UNIT-2

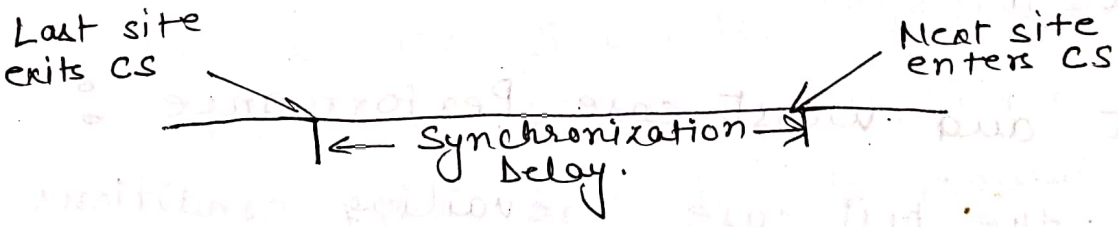
Distributed Mutual Exclusion

- Requirements of Mutual Exclusion Algorithm

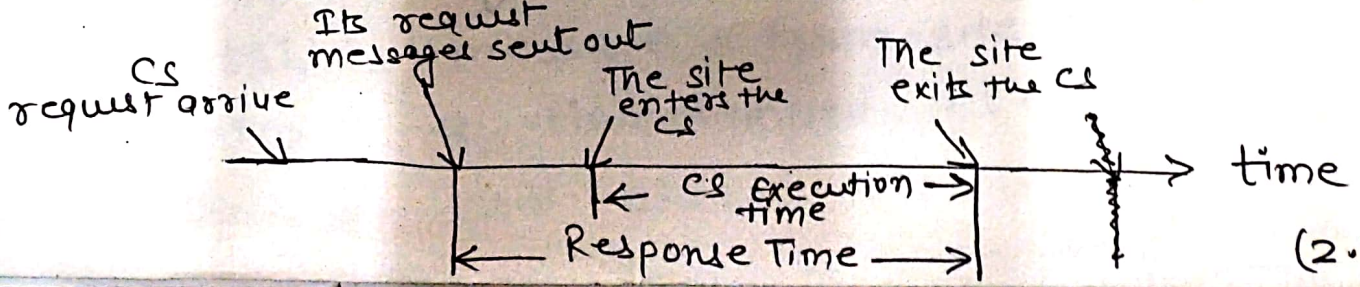
1. Freedom from Deadlocks.
2. Freedom from Starvation.
3. Fairness
4. Fault Tolerance.

- How to measure the Performance :

1. Number of messages
2. Synchronization delay : It is the time required after a site leaves the critical section and before the next site enters the critical section.



3. Response time : It is the time interval a request waits for its CS execution to be over after its request messages have been sent out.



4. System Throughput, which is the rate which the system executes requests for CS. If 'sd' is the synchronization delay and E is the average critical section execution time, then the throughput is given by the following equation

$$\text{System throughput} = 1/(sd + E)$$

Low and High load performance :

Under low load condition, there is seldom more than one request for mutual exclusion simultaneously in the system.

Under high load conditions, there is always a pending request for mutual exclusion at a site.

Best and worst case Performance :

In the best case, prevailing conditions are such that a performance metric attains the best possible value. Often for mutual exclusion algorithm, the best and worst cases coincide with low and high loads, respectively.

oken-Based and Non-Token Based Algo.

1. Non-Token Based Algorithms :

* It use timestamps to order requests for the CS and to resolve conflicts between simultaneous requests for the CS.
* Each request for the CS gets a timestamp, and smaller timestamp requests have priority over larger timestamp requests.

(A) Lamport's Algorithm :

Every site Si keeps a queue, request-queuei which contains mutual exclusion requests ordered by their timestamps.

The Algorithms :

(a) Requesting the critical section :

(i) when a site Si wants to enter the CS, it sends a REQUEST (tsi, i) message to all the sites in its request set Ri and places the request on request_queuei where (tsi, i) is the time-stamp of the request.

(ii) when a site Sj receives the REQUEST (tsi, i) message from site Si, it returns a timestamped REPLY message to Si and places site Si's request

on request_queuej.

(b) Executing the critical section :

Site Si enters the critical section when the two following conditions hold :

(i) Si has received a message with timestamp larger than (tsi, i) from all other sites.

(ii) Si's request is at the top of request_queuei.

