manipulation of the shared data through unorganized access to it by the threads. To solve this, a variety of methods have been implemented. An example of the several possible solutions is Mutex, fork-join, Conditional variable, barrier, spinlock as well as a semaphore. The fork-join strategy involves the parent thread waiting until the child threads perform the work concurrently. After the child threads are finished, the parent thread can continue. In this project, none of the child threads should manipulate the shared data that the other will manipulate. The main thread would need to wait till the other threads finish validation of the parts which can be performed concurrently. Both of these facts make the fork-join strategy ideal for implementation in this project.

Objectives:

* To develop a program to validate the solution to the sudoku puzzle.
* To implement threading and subsequently multithreading using the fork-join strategy to the solution to optimize the timing and resource usage.