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 ≤ weight ≤ 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 + 2r (mod n), (where n is the number of threads) and waits for notification from thread i + 2r (mod 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 = x0, . . . , xn 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.
public class RateLimiter {
    static final int LIMIT = 100; // example value
    public int quota = LIMIT;
    private Lock lock = new ReentrantLock();
    private Condition needQuota = lock.newCondition();
    public void increaseQuota() { // called once per minute
        synchronized (lock) { // grab the lock
            if (quota < LIMIT) { // if some of the quote has been used up:
                quota = LIMIT; // increase quota to LIMIT
                needQuota.signal(); // wake up a sleeper
            }
        } // unlock
    }
    private void throttle (int weight) {
        synchronized (lock) { // grab the lock
            while (quota < weight) { // while not enough quota:
                needQuota.await(); // sleep until increased
            }
            quota -= weight; // claim my job's part of the quota
            if (quota > 0) { // if still quota left over:
                needQuota.signal(); // wake up another sleeper
            }
        } // unlock
    }
    public void run(Runnable job, int weight) {
        throttle (weight); // sleep if under quota
        job.run(); // run my job
    }
}

Figure 1: A proposed RateLimiter class implementation
private class Node {
    AtomicInteger count;
    Node parent;
    volatile boolean sense;
    int d;
    // construct root node
    public Node() {
        sense = false;
        parent = null;
        count = new AtomicInteger(radix);
        ThreadLocal<Boolean> threadSense;
        threadSense = new ThreadLocal<Boolean>() {
            protected Boolean initialValue() { return true; }
        };
    }
    public Node(Node myParent) {
        this();
        parent = myParent;
    }
    public void await() {
        boolean mySense = threadSense.get();
        int position = count.getAndDecrement();
        if (position == 1) { // I'm last
            if (parent != null) { // root?
                parent.await();
            }
            count.set(radix); // reset counter
            sense = mySense;
        } else {
            while (sense != mySense) {};
        }
        threadSense.set(!mySense);
    }
}

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