

CST 402 - DISTRIBUTED COMPUTING

Question – Answer Bank

Module - III

Part A

1. What do you meant by mutual exclusion in a distributed environment?

Mutual exclusion is a fundamental problem in distributed computing systems.

Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner.

Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time

There are three basic approaches for implementing distributed mutual exclusion:

Token-based approach.

Non-token-based approach

.Quorum-based approach.

2. Discuss about various approaches for approaches for implementing distributed mutual exclusion.

- There are three basic approaches for implementing distributed mutual exclusion:

1. Token-based approach.
2. Non-token-based approach
3. .Quorum-based approach.

In the token-based approach, a unique token is shared among the sites.

A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over.

Mutual exclusion is ensured because the token is unique

In the non-token-based approach, two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.

A site enters the critical section (CS) when an assertion, defined on its local variables, becomes true.

In the quorum-based approach, each site requests permission to execute the CS from a subset of sites (called a quorum).

The quorums are formed in such a way that when two sites concurrently request access to the CS, at least one site receives both the requests and this site is responsible to make sure that only one request executes the CS at any time.

3. What are the requirements of mutual exclusion algorithms?

A mutual exclusion algorithm should satisfy the following properties:

Safety property:

The safety property states that at any instant, only one process can execute the critical section.

This is an essential property of a mutual exclusion algorithm.

Liveness property :

This property states the absence of deadlock and starvation.

Two or more sites should not endlessly wait for messages that will never arrive.

In addition, a site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS.

That is, every requesting site should get an opportunity to execute the CS in finite time.

Fairness :

Fairness in the context of mutual exclusion means that each process gets a fair chance to execute the CS.

In mutual exclusion algorithms, the fairness property generally means that the CS execution requests are executed in order of their arrival in the system

4. What are the performance evaluation metrics of a mutual exclusion algorithm?

Performance metrics

The performance of mutual exclusion algorithms is generally measured by the following four metrics:

Message complexity : This is the number of messages that are required per CS execution by a site

Synchronization delay : After a site leaves the CS, it is the time required and before the next site enters the CS

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

Performance metrics

System throughput This is the rate at which the system executes requests for the 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:

$$\text{System throughput} = \frac{1}{(SD + E)}.$$

5. List out the strategies for handling deadlocks in a distributed environment

Deadlocks can be dealt with using any one of the following three strategies: deadlock prevention, deadlock avoidance, and deadlock detection.

Deadlock prevention is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by preempting a process that holds the needed resource.

In the **deadlock avoidance** approach to distributed systems, a resource is granted to a process if the resulting global system is safe.

Deadlock detection requires an examination of the status of the process–resources interaction for the presence of a deadlock condition.

6. Discuss the issues in deadlock detection.

Issues in deadlock detection

Deadlock handling using the approach of deadlock detection entails addressing two basic issues:

detection of existing deadlocks

resolution of detected deadlocks.

Detection of deadlocks

Detection of deadlocks involves addressing two issues: maintenance of the WFG and searching of the WFG for the presence of cycles

Since, in distributed systems, a cycle or knot may involve several sites, the search for cycles greatly depends upon how the WFG of the system is represented across the system.

Depending upon the way WFG information is maintained and the search for cycles is carried out

Correctness criteria

A deadlock detection algorithm must satisfy the following two conditions:

1. **Progress (no undetected deadlocks)** : The algorithm must detect all existing deadlocks in a finite time.

after all wait-for dependencies for a deadlock have formed, the algorithm should not wait for any more events to occur to detect the deadlock

2. **Safety (no false deadlocks)** : The algorithm should not report deadlocks that do not exist (called phantom or false deadlocks).

In distributed systems where there is no global memory and there is no global clock, it is difficult to design a correct deadlock detection algorithm because sites may obtain an out-of-date and inconsistent WFG of the system.

As a result, sites may detect a cycle that never existed

7. Describe how quorum-based mutual exclusion algorithms differ from the other categories of mutual exclusion algorithms.

In the quorum-based approach, each site requests permission to execute the CS from a subset of sites (called a quorum).

The quorums are formed in such a way that when two sites concurrently request access to the CS, at least one site receives both the requests and this site is responsible to make sure that only one request executes the CS at any time.

In the token-based approach, a unique token is shared among the sites.

A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over.

Mutual exclusion is ensured because the token is unique

In the non-token-based approach, two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.

A site enters the critical section (CS) when an assertion, defined on its local variables, becomes true.

