Assignment 5 - ECE 454

Piyush Gadigone, 20366910
Yash Malik, 20379044

1.
Consider 'n' processes in a distributed environment. Let process P_i send a message 'm' to all the 'n' processes. Let 'a' be the event where the message 'm' is sent by P_i and 'b_r' be the receipt of the message 'm' for any process P_r. We have to prove that C(a) < C(b).

For process P_i, since a -> b, $C_i(a) < C_i(b)$ using κ_1.
For any process P_r, $C_r(b_r) > C_i(a)$ as deduced from κ_2. This means that for all the n processes that received the message 'm' now have a timestamp greater than $C_i(a)$.
Using these inequalities, for any process, C(b) > C(a), hence proving that C(.) satisfied κ_0.

2.
Consider 'n' processes in a distributed environment. Let 'a' be an event that happens at P_i and 'b' be an event that happens at P_j. Additionally, let $VC_i[j]$ denote the number of events that P_i knows that happened at P_j. To prove that $a \rightarrow b$ if and only if VC(a) <VC(b), we have to prove the following:

<u>(1) If $a \rightarrow b$ then VC(a) < VC(b)</u>
We know that VC(a)[j] < VC(b)[j] by definition of how vector clocks work where the vector clock at index 'j' is always incremented by process P_j when an event happens at j. Also, all the other indices are less than or equal to the value at VC(b) as VC(b) always takes the maximum value at each index. Using this, we prove that VC(a) < VC(b).

<u>(2) If VC(a) < VC(b) then $a \rightarrow b$</u>
If VC(a) < VC(b), then there exists at least an index in VC(b) that is greater than VC(a). One of these indices is certainly 'j'. All the other indices in VC(a) is less than or equal to the indices in VC(b) which implies that P_j knows of all the events that happened before P_i. This implies that $a \rightarrow b$.

Using the above two statements, we prove that $a \rightarrow b$ if and only if VC(a) <VC(b).

3.
Yes, this achieves causally ordered multicast.
Consider 'n' processes that are in a multicast group in a distributed environment. Let m_1 causally precede m_2. Then using Lamport clock, we know that $C(m_2) > C(m_1)$. In the proof on slide 22, since we use acknowledgements from all the processes in the multicast group, we can ensure that totally ordered multicast is achieved i.e. m_1 is delivered by all the processes to its application before m_2 as the timestamp on m_2 is higher than the timestamp on m_1. Hence the messages are also delivered in causal order.

4.

Assume there are ‘n’ processes. One of the processes P that wishes to form a logical ring can send a broadcast request to all the processes. The processes who wish to join the ring can respond with their process ids. The process P now creates a sorted list based on all the process ids. It sends the list to the process id which has the smallest value. Let this be process Q. Q then sets its successor to the next smallest process id in the list and sends the list to the successor. The successor follows the same strategy as Q. Following this pattern, a logical ring is formed. The space complexity for this algorithm is O(n) per process for the list and the time complexity is O(n) as the list has to traverse once through all the ‘n’ processes.

5.
A decentralized algorithm is unbounded because the peer can keep requesting for a resource if it constantly receives less than ‘m’ votes. This can lead to starvation.

One way to bound this is by limiting the amount of time a resource can be held by any peer. If a peer needs to hold the resource for longer than this duration, it has to release the resource and request it again. The advantage of this system is that the resource will eventually be granted to anyone who requests for it, thereby avoiding starvation. When a peer requests a resource, it attaches its timestamp to access the resource. If the request is not granted, the peer requests the resource again with the original timestamp. When multiple peers request a resource at the same time, the peer with the smallest timestamp gets access. Hence this gives an order when multiple requests are made for the same resource.

The tight lower bound for this system is 3m, where ‘m’ are the minimum number of votes required to access a resource. The reason why it is 3m is because one REQUEST message is sent to ‘m’ coordinators and all the ‘m’ coordinators respond back with an OK message. Once the resource has been utilized, the peer sends ‘m’ RELEASE messages.

For finding the tight upper bound, we need the following additional variables:

- T - Max time duration that a resource can be sent.
- I - Time interval between sending a request for a resource
- n - Total number of peers

The tight upper bound to access a resource would be when all the peers request access to a resource and for each request ‘floor(T/I + 1)’ messages are sent.
Tight upper bound: 3mn* floor(T/I + 1)

6.
Token ring algorithm is unbounded because the token can infinitely keep traveling in the ring when no process requires a token. The goal is to prevent this infinite transfer of token when the token is not required.

Consider a process P which currently has the token and is accessing a resource. Once the process finishes accessing the resource, it sends the token along with its process id to its successor. The process id

indicates the last process that used the token. If a process doesn't require the token, it just forwards the token along with the process id in it. If a process requires a token, it utilizes the resource and forwards the token along with its process id. As this pair travels, the process checks if the last used process id in the pair matches its own id. If it does, it means that no one in the ring requires the token at the given moment. It then stops forwarding the token and a STOP message along the ring with its process id to tell that the token forwarding has been stopped. If after a certain duration, some other process requires the token, it sends a REQUEST message along the ring to the process which has the token. The token forwarding then resumes back.

The tight-lower bound to access a resource is 1 if a process wants the token and the token is with its predecessor. The tight-upper bound is 'n' which happens when a process sends a REQUEST message to get the token.

7.
The output 001110 is not legal. The last two digits should be 11 if the processes run in any order. Before the last print statement, all the variables x, y and z should have been initialized to 1. Hence, the last two digits have to be 11.