3.6 DEADLOCK

Deadlocks are a fundamental problem in distributed systems and deadlock detection in distributed systems has received considerable attention in the past. In distributed systems, resources may be requested by a process in any order, which may not be known a priori, and a process can request a resource while holding others. If the allocation sequence of process resources is not controlled in such environments, deadlocks can occur. A deadlock can be defined as a condition where resources are requested by a set of processes that are held by other processes in the set.

3.6.1 Necessary Conditions for Deadlock

The following conditions are necessary for a deadlock situation to occur in a system:

1. **Mutual-exclusion condition:** If a process is holding a resource, any other process that is requesting for that resource must wait until the resource has been released.
2. **Hold-and-wait condition:** Processes are allowed to request for new resources without releasing the resources that they are currently holding.

3. **No-preemption condition:** A resource that has been allocated to a process becomes available for allocation to another process only after it has been voluntarily released by the process holding it.
4. **Circular-wait condition:** A circular chain must be formed by two or more processes in which each process is waiting for a resource that is held by the next process of the chain.

All four conditions must hold simultaneously in a system for a deadlock to occur. If anyone of them is absent, no deadlock can occur.

3.6.2 Deadlock Modelling

For deadlock modelling, a directed graph, called a resource allocation graph, is used in which both the set of nodes and the set of edges are partitioned into two types, resulting in the following graph elements:

1. **Process nodes:** A process node represents a process of the system. It is represented by a circle, with the name of the process written inside the circle (nodes P_1, P_2, and P_3 of Fig. 3.12).
2. **Resource nodes:** A resource node represents a resource of the system. It is represented by rectangle with the name of the resource written in the rectangle. A resource R, can have many units in the system, each unit is represented as a bullet inside the rectangle.

   For example, Fig. 3.12, there is one unit of resource R_1, two units of R_2.
3. **Assignment edges:** This is an directed edge from a resource node to a process node. It indicated that the resource is currently held by the process. In multiple units of a resource type, the tail of an assignment edge touches one of the bullets in the rectangle to indicate that only one unit of the resource is held by that process. Edges (R_1, PI), (R_2, P_3) are the assignment edges.
4. **Request edges:** A request edge is a directed edge from a process node to a resource node. It indicates that the process made a request for a unit of the resource type and is currently waiting for that resource. Edges (PI, R_2) and (P_2, R_l) are the two request edges.

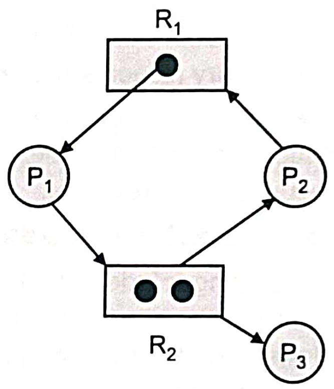

Fig. 3.12: Resource Allocation Graph

The following diagram represents a Resource Allocation Graph.

It gives the following information.

- There exist three processes in the system P_1, P_2 and P_3.
- There are two resources in the system R_1 and R_2.
- R_1 has a single instance and R_2 has two instances.
- Process P_1 is holding R_1's one instance and is waiting for R_2's one instance.
- Process P_2 is holding R_2's one instance and is waiting for R_1's one instance.
- Process P_3 is holding R_2's one instance and is not waiting for any resource.

Necessary and Sufficient Conditions for Deadlock in a Resource Allocation Graph:

- A cycle is a necessary condition for deadlock.
- If there is only a single unit of each resource type involved in the cycle, a cycle is both a necessary and a sufficient condition for a deadlock to exist
- In the cycle, if one or more of the resource types involved have more than one unit then for a deadlock to exist, a knot is the sufficient condition.

