UNIT III DISTRIBUTED MUTEX AND DEADLOCK

Distributed Mutual exclusion Algorithms: Introduction – Preliminaries – Lamport's algorithm – Ricart-Agrawala's Algorithm –– Token-Based Algorithms – Suzuki-Kasami's Broadcast Algorithm; Deadlock Detection in Distributed Systems: Introduction – System Model – Preliminaries – Models of Deadlocks – Chandy-Misra-Haas Algorithm for the AND model and OR Model.

3.1 DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS

- Mutual exclusion is a concurrency control property which is introduced to preventrace conditions.
- It is the requirement that a process cannot access a shared resource while anotherconcurrent process is currently present or executing the same resource.

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

- Message passing is the sole means for implementing distributed mutual exclusion.
- The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes insome consistent way.
- There are three basic approaches for implementing distributed mutual exclusion:

1. Token-based approach:

- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.
- Each requests for critical section contains a sequence number. This sequencenumber is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique.
- Eg: Suzuki-Kasami's Broadcast Algorithm

2. Non-token-based approach:

- A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successiveround of messages among sites.
- This approach use timestamps instead of sequence number to order requests for the critical section.
- When ever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme.
- Eg: Lamport's algorithm, Ricart–Agrawala algorithm

3. Quorum-based approach:

- Instead of requesting permission to execute the critical section from all othersites, Each site requests only a subset of sites which is called a quorum.
- Any two subsets of sites or Quorum contains a common site.
- This common site is responsible to ensure mutual exclusion.
- Eg: Maekawa's Algorithm

3.1.1 Preliminaries

- The system consists of N sites, S_1, S_2, S_3, …, S_N.
- Assume that a single process is running on each site.
- The process at site S_i is denoted by p_i. All these processes communicate asynchronously over an underlying communication network.
- A process wishing to enter the CS requests all other or a subset of processes by sending REQUEST messages, and waits for appropriate replies before entering theCS.
- While waiting the process is not allowed to make further requests to enter the CS.
- A site can be in one of the following three states: requesting the CS, executing the CS,or neither requesting nor executing the CS.
- In the requesting the CS state, the site is blocked and cannot make further requests forthe CS.
- In the idle state, the site is executing outside the CS.
- In the token-based algorithms, a site can also be in a state where a site holding thetoken is executing outside the CS. Such state is referred to as the idle token state.
- At any instant, a site may have several pending requests for CS. A site queues upthese requests and serves them one at a time.
- N denotes the number of processes or sites involved in invoking the critical section, Tdenotes the average message delay, and E denotes the average critical section execution time.

3.1.2 Requirements of mutual exclusion algorithms

- **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 fairchance 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 inthe system.

3.1.3 Performance metrics

- **Message complexity:** This is the number of messages that are required per CSexecution by a site.
- **Synchronization delay:** After a site leaves the CS, it is the time required and beforethe next site enters the CS. (Figure 3.1)
- **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. Thus, response time does not include the time a request waits at a site before its request messages have been sentout. (Figure 3.2)
- **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 executiontime.

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

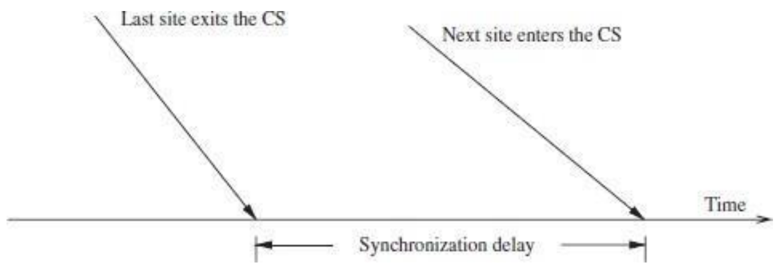


Figure 3.1 Synchronization delay

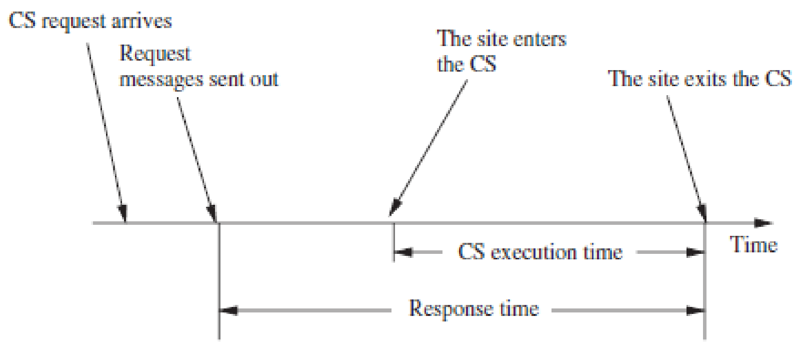


Figure 3.2 Response

TimeLow and High Load Performance:

- The performance of mutual exclusion algorithms is classified as two special loadingconditions, viz., “low load” and “high load”.
- The load is determined by the arrival rate of CS execution requests.
- Under ***low load*** conditions, there is seldom more than one request for the

criticalsection present in the system simultaneously.

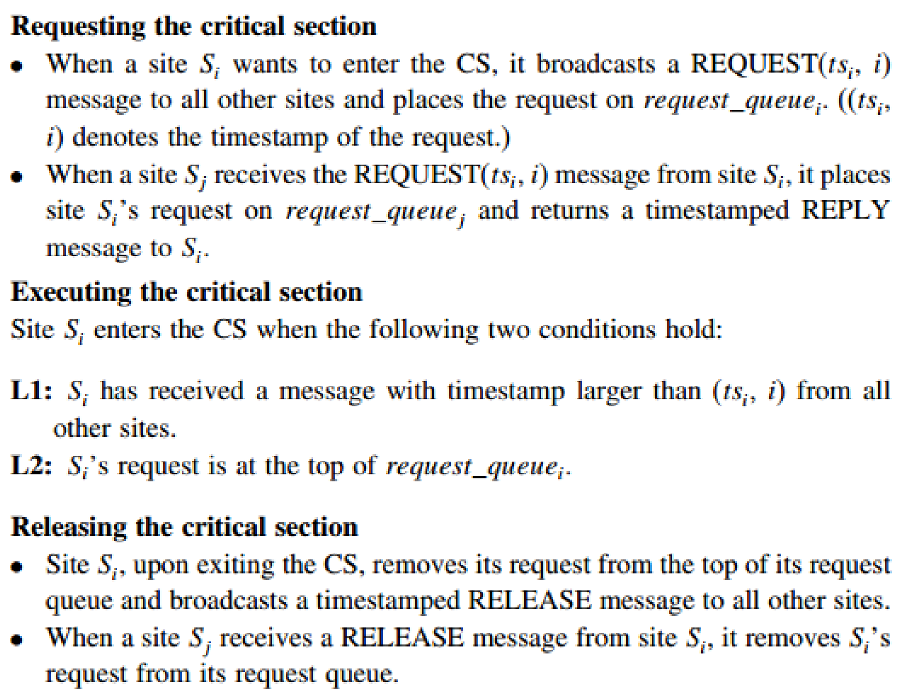
- Under ***heavy load*** conditions, there is always a pending request for critical section at asite.

