DISTRIBUTED COMPUTING SYSTEMS

ASSIGNMENT III

Algorithm Implemented: *Ricart and Agrawala's Permission Based Distributed Mutual Exclusion Algorithm*

Girish Motwani

4710-412-021-02205

Algorithm:

Ricart and **Agrawala's** Algorithm solves the synchronization problem in distributed systems. This algorithm insures that only one process will be allowed in a critical region at a time. It works by a using a system of messages and acknowledgments. The sending of a message is assumed to be reliable; that is, every message is acknowledged. The algorithm is based on the fact that a node receiving a REQUEST message can immediately determine whether the requesting node or it should be allowed to enter its critical section first. The node originating the REQUEST message is never told the result of the comparison. A REPLY message is returned immediately if the originator of the REQUEST message has priority; otherwise, the REPLY is delayed. The priority order decision is made by comparing a sequence number present in each REQUEST message. If the sequence numbers are equal, the node numbers are compared to determine which will enter first.

The algorithm works as follows:

When a process wants to enter a critical region, it builds a message containing the type of the message i.e. a REQUEST indicating that it wants to enter the Critical Section, its time stamp and its identity number. It then sends the message to all the other processes. When a process receives a request message from another process, the action it takes depends on its state with respect to the critical region named in the message.

There are three possible states:

- If the receiver is not in the critical region and does not want to enter it, it sends back a REP (Reply) message to the sender.
- If the receiver is already in the critical region, it does not reply. Instead, it sets rep_deferred [sender] to true and thus queues the request.
- If the receiver wants to enter the critical region, but has not yet done so, it compares the Time stamp in the incoming message with the one contained in the message that it has sent everyone. The lowest one wins. If the incoming message is lower, the receiver sends back a REP message. If its own message has a lower timestamp, the receiver queues the incoming message by setting rep_ deferred [sender] to true and sends nothing.

After sending out requests asking permission to enter a critical region, a process sits back and waits until everyone else has given permission. As soon as all the permissions are in, it may enter the critical region. When it exits the critical region, it sends REP (Reply) messages to all processes on its queue and deletes them all from the queue.

To handle the termination, each of the members sends a special message with type TERMINATED when it terminates. The other members on receiving the message remove this member from the list of members and then carry out the protocol. Thus no further requests will be sent to the member which has terminated.

Implementation:

The implementation comprises of

- A C source file RicartArgwl.c. This is the program running at each of the sites comprising the distributed system.
- A file containing the names of the hosts that comprise the distributed system. The file is hosts.txt
- A file containing the definitions of the global variables and data structures.

The executable can be obtained using the run script. It is named dme. When the program is run, there are 3 threads of execution at each of the systems. The threads perform the following functions:

- One thread is used for sending messages
- The second is to receive messages sent by the other hosts and display them.
- Finally, the third thread is used to process the Request and Reply messages for entering Critical Section.

The algorithm uses only 2* (N - 1) messages between nodes, where N is the number of nodes and is optimal in the sense that a symmetrical, distributed algorithm cannot use fewer messages if requests are processed by each node concurrently. In addition, the time required to obtain the mutual exclusion is minimal if it is assumed that the nodes do not have access to timing-derived information and that they act symmetrically.