Wait-for Graph:

When all the resource types have only a single unit each, a simplified form of resource allocation graph is normally used. By eliminating the resource nodes and collapsing the appropriate edges, the original resource allocation graph is transformed into simplified graph. This simplification is according to the observation that a resource can always be identified by its current owner (process holding it). Fig. 3.13 shows an example of a resource allocation graph and its simplified form. The simplified graph is commonly known as a wait-for graph (WFG) because it clearly shows which processes are waiting for which other processes. For instance, in the WFG of Fig 3.13, processes PI and P_3 are waiting for P_2 and process P_2 is waiting for P_1. Since WFG is constructed only when each resource type has only a single unit, a cycle is both a necessary and sufficient condition for deadlock in a WFG.

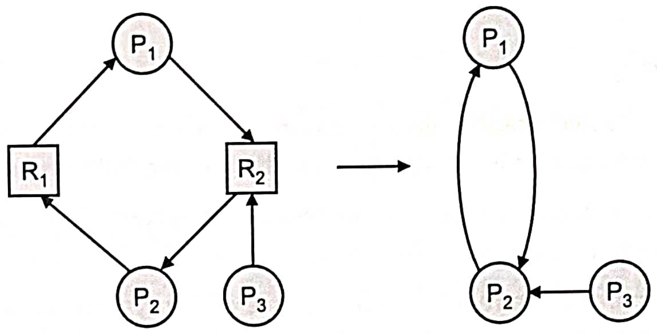

Fig. 3.13: Resource Allocation Graph to Wait for Graph

The problem of deadlocks has been generally studied in distributed systems under the following models :

- The system has only reusable resources.
- Only exclusive access of resources is allowed for processes.
- Every resource has only one copy.

A process can be in two states, running or blocked. A process in the running state (or active state) has all the resources it needs and the process is executing or is ready for execution. In the blocked state, a process is waiting to acquire some resource.

3.6.3 Handling Deadlocks In Distributed Systems

Handling of deadlocks in distributed systems is more complex than in centralized systems because the resources, the processes, and other relevant information are scattered on different nodes of the system. The following are the three methods to handle deadlocks:

1. **Avoidance:** The allocation of resources is done to avoid deadlocks.
2. **Prevention:** Constraints are imposed on the ways in which processes request resources in order to prevent deadlocks.
3. **Detection and recovery:** Deadlocks are allowed to occur and a detection algorithm is used to detect them. When a deadlock is identified, it is resolved by certain means.

3.6.4 Deadlock Detection

Deadlock detection uses an algorithm that keeps examining the state of the system to determine whether a deadlock has occurred. When a deadlock is detected, the system takes some action to recover from the deadlock. Some methods for deadlock detection in distributed systems are presented below.

3.6.4.1 Centralized Approach

(a) Completely Centralized Algorithm:

In the centralized deadlock detection approach, there is a local co-ordinator at each site that maintains a WFG for its local resources, and there is a central co-ordinator (also known as a centralized deadlock detection) that is responsible for combining all the individual WFGs. Using information obtained from the local co-ordinators of all the sites, the central co-ordinator creates the global WFG. In this approach, deadlock detection is performed as follows:

1. If a cycle exists in the local WFG of any site, it represents a local deadlock. Such deadlocks are detected and resolved locally by the local co-ordinator of the site.
2. Deadlocks involving resources at two or more sites get reflected as cycles in the global WFG. Therefore, such deadlocks are detected and resolved by the central co-ordinator.

The local co-ordinators send local state information to the central co-ordinator in the form of messages.

Disadvantages:

1. It is susceptible to the central co-ordinator failing. Hence special provision for handling such faults have to be made. One approach is to provide a backup central co-ordinator that duplicates the job of the central co-ordinator.
2. The centralized co-ordinator can constitute performance bottleneck in large systems having too many sites.
3. The centralized co-ordinator may detect false deadlocks.

(b) HO Ramamurthy (Two-Phase Algorithm):

A resource status table is kept by the central or control site in this technique, and if a cycle is found, the system is not immediately considered to be in a deadlock., instead the cycle is verified again as the system is distributed some or the other resources is vacant or freed by sites at every instant of time. Now, after checking if a cycle is detected again then, the system is said to be deadlock. Phantom deadlock is less likely to occur using this method, but time consumption is higher.

