COM S 352: Intro to Operating Systems

Project 1: POSIX Threads

100 Points

Due : Friday, 15 March 2019, 11:59 pm

1 Overall Objective

You are to write a C or C++ concurrent *Shearsort* program using POSIX Threads (PThreads) library. You are NOT to use `fork()` calls, pipes, or message queues in this exercise.

The *Shearsort* is a simple mesh-sorting algorithm that consists of nothing more than alternately sorting rows and columns of the mesh. In particular, it sorts all the **rows** in Phases 1, 3, ..., $\log_2 \sqrt{N} + 1$, and all the **columns** in Phases 2, 4, ..., $\log_2 \sqrt{N}$, where N is the total number of elements. The **columns** are sorted so that smaller numbers move upward. The **odd rows** (1, 3, ..., $\sqrt{N}-1$) are sorted so that smaller numbers move leftward, and the **even rows** (2, 4, ..., $\sqrt{N}$) are sorted in reverse order (i.e., so that smaller numbers move rightward). The numbers will appear in a snakelike order after $2\log_2 \sqrt{N} + 1 = \log_2 N + 1$ phases. An example is given in Figure 1. (A proof that the Shearsort algorithm works can be found in *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* by F. Thomson Leighton. However, the information about Shearsort in this handout should be sufficient for you to complete the exercise.)

2 Assignment

In this assignment you will use shared memory to store ONE copy of the whole 2-dimensional array. Also, you will be required to create a limited number of threads for the sort. For a square array of $n \times n$ elements, ONLY n THREADS are to be created. Each thread is to alternate between row (odd) and column (even) phases. To synchronize the phases properly, you may use either *semaphores* (at most ONE SEMAPHORE PER THREAD), or a more elegant data structure for this purpose, called *condition variables*.

Your program should implement the Shearsort algorithm to handle 16 integers as follows.

1. Read the integers into a 2-dimensional $n \times n$ array in global memory from the file "input.txt". This can be done by the main thread before creating the n sorting threads.