8. Calculate the rate at which a system can execute the critical section requests if the synchronization delay and average critical section execution times are 3 and 1 second respectively.

$$\begin{aligned}\text{Rate of CS execution, System throughput} &= 1 / (\text{Sync. Delay} + \text{Avg. CS execution time}) \\ &= 1 / (3 + 1) = 1/4\end{aligned}$$

PART B

1. Explain Lamports algorithm for mutual exclusion

Lamport developed a distributed mutual exclusion algorithm as an illustration of his clock synchronization scheme

The algorithm is fair in the sense that a request for CS are executed in the order of their timestamps and time is determined by logical clocks.

When a site processes a request for the CS, it updates its local clock and assigns the request a timestamp.

The algorithm executes CS requests in the increasing order of timestamps.

Every site S_i keeps a queue, $request_queue_i$, which contains mutual exclusion requests ordered by their timestamps.

This algorithm requires communication channels to deliver messages in FIFO order.

Requesting the critical section

- When a site S_i wants to enter the CS, it broadcasts a $REQUEST(ts_i, i)$ message to all other sites and places the request on $request_queue_i$. ((ts_i, i) denotes the timestamp of the request.)
- When a site S_j receives the $REQUEST(ts_i, i)$ message from site S_i , it places site S_i 's request on $request_queue_j$ and returns a timestamped $REPLY$ message to S_i .

Executing the critical section

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

- L1:** S_i has received a message with timestamp larger than (ts_i, i) from all other sites.
- L2:** S_i 's request is at the top of $request_queue_i$.

Releasing the critical section

- Site S_i , upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped $RELEASE$ message to all other sites.
- When a site S_j receives a $RELEASE$ message from site S_i , it removes S_i 's request from its request queue.

Algorithm 9.1 Lamport's algorithm.

2. Explain Ricart–Agrawala algorithm.

- The Ricart–Agrawala algorithm assumes that the communication channels are FIFO.
- The algorithm uses two types of messages: $REQUEST$ and $REPLY$.
- A process sends a $REQUEST$ message to all other processes to request their permission to enter the critical section.

- A process sends a REPLY message to a process to give its permission to that process.
- Processes use Lamport-style logical clocks to assign a timestamp to critical section requests.
- Timestamps are used to decide the priority of requests in case of conflict
- if a process p_i that is waiting to execute the critical section receives a REQUEST message from process p_j ,
- then if the priority of p_j 's request is lower, p_i defers the REPLY to p_j and sends a REPLY message to p_j only after executing the CS for its pending request.
- Otherwise, p_i sends a REPLY message to p_j immediately, provided it is currently not executing the CS.
- Each process p_i maintains the request-deferred array, RD_i , the size of which is the same as the number of processes in the system.
- Initially, $\forall i \forall j: RD_i[j] = 0$.

Requesting the critical section

- (a) When a site S_i wants to enter the CS, it broadcasts a timestamped REQUEST message to all other sites.
- (b) When site S_j receives a REQUEST message from site S_i , it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the CS, or if the site S_j is requesting and S_i 's request's timestamp is smaller than site S_j 's own request's timestamp. Otherwise, the reply is deferred and S_j sets $RD_j[i] := 1$.

Executing the critical section

- (c) Site S_i enters the CS after it has received a REPLY message from every site it sent a REQUEST message to.

Releasing the critical section

- (d) When site S_i exits the CS, it sends all the deferred REPLY messages: $\forall j$ if $RD_i[j] = 1$, then sends a REPLY message to S_j and sets $RD_i[j] := 0$.
-

Algorithm 9.2 The Ricart-Agrawala algorithm.

3. Explain Quorum-based mutual exclusion algorithms. / Explain Maekawa's algorithm for mutual exclusion./ Discuss the three types of messages required for deadlock handling in Maekawa's algorithm. Explain how Maekawa's algorithm handles deadlocks

- Quorum-based mutual exclusion algorithms represented a departure from the trend in the following two ways:
- A site does not request permission from all other sites, but only from a subset of the sites.
- This is a radically different approach as compared to the Lamport and Ricart–Agrawala algorithms, where all sites participate in conflict resolution of all other sites
- In quorum-based mutual exclusion algorithm, a site can send out only one REPLY message at any time.
- A site can send a REPLY message only after it has received a RELEASE message for the previous REPLY message.
- Therefore, a site S_i locks all the sites in R_i in exclusive mode before executing its CS.
- Quorum-based mutual exclusion algorithms significantly reduce the message complexity of invoking mutual exclusion by having sites ask permission from only a subset of sites.
- Since these algorithms are based on the notion of “Coterie” and “Quorums,” we first describe the idea of coterie and quorums.
- A coterie C is defined as a set of sets, where each set $g \in C$ is called a quorum. The following properties hold for quorums in a coterie:
- Intersection property
- Minimality property
- Coterie and quorums can be used to develop algorithms to ensure mutual exclusion in a distributed environment
- A simple protocol works as follows: let “a” be a site in quorum “A.”