(c) HO Ramamurthy (One Phase Algorithm):

A process table and resource status table is kept by the control or central site. The system is said to be deadlocked if the cycle is detected in both process and resource tables. This method has less time consumption but more space complexity.

3.6.4.2 Distributed Approach

(a) Chandy Misra_Haas Algorithm:

It is considered to be the best algorithm for detecting global deadlocks in distributed systems. The algorithm allows a process to request for multiple resources at a time. It is presumed that each process in the system is assigned a unique identifier.

When a request for a resource made by a process times out or fails, the process will create and send a probe message to every processes that is in procession of requested resources.

Every probe message has the information given below:

- the process id of the blocked process (the process initiating the probe message);
- the process id of the process that is sending the probe message; and
- the process id of the process receiving this probe message.

A process determines if it is waiting for resources when it receives a probe message. If not, the required resource is currently being used and will ultimately be finished and released. If it is waiting for resources, it forwards the probe message to every process it is aware of that is currently holding the resources it has itself requested. The process changes the sender and receiver ids of the probe message as follows:

1. The first field is left unchanged.
2. The process changes the second field to its own process id.
3. The third field is changed to the process id that will be the new recipient of this message.

Every process receiving the probe message repeats this procedure. If the probe message returns back to the original sender (the process whose identifier is in the first field of the message), it becomes aware that the system has a cycle and thus a deadlock exists.

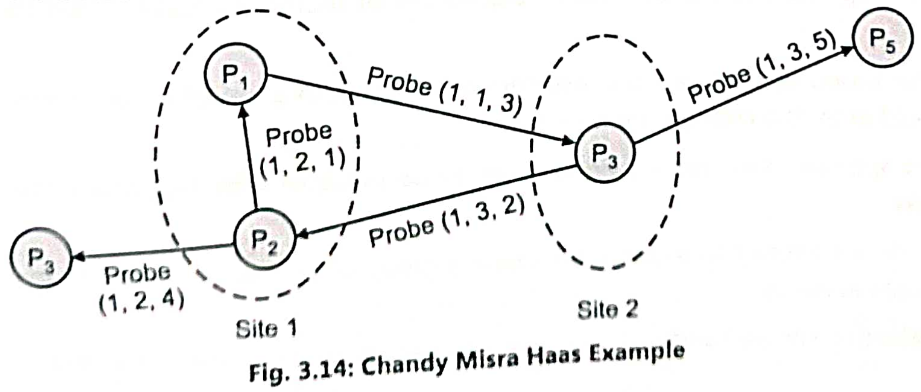

Fig. 3.14: Chandy Misra Haas Example

Consider example in Fig. 3.14 as shown in figure probe message is initiated by P_1 hence all the messages show P_1 as the initiator. When this probe message reaches process P_3, it updates it and forwards it to two more processes P_2 and P_5. P_2 updates the probe message and forwards it to P_4 and P_1. The probe message eventually returns to process P_1. P_1 detects the cycle and thus the Deadlock.

Performance:

The algorithm requires at most exchange of $\frac{m(n-1)}{2}$ messages to detect deadlock. Here, number of processes is denoted by m and number of sites is denoted by n. The delay in detecting the deadlock is O(n).

Advantages:

- No special data structure required. The deadlock detection uses a small probe message which consists of three integers.
- Only a small amount of computation is needed at each site and overhead is minimal.
- In contrast to other deadlock detection algorithms, this algorithm does not need construction of any graphs or the passing of graph information to other sites.
- False deadlock, also known as phantom deadlock, is not reported by the algorithm.

Disadvantages:

- Distributed detection algorithm's main drawback is that not all sites are aware of the processes involved in the deadlock; this makes resolution difficult.

SUMMARY