Best and worst case performance

- In the best case, prevailing conditions are such that a performance metric attains the best possible value. For example, the best value of the response time is a roundtrip message delay plus the CS execution time, 2T +E.
- For examples, the best and worst values of the response time are achieved when load is, respectively, low and high;
- The best and the worse message traffic is generated at low and heavy load conditions, respectively.

3.2 LAMPORT'S ALGORITHM

- Lamport's Distributed Mutual Exclusion Algorithm is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.
- In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.
- In Lamport's Algorithm critical section requests are executed in the increasing order oftimestamps i.e a request with smaller timestamp will be given permission toexecute critical section first than a request with larger timestamp.
- Three type of messages (REQUEST, REPLY and RELEASE) are used andcommunication channels are assumed to follow FIFO order.
- A site send a REQUEST message to all other site to get their permission to entercritical section.
- A site send a REPLY message to requesting site to give its permission to enter thecritical section.
- A site send a RELEASE message to all other site upon exiting the critical section.
- Every site Si, keeps a queue to store critical section requests ordered by theirtimestamps.
- request_queuei denotes the queue of site Si.
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.



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.

Fig 3.1: Lamport's distributed mutual exclusion algorithm

To enter Critical section:

- When a site Si wants to enter the critical section, it sends a request message Request(tsi,i) to all other sites and places the request on request_queuei. Here, Tsi denotes the timestamp of Site Si.
- When a site Sj receives the request message REQUEST(tsi, i) from site Si, it returns a timestamped REPLY message to site Si and places the request of site Si on request_queuej

To execute the critical section:

- A site Si can enter the critical section if it has received the message with timestamp larger than (tsi, i) from all other sites and its own request is at the top of request_queuei.

To release the critical section:

- When a site Si exits the critical section, it removes its own request from the top of its request queue and sends a timestamped RELEASE message to all other sites. When a site Sj receives the timestamped RELEASE message from site Si, it removes the requestof Sia from its request queue.

Correctness

Theorem: Lamport's algorithm achieves mutual exclusion.

Proof: Proof is by contradiction.

- Suppose two sites Si and Sj are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites concurrently.
- This implies that at some instant in time, say t, both Si and Sj have their own

requests at the top of their request queues and condition L1 holds at them. Without loss of generality, assume that S_i 's request has smaller timestamp than the request of S_j .

- From condition L1 and FIFO property of the communication channels, it is clear that atinstant t the request of S_i must be present in request $queue_j$ when S_j was executing its CS. This implies that S_j 's own request is at the top of its own request queue whena smaller timestamp request, S_i 's request, is present in the request $queue_j$ – a contradiction!

Theorem: Lamport's algorithm is fair.

Proof: The proof is by contradiction.

- Suppose a site S_i 's request has a smaller timestamp than the request of another site S_j and S_j is able to execute the CS before S_i .
- For S_j to execute the CS, it has to satisfy the conditions L1 and L2. This implies thatat some instant in time say t, S_j has its own request at the top of its queue and it has alsoreceived a message with timestamp larger than the timestamp of its request from all other sites.
- But request queue at a site is ordered by timestamp, and according to our assumption S_i has lower timestamp. So S_i 's request must be placed ahead of the S_j 's request in therequest $queue_j$. This is a contradiction!

Message Complexity:

Lamport's Algorithm requires invocation of 3(N – 1) messages per critical section execution.These 3(N – 1) messages involves

- (N – 1) request messages
- (N – 1) reply messages
- (N – 1) release messages

Drawbacks of Lamport's Algorithm:

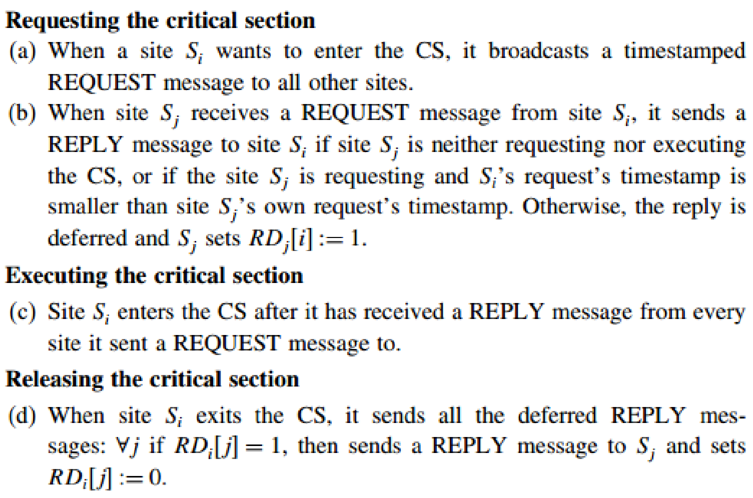
- **Unreliable approach**: failure of any one of the processes will halt the progress ofentire system.
- **High message complexity:** Algorithm requires 3(N-1) messages per critical sectioninvocation.

Performance:

Synchronization delay is equal to maximum message transmission time. It requires 3(N – 1)messages per CS execution. Algorithm can be optimized to 2(N – 1) messages by omitting the REPLY message in some situations.

