**System programming**

**Lab 4**

**Threads**

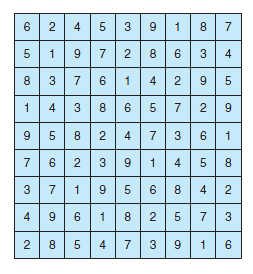
**Task #1**

A ***Sudoku*** puzzle uses a 9 × 9 grid in which each column and row, as well as each of the nine 3 × 3 subgrids, must contain all of the digits 1 … 9. Figure presents an example of a valid Sudoku puzzle. This assignment consists of designing a multithreaded application that determines whether the solution to a Sudoku puzzle is valid.

There are several different ways of multithreading this application. One suggested strategy is to create child threads that check the following criteria:

* A thread to check that each column contains the digits 1 through 9.
* A thread to check that each row contains the digits 1 through 9.
* Nine threads to check that each of the 3 × 3 subgrids contains the digits 1through 9.

This would result in a total of eleven separate threads for validating a Sudoku puzzle. However, you are welcome to create even more 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.

****

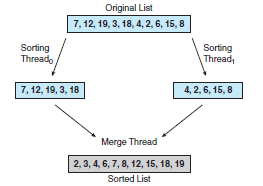
The parent thread will create the worker threads, passing each worker the location that it must check in the Sudoku grid. Each worker thread is assigned the task of determining the validity of a particular region of the Sudoku puzzle. Once a worker thread has performed this check, it must pass its results back to the parent. 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 determines if the Sudoku puzzle is valid.

**Task #2**

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term ***sorting threads***) sort each sublist using a sorting algorithm of your choice. The two sublists are then merged by a third thread—a ***merging thread*** —which merges the two sublists into a single sorted list.

Because global data are shared cross all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured

according to the following figure:

****

This programming task will require passing parameters to each of the sorting threads. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. The parent thread will output the sorted array once all sorting threads have exited.

**Task #3**

In this assignment, you are going to use mutexes in the **pthread** library. Write a program using **pthread** to create 5 child threads. ***The main function*** creates a file called data.txt and stores an initial value 0 in it. Every child thread is required to access this file 20 times with synchronization, i.e. only one child thread is able to access the file at a time. They need to read the value stored in the file, increment the value by one, write the new value to the file and print the new value out then use the function **usleep()** to sleep for a random time (within 1 s).

Reference:

**usleep**

Suspends the calling thread for the specified number of seconds.

**Syntax**

#include <unistd.h>

int usleep (useconds\_t useconds);

**Parameters**

useconds -- specifies the microseconds to suspend the calling thread. The value must be less than 1,000,000.