BITS Pilani
Pilani Campus

CS G623: Advanced Operating Systems

Lecture 10

Amit Dua

August 25, 2018

Topics to be discussed

- DME
- Goals
- Approaches
  - Centralized
  - Distributed
  - Token based
- Leader-election algorithm
- Centralized approach
- Lamport's algorithm
- Richart-Agarwala'a algorithm
- Maekwa's algorithm

Distributed Mutual Exclusion

What is mutual exclusion?

It is exclusive access to a shared resource or to the critical region.

DME Continued…..

- An algorithm used for implementing mutual exclusion must satisfy:
  - Mutual exclusion
  - No starvation,
  - Freedom from deadlock
  - Fault tolerant

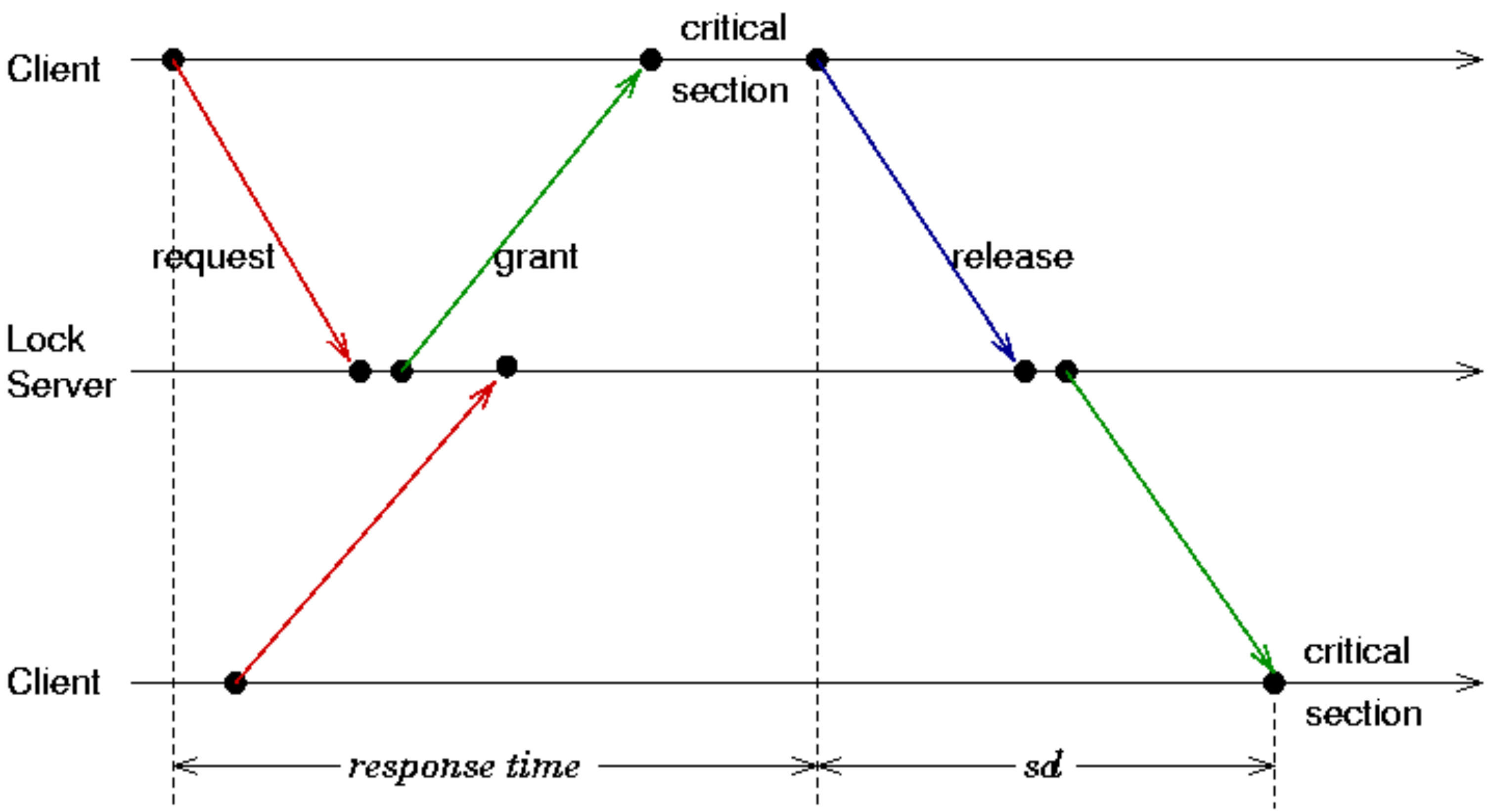
- To handle mutual exclusion in a distributed system,
  - Centralized Approach
  - Distributed Approach
  - Token-Passing Approach

