Homework 2 (Multicast, Mutual Exclusion, Leader Election) - 100 Points

Solutions

CS425/ECE 428 Distributed Systems, Fall 2009, Instructor: Klara Nahrstedt

Out: Tuesday, September 22, **Due Date**: Tuesday, October 6

Instructions: (1) Please, hand in hardcopy solutions that are typed (you may use your favorite word processor). We will not accept handwritten solutions. Figures and equations (if any) may be drawn by hand. (2) Please, start each problem on a fresh sheet and type your name at the top of each sheet. (3) Homework will be due at the **beginning of class** on the day of the deadline.

Relevant Reading for this Homework: Chapter 12.1-12.4

Problem 1: Ordered Multicast – 20 Points

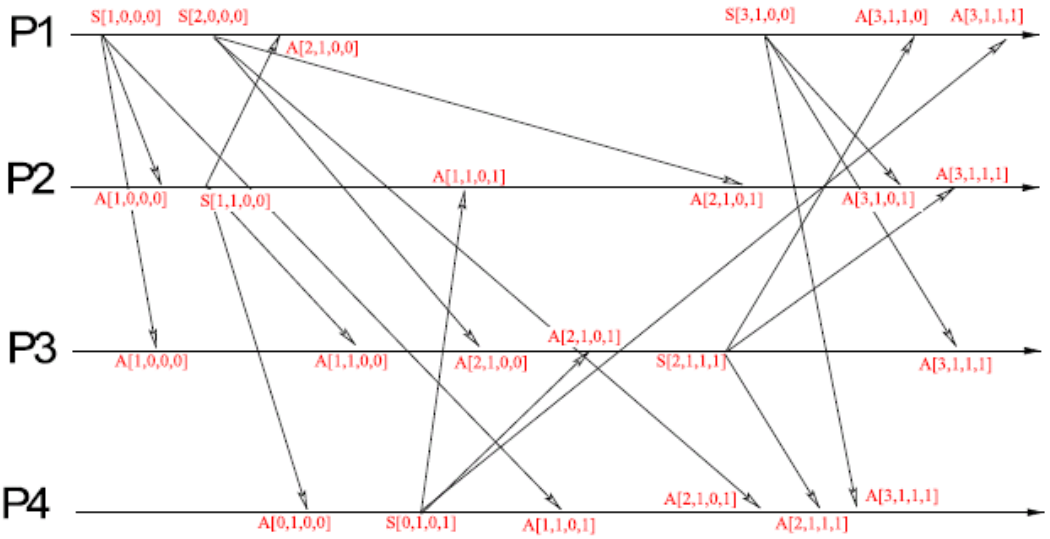
Consider the Figure 1. Using sequence numbers (for FIFO Ordering multicast) or vector clocks (for Causal Ordering multicast) , mark timestamps at the point of each multicast send and each multicast receipt. Also mark multicast receipts that are buffered, along with the points at which they are delivered to the application

a. **(10 Points)** FIFO Ordering multicast algorithm as discussed in class

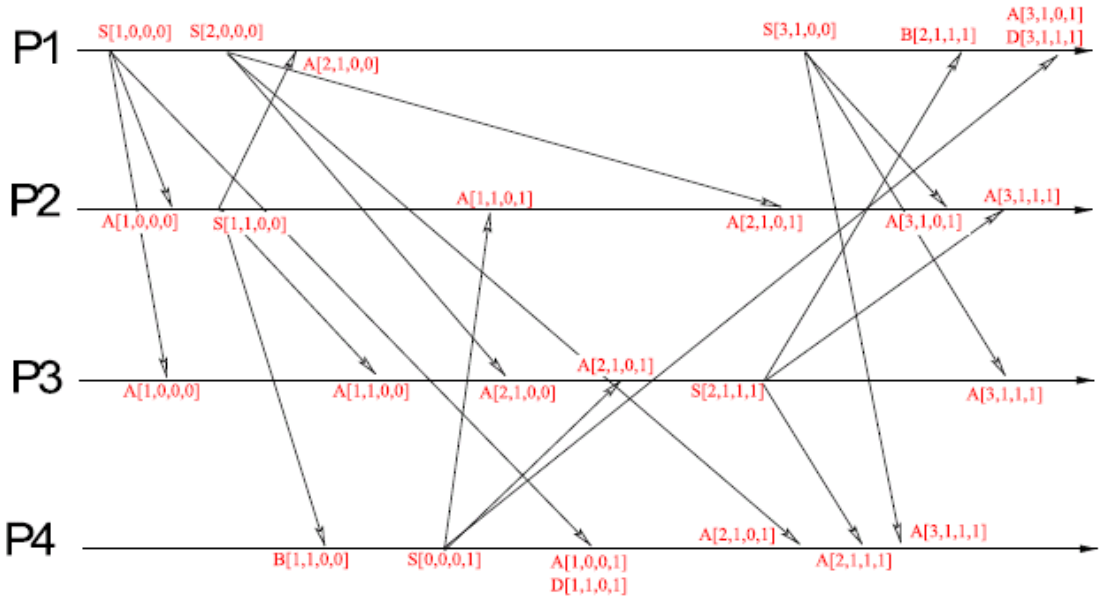
b. **(10 Points)** Causally Ordered multicast algorithm as discussed in class

