EE 360P: Concurrent and Distributed Systems
Assignment 1

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)
TA: Asad S. Malik (email: asadsm@utexas.edu)
TA: Tushar S Chouhan (email: tushar.chauhan94@utexas.edu)
TA: Xiong Zheng (email: zhengxiongtym@utexas.edu)
TA: Eyenunwana I Nwoko (email: einwoko96@utexas.edu)

Deadline: Feb. 6^{th}, 2018

This homework contains a programming part (Q1-Q2) and a theory part (Q3-Q7). The theory part should be written or typed on a paper and submitted at the beginning of the class. The source code (Java files) of the programming part must be uploaded through the canvas before the end of the due date (i.e., 11:59pm in Feb. 6^{th}). The assignment should be done in teams of two. You should use the templates downloaded from the course github (https://github.com/vijaygarg1/EE-360P.git). 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. Please zip and name the source code as [EID1_EID2].zip.

1. **(25 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 quicksort, the input array is divided into two sub-arrays , and then the two sub-arrays are sorted recursively using the same algorithm. If at any stage, the array size is less than or equal to 16, then simple sequential insert sort is used for sorting. In your implementation, you should create two new runnable tasks for sorting two subarrays. You should use ExecuterService and Futures to sort these subarrays in parallel. The input array `A` is also the output array. 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. **(25 points)** Write a Java class that performs a parallel merge of two arrays of integers. It provides the following `static` method:

```
public class PMerge {
       public static int parallelMerge(int[] A, int[] B, int[] C, int numThreads) {
            // your implementation goes here.
      }
}
```

This method uses as many threads as specified by `numThreads`. It merges arrays `A` and `B` into the array `C`. You may assume that both arrays `A` and `B` are sorted and that there are no duplicates in `A` and `B`.

3. **(5 points)**

   Create a TACC UserID and submit it along with Homework questions.
   Link: https://portal.tacc.utexas.edu/

4. **(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.

5. **(15 points)** Peterson's algorithm uses a multi-write variable `turn`. Modify the algorithm to use two variables `turn0` and `turn1` instead of `turn` such that P_0 does not write to `turn1` and P_1 does not write to `turn0`.

6. **(15 points)** Show that the bakery algorithm does not work in the absence of *choosing* variables.