- All use message passing approach rather than shared variable

Performance of DME Algorithms

- Performance of each algorithm is measured in terms of
  - no. of messages required for CS invocation
  - synchronization delay (leaving & entering)
  - response time (arrival, sending out, Entry & Exit)
- system throughput = 1/(sd+E)

Performance Continued…



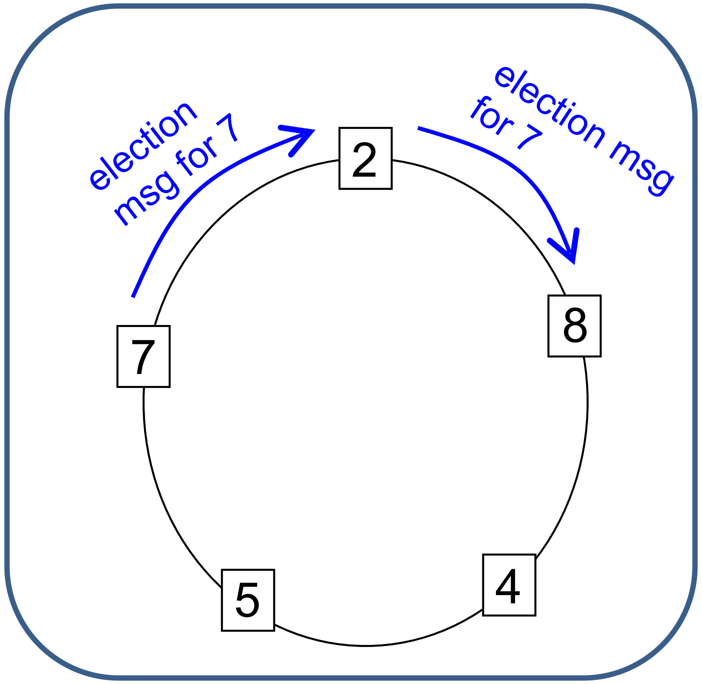
$$\text{System Throughputs} = \frac{1}{Sd + E}$$

DME: A Centralized Algorithm

- One process is elected as the coordinator/leader (e.g., the one running on the machine with the highest network address) which resolves conflicts.

An example leader election algorithm

- Each node has a unique identifier
- Nodes only send messages clockwise
- Each node acts on its own
- Protocol:
  - A node send election message with its own id clockwise
  - Election message is forwarded if id in message larger than own message
  - Otherwise message discarded
- A node becomes leader if it sees it own election message



Lelang-Chang Algorithm

Central Approach: Adv.s & Disadvs

- Mutual exclusion can be achieved.
- No process waits forever.
- Easy to implement.
- It can be used for general resource allocation rather than just managing mutual exclusion.

- Single point of failure
- If process normally blocks after making a request, difficult to distinguish a dead coordinator from "access denied".

Lamport's DME

Requesting the critical section.

1. When a site Si wants to enter into the CS, it sends a REQUEST(T=tsi, i) message to all the sites in its request set Ri and places the request on request_queuei.

2. 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.

Executing the critical section.

Site Si enters the CS when the two following conditions hold:

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

2. Si 's request is at the top of request_queuei.

Continued…

Releasing the critical section.

