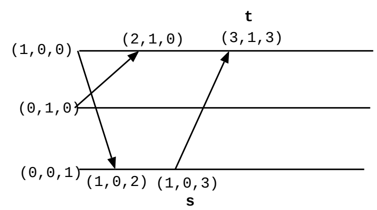
EE 382N Practice Questions

1. Assume that you have implemented Mattern's vector clock algorithm. However some application needs Lamport's time stamps. Write a function *convert ()* that takes as input Mattern's time stamp and outputs Lamport's time stamps.
Solution:
Note that $\forall s,t : s \rightarrow t \Rightarrow LC[s] < LC[t]$ where LC is the Lamport's clock.
For a vector clock VC, the following is true:
$$\forall s,t : s \rightarrow t \Rightarrow \sum_i s.VC[i] < \sum_i t.VC[i]$$
where i is the processor index. This claim follows from $s \rightarrow t \Rightarrow s.VC < t.VC$ (each component of $s.VC$ is less than or equal to the corresponding component of $t.VC$ and one of the components is strictly smaller). Thus, define a function *convert()* that returns the sum of the individual vector components. NOTE: returning $max(i)$ will not always work, as the following counterexample shows. Here, $s \rightarrow t$, but $max(s.VC) \not< max(t.VC)$



2. Let s.v be the vector clock for the local state s. Given n vectors, one from each state, one can form a matrix to represent the cross product of the local states which can be viewed as the global state. Give a suitable condition on the matrix for this global state to be consistent.
Solution:

Let M be the matrix representing the global state G. Thus, $G[i]$ is the local state on process i and $M[i,j]$ denote the jth component of the vector clock at state $G[i]$.

We claim that G is a consistent global state iff:
$\forall i,j : i \neq j : M[i,i] > M[j,i]$.
Proof:
G is a consistent global state
= { definition of consistency }
$\forall i,j : i \neq j : G[i] \not\rightarrow G[j]$.
= { property vector clocks }
$\forall i,j : i \neq j : G[i].v[i] > G[j].v[i]$.
= { definition of M }
$\forall i,j : i \neq j : M[i,i] > M[j,i]$.

Note that we have proved an iff statement using equivalences rather than implications.

3. A vector clock designer has come up with the idea that instead of process P_i sending the entire vector to P_j every time, it should only send those components of the vector that are different from the last time it sent a message to P_j. What are the advantages and disadvantages of this scheme ?
Solution:
Disadv: (1) One big disadvantage is that it assumes FIFO communication between processes. If FIFO does not hold, vector clock properties will not be satisfied. (2) Requires each process to maintain the last vector sent to every other process and last vector received from every other process. This has $O(n^2)$ memory overhead. (3) A minor disadvantage is that additional calcualation is required during each send and receive.
Advantage: Reduced communication complexity on average.

4. To totally order all the events in a distributed system (preserving happened-before relation), one can use Lamport's logical clock with process id. Two events e and f are then ordered as

$$e < f \equiv (e.c < f.c) \vee ((e.c = f.c) \wedge (e.p < f.p))$$

This ordering favors the small numbered processes when two events have the same value for Lamport's clock. Modify the scheme so that this unfairness is removed.
Solution:
Extending the logical clock with process number always favors processes with smaller idenfiers. However, the oddness and evenness attributes of the logical clock values can be used along with the process numbers. In case of a tie, new scheme favors the process with smaller identifier if the common logical clock value is even.

$$\begin{aligned} e < f \quad \equiv \quad & (e.c < f.c) \vee \\ & ((e.c = f.c) \wedge (e.p < f.p) \wedge (even(e.c)) \vee \\ & ((e.c = f.c) \wedge (e.p > f.p) \wedge (odd(e.c)) \end{aligned}$$

5. The mutual exclusion algorithm by Lamport requires that any request message be acknowledged. Under what conditions does a process not need to send an *acknowledgement* message for a *request* message?
Solution:
A process P_i need not send an acknowledgment for a request with logical time t if it has also requested critical section. If its request has smaller timestamp than t, then we know that P_i will enter the critical section first. It will then send the release message which will serve as an acknowledgment to the request. If its request has larger timestamp than t, then that message will serve as an acknowledgment (it will make the component for P_i in the direct dependency vector of the requesting process bigger). Also, if the logical time of the request is lower than any message P_i has sent to the requesting process then there is no need to send the acknowledgment.

6. The centralized WCP detection algorithm requires an application process to send its vectors whenever its local predicate becomes true for the first time since the most recent send or the beginning of the execution. Will the algorithm still work if every process sent Mattern's vector whenever its local predicate became true for the first time since the most recent *receive* or the beginning of the execution ? Justify your answer.
Solution:
The algorithm will not work. The counter example is given below.

P1: F, message m1 received, T, message m2 sent, T

P2: F, message m1 sent, F, message m2 received, T

7. Show how you will modify the centralized algorithm for mutual exclusion so that requests are granted in the order (happened-before order) they are made.
Solution:
We require that if s and t are two local states corresponding to request events and $s \rightarrow t$, then the request s must be honored before the request t. Thus, even if the request t reaches the coordinator before s, the coordinator should be able to infer that there is a pending request from the process $P_{s.p}$ and thus wait for that request to arrive. Because $s \rightarrow t$, the state t has the information about the request s. Thus, the process $P_{t.p}$ only needs to relay to the coordinator all the requests that it knows. The coordinator can delay receiving the message t until all requests before t are received. The information about requests known at the state s can be concisely captured using a vector $s.v$ where $s.v[j]$ represents the number of requests made by the process P_j that causally precede the state s. All the client processes need to implement a vector-clock type algorithm that maintains this vector (by propagating the vector in the messages etc.).

8. Prove or disprove concurrent with is a transitive relation.
Solution:
Counter example: Consider processes P_1 and P_2. Consider states s and t on process P_1 such that $s \rightarrow t$. Consider state u on P_2 such that $s||u$ and $u||t$. If concurrent with were a transitive relation, we would have $s||t$ which is not true.