Solution:

(a) FIFO multicast



(b) Causal Multicast



Problem 2: Ordered Multicast - 20 Points

2.1 (**10 Points**) Let us assume FIFO-ordered multicast per group, i.e., if a correct process issues **multicast(g,m)** and then **multicast (g, m')** , then every correct process that delivers m' will have already delivered m. Show that the *FIFO-ordered multicast algorithm per group* does not work for overlapping groups, by considering two messages sent from the same source to two overlapping groups, and considering a process in the intersection of those groups. Adapt the protocol to work for this case and write down the algorithm. Hint: processes should include with their messages the latest sequence numbers of messages sent to *all groups.*

Solution:

If there are two overlapping groups, it is possible that message sequence numbers will collide for members of both groups. Say that there is a process p1 in groups G1 and G2, and that p1 receives m1 that was multicast to G1 with sequence number s, followed by m2 that was multicast to G2 with sequence number s. p1 will accepted m1, but will proceed to drop m2 since it has the same sequence number s. Even worse, a message m3 to G2 with sequence number s + 1 could now be accepted, breaking the FIFO ordering.

In order to solve this problem, we need to include some way of distinguishing groups. Note that we cannot just increment a global counter across all groups: a process that is not in the intersection may not see all messages, which is required here for FIFO delivery. A simple solution would be to piggyback both the group id and the sequence number in the multicast message.

2.2 (**10 Points**) Let us assume *FIFO-ordered multicast across groups*, i.e., (a) FIFO-ordered multicast per group, and (b) if a correct process issues **multicast(g,m)** and then multicast(g'm'), then every correct process in the intersection of groups g and g' that delivers m' will have already delivered m.
Does the *FIFO-ordered multicast algorithm across groups* work for overlapping groups, when considering two messages sent from the same source to two overlapping groups and considering a process in the intersection of those groups? Explain if yes or no. In case it does not work, adapt the protocol to work for this case and write down the algorithm.

Solution:

If one assumes that conditions (a) and (b) are true and somehow implemented, then in this case it should work (answer yes). Here is the reasoning: if source **S** first multicast **(g,m)** with sequence number **s** to group **g**, and then the source **S** multicast **(g',m')** with sequence number **s** to group **g'**, the messages **m** and **m'** are FIFO ordered at source **S** going to different groups (**g** and **g'**). Based on the execution of the assumption/condition (b) "if a correct process issues multicast **(g,m)** and then multicast **(g',m')**, then every correct process in the intersection of group **g** and **g'** receives **m** before **m'"**, this means that the messages **m** and **m'** from source **S** will arrive at a correct process **p'** in the intersection of group **g** and **g'** in the order first **m** and then **m'**. Even if message **m** might be delayed and **m'** might arrive before **m** (even though it was multicast second from the source S), message **m'** will wait until **m** arrives and is delivered at the process **p'** in the intersection due to rule (b). See Figure 1.

It is important to stress that to **implement the condition (b**), group ids and message sequence numbers (e.g., **g.m.s** and **g'.m',s**) will not be sufficient to differentiate messages **m** and **m'** at the receiver process **p'** and decide which message was sent first. It means we will need to include a **global sequence number** (GSN)across both groups **g** and **g'** in the sender **S** to ensure that if **multicast(g, m.GSN)** was sent before multicast **(g',m'.GSN+1)** from a correct process **S,**

and message **m'** is delivered ahead of **m** at the process **p'** in the intersection of the groups **g** and **g'**, then based on the global sequence number GSN we can decide that message **m'.GSN+1** must wait until message **m.GSN** arrives and we deliver message **m** before **m'**.

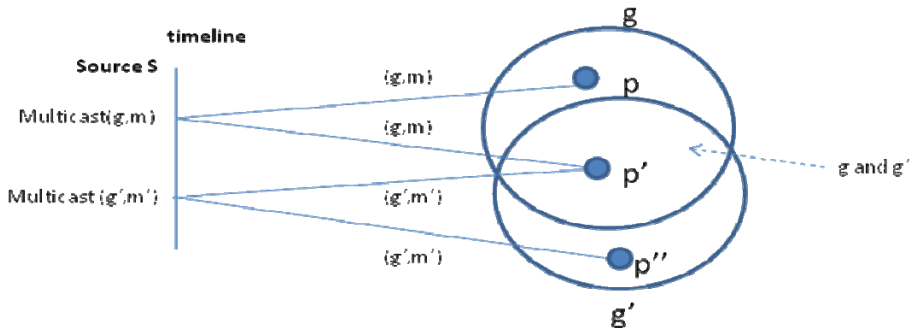


Figure 1: solution to 2.2.

Project 3: Distributed Mutual Exclusion – 20 Points

The following Figure 2 shows a Raymond's Ring for management of mutual exclusion. Node 1 is holding the token (is in CS). Current queue entries are shown under each node.