1. Site Si, upon exiting 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.

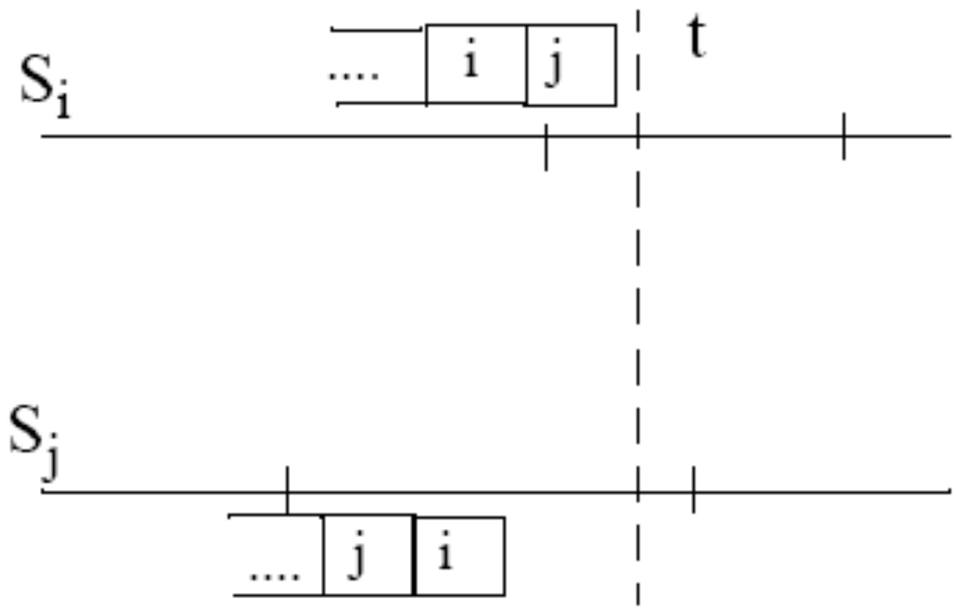
2. When a site Sj receives a RELEASE message from site Si , it removes Si 's request from its request queue.

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

Example Lamport's DME

Correctness

- Suppose that both Si and Sj were in CS at the same time (t).
- Then we have:



No. of messages, Synch delay, Optimization if any?)

Ricart-Agrawala DME

Requesting the CS

1. A site Si wanting to enter into CS sends a timestamped request message to all sites in it's request set.
2. Upon reception of a request message from Si, site Sj immediately sends a timestamped *reply* message if and only if:
   1. Sj is not currently interested in the critical section OR
   2. Sj's request has a lower priority (*usually this means having a later timestamp)*
   3. Otherwise, Sj will defer the reply message. This means that a reply will be sent only after Sj has finished using the critical section itself.

Executing in the Critical Section

1. Si enters into CS after receiving reply from all other sites

Releasing the critical section

4. Upon exiting the critical section, the site Si sends reply messages to all deferred requests.

Maekawa's DME Algorithm

- A site requests permission only from a subset of sites.
- Request set of sites S_i & S_j: Ri, Rj such that Ri and Rj will have atleast one common site (Sk). Sk mediates conflicts between Ri and Rj.
- A site can send only one REPLY message at a time, i.e., a site can send a REPLY message only after receiving a RELEASE message for the previous REPLY message.

Request Subsets

- Rules for generating request sets:
  - Sets Ri and Rj have atleast one common site.
  - Si is always in Ri.
  - Cardinality of Ri, i.e., the number of sites in Ri is K.
  - Any site Si is in K number of Ri's. $N = K(K - 1) + 1$

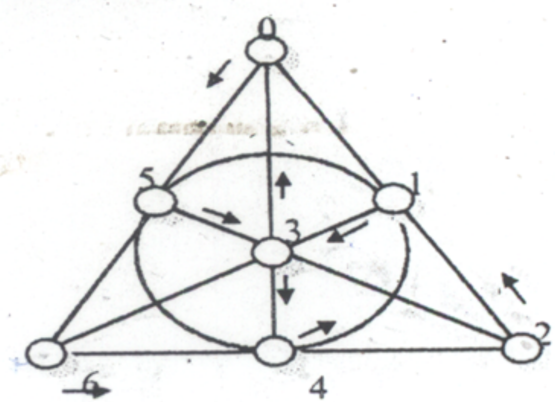
- What would be the request set for N = 3?

Request Subsets continued…

- What would be the request set for N = 7?

