Due: Tuesday, October 16

Unless otherwise indicated, each question is worth 10 points. You may work with one partner and sign both of your names to your paper. You will receive the same grade.

1. Recall the matrix multiplication involving Linda that we discussed in class. Imagine a failure model in which (a) any number of processes may fail at any time, but (b) the tuple space will not fail (c) and after a process fails, another process on the same computational device may take its place. Design an algorithm to perform the matrix multiplication having the properties that will produce the right answer if it terminates and that will terminate if at least one process stays up for a sufficiently long time.

2. (20 points) Find the Chord implementation in chapter/section 6.3 and appendix B.2 of Boon Loo Thau’s thesis (available from his web page at the University of Pennsylvania). Modify it so that for a network having n nodes, no node has more than ceiling(n/3) entries in the finger table but every lookup requires at most 3 hops. (Hint color each node one of three colors and have each red node point to all blue nodes, each blue node point to all the green nodes, and each green node point to all the red nodes.) Your implementation should maintain this structure after failure.

3. Using synchronized time, simultaneity is an equivalence relation (transitive, reflexive, symmetric). Define two events A and B to be Lamport-simultaneous if NEITHER of the following hold A → B nor B → A, where → is the Lamport ordering.

Is Lamport-simultaneity reflexive? symmetric? transitive? (Show counter-examples if not.)

4. The formal motivation for wait-free algorithms is Herlihy’s model:

Any process may suspend execution at any time (due to programmer error, an operating system error, sunspots, or whatever) except during one of the approved atomic operations.

One problem he poses is: given a single register initialized to zero, where the only given operation is F&A (but you can use as many of these as you like), for how many processes can one solve the “who went first” problem. Solving the problem means the following: if one process actually does touch the register first, then all processes that don’t hang should be able to determine which one went first. The protocol need not guarantee anything about the output of a process that hangs.

a) Could the following work?

Process 1:

---

1A relation R is transitive if for all a, b, c, aRb and bRc implies aRc. A relation is reflexive if for all x, xRx. A relation is symmetric if for all x, y, xRy implies yRx.
temp1 = F&A(reg, 7);
if temp1 > 0 then F&A(reg, -7);
if temp1 = 0 then return process 1 went first;
while temp1 != 13 and temp1 != 15 do
    temp1 := F&A(reg,0)
end while;
if temp1 = 13 then return process 2 went first;
if temp1 = 15 then return process 3 went first;

Process 2:

    temp2 = F&A(reg, 13);
    if temp2 > 0 then F&A(reg, -13);
    if temp2 = 0 then return process 2 went first;
    while temp2 != 7 and temp2 != 15 do
        temp2 := F&A(reg,0)
    end while;
    if temp2 = 7 then return process 1 went first;
    if temp2 = 15 then return process 3 went first;

Process 3:

    temp3 = F&A(reg, 15);
    if temp3 > 0 then F&A(reg, -15);
    if temp3 = 0 then return process 3 went first;
    while temp3 != 7 and temp3 != 13 do
        temp3 := F&A(reg,0)
    end while;
    if temp3 = 7 then return process 1 went first;
    if temp3 = 13 then return process 2 went first;

Intuitively, a process that discovers that its local temp is not 0 backs itself out, eliminating the commutativity problem.

Does this protocol work? Prove that it does or show a counterexample.

b) Suppose we relax the assumption that all processes try to be the first to modify the shared register and assume instead that only two processes make this attempt. That is, the third process only performs F&A(reg,0)’s, effectively reading the register’s value.

Is there a protocol that determines which of the first two goes first? Show one or prove that one can’t exist.

c) Suppose there were a new instruction ExF&A(sourcereg, destreg, x) that returns the current contents of sourcereg, and puts the sum of those contents and x into destreg. Assuming that this instruction is atomic in the sense that if one ExF&A accesses a register over a given time interval, then no other ExF&A (or F&A) can do so during that time interval. (It’s as if you lock both regs, do the ExF&A, then release locks.)

Now, assuming you can use ExF&A as well as F&A and that you have two registers reg1 and reg2 both initialized to 0, readdress the problem of 2b with the assumption that only
two processes attempt to modify any register (the third process only issues $F&A(\text{reg}1,0)$ or $F&A(\text{reg}2,0)$). That is, try to design a protocol or show that no one exists.

5. Suppose $v = 10$, $e = 2$, and $k = 0.01$. All units in seconds. Design a protocol that will keep two agents synchronized to within 3 seconds (i.e., clocks must be at most 3 seconds apart). Assume they are synchronized exactly when they begin. Clocks should not go backwards. You need not use Lamport’s approach.

Try to use as few messages as possible.

6. Design a Petri net to decide whether an elevator should stop at a given floor assuming there are many possible elevators. You should take the following considerations into account: (i) only one elevator need stop on a floor per given direction (up or down); (ii) the elevator should be going in that direction (or be stopped); (iii) the elevator that stops should either not be full or contain people who want to get off at that floor.

7. (Extra credit, 10 points) Give us a running implementation of your modified Chord algorithm using the software from http://p2.berkeley.intel-research.net/p2sw.php.