22C: 166 Distributed Systems and Algorithms
Homework 2, Total points = 60

Assigned 9/21/11 due 9/28/11

Please submit typewritten solutions through ICON, preferably in the pdf format. Late assignments will not be accepted without prior approval.

Question 1 (20 points): In the class we have discussed only about message passing algorithms for mutual exclusion, but have not looked at the shared-memory algorithms. We now study a mutual exclusion algorithm on the shared memory model. The following two-process algorithm claims to solve the mutual exclusion algorithm. Both processes have read and write access to the shared variables x and y:

```
        program     resolve  {program for process i:  i ∈ {1, 2}}
        define      x,: integer, y: boolean
        initially   y = false

        do true →
start:  x:= i;
                if  y →
                        do y → skip od; {busy waiting lop}
                        goto start;
                fi;
        y := true
                if x ≠ i →
                        y:= false;
                        do x ≠ 0 → skip od; {busy waiting lop}
                        goto start;
                fi;
        critical section;
        y:= false; x:= 0
        non-critical section;
        od
```

(a) Does it satisfy the requirements of a correct solution?

(b) If N processes 1, 2, …, N (N > 2) execute the above algorithm, then what is the *maximum number* of processes that can be in their critical sections concurrently?

Briefly explain your answers.

Question 2 (25 points): The dining philosophers' problem is a classic problem in concurrent programming. It was designed by Dijkstra in 1965 as a class exercise for students, and since then this problem and its variations have been studied by numerous researchers. Here is a statement of the problem:

Five philosophers sit at a table around a bowl of spaghetti. A fork is placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. Eating is not limited by the amount of spaghetti left: assume an infinite supply. However, a philosopher can only eat while holding both the fork to the left and the fork to the right. Each philosopher can pick up an adjacent fork, when available, and put it down, when holding it. These are separate actions: forks must be picked up and put down one by one. The problem is how to design a discipline of behavior (by specifying the program for a philosopher) such that each philosopher can forever continue to alternate between eating and thinking, and thus does not starve to death. Do not use global variable, or a global cache, since that would spoil the fun. (Feel free to look at the Wikipedia page for discussion on this problem).

(a) First, think of a simple approach: let each philosopher, when he/she becomes hungry, grabs the left fork first and then the right fork, one after another (when they become available). After finishing eating, each philosopher puts down the right fork first and then the left fork. What problem do you foresee?

(b) Now, using the message-passing model, propose a solution of the dining philosophers problem. You can assume that a philosopher can send a time-stamped request to the left neighbor for the left fork and to the right neighbor for the right fork. Note that each philosopher can only communicate with his/her right and left neighbors. Briefly argue why your solution will work.

Question 3 (15 points): Propose a method using which any node in a large network can compute the topology of the network. Assume that initially each node knows about its immediate neighbors only.