3.3 RICART–AGRAWALA ALGORITHM

- Ricart–Agrawala algorithm is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala.
- This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm.
- It follows permission based approach to ensure mutual exclusion.
- Two type of messages (REQUEST and REPLY) are used and communication channels are assumed to follow FIFO order.
- A site send a REQUEST message to all other site to get their permission to enter critical section.
- A site send a REPLY message to other site to give its permission to enter the critical section.
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests.
- Smaller timestamp gets high priority over larger timestamp.
- The execution of critical section request is always in the order of their timestamp.



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 $RD_i[j] := 0$.

Fig 3.2: Ricart–Agrawala algorithm

To enter Critical section:

- When a site Si wants to enter the critical section, it send a timestamped REQUEST message to all other sites.
- When a site Sj receives a REQUEST message from site Si, It sends a REPLY message to site Si if and only if Site Sj is neither requesting nor currently executing the critical section.
- In case Site Sj is requesting, the timestamp of Site Si's request is smaller than its own request.
- Otherwise the request is deferred by site Sj.

To execute the critical section:

Site S_i enters the critical section if it has received the REPLY message from all other sites.

To release the critical section:

Upon exiting site S_i sends REPLY message to all the deferred requests.

Theorem: Ricart-Agrawala algorithm achieves mutual exclusion.

Proof: Proof is by contradiction.

- Suppose two sites S_i and S_j ‘ are executing the CS concurrently and S_i ’s request has higher priority than the request of S_j . Clearly, Si received S_j ’s request after it has made its own request.
- Thus, S_j can concurrently execute the CS with S_i only if Si returns a REPLY to S_j (in response to S_j ’s request) before Si exits the CS.
- However, this is impossible because S_j ’s request has lower priority. Therefore,Ricart- Agrawala algorithm achieves mutual exclusion.

Message Complexity:

Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involve:

- (N – 1) request messages
- (N – 1) reply messages

Drawbacks of Ricart–Agrawala algorithm:

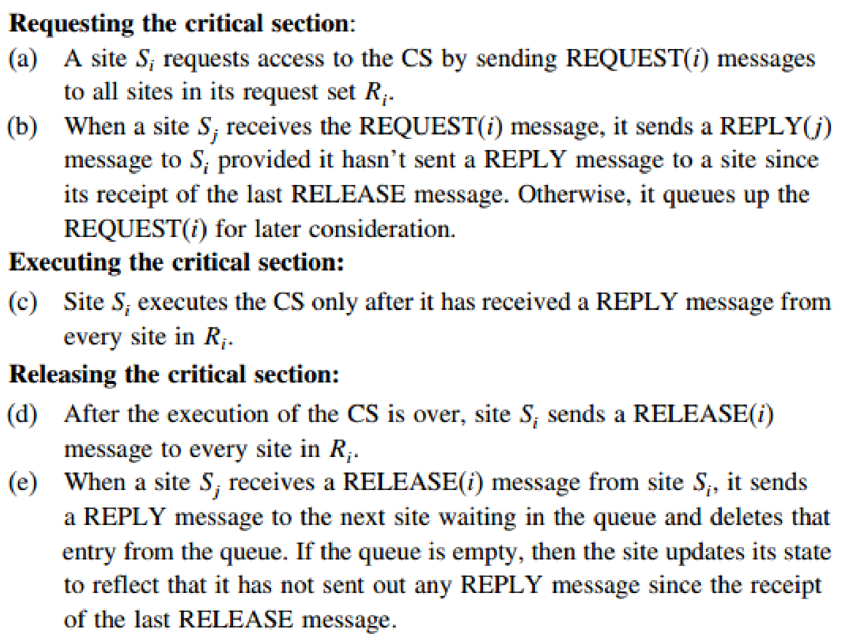
- **Unreliable approach:** failure of any one of node in the system can halt the progress of the system. In this situation, the process will starve forever. The problem of failure of node can be solved by detecting failure after some timeout.

Performance:

Synchronization delay is equal to maximum message transmission time It requires 2(N – 1) messages per Critical section execution.

3.4 MAEKAWA‘s ALGORITHM

- Maekawa’s Algorithm is quorum based approach to ensure mutual exclusion in distributed systems.



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.

Fig 3.3: Maekawa's Algorithm

- In permission based algorithms like Lamport's Algorithm, Ricart-Agrawala Algorithm etc. a site request permission from every other site but in quorum based approach, a site does not request permission from every other site but from a subset ofsites which is called quorum.
- Three type of messages (REQUEST, REPLY and RELEASE) are used.
- A site send a REQUEST message to all other site in its request set or quorum to get their permission to enter critical section.
- A site send a REPLY message to requesting site to give its permission to enter the critical section.
- A site send a RELEASE message to all other site in its request set or quorum upon exiting the critical section

The following are the conditions for Maekawa's algorithm:

M1 $(\forall i\ \forall j : i \neq j,\ 1 \leq i, j \leq N :: R_i \cap R_j \neq \phi)$.
M2 $(\forall i : 1 \leq i \leq N :: S_i \in R_i)$.
M3 $(\forall i : 1 \leq i \leq N :: |R_i| = K)$.
M4 Any site S_j is contained in K number of R_is, $1 \leq i, j \leq N$.

Maekawa used the theory of projective planes and showed that N = K(K – 1)+ 1.
This relation gives |Ri|= √N.

To enter Critical section:

- When a site Si wants to enter the critical section, it sends a request message REQUEST(i) to all other sites in the request set Ri.
- When a site Sj receives the request message REQUEST(i) from site Si, it returns a REPLY message to site Si if it has not sent a REPLY message to the site from the time it received the last RELEASE message. Otherwise, it queues up the request.

To execute the critical section:

- A site S_i can enter the critical section if it has received the REPLY message from all the site in request set R_i

To release the critical section:

- When a site S_i exits the critical section, it sends RELEASE(i) message to all other sites in request set R_i
- When a site S_j receives the RELEASE(i) message from site S_i, it send REPLY message to the next site waiting in the queue and deletes that entry from the queue
- In case queue is empty, site S_j update its status to show that it has not sent any REPLY message since the receipt of the last RELEASE message.

Correctness
Theorem: Maekawa's algorithm achieves mutual exclusion.

