

## Homework 7

Professor: Maurice Herlihy

TA: cs1760tas@lists.brown.edu

**Problem 1.** Figure 1 shows a proposed implementation of a `RateLimiter` class, which runs jobs but limits the “weight” of the jobs started per minute using a `quota`, which is increased to `LIMIT` every minute by a separate thread. We want to guarantee that jobs will run promptly if there is enough `quota`. You may assume a fast processor and fair scheduler, so that the `RateLimiter` reaches a quiescent state (all jobs are sleeping in `await()` or running), if possible, before each call to `increaseQuota()`.

- a) Describe the distributions of `weight` values ( $0 \leq \text{weight} \leq \text{LIMIT}$ ) under which this implementation works or fails and explain why.
- b) Fix this implementation so it allows jobs to have any `weight` value from 0 to `LIMIT`, and describe how it may impact performance.

**Problem 2.** The combining tree barrier uses a single thread-local sense field for the entire barrier. Suppose instead we were to associate a thread-local sense with each node as in Figure 2. Either:

- Explain why this implementation is equivalent to the other one, except that it consumes more memory, or.
- Give a counterexample showing that this implementation is incorrect.

**Problem 3.** A *dissemination barrier* is a symmetric barrier implementation in which threads spin on statically-assigned locally-cached locations using only loads and stores. As illustrated in Figure 3, the algorithm runs in a series of rounds. At round  $r$ , thread  $i$  notifies thread  $i + 2(\bmod n)$ , (where  $n$  is the number of threads) and waits for notification from thread  $i - 2(\bmod n)$ .

For how many rounds must this protocol run to implement a barrier? What if  $n$  is not a power of 2? Justify your answers. (Hint: argue by induction on  $n$ .)

**Problem 4.** This is problem related to counting networks. Recall that a sequence  $X = x_0, \dots, x_n$  is *k-smooth* if the difference between the largest and smallest value in the sequence is  $k$  or less.

Let  $X$  and  $Y$  be sequences of length  $n$ . A *matching* layer of balancers for  $X$  and  $Y$  is one where each element of  $X$  is joined by a balancer to some element of  $Y$  in an arbitrary one-to-one correspondence.

Prove the following: if  $X$  and  $Y$  are each  $k$ -smooth, and  $Z$  is the  $2n$ -long result of matching  $X$  and  $Y$ , then  $Z$  is  $(k + 1)$ -smooth.

```

1 public class RateLimiter {
2     static final int LIMIT = 100; // example value
3     public int quota = LIMIT;
4     private Lock lock = new ReentrantLock();
5     private Condition needQuota = lock.newCondition();
6     public void increaseQuota() { // called once per minute
7         synchronized(lock) { // grab the lock
8             if (quota < LIMIT) { // if some of the quote has been used up:
9                 quota = LIMIT; // increase quota to LIMIT
10                needQuota.signal(); // wake up a sleeper
11            }
12        } // unlock
13    }
14    private void throttle(int weight) {
15        synchronized(lock) { // grab the lock
16            while (quota < weight) { // while not enough quota:
17                needQuota.await(); // sleep until increased
18            }
19            quota -= weight; // claim my job's part of the quota
20            if (quota > 0) { // if still quota left over:
21                needQuota.signal(); // wake up another sleeper
22            }
23        } // unlock
24    }
25    public void run(Runnable job, int weight) {
26        throttle(weight); // sleep if under quota
27        job.run(); // run my job
28    }
29 }

```

Figure 1: A proposed RateLimiter class implementation

```

1  private class Node {
2      AtomicInteger count;
3      Node parent;
4      volatile boolean sense;
5      int d;
6      // construct root node
7      public Node() {
8          sense = false;
9          parent = null;
10         count = new AtomicInteger(radix);
11         ThreadLocal<Boolean> threadSense;
12         threadSense = new ThreadLocal<Boolean>() {
13             protected Boolean initialValue () { return true; };
14         };
15     }
16     public Node(Node myParent) {
17         this ();
18         parent = myParent;
19     }
20     public void await() {
21         boolean mySense = threadSense.get();
22         int position = count.getAndDecrement();
23         if ( position == 1) { // I'm last
24             if (parent != null) { // root?
25                 parent.await ();
26             }
27             count.set (radix); // reset counter
28             sense = mySense;
29         } else {
30             while (sense != mySense) {};
31         }
32         threadSense.set (!mySense);
33     }
34 }

```

Figure 2: Thread-local tree barrier.

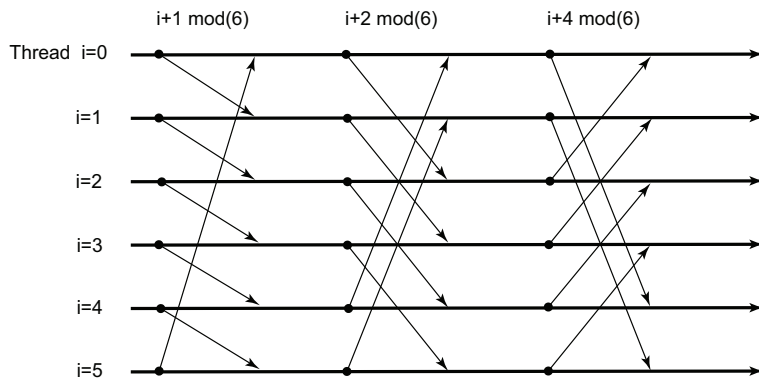


Figure 3: Communication in the dissemination barrier. In each round  $r$  a thread  $i$  communicates with thread  $i + 2^r \pmod{n}$ .