EE 382C: Multicore Computing
Assignment 1

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)
TA: Changyong Hu (email: colinhu9@utexas.edu)

Deadline: September 8, 2016

This homework contains a programming part (Q1-Q2) and a theory part (Q3-Q5). The theory part should be written or typed on a paper and submitted at the beginning of the next class. The source code (Java files) of the programming part must be uploaded to the canvas before the end of the due date (i.e., 11:59pm on September 8). The assignment should be done in teams of two. Please zip and name the source code as [EID1_EID2].zip. You should use the templates downloaded from the course github (https://github.com/vijaygarg1/Option3-Multicore). You should not change the file names and function signatures. In addition, you should not use package for encapsulation. Note that, we provide some test cases for your convenience in SimpleTest class. You do not need to modify the class.

1. **(35 points)** Write a Java class `PSort` that allows parallel sorting of an array of integers. It provides the following `static` method:

   ```
   public class PSort {
     public static void parallelSort(int[] A, int begin, int end) {
       // your implementation goes here.
     }
   }
   ```

   Use *QuickSort* algorithm for sorting and *Runnable* interface for your implementation. In quick sort, the input array is divided into two sub-arrays, and then the two sub-arrays are sorted recursively using the same algorithm. In your implementation, you should create two new threads for sorting the divided sub-arrays.

   The input array `A` is also the output array, which means your should put the sorted data back in `A`. The range to be sorted extends from index begin, inclusive, to index end, exclusive. In other words, the range to be sorted is empty when `begin == end`. To simplify the problem, you can assume that the size of array `A` does not exceed 10,000.

2. **(35 points)** Write a Java class that allows parallel search in an array of integers. It provides the following `static` method:

   ```
   public class PSearch {
     public static int parallelSearch(int x, int[] A, int numThreads) {
       // your implementation goes here.
     }
   }
   ```

This method creates as many threads as specified by `numThreads`, divides the array `A` into that many parts, and gives each thread a part of the array to search for `x` sequentially. If any thread finds `x`, then it returns an index `i` such that `A[i] = x`. Otherwise, the method returns -1. Use *Callable* interface for your implementation.

3. **(15 points)** Show that any of the following modifications to Peterson's algorithm makes it incorrect:

   a) A process in Peterson's algorithm sets the *turn* variable to itself instead of setting it to the other process.

   b) A process sets the *turn* variable before setting the *wantCS* variable.

4. **(15 points)** Show that the bakery algorithm does not work in the absence of *choosing* variables.

5. **(0 points), do not submit** You are one of the recently arrested prisoners. The warden, a deranged computer scientist, makes the following announcement:

   - All prisoners may meet together today and plan a strategy, but after today you will be in isolated cells and have no communication with one another.
   - I have setup a "switch room" which contains a light switch, which is either on or off. The switch is not connected to anything.
   - Every now and then, I will select one prisoner at random to enter the "switch room". This prisoner may throw the switch (from on to off, or vice-versa), or may leave the switch unchanged. Nobody else will ever enter this room while the prisoner is in the room.
   - Each prisoner will visit the "switch room" arbitrarily often. More precisely, for any N, eventually each of you will visit the "switch room" at least N times.
   - At any time, any of you may declare: "we have all visited the 'switch room' at least once". If the claim is correct, I will set you free. If the claim is incorrect, I will feed all of you to the crocodiles. Choose wisely!

   Devise a winning strategy when you know that the initial state of the switch is off. (Hint: not all prisoners need to do the same thing.)