Proof: Proof is by contradiction.

- Suppose two sites Si and Sj are concurrently executing the CS.
- This means site Si received a REPLY message from all sites in Ri and concurrently site Sj was able to receive a REPLY message from all sites in Rj .
- If Ri ∩ Rj = {Sk }, then site Sk must have sent REPLY messages to both Si and Sj concurrently, which is a contradiction

Message Complexity:
Maekawa's Algorithm requires invocation of 3√N messages per critical section execution as the size of a request set is √N. These 3√N messages involves.

- √N request messages
- √N reply messages
- √N release messages

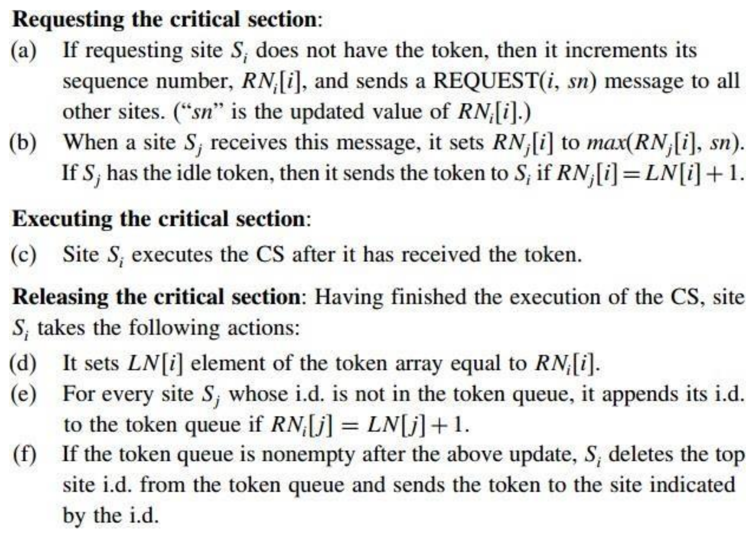
Drawbacks of Maekawa's Algorithm:
This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.

Performance:
Synchronization delay is equal to twice the message propagation delay time. It requires 3√n messages per critical section execution.

3.5 SUZUKI–KASAMI's BROADCAST ALGORITHM

- Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems.
- This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses REQUEST and REPLY messages to ensure mutual exclusion.
- In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token.
- Non-token based algorithms uses timestamp to order requests for the critical section where as sequence number is used in token based algorithms.

- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.



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.

Fig 3.4: Suzuki–Kasami‘s broadcast algorithm

To enter Critical section:

- When a site Si wants to enter the critical section and it does not have the token then it increments its sequence number RNi[i] and sends a request message REQUEST(i, sn) to all other sites in order to request the token.
- Here sn is update value of RNi[i]
- When a site Sj receives the request message REQUEST(i, sn) from site Si, it sets RNj[i] to maximum of RNj[i] and sni.eRNj[i] = max(RNj[i], sn).
  After updating RNj[i], Site Sj sends the token to site Si if it has token and RNj[i] = LN[i] + 1

To execute the critical section:

- Site Si executes the critical section if it has acquired the token.

To release the critical section:

After finishing the execution Site Si exits the critical section and does following:

- sets LN[i] = RNi[i] to indicate that its critical section request RNi[i] has been executed
- For every site Sj, whose ID is not prsent in the token queue Q, it appends its ID to Q if RNj[j] = LN[j] + 1 to indicate that site Sj has an outstanding request.
- After above updation, if the Queue Q is non-empty, it pops a site ID from the Q and sends the token to site indicated by popped ID.
- If the queue Q is empty, it keeps the token

Correctness

Mutual exclusion is guaranteed because there is only one token in the system and a site holds the token during the CS execution.

Theorem: A requesting site enters the CS in finite time.

Proof: Token request messages of a site Si reach other sites in finite time.

Since one of these sites will have token in finite time, site Si ’s request will be placed in the

token queue in finite time.
Since there can be at most N − 1 requests in front of this request in the token queue, site Si will get the token and execute the CS in finite time.

Message Complexity:
The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves

- (N – 1) request messages
- 1 reply message

Drawbacks of Suzuki–Kasami Algorithm:

- Non-symmetric Algorithm: A site retains the token even if it does not have requested for critical section.

Performance:
Synchronization delay is 0 and no message is needed if the site holds the idle token at the time of its request. In case site does not holds the idle token, the maximum synchronization delay is equal to maximum message transmission time and a maximum of N message is required per critical section invocation.

3.6 DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Deadlock can neither be prevented nor avoided in distributed system as the system is so vast that it is impossible to do so. Therefore, only deadlock detection can be implemented.The techniques of deadlock detection in the distributed system require the following:

- **Progress:**The method should be able to detect all the deadlocks in the system.
- **Safety:** The method should not detect false of phantom deadlocks.

There are three approaches to detect deadlocks in distributed systems.
Centralized approach:

- Here there is only one responsible resource to detect deadlock.
- The advantage of this approach is that it is simple and easy to implement, while the drawbacks include excessive workload at one node, single point failure which in turnsmakes the system less reliable.

Distributed approach:

- In the distributed approach different nodes work together to detect deadlocks. Nosingle point failure as workload is equally divided among all nodes.
- The speed of deadlock detection also increases.

Hierarchical approach:

- This approach is the most advantageous approach.
- It is the combination of both centralized and distributed approaches of deadlockdetection in a distributed system.
- In this approach, some selected nodes or cluster of nodes are responsible for deadlockdetection and these selected nodes are controlled by a single node.

System Model

- A distributed program is composed of a set of n asynchronous processes p1, p2, . .

. , pi , . . . , pn that communicates 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 belost garbled or duplicated due to timeout and retransmission, processors may fail and communication links may go down.

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 neededresources and is either executing or is ready for execution.
- In the blocked state, a process is waiting to acquire some resource.

Wait for graph

This is used for deadlock deduction. A graph is drawn based on the request and acquirement of the resource. If the graph created has a closed loop or a cycle, then there is adeadlock.