- Example: Finite Projective Planes

S(0) = {0,5,6}
S(1) = {1,3,6}
S(2) = {2,1,0}
S(3) = {3,0,4}
S(4) = {4,1,5}
S(5) = {5,3,2}
S(6) = {6,4,2}



Request Set with N=13

$R_1 = \{ 1, 2, 3, 4 \}$
$R_2 = \{ 2, 5, 8, 11 \}$
$R_3 = \{ 3, 6, 8, 13 \}$
$R_4 = \{ 4, 6, 10, 11 \}$
$R_5 = \{ 1, 5, 6, 7 \}$
$R_6 = \{ 2, 6, 9, 12 \}$
$R_7 = \{ 2, 7, 10, 13 \}$
$R_8 = \{ 1, 8, 9, 10 \}$
$R_9 = \{ 3, 7, 9, 11 \}$
$R_{10} = \{ 3, 5, 10, 12 \}$
$R_{11} = \{ 1, 11, 12, 13 \}$
$R_{12} = \{ 4, 7, 8, 12 \}$

Maekawa's DME 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

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

Releasing the critical section

1. After the execution of the CS is over, site Si sends RELEASE(i) message to all the sites in Ri .
2. When a site Sj receives a RELEASE(i) 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 updates its state to reflect that the site has not sent out any REPLY message.

Maekawa’s Example

Maekawa’s DME Correctness

- **Proof of correctness (by contradiction)**
  Suppose Si and Sj are in the CS at the same time Ri ^ Rj = { Sk }.
  Then, Sk must have sent REPLY to both Si and Sj, which is not allowed.

- **Performance**
  messages/CS = ?
  synchronization delay = ?

Do you foresee any problem?

Solution to Deadlock

- **FAILED**

  A FAILED message from site Si to site Sj indicates that Si cannot grant Sj's request.
- **INQUIRE**

  An INQUIRE message from Si to Sj indicates that Si would like to find out from Sj…
- **YIELD**

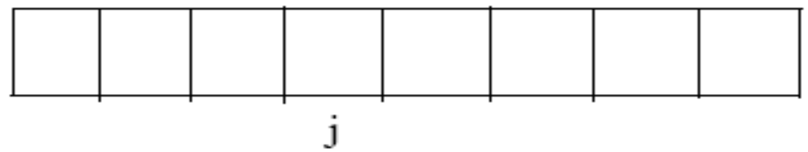
  A YIELD message from site Si to Sj indicates that Si is returning the permission to Sj.

Token-based DME Algorithms

- A site enters CS if it possesses the token (only one token for the System).
- The major difference is the way the token is searched.
- Use sequence numbers instead of timestamps
  - Used to distinguish requests from same site
  - Keep advancing independently at each site
- The proof of mutual exclusion is trivial

Suzuki-Kasami's Broadcast Algorithm

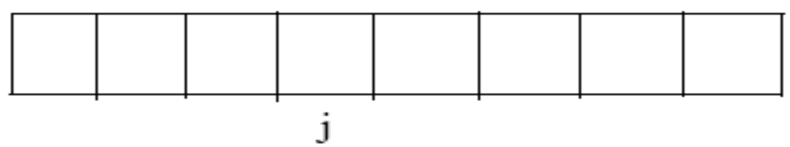
A site has RN:



Largest requesting seq # so far from S_j .

Token: A queue & LN

Q



Largest executed seq # so far from j .

- LN[j] is the sequence no. of the request that site S_j executed most recently.

The Algorithm

- **Requesting the critical section.**

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

  2. When a site S_j receives this message, it sets RN_j [i] to max(RN_j [i], sn). If S_j has the idle token, it sends the token to S_i if RN_j [i] = LN[i] + 1.

- **Executing the critical section.**

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

- **Releasing the critical section.**

  Having finished the execution of the CS, site S_i takes the following actions:

  4. It sets LN[i] element of the token array equal to RN_i [i].

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

Example1

Example2

Analysis

- **Correctness**

  Mutex is trivial.

  – Theorem:

  A requesting site enters the CS in finite time.

  – Proof:

  A request enters the token queue in finite time. The queue is in FIFO order, and there can be a maximum N-1 sites ahead of the request.

