messages). The algorithm is given by the following rules.

- To send a broadcast message, a process sends a timestamped message to all other processes including itself. This step corresponds to requesting the critical section in the mutual exclusion algorithm.
- On receiving a broadcast message, the message and its timestamp are stored in the queue and an acknowledgment is returned.
- A process can remove the message from the request queue with the smallest timestamp, t, if it has received a message from every other process with timestamp greater than t. This step corresponds to executing the critical section for the mutual exclusion algorithm.
- When a process removes its own message from its queue, it informs all other processes that this message is deliverable. Other processes can deliver the message on receiving this notification. This step corresponds to releasing of the critical section in the mutual exclsuion algorithm.

In this algorithm, the total order of messages delivered is given by the logical clock of send events of the broadcast messages.

17.4.3 Skeen's Algorithm

The distributed algorithm of Skeen is given by the following rules. It also assumes that processes have access to Lamport's logical clock.

- To send a multicast message, a process sends a timestamped message to all the destination processes.
- On receiving a message, a process marks it as *undeliverable* and sends the value of the logical clock as the proposed timestamp to the initiator.
- When the initiator has received all the proposed timestamps, it takes the maximum of all proposals and assigns that timestamp as the final timestamp to that message. This value is sent to all the destinations.

- Upon receiving the final timestamp of a message, it is marked as deliverable.
- A deliverable message is delivered to the site if it has the smallest timestamp in the message queue.

In this algorithm, the total order of message delivery is given by the final timestamps of the messages.

We leave the formal description and proof of correctness of these algorithms as exercises.

17.5 Application: Replicated State Machines

Assume that we are interested in providing a fault-tolerant service in a distributed system. The service is expected to process *requests* and provide *outputs*. We would also like the service to tolerate up to t faults where each fault corresponds to a crash of a processor. We can build such a service using $t+1$ processors in a distributed system as follows. We structure our service as a *deterministic* state machine. This means that if each nonfaulty processor starts in the same initial state and executes the requests in the same order, then each will produce the same output. Thus by combining outputs of the collection we can get a t fault-tolerant service. The key requirement for implementation is that all state machines process all requests in the same order. The total ordering of messages (for example, Lamport's algorithm) satisfies this property.

17.6 Problems

17.1. Design an algorithm for synchronous ordering for point-to-point messages that does not use a static priority scheme. (Hint: Impose an acyclic directed graph on processes. The edge from P_i to P_j means that P_i is bigger than P_j for the purpose of sending messages. Give a rule by which the direction of edges is reversed, such that acyclicity of the graph is maintained.)

17.2. Extend the algorithm for synchronous ordering to the case when a process may send a message to itself. The message, whenever enabled, is considered to take zero time.