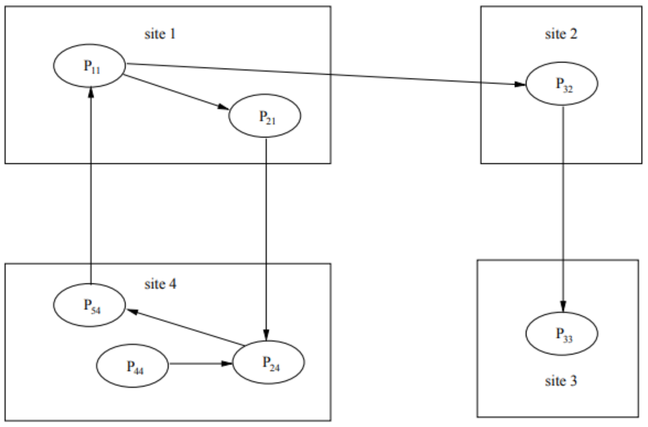


Fig 3.5: Wait for graph

Preliminaries

3.6.1 Deadlock Handling Strategies

Handling of deadlock becomes highly complicated in distributed systems because nosite has accurate knowledge of the current state of the system and because every inter-site communication involves a finite and unpredictable delay. There are three strategies for handling deadlocks:

- **Deadlock prevention:**
  - This is achieved either by having a process acquire all the needed resources simultaneously before it begins executing or by preempting a process whichholds the needed resource.
  - This approach is highly inefficient and impractical in distributed systems.
- **Deadlock avoidance:**
  - A resource is granted to a process if the resulting global system state is safe.This is impractical in distributed systems.
- **Deadlock detection:**
  - This requires examination of the status of process-resource interactions forpresence of cyclic wait.
  - Deadlock detection in distributed systems seems to be the best approach tohandle deadlocks in distributed systems.

3.6.2 Issues in deadlock Detection

Deadlock handling faces two major issues

1. Detection of existing deadlocks
2. Resolutionof detected deadlocks

Deadlock Detection

- Detection of deadlocks involves addressing two issues namely maintenance of theWFG and searching of the WFG for the presence of cycles or **knots**.
- In distributed systems, a cycle or knot may involve several sites, the search for cyclesgreatly 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, there are centralized, distributed, and hierarchical algorithms for deadlockdetection in distributed systems.

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 finite time. In other words, after allwait-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 which do not exist. This is also called as called **phantom or false deadlocks.**

Resolution of a Detected Deadlock

- Deadlock resolution involves breaking existing wait-for dependencies between theprocesses to resolve the deadlock.
- It involves rolling back one or more deadlocked processes and assigning theirresources to blocked processes so that they can resume execution.
- The deadlock detection algorithms propagate information regarding wait-fordependencies along the edges of the wait-for graph.
- When a wait-for dependency is broken, the corresponding information should beimmediately cleaned from the system.
- If this information is not cleaned in a timely manner, it may result in detection ofphantom deadlocks.

3.7 MODELS OF DEADLOCKS

The models of deadlocks are explained based on their hierarchy. The diagrams illustrate theworking of the deadlock models. P_a, P_b, P_c, $P_d$are passive processes that had already acquired the resources. $P_e$is active process that is requesting the resource.

3.7.1 Single Resource Model

- A process can have at most one outstanding request for only one unit of a resource.
- 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.

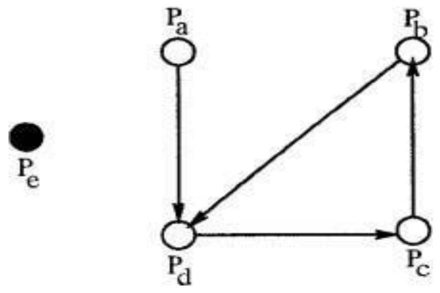


Fig 3.6: Deadlock in single resource model

3.7.2 AND Model

- In the AND model, a passive process becomes active (i.e., its activation condition isfulfilled) only after a message from each process in its dependent set has arrived.
- In the AND model, a process can request more than one resource simultaneously and therequest 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.
- Each node of the WFG in such a model is called an AND node.
- In the AND model, if a cycle is detected in the WFG, it implies a deadlock but not viceversa. That is, a process may not be a part of a cycle, it can still be deadlocked.

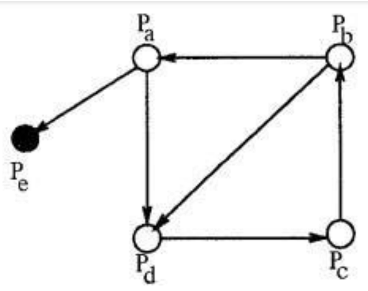


Fig 3.7: Deadlock in AND model

3.7.3 OR Model

- A process can make a request for numerous resources simultaneously and the requestis satisfied if any one of the requested resources is granted.
- Presence of a cycle in the WFG of an OR model does not imply a deadlockin the OR model.
- In the OR model, the presence of a knot indicates a deadlock.

Deadlock in OR model: a process Pi is blocked if it has a pending OR request to be satisfied.

- With every blocked process, there is an associated set of processes called **dependentset.**
- A process shall move from an idle to an active state on receiving a grant messagefrom any of the processes in its dependent set.
- A process is permanently blocked if it never receives a grant message from any of theprocesses in its dependent set.
- A set of processes S is deadlocked if all the processes in S are permanently blocked.
- In short, a processis deadlocked or permanently blocked, if the following conditionsare met:
  1. Each of the process is the set S is blocked.
  2. The dependent set for each process in S is a subset of S.
  3. No grant message is in transit between any two processes in set S.
- A blocked process P is the set S becomes active only after receiving a grant messagefrom a process in its dependent set, which is a subset of S.

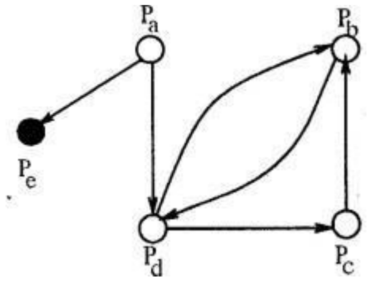


Fig 3.8: OR Model

3.7.4 $\binom{p}{q}$ Model (p out of q model)

- This is a variation of AND-OR model.
- This allows a request to obtain any k available resources from a pool of n resources. Both the models are the same in expressive power.
  - This favours more compact formation of a request.
  - Every request in this model can be expressed in the AND-OR model and vice-versa.
  - Note that AND requests for p resources can be stated as $\binom{p}{q}$ and OR requests for presources can be stated as $\binom{p}{1}$.