2. Print the integers in the order entered to `stdout`.

3. Create and initialize the semaphores or condition variable necessary for the algorithm.

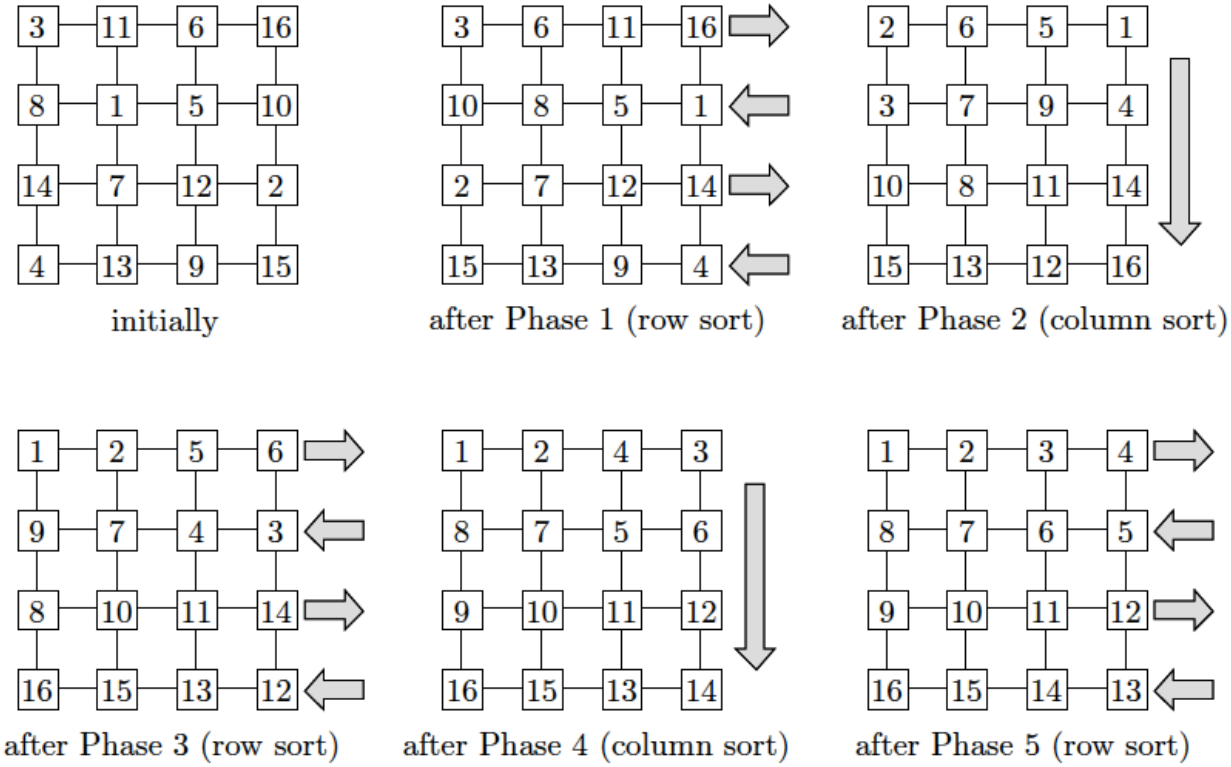

Figure 1: Alternately sorting the rows and columns in Shearsort. The numbers to be sorted appear in a snakelike order after $log_2 N + 1$ phases. Notice that even rows are always sorted in reverse order. The arrow direction indicates the sorting order (from small to large).

4. Create the n threads to sort the array using Shearsort.

5. Wait for the threads to finish a phase.

6. Print the array of sorted integers to `stdout` and go to the next phase..

Each thread would do the following in each phase of the algorithm.

1. By performing the appropriate number of wait operations on sempahores (or by a proper wait on a condition variable), block until the prior phase (if any) is finished.

2. Sort the row/column in the appropriate order using Bubble Sort.

3. Perform appropriate signal operations to signal the other threads to begin the next phase.

   WARNING: You are NOT to use the sleep() call or any code (such as a loop counting from 1 to 10000) to introduce any delay in your program. If any such delaying code is found in your program, YOU WILL LOSE %30 OF TOTAL POINTS.

3 POSIX pthreads

POSIX Threads, usually referred to as Pthreads, is an execution model that exists independently from programming language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API. You can find plenty of material Online to learn about this API. Essentially you need to know; how to (1) create threads (2) terminate threads (3) use semaphores or condition variables. Here's one resource: https://computing.llnl.gov/tutorials/pthreads/

4 Guidelines

- You should use C/C++ to develop the code.
- **You should work on this project individually.**
- You need to turn in electronically by submitting a zip file named: Firstname_Lastname_Project1.zip.
- Source code must include proper documentation to receive full credit (you will lose 10% of your score, if the code is not well documented as follows).
    - File Comments: Every .h and .c file should have a high-level comment at the top describing the file's contents, and should include your name(s) and the date.
    - Function Comments: Every function (in both the .h and the .c files) should have a comment describing:
        - what function does;
        - what its parameter values are
        - what values it returns (if a function returns one type of value usually, and another value to indicate an error, your comment should describe both of these types of return values).
    - In-line Comments: Any complicated, tricky code sequences in the function body should contain in-line comments describing what it does (here is where using good function and variable names can save you from having to add comments).
- All projects require the use of a **make file** or a certain script file (accompanying with a readme file to specify how to use the script/make file to compile), such that the grader will be able to compile/build your executable by simply typing "make" or some simple command that you specify in your readme file.
- **Source code must compile and run correctly on the department machine "pyrite", which will be used by the TA for grading. If your program compiles, but does not run correctly on pyrite, you will lose 15% of your score. If your program doesn't compile at all on pyrite, you will lose 50% of your score (only for this issue/error, points will be deducted separately for other errors, if any)**.
- You are responsible for thoroughly testing and debugging your code. The TA may try to break your code by subjecting it to bizarre test cases.
- You can have multiple submissions, but the TA will grade only the last one.