- If “a” wants to invoke mutual exclusion, it requests permission from all sites in its quorum “A.”
- Minimality property ensures efficiency
- Maekawa’s algorithm was the first quorum-based mutual exclusion algorithm.
- This algorithm requires delivery of messages to be in the order they are sent between every pair of sites.

Requesting the critical section:

- (a) A site S_i requests access to the CS by sending REQUEST(i) messages to all sites in its request set R_i .
- (b) When a site S_j receives the REQUEST(i) message, it sends a REPLY(j) message to S_i provided it hasn’t sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST(i) for later consideration.

Executing the critical section:

- (c) Site S_i executes the CS only after it has received a REPLY message from every site in R_i .

Releasing the critical section:

- (d) After the execution of the CS is over, site S_i sends a RELEASE(i) message to every site in R_i .
 - (e) When a site S_j receives a RELEASE(i) message from site S_i , 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 it has not sent out any REPLY message since the receipt of the last RELEASE message.
-

Algorithm 9.5 Maekawa’s algorithm.

4. Illustrate Suzuki–Kasami’s broadcast algorithm

- In Suzuki–Kasami’s algorithm if a site that wants to enter the CS does not have the token, it broadcasts a REQUEST message for the token to all other sites.
- A site that possesses the token sends it to the requesting site upon the receipt of its REQUEST message.
- If a site receives a REQUEST message when it is executing the CS, it sends the token only after it has completed the execution of the CS

- Although the basic idea underlying this algorithm may sound rather simple, there are two design issues that must be efficiently addressed:
- 1. How to distinguishing an outdated REQUEST message from a current REQUEST message
- 2. How to determine which site has an outstanding request for the CS

Requesting the critical section:

- (a) If 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 $RN_i[i]$.)
- (b) 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, then it sends the token to S_i if $RN_j[i] = LN[i] + 1$.

Executing the critical section:

- (c) Site S_i executes the CS after it has received the token.

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

- (d) It sets $LN[i]$ element of the token array equal to $RN_i[i]$.
 - (e) For every site S_j whose i.d. is not in the token queue, it appends its i.d. to the token queue if $RN_i[j] = LN[j] + 1$.
 - (f) If the token queue is nonempty after the above update, S_i deletes the top site i.d. from the token queue and sends the token to the site indicated by the i.d.
-

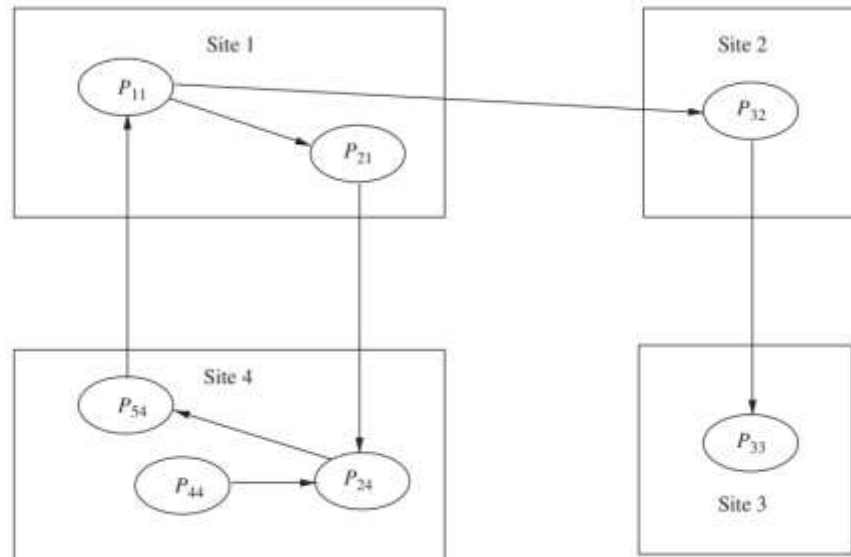
Algorithm 9.7 Suzuki–Kasami’s broadcast algorithm.

5. Explain with example, how wait-for-graphs can be used in deadlock detection.

Wait-for graph (WFG)

- In distributed systems, the state of the system can be modeled by directed graph, called a wait-for graph (WFG).
- In a WFG, nodes are processes and there is a directed edge from node P1 to node P2 if P1 is blocked and is waiting for P2 to release some resource.
- A system is deadlocked if and only if there exists a directed cycle or knot in the WFG

Figure 10.1 Example of a WFG.



