Machine Problem 2 (MP2)

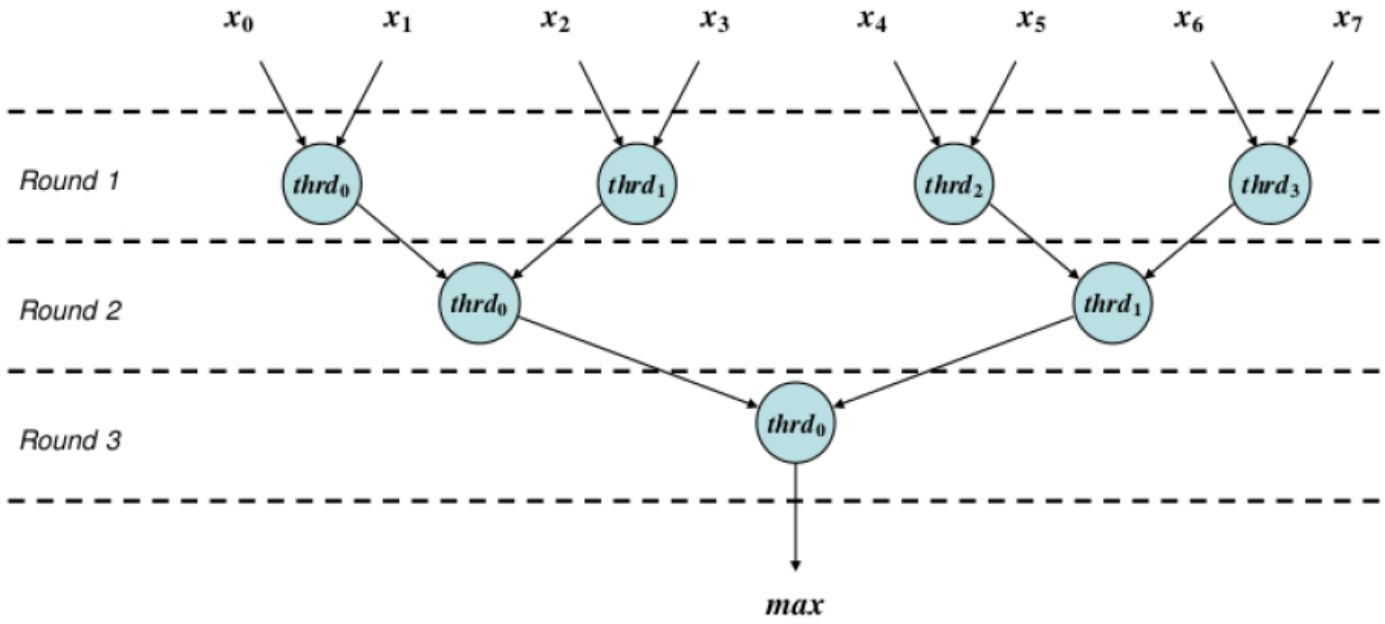
Due Thursday September 29, 2017

Purpose

The primary objective of this homework is to familiarize you with synchronization primitives, specifically: (a) how they can be used to construct more sophisticated primitives; and (b) how multiple threads of execution can use them in order to solve problems cooperatively.

Problem

You are to implement a parallel, binary reduction. Given a list of N numbers, your program will use $N/2$ threads to cooperatively determine the maximum number in the list.



A maximum-finding binary reduction works much like a basketball tournament: pairs of items can be compared in parallel just like pairs of teams play each other in a game. The maximum item (or winner of the game) goes on to the next round. If there are N numbers (or N teams), the total number of rounds will be $\log_2 N$.

For a given round, we will use individual threads (basketball courts) to execute the comparisons (play the games) concurrently. But we must wait until all of the comparisons for a given round have been made before we can *re-use* the threads (courts) to start the next. For this, you will need to implement a *barrier* primitive.

A *barrier* is a synchronization mechanism used to coordinate the activities of a set of threads. The basic idea is simple. The barrier primitive is initialized to some value, say 8. Threads issue a wait() call on the barrier. If less than 8 threads have performed the wait, then any calling threads are blocked. When the 8th thread calls wait(), it wakes up the other seven and everyone proceeds. You will use binary semaphores to

create a barrier class that implements the wait() method.

Your program will have no command-line parameters. It will obtain read the input sequence of numbers from stdin, each number on its own line. It will recognize the end of the sequence when it receives an empty line. If it helps, you can assume that the length of the input sequence will be a power-of-two. After the last round, print the final result to stdout.

Your solution will use a single barrier that will be shared by all threads ***and that will be re-used between rounds***. Similarly you will use the same N/2 for all rounds, not create a new set of threads on every round. Your barrier will have the property that all waiters for a particular round will be released before any waiters in subsequent round.

Note well. You must implement your own Barrier class. You may only use integers and binary semaphores in your implementation. More precisely you must use pthreads counting semaphores as if they were binary semaphores, i.e. they can only take values of 0 and 1.

In your written evaluation, be sure to discuss your implementation, specifically:

- How you implemented the barrier. How you tested your barrier.
- Whether or not your main thread participates in comparisons
- How you organized (in program memory) the intermediate results.

Miscellaneous

- You may want to familiarize yourself with the shell's pipe operator ('|'), allowing you to feed a textfile as input to your program using cat.
- You must use Unix/Linux for this homework.
- Your implementation must be in C/C++ and use the POSIX APIs for threads and semaphores. You must use the GNU C/C++ development tools (e.g., gcc, g++).
- You must submit your solution by uploading an archive (.tar) of your source file(s) to the course's Collab portal. Along with your sources (.c/.cpp/.h files), you must submit a Makefile that will allow us to compile your solution, specifically to an executable named "max". Failure to follow these directions will cause you to get a zero on your code portion of the grade.
- You must submit in class a printed writeup regarding your solution in accordance with the requirements set in the Beginning of Course Memo (BOCM). The writeup WILL include the source and header files that you have created.
- If you have any other questions about the homework please ask.