Iran University of Science and Technology

School of Computer Engineering

Operating Systems Course – Fall 2018

Homework #3

Due: 6/8/1397 10:00

Thread Programming & Synchronization

1. You have a serial version of vector dot product computation. Change this program to do the computation using multiple threads.
   Your program should input the length of vectors and the number of threads from the user (t), then initialize the two vectors with random values, divide the vectors to (t) parts. Each thread computes its own part and updates the final value. The main thread prints the final dot product.
   For Simplicity, assume that the vector length is divisible to t.

   Example Output:

   ```
   $ ./vector_thread.out
   Enter vector length: 8
   Enter number of threads: 2
   Main thread: filling the vectors with random numbers.
   Main thread: vector1 = <1,2,3,4,5,6,7,8>
   Main thread: vector2 = <1,2,3,4,5,6,7,8>
   Thraed1: computing my part
   Thraed2: computing my part
   Main thread: dot product result: 204. Exiting
   ```

   In your readme file, explain what locking primitives you used and WHY?

2. Now write a multithreaded matrix multiplication code which computes the sum of the rows of result matrix as a vector: (see the figure and example)

   See the example:

   ```
   $ ./matrix_thread.out
   Enter matrix length: 6
   Enter number of multiplier threads: 3
   Main thread: filling the matrices with random numbers.
   Main thread: matrix1 = <1,2,3,4,5,6
                           1,2,3,4,5,6
                           1,2,3,4,5,6
                           1,2,3,4,5,6
                           1,2,3,4,5,6
                           1,2,3,4,5,6>
   ```

Main thread: matrix2 = <1,2,3,4,5,6
1,2,3,4,5,6
1,2,3,4,5,6
1,2,3,4,5,6
1,2,3,4,5,6
1,2,3,4,5,6>
Main thread: Creating 3 multiplier threads
Main thread: Creating the Aggregator threads
Main thread: Waiting for Aggregator to finish
Thraed1: computing my part
Thraed2: computing my part
Thraed3: computing my part
Thread1: got one line ready, calling aggregator
AggregatorThraed: I'm awake, computing the sum of row from thread 1
Thread2: got one line ready, calling aggregator
AggregatorThraed: I'm awake, computing the sum of row from thread 2
Thread1: got one line ready, calling aggregator
AggregatorThraed: I'm awake, computing the sum of row from thread 1
Thread3: got one line ready, calling aggregator
AggregatorThraed: I'm awake, computing the sum of row from thread 3
Thread3: got one line ready, calling aggregator
AggregatorThraed: I'm awake, computing the sum of row from thread 3
Thread2: got one line ready, calling aggregator
AggregatorThraed: I'm awake, computing the sum of row from thread 2
AggregatorThraed: I'm done aggregating all 6 rows. Joining!
Main thread: Aggregator Joined. The aggregate vector =
<441,441,441,441,441,441>

In your readme file, explain what locking primitives you used and WHY?