

EE 360P: Concurrent and Distributed Systems

Assignment 1

Instructor: Professor Vijay K. Garg

Deadline: 11:59pm Feb. 5th, 2019

This homework contains a programming part (Q1-Q2) and a theory part (Q3-Q6). The theory part should be either typed and submitted as a pdf file or written on a paper, scanned and then submitted online as a pdf file. The file should be named [Theory-EID1_EID2].pdf. This file should be submitted in the directory that has the source code for the programming part. The source code (Java files) of the programming part and the pdf file for the non-programming part must be uploaded through the canvas before the deadline (i.e., 11:59pm on Feb. 5th). There will be a late penalty of 10% of the total grade for each day the assignment is late. 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. The zip folder you submit should contain only java files with no sub-folders. The source files for both questions should be in the same folder. 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. 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 the simple sequential insert sort is used for sorting. In your implementation, you should sort two subarrays in parallel. You should use `ForkJoinPool` for parallel implementation. 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) {
```

```

    //arrays A and B are sorted in the ascending order
    // These arrays may have different sizes.
        //array C is the merged array sorted in the descending order
    // your implementation goes here.
}
}

```

This method uses as many threads as specified by `numThreads`. It merges sorted arrays `A` and `B` into the array `C` sorted in the reverse order. You may assume that both arrays `A` and `B` are sorted and that there are no duplicates in `A` and `B`. (i.e., all elements in `A` are unique, all elements in `B` are unique; however, `A` and `B` may have elements in common. The elements that are common must appear in `C` as duplicates.

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. The remaining algorithm stays the same.
- b) A process sets the *turn* variable before setting the *wantCS* variable.

5. (15 points) Prove that Peterson's algorithm is free from starvation.

6. (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`.