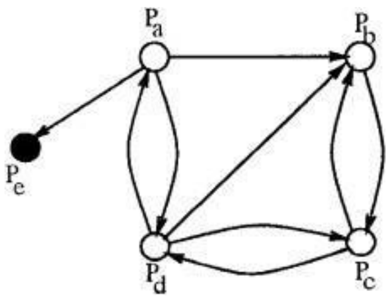


Fig 3.9: p out of q Model

3.7.5 Unrestricted model

- No assumptions are made regarding the underlying structure of resource requests.
- In this model, 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).

3.8 KNAPP'S CLASSIFICATION OF DISTRIBUTED DEADLOCK DETECTIONALGORITHMS

The four classes of distributed deadlock detection algorithm are:

1. Path-pushing
2. Edge-chasing
3. Diffusion computation
4. Global state detection

3.8.1 Path Pushing algorithms

- In path pushing algorithm, the distributed deadlock detection are detected bymaintaining an explicit global wait for graph.
- The basic idea is to build a global WFG (Wait For Graph) for each site of thedistributed system.
- At each site whenever deadlock computation is performed, it sends its local WFG toall the neighbouring sites.

- After the local data structure of each site is updated, this updated WFG is then passed along to other sites, and the procedure is repeated until some site has a sufficiently complete picture of the global state to announce deadlock or to establish that nodeadlocks are present.
- This feature of sending around the paths of global WFGhas led to the term path-pushing algorithms.
  Examples: Menasce-Muntz , Gligor and Shattuck, Ho and Ramamoorthy, Obermarck

3.8.2 Edge Chasing Algorithms

- The presence of a cycle in a distributed graph structure is be verified by propagating special messages called probes, along the edges of the graph.
- These probe messages are different than the request and reply messages.
- The formation of cycle can be deleted by a site if it receives the matching probe sentby it previously.
- Whenever a process that is executing receives a probe message, it discards this message and continues.
- Only blocked processes propagate probe messages along their outgoing edges.
- Main advantage of edge-chasing algorithms is that probes are fixed size messages which is normally very short.
  Examples: Chandy et al., Choudhary et al., Kshemkalyani–Singhal, Sinha–Natarajan algorithms.

3.8.3 Diffusing Computation Based Algorithms

- In diffusion computation based distributed deadlock detection algorithms, deadlockdetection computation is diffused through the WFG of the system.
- These algorithms make use of echo algorithms to detect deadlocks.
- This computation is superimposed on the underlying distributed computation.
- If this computation terminates, the initiator declares a deadlock.
- To detect a deadlock, a process sends out query messages along all the outgoing edges inthe WFG.
- These queries are successively propagated (i.e., diffused) through the edges of the WFG.
- When a blocked process receives first query message for a particular deadlock detection initiation, it does not send a reply message until it has received a reply message for every query it sent.
- For all subsequent queries for this deadlock detection initiation, it immediately sends back a reply message.
- The initiator of a deadlock detection detects a deadlock when it receives reply for every query it had sent out.
  Examples: Chandy–Misra–Haas algorithm for one OR model, Chandy–Herman algorithm

3.8.4 Global state detection-based algorithms

Global state detection based deadlock detection algorithms exploit the following facts:

1. A consistent snapshot of a distributed system can be obtained without freezing theunderlying computation.
2. If a stable property holds in the system before the snapshot collection is initiated, thisproperty will still hold in the snapshot.

Therefore, distributed deadlocks can be detected by taking a snapshot of the system and examining it for the condition of a deadlock

3.9 MITCHELL AND MERRITT'S ALGORITHM FOR THE SINGLE-RESOURCEMODEL

- This deadlock detection algorithm assumes a single resource model.
- This detects the local and global deadlocks each process has assumed two differentlabels namely private and public each label is accountant the process id guaranteesonly one process will detect a deadlock.
- Probes are sent in the opposite direction to the edges of the WFG.
- When a probe initiated by a process comes back to it, the process declares deadlock.

Features:

1. Only one process in a cycle detects the deadlock. This simplifies the deadlock resolution – this process can abort itself to resolve the deadlock. This algorithm canbe improvised by including priorities, and the lowest priority process in a cycle detects deadlock and aborts.
2. In this algorithm, a process that is detected in deadlock is aborted spontaneously, eventhough under this assumption phantom deadlocks cannot be excluded. It can be shown, however, that only genuine deadlocks will be detected in the absence of spontaneous aborts.

Each node of the WFG has two local variables, called labels:

1. a private label, which is unique to the node at all times, though it is not constant.
2. a public label, which can be read by other processes and which may not be unique.

Each process is represented as u/v where u and u are the public and private labels, respectively. Initially, private and public labels are equal for each process. A global WFGis maintained and it defines the entire state sof the system.

- The algorithm is defined by the four state transitions as shown in Fig.3.10, where z =inc(u, v), and inc(u, v) yields aunique label greater than both u and v labels that are notshown do not change.
- The transitions in the defined by the algorithm are block, activate , transmit anddetect.
- **Block** creates an edge in the WFG.
- Two messages are needed, one resource request and onemessage back to the blockedprocess to inform it of thepublic label of the process it is waiting for.

  Activate denotes that a process has acquired the resourcefrom the process it was waiting for.

- **Transmit** propagates larger labels in the opposite directionof the edges by sending aprobe message.

