CSCI 344/715
Lecturer: Simina Fluture
Topics: Distributed Algorithm

Bring with you dimu.java

DISTRIBUTED MUTUAL EXCLUSION ALGORITHM

Requires total ordering of the events in the system.
A collection of threads that do not share memory, are connected to a local area network (LAN). The threads need mutual exclusion over some shared resources. In order to avoid a bottleneck a central server will not be used, rather the Distributed Algorithm fore Mutual Exclusion will be used.
There will be N nodes connected by the LAN and we assume that:

- there are no lost or garbled messages.
- Messages sometimes arrive in a different order than they were sent.
- Nodes do not fail or halt.

When a process wants to enter the CS:
1) It builds a message:

Message = name of the CS + its process ID + the current time

2) It sends the message to all processes in the system including itself.

When a process receives a request message from another process (there are three cases):
If the receiver is not in the CS and doesn't intend to enter the CS, it sends back an "OK" message.
If the receiver is in the CS, it doesn't reply and queues the request.
If the receiver wants to enter the CS, it compares the timestamp of the incoming message with the one contained in the message sent by itself. If the incoming message is lower, the receiver sends back an "OK" message, otherwise the receiver queues the incoming request and sends nothing.

A process can enter the CS **after receiving the OK** messages from **all** of the other processes.
Fig. The distributed algorithm for Mutual Exclusion

-) In practice there are now *n* points of failure.
If one process goes down, the system goes down.
Not so easy to implement, slower and more expensive. Need to keep track of a membership list.

Improving the algorithm:
Allow a process to enter a CS when it has collected permission from a simple majority of other processes rather from all of them.

Dimu.java

For parallelism each site has three threads. The following activities occur in each node:

```
While(true) {
        Noncritical section code            // outsideCS( );
        ** preprotocol:
        Choose a sequence number            //  chooseNumber( );
        SentItToAllOthers                   // sendRequest( );
        Wait for replies                    // waitForReply( );

        access Critical Section       // insideCS( );

        ** postprotocol:
        reply to all others (the deferreds)   // replyToDefferedNodes( );
}
```

Even if in normal situation the Mutual Exclusion, Progress, No Starvation and Bounded Waiting are satisfied in reality the Distributed Mutual Exclusion Algorithm it is hard to be implemented and many problems might occur.

Initial values: **Requesting =**
Number =
HighNumber =
Bin Sem S =
Bin Sem Wakeup =
Deferred[i] =

Going over one exmple:
Each site has three threads:

HandleRequests( );	main( );	HandleReplies( );

Example: numNodes = 3;

Site 0	**Site 1**	**Site 2**
RequestChannel[0]	**RequestChanne[l]**	**RequestChannel[2]**
ReplyChannel[0]	**ReplyChannel[1]**	**ReplyChannel[2]**

Homework

1. [SH]: 6.6: problem 1, 4 (page 205).

a) *Program 6.4 has a binary semaphore* ***s****, local to each node. Why is it necessary?*
b) *Suppose that the binary semaphore* ***s*** *is deleted from the method* ***replyToDefferedNodes*** *and the* ***V(s)*** *in thread* ***handleRequests*** *is moved up just before the* ***if*** *statement. Show that a* ***deadlock*** *is possible.*
c) *Does the* ***V(s)*** *in* ***replyToDefferedNodes*** *need to be moved to the end of the method? In other words, why have a* ***P(s)…V(s)*** *bracket the single assignment statement* ***requesting=false****? Where is the* ***race condition****?*
d) *Suppose node* ***m*** *decides to enter its critical section and sends a request message to node* ***n****. Node* ***n*** *sends a reply message to node* ***m*** *indicating that node* ***n*** *does not want to enter its critical section. Suppose further that nodes* ***n*** *later decides to enter its critical section and sends a request message to node* ***m****. What happens if node* ***n****'s request message is received by node* ***m*** *before the earlier-sent reply message of node* ***n*** *to node* ***m****?*

2. Even if in normal situations the Mutual Exclusion, Progress, No Starvation and Bounded Waiting conditions are satisfied, in reality the Distributed Mutual Exclusion Algorithm is hard to be implemented and many problems might occur.
Discuss the problems that might occur.
Hint: [SH] page 181.