ALGORITHMS FOR THE SINGLE RESOURCE MODEL, THE AND MODEL AND THE OR MODEL

1. MITCHELL AND MERRITT'S ALGORITHM FOR THE SINGLE-RESOURCE MODEL

- This deadlock detection algorithm assumes a single resource model.
- This detects the local and global deadlocks each process has assumed two different labels namely private and public each label is accountant the process id guarantees only 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 can be 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, even though 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 WFG is maintained and it defines the entire states of the system.

The algorithm is defined by the four state transitions as shown in Fig., 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 and detect.
- **Block** creates an edge in the WFG.
- Two messages are needed; one resource request and onemessage back to the blocked process 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 a probe message.

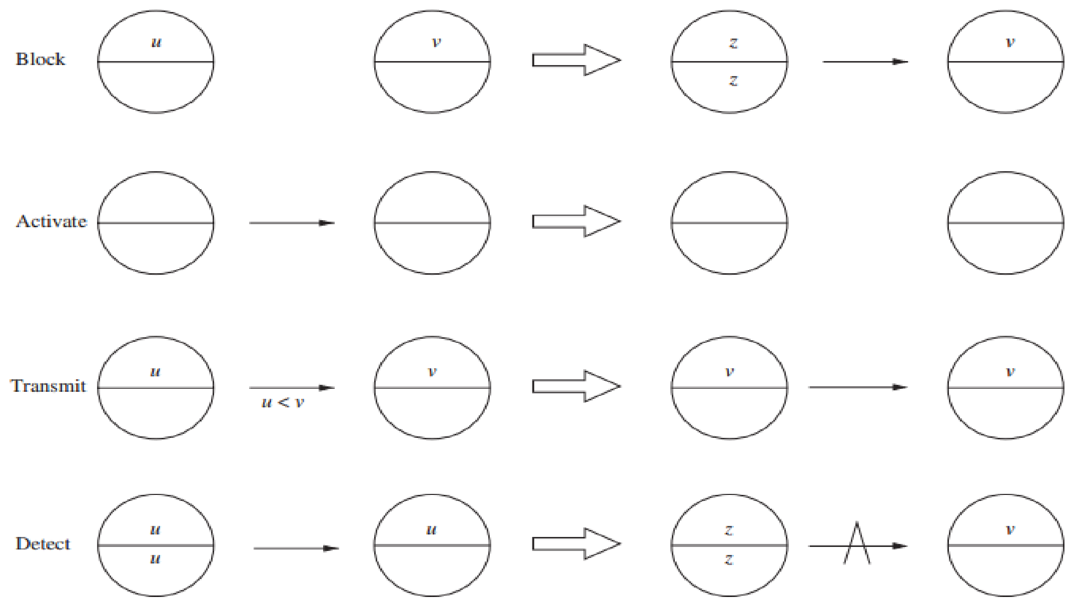

Fig : Four possible state transitions

- **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 a deadlock 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 its own.

Message Complexity:

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

2. 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 distributed systems.
- 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 more of its requested resources.
- This algorithm uses a special message called probe, which is a triplet (i, j,k), denoting that it belongs to a deadlock detection initiated for process Pi andit is being sent by the home 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 is detected when a probe message returns to the process that initiated it.
- A process P_j is said to be dependent on another process P_k if there exists a sequence of processes P_j, P_{i1} , P_{i2} , . . . , P_{im}, $P_k$such that each process except $P_k$in the sequence is blocked and each process, except the P_j, holds a resource for which the previous process in the sequence is waiting.
- Process P_j is said to be locally dependent upon process P_k if P_j is dependent upon $P_k$and both the processes are on the same site.
- When a process receives a probe message,it checks to see if it is also waiting for resources
- If not, it is currently using the needed resource and will eventually finish and release the 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 knows there is a cycle in the system and thus, deadlock.

Datastructures

Each process Pi maintains a boolean array, dependent i, where dependent (j) is true only if Pi knows that Pj is dependent on it. Initially, dependent i (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 : Chandy–Misra–Haas algorithm for the AND model

Performance analysis

- In the algorithm, one probe message is sent on every edge of the WFG which connects processes on two sites.
- The algorithm exchanges at most m(n − 1)/2 messages to detect a deadlock that involves 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. 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 being sent from process p_j to process p_k.
- A blocked process initiates deadlock detection by sending query messages to all processes 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 following actions:

  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 dependent set 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 the corresponding 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 from process Pi.
  - When a blocked process Pk receives a reply(i, j, k) message, it decrements num_k(i) only if $wait_k$ (i) holds.
  - A process sends a reply message in response to an engaging query only after it has 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 to all 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 : Chandy–Misra–Haas algorithm for the OR model

Performance analysis

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

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 turns makes the system less reliable.

Distributed approach:

- In the distributed approach different nodes work together to detect deadlocks. No single 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 deadlock detection in a distributed system.
- In this approach, some selected nodes or cluster of nodes are responsible for deadlock detection 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 be lost 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 needed resources 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 a deadlock.