6. Explain / compare various models of deadlocks. / Explain any different models of deadlock

Models of deadlocks

Distributed systems allow many kinds of resource requests.

A process might require a single resource or a combination of resources for its execution

Models of deadlocks introduces a hierarchy of request models starting with very restricted forms to the ones with no restrictions

1. The single-resource model

The single-resource model is the simplest resource model in a distributed system, where a process can have at most one outstanding request for only one unit of a resource.

Since the maximum out-degree of a node in a WFG for the single resource model can be 1, the presence of a cycle in the WFG shall indicate that there is a deadlock

2. The AND model

- In the AND model, a process can request more than one resource simultaneously and the request is satisfied only after all the requested resources are granted to the process.

- The requested resources may exist at different locations.
- The out degree of a node in the WFG for AND model can be more than 1.
- The presence of a cycle in the WFG indicates a deadlock in the AND model.

3. The OR model

In the OR model, a process can make a request for numerous resources simultaneously and the request is satisfied if any one of the requested resources is granted.

The requested resources may exist at different locations.

If all requests in the WFG are OR requests, then the nodes are called OR nodes.

Presence of a cycle in the WFG of an OR model does not imply a deadlock in the OR model.

3. The AND-OR model

A generalization of the previous two models (OR model and AND model) is the AND-OR model.

In the AND-OR model, a request may specify any combination of and and or in the resource request.

For example, in the ANDOR model, a request for multiple resources can be of the form x and $(y \text{ or } z)$.

4. The $\binom{p}{q}$ model

Another form of the AND-OR model is the $\binom{p}{q}$ model (called the P-out-of-Q model), which allows a request to obtain any k available resources from a pool of n resources.

Both the models are the same in expressive power.

model lends itself to a much more compact formation of a request

Every request in the model can be expressed in the AND-OR model and vice-versa

5. Unrestricted model

In the unrestricted model, no assumptions are made regarding the underlying structure of resource requests.

Only one assumption that the deadlock is stable is made and hence it is the most general model.

This model helps separate concerns: Concerns about properties of the problem (stability and deadlock) are separated from underlying distributed systems computations (e.g., message passing versus synchronous communication).

7. Explain in detail about deadlock handling strategies in a distributed environment

System model

A distributed system consists of a set of processors that are connected by a communication network.

The communication delay is finite but unpredictable.

A distributed program is composed of a set of n asynchronous processes $P_1, P_2, \dots, P_i, \dots, P_n$ that communicate by message passing over the communication network.

- Without loss of generality we assume that each process is running on a different processor.
- The processors do not share a common global memory and communicate solely by passing messages over the communication network.
- There is no physical global clock in the system to which processes have instantaneous access.
- The communication medium may deliver messages out of order, messages may be lost, garbled, or duplicated due to timeout and retransmission, processors may fail, and communication links may go down.
- The system can be modeled as a directed graph in which vertices represent the processes and edges represent unidirectional communication channels.

We make the following assumptions:

- The systems have only reusable resources.
- Processes are allowed to make only exclusive access to resources.
- There is only one copy of each resource.
- A process can be in two states, running or blocked. In the running state (also called active state),
- a process has all the needed resources and is either executing or is ready for execution.
- In the blocked state, a process is waiting to acquire some resource.

Deadlock handling strategies

There are three strategies for handling deadlocks,

- deadlock prevention,
- deadlock avoidance,
- deadlock detection.

Handling of deadlocks becomes highly complicated in distributed systems because no site has accurate knowledge of the current state of the system and because every inter-site communication involves a finite and unpredictable delay.

Deadlock prevention is commonly achieved either by having a process acquire all the needed resources simultaneously before it begins executing or by pre-empting a process that holds the needed resource.

This approach is highly inefficient and impractical in distributed systems.

In **deadlock avoidance** approach to distributed systems, a resource is granted to a process if the resulting global system state is safe.

Due to several problems, however, deadlock avoidance is impractical in distributed systems.

Deadlock detection requires an examination of the status of process– resource interactions for the presence of cyclic wait.

Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems.

8. Site S1, S2 , s3 contains process as listed below.

S1 : P17,P32,P61,P72

S2: P33, P44, P7

S3: P91,P6,P23,P28

Resources available are R1, R2, R7

All process in site 2 is demanding for Resource R7, while R7 is assigned to the first process in site three. Second process in site 2 is demanding resource held by third process in site 1. Evaluate the global state scenario and examine whether the resultant is in a deadlocked condition.