(c) Releasing the critical section :

(i) Site Si, upon releasing the CS, removes its request from the top of its request queue and sends a timestamped RELEASE message to all the sites in its request set.

(ii) When a site Sj receives a RELEASE message from site Si, it removes Si's request from its request queue.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS. The algorithm executes CS requests in the increasing order of timestamps.

The Ricart-Agrawala Algorithm :

It is an optimization of Lamport's algorithm that dispenses with RELEASE messages by cleverly merging them with REPLY message.

The Algorithm

(i) Requesting the critical section :

1. When a site Si wants to enter the CS, it sends a timestamped REQUEST message to all the sites in the request set.
2. When site Sj receives a REQUEST message from site Si, it sends a REPLY message to site Si if site Sj is neither requesting nor executing the CS or if site Sj is requesting and Si's request's timestamp is smaller than site Sj's own request's timestamp. The request is deferred otherwise.

(ii) Executing the critical section :

3. Site Si enters the CS after it has received REPLY messages from all the sites in its request set.

(iii) Releasing the critical section :

4. When site Si exits the CS, it sends REPLY messages to all the deferred requests.

A site's REPLY messages are blocked by sites that are requesting the CS with higher priority (i.e. a smaller timestamp).

Thus, when a site sends out REPLY message to all the deferred requests, the site with the next highest priority request receives the last needed REPLY message and enters the CS.

The execution of CS requests in the algorithm is always in the order of their timestamp.

(C) Maekawa's Algorithm :

First, a site does not request permission from every other site, but only from a subsets of the sites.

Second, in Maekawa's algorithm a site can send out only one REPLY message at a time i.e. after it has received a RELEASE message for the previous REPLY message. Therefore, a site Si locks all the site in Ri in exclusive mode before executing its CS.

~~MEK~~

MAEKAWA'S ALGORITHM:

The Algorithm

Requesting the critical section:

1. A site Si requests access to the CS by sending REQUEST(i) messages to all the sites in its request set Ri.

2. When a site Sj receives the REQUEST(i) message, it sends a REPLY(j) message to Si provided it hasn't sent a REPLY message to a site from the time it received the last RELEASE message. Otherwise, it queues up the REQUEST for later consideration.

Executing the critical section:

3. Site Si accesses the CS only after receiving REPLY messages from all the sites in Ri.

Releasing the critical section:

4. After the execution of CS is over, site

Si sends RELEASE(i) message to the sites in Ri.

5. When a site Sj receives a RELEASE message from site Si, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site update its state to reflect that the site has not sent out any REPLY messag

Token - Based Algorithm :

* A unique token is shared among all sites. A site is allowed to enter its CS if it possesses the token.
* It use sequence numbers instead of timestamps. A site increments its sequence number counter every time it makes a request for the token.

(A) Suzuki - Kasami's Broadcast Algorithms :

(i) Requesting the critical section :

1. If the requesting site Si does not have the token, then it increments its sequence number, RNi[i] and sends a REQUEST (i, Sn) message to all other sites. (Sn is the updated value of RNi[i].)

2. When a site Sj receives this message, it sets RNj[i] to max(RNj[i], Sn). If Sj has the idle token, then it sends the token to Si if RNj[i] = LN[i] + 1.

(ii) Executing the critical section :

3. Site Si executes the CS when it has received the token.

(iii) Releasing the critical section : Having finished the execution of the CS, site Si takes the following actions :

4. It sets LN[i] elements of the token array equal to RNi[i].

5. For every site Sj whose ID is not in the token queue, it appends its ID to the token queue if RNi[j] = LN[i]+1.

6. If token queue is nonempty after the above update, then it deletes the top site ID from the queue and sends the token to the site indicated by the ID.

Thus, after having executed its CS, a site gives priority to other sites with out-standing requests for the CS.

* The Suzuki-Kasami algorithm is not symmetric because a site retains the token even if it does not have a request for the CS, which is contrary to the spirit of Ricart and Agrawala's definition of a symmetric algorithm : "no site possesses the right to access its CS when it has not been requested."

3) Singhal's Heuristic Algorithm :

- In this algo, each site maintains information about the state of other sites in the system and uses it to select a set of sites that are likely to have the token.
- The site requests the token only from these sites, reducing the number of messages required to execute the CS. It is called a heuristic algorithm because site are heuristically selected for sending token request messages.