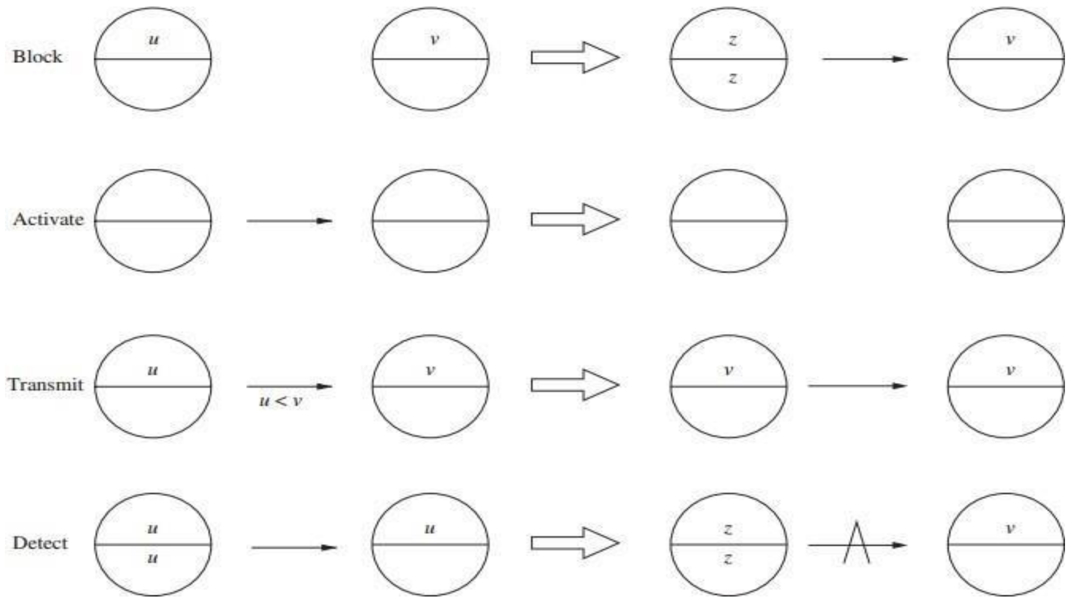


Fig 3.10: Four possible state transitions

- **Detect** means that the probe with the private label of some process has returned to it,indicating a deadlock.
- This algorithm can easily be extended to include priorities, so that whenever adeadlock occurs, the lowest priority process gets aborted.
- This priority based algorithm has two phases.
  1. The first phase is almost identical to the algorithm.
  2. The second phase the smallest priority is propagated around the circle. The propagation stops when one process recognizes the propagated priority as itsown.

Message Complexity:

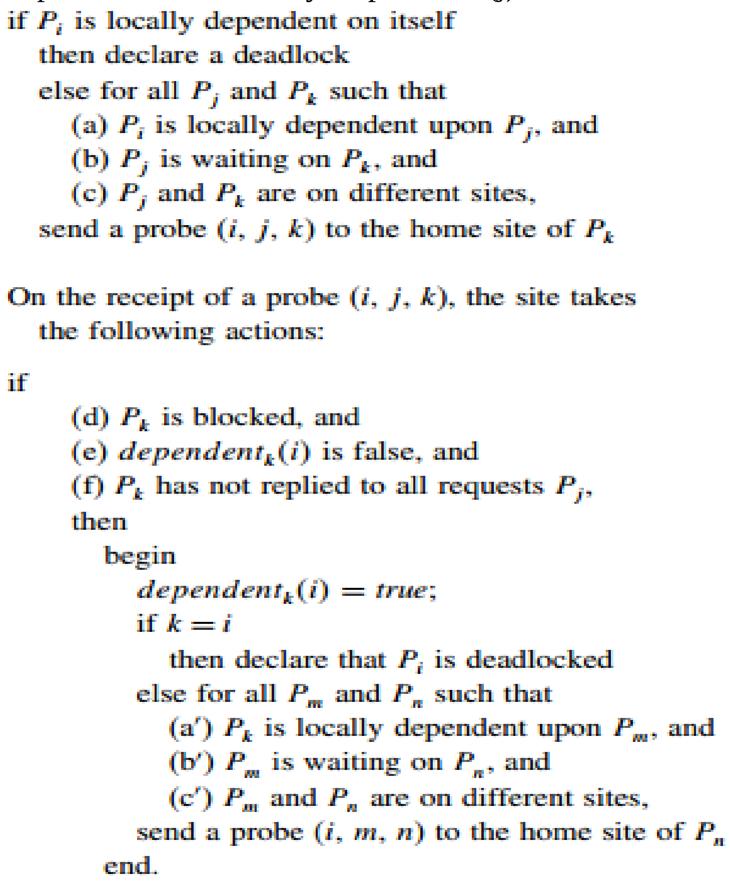
If we assume that a deadlock persists long enough to be detected, the worst-case complexityof the algorithm is s(s - 1)/2 Transmit steps, where s is the number of processes in the cycle.

3.10 CHANDY–MISRA–HAAS ALGORITHM FOR THE AND MODEL

- This is considered an edge-chasing, probe-based algorithm.
- It is also considered one of the best deadlock detection algorithms for distributedsystems.
- If a process makes a request for a resource which fails or times out, the process generates a probe message and sends it to each of the processes holding one or moreof its requested resources.
- This algorithm uses a special message called probe, which is a triplet (i, j,k), denotingthat it belongs to a deadlock detection initiated for process Pi andit is being sent by thehome site of process Pj to the home site of process Pk.
- Each probe message contains the following information:
    - the id of the process that is blocked (the one that initiates the probe message);
    - the id of the process is sending this particular version of the probe message;
    - the id of the process that should receive this probe message.
- A probe message travels along the edges of the global WFG graph, and a deadlock isdetected when a probe message returns to the process that initiated it.
- A process Pj is said to be dependent on another process Pk if there exists a sequence ofprocesses Pj, Pi1 , Pi2 , . . . , Pim, Pksuch that each process except Pkin the sequence is blocked and each process, except the Pj, holds a resource for which the previous processin the sequence is waiting.
- Process Pj is said to be locally dependent upon process Pk if Pj is dependent uponPkand both the processes are on the same site.
- When a process receives a probe message,it checks to see if it is also waiting forresources
- If not, it is currently using the needed resource and will eventually finish and releasethe resource.
- If it is waiting for resources, it passes on the probe message to all processes it knows to be holding resources it has itself requested.
- The process first modifies the probe message, changing the sender and receiver ids.
- If a process receives a probe message that it recognizes as having initiated,it knowsthere is a cycle in the system and thus, deadlock.

