# EE 360P: Concurrent and Distributed Systems
# Assignment 2

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)

**Deadline: 11:00 am Feb. 15th, 2018**

The submissions must be uploaded to the canvas by the deadline mentioned above. Please submit one zip file per team, with the name format [EID1_EID2].zip. You should use the templates downloaded from the course github (https://github.com/vijaygarg1/EE-360P.git).

1. (a) **(15 points)** A ThreadSynch is a synchronization aid that allows a set of threads to wait for all other threads to reach a common point. Whenever a thread calls the method await, it gets blocked until the number of parties specified in the constructor have also called await. When the specified number of threads have called await, all threads are released.

   Your implementation must allow a ThreadSynch to be reused after the waiting threads are released. So, a thread may call await a multiple number of times. You need to implement the following methods using *semaphores*:

```
public class ThreadSynch {
  public ThreadSynch(int parties) {
    // Creates a new ThreadSynch that will release threads only when
    // the given number of threads are waiting upon it
  }
  int await() throws InterruptedException {
    // Waits until all parties have invoked await on this ThreadSynch.
    // If the current thread is not the last to arrive then it is
    // disabled for thread scheduling purposes and lies dormant until
    // the last thread arrives.
    // Returns: the arrival index of the current thread, where index
    // (parties - 1) indicates the first to arrive and zero indicates
    // the last to arrive.
  }
}
```

   (b) **(15 points)** Implement `MonitorThreadSynch` using Java Monitor. Its interface is identical to `ThreadSynch`. You are not allowed to use semaphores for its implementation.

2. **(35 points)** Implement a Java class `FairReadWriteLock` that synchronizes reader and writer threads using monitors with `wait, notify` and `notifyAll` methods. The class should provide the following methods: `void beginRead()`, `void endRead()`, `void beginWrite()`, and `void`

`endWrite()`. A reader thread only invokes `beginRead()` and `endRead()` while a writer thread only invokes `beginWrite()` and `endWrite()`. In addition, the lock (instance of this class) should provide the following properties:

(a) There is no read-write or write-write conflict.

(b) A writer thread that invokes `beginWrite()` will be blocked until all preceding reader and writer threads have acquired and released the lock.

(c) A reader thread that invokes `beginRead()` will be blocked until all preceding writer threads have acquired and released the lock.

The precedence of threads is determined by the timestamp (sequence number) that threads obtain on arrival.

3. **(35 points)** Use Java ReentrantLock and Condition to implement a linked list based priority queue `PriorityQueue`. Each node in the queue has two fields: name (a String) and priority (an integer in the range 0..9). The order of the nodes is always kept sorted based on the priority with 9 as the highest priority and 0 as the least. Nodes with the same priority number are kept in the order of inserts. Furthermore, you should not use a global lock for the queue. Your `PriorityQueue` should support the following methods.

```
  public class PriorityQueue {
    public PriorityQueue(int capacity) {
      // Creates a Priority queue with maximum allowed size as capacity
    }
    public int add(String name, int priority) {
      // Adds the name with its priority to this queue.
      // Returns the current position in the list where the name was inserted;
      // otherwise, returns -1 if the name is already present in the list.
      // This method blocks when the list is full.
}
    public int search(String name) {
      // Returns the position of the name in the list;
      // otherwise, returns -1 if the name is not found.
}
    public String getFirst() {
      // Retrieves and removes the name with the highest priority in the list,
      // or blocks the thread if the list is empty.
} }
```