The Algorithm :

Requesting the critical section :

1. If the requesting site Si does not have the token, then it takes the following actions :
   - It sets SVi[i] := R.
   - It increments SNi[i] := SNi[i] + 1.
   - It sends REQUEST (i, sn) message to all sites Sj for which SVi[j] = R. (sn is the updated value of SNi[i].)
2. When a site Sj receives the REQUEST (i, sn) message, it discard the message if SNj[i] ≥ sn because the message is outdated.

Otherwise, it sets SNj[i] to 'sn' and takes the following actions based on its own state :

- SVj[j] = N: set SVj[i] := R
- SVj[j] = R : If SVj[i] ≠ R then set SVj[i] := R and send a REQUEST (j, SNj[i]) message to Si (else do nothing).
- SVj[j] = E : set SVj[i] := R
- SVj[j] = H : set SVj[i] := R, TSV[i] := R, TSN[i] := sn, SVj[j] := N and send the token to site Si.

Executing the critical Section

3. Si executes the CS after it has received the token. Before entering the CS, Si ~~gets~~ sets SVi[i] to E.

Releasing the critical section

4. Having finished the execution of the CS, site Si sets SVi[i] := N and TSV[i] := N, and updates its local and token vectors in the following way :

For all Sj, j = 1 to N do
if SNi[j] > TSN[j]
then
(* update token information from local information *)
{ TSV[j] := SVi[j]; TSN[j] := SNi[j] }

else

(* update local information from token information *)

{ SV_i[j] := TSV[j]; SN_i[j] := TSN[j] }

5. If (∀j :: SV_i[j] = N), then set SV_i[j] := H, else send the token to a site S_j such that SV_i[j] = R.

(c) Raymond's Tree-Based Algorithm

Sites are logically arranged as a directed tree such that the edges of the tree are assigned directions towards the site (root of the tree) that has the token.

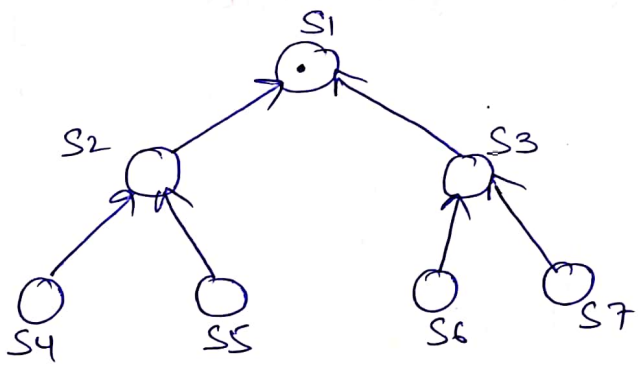
Every site has a local variable holder that points to an immediate neighbor node on a directed path to the root node.

Every site keeps a FIFO queue, called request_q, which stores the requests of those neighboring sites that have sent a request to this site, but have not yet been sent the token.

The Algorithm :

Requesting the critical section

1. When a site wants to enter the CS, it sends a REQUEST message to the node along the directed path to the root, provided it does not hold the token and its request_q is empty. It then adds its request to its request_q.



2. When a site on the path receives this messages, it places the REQUEST in its request_q and sends a REQUEST message along the directed path to the root provided it has not sent out a REQUEST message on its outgoing edge (for a previously received REQUEST on its request_q).

3. When the root site receives a REQUEST message, it sends the token to the site from which it received the REQUEST message and sets its holder

variable to point at that site.

4. When a site receives the token, it deletes the top entry from its request_q, sends the token to the site indicated in this entry, and sets its holder variable to point at that site. If the request_q is nonempty at this point, then the site sends a REQUEST message to the site which is pointed at by holder variable.

Executing the critical section :

5. A site enters the CS when it receives the token and its own entry is at the top of its request_q. In this case, the site deletes the top entry from its request_q and enters the CS.

Releasing the critical section :

6. After a site has finished execution of the CS, it takes the following actions :

6. If its request_q is non-empty, then it deletes the top entry from its request_q, sends the token to that site, and sets its holder variable to point at

(9.15)

that site.

7. If the request_q is nonempty at this point, then the site sends a REQUEST message to the site which is pointed at by the holder variable.

Performance :

The average message complexity of Raymond's algorithm is O(logN) because the average distance between any two nodes in the tree with N nodes is O(logN).