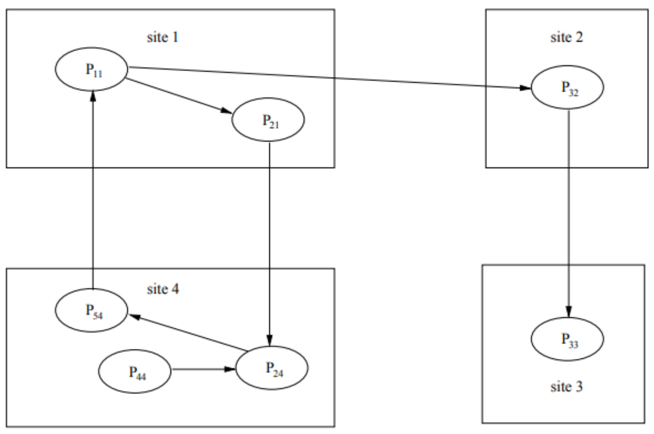

Fig : Wait for graph

Preliminaries

Deadlock Handling Strategies

Handling of deadlock 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. 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 which holds 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 for presence of cyclic wait.
  - Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems.

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 the WFG 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 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, there are centralized, distributed, and hierarchical algorithms for deadlock detection 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 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 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 the processes to resolve the deadlock.
- It involves rolling back one or more deadlocked processes and assigning their resources to blocked processes so that they can resume execution.
- The deadlock detection algorithms propagate information regarding wait-for dependencies along the edges of the wait-for graph.
- When a wait-for dependency is broken, the corresponding information should be immediately cleaned from the system.
- If this information is not cleaned in a timely manner, it may result in detection of phantom deadlocks.

KNAPP'S CLASSIFICATION OF DISTRIBUTED DEADLOCK DETECTION ALGORITHMS

The four classes of distributed deadlock detection algorithm are:

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

Path Pushing algorithms

- In path pushing algorithm, the distributed deadlock detection are detected by maintaining an explicit global wait for graph.
- The basic idea is to build a global WFG (Wait For Graph) for each site of the distributed system.
- At each site whenever deadlock computation is performed, it sends its local WFG to all 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 no deadlocks 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

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 sent by 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.

Diffusing Computation Based Algorithms

- In diffusion computation based distributed deadlock detection algorithms, deadlock detection 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 in the 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

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 the underlying computation.
2. If a stable property holds in the system before the snapshot collection is initiated, this property 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

MODELS OF DEADLOCKS

The models of deadlocks are explained based on their hierarchy. The diagrams illustrate the working 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.

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.

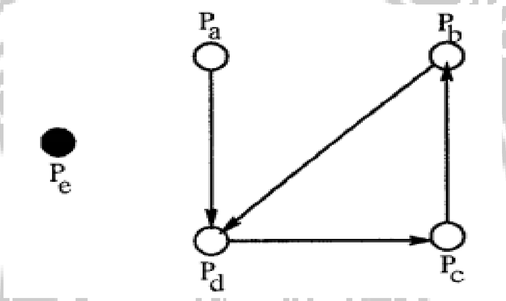

Fig: Deadlock in single resource model

AND Model

- In the AND model, a passive process becomes active (i.e., its activation condition is fulfilled) 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 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.
- 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 vice versa. That is, a process may not be a part of a cycle, it can still be deadlocked.

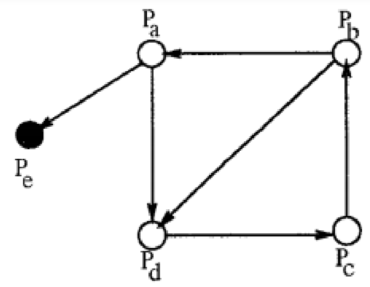

Fig : Deadlock in AND model

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.
- Presence of a cycle in the WFG of an OR model does not imply a deadlock in 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 **dependent set.**
- A process shall move from an idle to an active state on receiving a grant message from any of the processes in its dependent set.
- A process is permanently blocked if it never receives a grant message from any of the Processes 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 conditions are 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 message from a process in its dependent set, which is a subset of S.

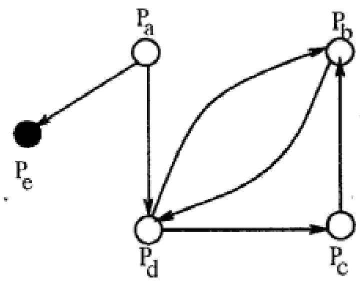

Fig: OR Model

$\binom{p}{q}$ 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 p resources can be stated as $\binom{p}{1}$.

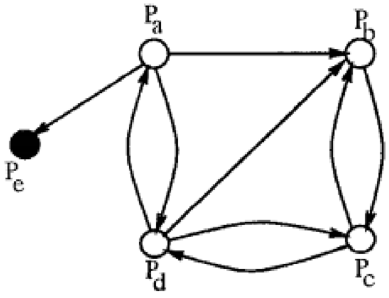

Fig: p out of q Model

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).