Operating Systems Lab (CS 470):

Lab 3: Write a program in C/C++ under Linux/MacOS handling a basic menu –in text mode, with 3 items *(Problem 1*, *Problem 2*, and *Exit* item). When the user selects *Problem1* (typing 1 from the keyboard) the software should start running the problem 1. If the *Problem2* menu item is selected, the problem 2 should run. The software should exit only if the *Exit* menu is selected. The problem 1 and problem 2 should run as many times the user selects them. After the execution of the selected problem, the menu should show up on the screen allowing the user a new selection.

Problem 1: Given a sequence of numbers $x_0, x_1, \ldots, x_n^2 \in \{0,1\}$ generated randomly, where x_i represents a square matrix element in a linear fashion (concatenate the lines of the matrix for example), write a simulation process as follows: a) start M threads, b) each thread generates a column *i* and a row *j* randomly (0<=i,j<n), and c) considering the element location in the matrix given by the generated column and row change the value of the matrix element to 0 if its 8 neighbors (elements which they have a direct connection in the matrix with element (i,j)) are predominantly 0, and 1 otherwise. If the number of neighbors is equal you can do nothing (not change the location (i,j) in the matrix or randomly flip the element to 0 or 1. The same threads will generate new locations from the matrix and do the "flipping" until all the elements in the matrix become 0 or all the elements become 1.

Problem 2: Given a sequence of numbers, $x_0, x_1, \ldots, x_n \in R$ generated randomly, write a simulation process as follows: a) start M threads, b) each thread is generating an index *i* randomly such as 0<=i<n. Each thread is taking the value x[i] from the array and inserts the element in such a way that every element before the element x[i] should be smaller or equal than x[i]. The same threads will generate new indices until the array is completely sorted in increasing order.

Overview

Reading and modifying (increment, decrement, assign) a shared variable/file between different threads needs extra attention due to inconsistency which might occur if the processes/threads run in parallel. In order to avoid this, different mechanism are implemented to solve the critical section problem.

Instructions

The elements for the matrix (see Problem1) or the array (see Problem2) should be generated randomly each time when the user runs Problem1 or Problem2 from the menu. The original matrix/array should be printed on the standard output. The simulations (see problem 1 or problem 2) end when all elements in the matrix turn 0 or 1 or the elements are completely sorted for the second problem.

To protect the critical section consider *pthread_mutex_lock()* and *pthread_mutex_unlock()* functions, respectively. The stopping criteria checking (all the elements are 0/1 or the list is sorted) should run in a separate thread for both problems.

Notes

- The number of elements (n) should be provided as command line argument, while the number of threads (M) should be read from the console at each time when Problem1 or Problem2 is invoked.
- The threads should be implemented using POSIX pthread API.
- Each step in the simulation process should be traceable. Printing on the standard output is to be provided. Please provide information such as: a) which thread is doing something, b) the thread is generating which (i,j) for the matrix or i for the array and c) how that tread is influencing the matrix or the array (see occurring modification if any).

Rubric

Task	**Points**
Error handling	2
Simulation running Problem1	4
Simulation running Problem2	4