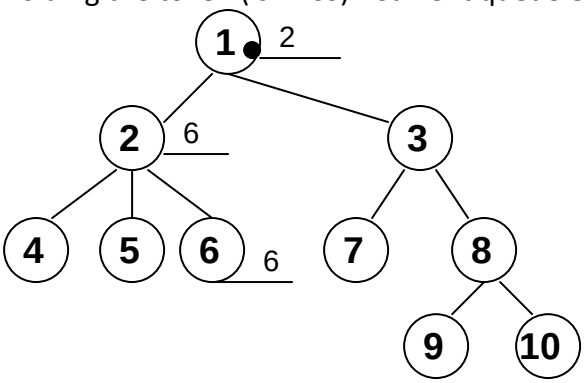


Figure 2: Raymond Algorithm – Initial State from which you start

a. Suppose nodes 9, 7, and 5 (in that order) request to enter the same CS while node 1 is still holding the token. List contents of each node's queue, after these requests are processed by all relevant nodes.
b. Assuming no other requests are received for entry to the same CS, other than those in part (a), show the contents of each node's queue when the token gets to node #6
c. Assuming no other requests are received for entry to the same CS, other than those in part (a), show the contents of each node's queue when the token gets to node #9.
d. Consider a request from node 10 while node 9 is holding the token, show the contents of each node's queue after this request is processed by all relevant nodes.

Solution:

a. Suppose nodes 9, 7, and 5 (in that order) request to enter the same CS while node 1 is still holding the token. List contents of each node's queue, after these requests are processed by all relevant nodes.
1: 2, 3 <> 2: 6, 5 <> 3: 8, 7 <> 4:empty <> 5: 5 <> 6: 6 <> 7: 7 <> 8: 9 <> 9: 9 <>10: empty

b. Assuming no other requests are received for entry to the same CS, other than those in part (a), show the contents of each node's queue when the token gets to node #6.
1: 3 <> 2: 5,1 <> 3: 8, 7 <> 4:empty <> 5: 5 <> 6: 2 <> 7: 7 <> 8: 9 <> 9: 9 <>10: empty

c. Assuming no other requests are received for entry to the same CS, other than those in part (a), show the contents of each node's queue when the token gets to node #9.
1: empty <> 2: empty <> 3: 7 <> 4:empty <> 5: empty <> 6: empty <> 7: 7 <> 8: 3 <> 9: 8 <>10: empty

d. Consider a request from node 10 while node 9 is holding the token, show the contents of each node's queue after this request is processed by all relevant nodes.
1: empty <> 2: empty <> 3: 7 <> 4:empty <> 5: empty <> 6: empty <> 7: 7 <> 8: 3,10 <> 9: 8 <>10: 10

Problem 4: Distributed Mutual Exclusion – 10 Points

Consider a group of distributed systems, P1, P2, P3, and P4 that share an object. They use the Ricarti-Agrawala algorithm for management of mutual exclusion. P1 is currently in the critical section and there is no other node in the "wanted" state. Now consider requests from P4, P2 and P3 (in that order) to enter the same CS.

a. (5 Points) Show the state (as required by the algorithm, i.e. "held", "wanted", etc.) and queue entries at each processor.

Solution:

P1: state: held; Q= {P4, P2, P3}

P2: state: wanted; Q= {P3}

P3: state: wanted; Q = {}

P4: state: wanted; Q = {P2, P3}

b. (5 Points) Now, P1 exits the CS and informs all relevant nodes that CS is released. Show the state and queue entries at each processor, at this stage.

Solution:

P1: state: released; Q= {}

P2: state: wanted; Q= {P3}

P3: state: wanted; Q = {}

P4: state: held; Q = {P2, P3}

Problem 5: Election Algorithm – 10 Points

Consider a group of distributed systems, P1, P2, P3, P4 and P5 (P5 is currently the coordinator). P5 fails and P2 notices the failure. If the Bully algorithm is used for election of a new coordinator and the election attribute is the (Max of) processor numbers, show the set of all messages communicated through each communication channel P_{ij} i,j = 1..5 for this election. Show the type of each message as "election", "response", "coordinator".

Solution:

P1 sends no messages;

P2 sends "election" messages to P3, P4 and P5

P3 sends "response to P2 and sends "election to P4, P5

P4 sends "response to P2 and P3, sends "election" to P5 and sends "coordinator" to all

Problem 6: Election Algorithm – 20 Points

For a synchronous system, design an election algorithm using a mutual exclusion algorithm, without knowing how the mutual exclusion algorithm is implemented.

Solution:

A host will request a certain "leader-specific" critical section entry, using mutual exclusion algorithm. Once the entry is granted, the host elects itself a leader and sends all other peers message 'coordinator' that the host became a coordinator/leader. Because it got entry to "leader-specific" CS, the acceptance should be automatic, i.e., no acknowledgement messages from nodes are needed. The synchronous system guarantees that the 'coordinator' message is sent within a certain time interval. So in some way whoever holds the critical region can become the leader. There is no bound in synchronous systems on the upper bound how long

one can stay in CS, so as long as the leader holds the “leader-specific” CS, it will be a leader. Other nodes can apply to the mutual exclusion algorithm to become leaders, and once the host leaves the “leader-specific” CS, other nodes gets in and elects itself as a leader.