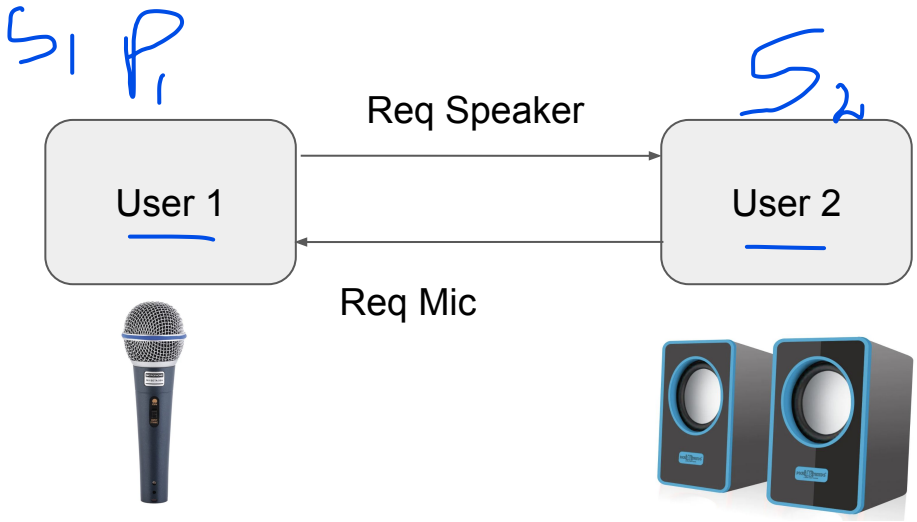
CS 3551 DISTRIBUTED COMPUTING

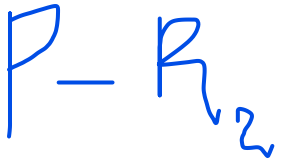
UNIT III DISTRIBUTED MUTEX AND DEADLOCK 10

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.

Deadlock



What is deadlock?

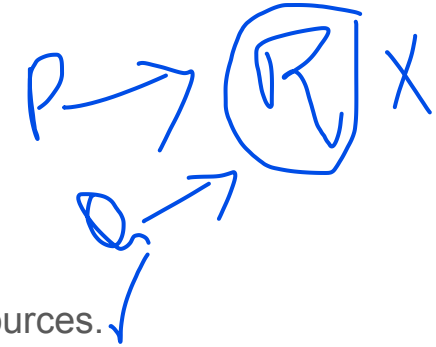


1. A deadlock can be defined as a condition where a set of processes request resources that are held by other processes in the set.
2. **Handling Deadlocks:**
   a. **Deadlock prevention** is commonly achieved by either having a process acquire all the needed resources simultaneously before it begins execution or by pre-empting a process that holds the needed resource.
   b. **In the deadlock avoidance** approach to distributed systems, a resource is granted to a process if the resulting global system is safe.
   c. **Deadlock detection** requires an examination of the status of the process–resources interaction for the presence of a deadlock condition. To resolve the deadlock, we have to abort a deadlocked process

System Model

1. A distributed system consists of a set of processors that are connected by a communication network.
2. The communication delay is finite but unpredictable.
3. A distributed program is composed of a set of n asynchronous processes P1, P2, , Pi, , Pn that communicate by message passing over the communication network.
4. Without loss of generality we assume that each process is running on a different processor.
5. The processors do not share a common global memory and communicate solely by passing messages over the communication network.
6. There is no physical global clock in the system to which processes have instantaneous access.
7. 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.
8. The system can be modeled as a directed graph in which vertices represent the processes and edges represent unidirectional communication channels

System Model continued…



- 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.

- Running state (also called active state), a process has all the needed resources and is either executing or is ready for execution.
- Blocked state, a process is waiting to acquire some resource.