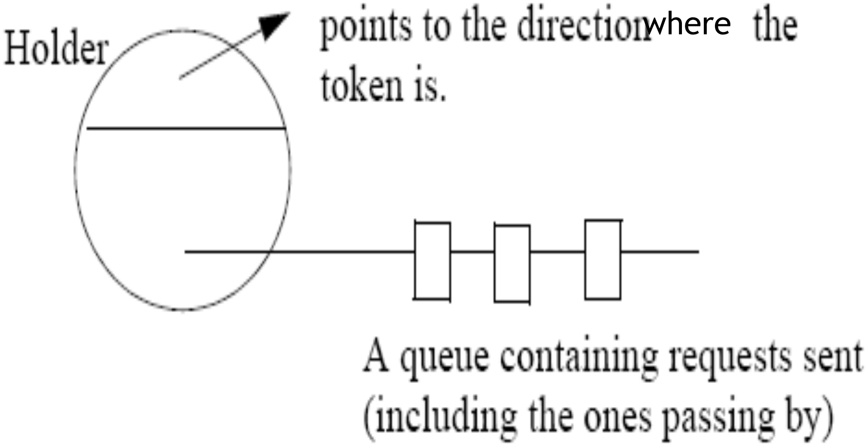
- **Performance**

  0 or N messages per CS invocation.

  Synchronous delay: 0 or T(only 1 message)

Raymond's Tree-based DME Algorithm

A node:



Raymonds' DME Continued…

Let us see a tree…

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 to its request_q.

2. When a site on the path receives this message, 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.

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 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.

The Algorithm Continued….

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

Example1

Example2

Analysis

- **Proof of Correctness**

  Mutex is trivial.

  Finite waiting: All the requests in the system form a FIFO queue and the token is passed in that order.

- **Performance**

  O(logN) messages per CS invocation.

  Sync. delay: (T logN) / 2

  The average distance between two sites is logN / 2.

Singhal's Heuristic Algorithm

- Data Structures:
  - Si maintains SVi[1..M] and SNi[1..M] for storing information on other sites: state and highest sequence number.
  - Token contains 2 arrays: TSV[1..M] and TSN[1..M].
  - States of a site
    - R : requesting CS
    - E : executing CS
    - H : Holding token, idle
    - N : None of the above
  - Initialization:
    - SVi[j] := N, for j = M .. i; SVi[j] := R, for j = i-1 .. 1;

      SNi[j] := 0, j = 1..M
    - Site S1 is in state H.
    - Token: TSV[j] := N & TSN[j] := 0, j = 1 .. M.

Algorithm Continued…

- **Requesting CS**
  - If Si has no token and requests CS:
    - SVi[i] := R. SNi[i] := SNi[i] + 1.
    - Send REQUEST(i,sn) to sites Sj for which SVi[j] = R. (sn: sequence number, updated value of SNi[i]).
  - Receiving REQUEST(i,sn): if sn <= SNj[i], ignore. Otherwise, update SNj[i] and do:
    - SVj[j] = N -> SVj[i] := R.
    - SVj[j] = R -> If SVj[i] != R, set it to R & send REQUEST(j,SNj[j]) to Si. Else do nothing.
    - SVj[j] = E -> SVj[i] := R.
    - SVj[j] = H -> SVj[i] := R, TSV[i] := R, TSN[i] := sn, SVj[j] = N. Send token to Si.

- **Executing CS**

  after getting token. Set SVi[i] := E.

Algorithm Continued…

- **Releasing CS**
  - SVi[i] := N, TSV[i] := N. Then, do:
    - For other Sj: if (SNi[j] > TSN[j]), then {TSV[j] := SVi[j]; TSN[j] := SNi[j]} //update token info from local info
    - else {SVi[j] := TSV[j]; SNi[j] := TSN[j]} // otherwise
  - If SVi[j] = N, for all j, then set SVi[i] := H. Else send token to a site Sj provided SVi[j] = R.

- Fairness of algorithm will depend on choice of Si, since no queue is maintained in token.
- Arbitration rules to ensure fairness used.

- Performance
  - Low to moderate loads: average of N/2 messages.
  - High loads: N messages (all sites request CS).
  - Synchronization delay: T.