Data structures

Each process Pi maintains a boolean array, dependenti, where dependent(j) is true only if Pi knows that Pj is dependent on it. Initially, dependenti (j) is false for all i and j.



```
if $P_i$ is locally dependent on itself
   then declare a deadlock
   else for all $P_j$ and $P_k$ such that
      (a) $P_i$ is locally dependent upon $P_j$, and
      (b) $P_j$ is waiting on $P_k$, and
      (c) $P_j$ and $P_k$ are on different sites,
   send a probe $(i, j, k)$ to the home site of $P_k$

On the receipt of a probe $(i, j, k)$, the site takes
   the following actions:

if
      (d) $P_k$ is blocked, and
      (e) $dependent_k(i)$ is false, and
      (f) $P_k$ has not replied to all requests $P_j$,
      then
         begin
            $dependent_k(i) = true$;
            if $k = i$
               then declare that $P_i$ is deadlocked
            else for all $P_m$ and $P_n$ such that
               (a′) $P_k$ is locally dependent upon $P_m$, and
               (b′) $P_m$ is waiting on $P_n$, and
               (c′) $P_m$ and $P_n$ are on different sites,
            send a probe $(i, m, n)$ to the home site of $P_n$
         end.
```

Fig 3.11: Chandy–Misra–Haas algorithm for the AND

modelPerformance analysis

- In the algorithm, one probe message is sent on every edge of the WFG whichconnects processes on two sites.
- The algorithm exchanges at most m(n − 1)/2 messages to detect a deadlock thatinvolves m processes and spans over n sites.
- The size of messages is fixed and is very small (only three integer words).
- The delay in detecting a deadlock is O(n).

Advantages:

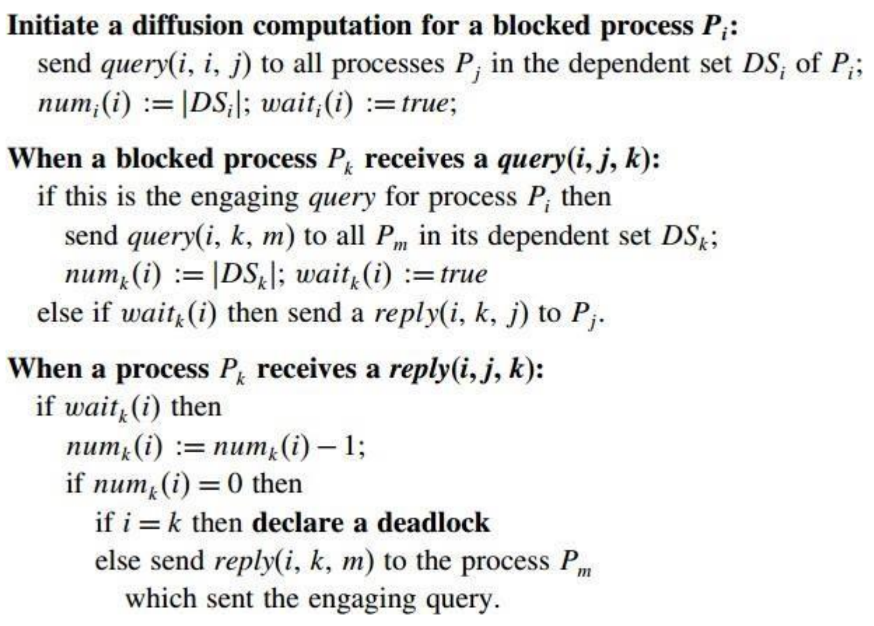
- It is easy to implement.
- Each probe message is of fixed length.
- There is very little computation.
- There is very little overhead.
- There is no need to construct a graph, nor to pass graph information to other sites.
- This algorithm does not find false (phantom) deadlock.
- There is no need for special data structures.

3.11 CHANDY–MISRA–HAAS ALGORITHM FOR THE OR MODEL

- A blocked process determines if it is deadlocked by initiating a diffusion computation.
- Two types of messages are used in a diffusion computation:
  - query(i, j, k)
  - reply(i, j, k)

denoting that they belong to a diffusion computation initiated by a process p_i and are beingsent from process p_j to process p_k.

- A blocked process initiates deadlock detection by sending query messages to allprocesses in its dependent set.
- If an active process receives a query or reply message, it discards it.
- When a blocked process Pk receives a query(i, j, k) message, it takes the followingactions:
  1. If this is the first query message received by Pk for the deadlock detection initiated by Pi, then it propagates the query to all the processes in its dependentset and sets a local variable num_k (i) to the number of query messages sent.
  2. If this is not the engaging query, then Pk returns a reply message to it immediately provided Pk has been continuously blocked since it received thecorresponding engaging query. Otherwise, it discards the query.
  - Process Pk maintains a boolean variable $wait_k$(i) that denotes the fact that it has been continuously blocked since it received the last engaging query fromprocess Pi.
  - When a blocked process Pk receives a reply(i, j, k) message, it decrementsnum_k(i) only if $wait_k$(i) holds.
  - A process sends a reply message in response to an engaging query only after ithas received a reply to every query message it has sent out for this engaging query.
  - The initiator process detects a deadlock when it has received reply messages toall the query messages it has sent out.



```
Initiate a diffusion computation for a blocked process $P_i$:
  send $query(i, i, j)$ to all processes $P_j$ in the dependent set $DS_i$ of $P_i$;
  $num_i(i) := |DS_i|$; $wait_i(i) := true$;

When a blocked process $P_k$ receives a $query(i, j, k)$:
  if this is the engaging query for process $P_i$ then
    send $query(i, k, m)$ to all $P_m$ in its dependent set $DS_k$;
    $num_k(i) := |DS_k|$; $wait_k(i) := true$
  else if $wait_k(i)$ then send a $reply(i, k, j)$ to $P_j$.

When a process $P_k$ receives a $reply(i, j, k)$:
  if $wait_k(i)$ then
    $num_k(i) := num_k(i) - 1$;
    if $num_k(i) = 0$ then
      if $i = k$ then declare a deadlock
      else send $reply(i, k, m)$ to the process $P_m$
        which sent the engaging query.
```

Fig 3.12: Chandy–Misra–Haas algorithm for the OR model

Performance analysis

- For every deadlock detection, the algorithm exchanges e query messages ande replymessages, where e = n(n – 1) is the number of edges.