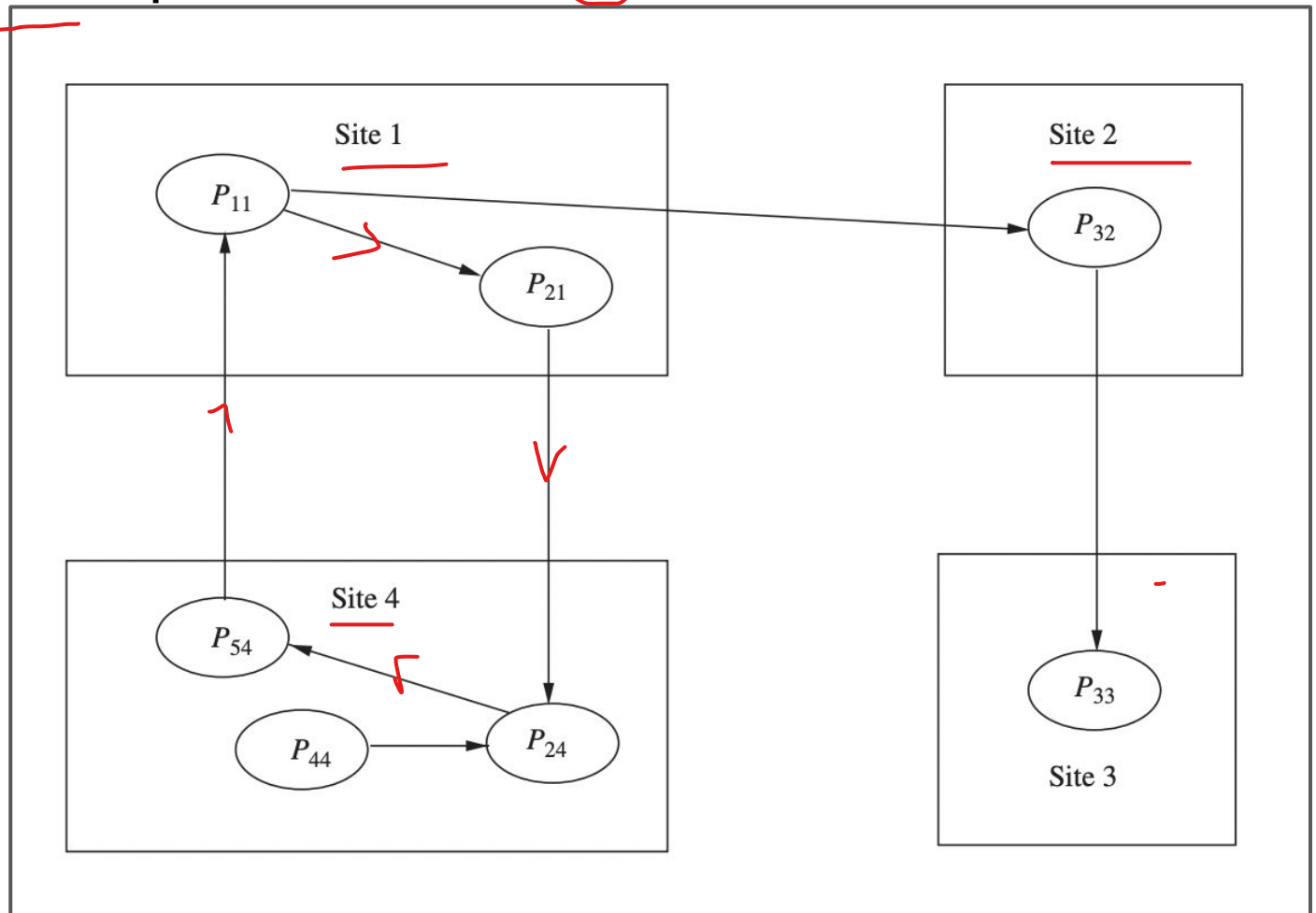
LIKE

COMMENT

SHARE

SUBSCRIBE

Wait for Graph



Wait for Graph

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 P_1 to mode P_2 if P_1 is blocked and is waiting for P_2 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 shows a WFG, where process P_{11} of site 1 has an edge to process P_{21} of site 1 and an edge to process P_{32} of site 2. Process P_{32} of site 2 is waiting for a resource that is currently held by process P_{33} of site 3. At the same time process P_{21} at site 1 is waiting on process P_{24} at site 4 to release a resource, and so on. If P_{33} starts waiting on process P_{24}, then processes in the WFG are involved in a deadlock depending upon the request model.

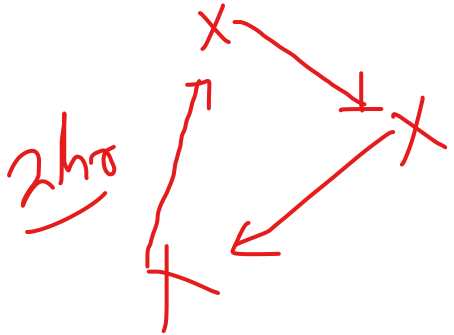
Issues in Deadlock Detection

- Detection of deadlocks involves addressing two issues: maintenance of the WFG and searching of the WFG for the presence of cycles (or knots).
- 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. Many algorithms are there for the same.

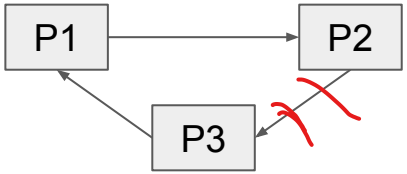
Correctness criteria

A deadlock detection algorithm must satisfy the following two conditions:

- **Progress (no undetected deadlocks)** The algorithm must detect all existing deadlocks in a finite time. Once a deadlock has occurred, the deadlock detection activity should continuously progress until the deadlock is detected. 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.
- **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 but whose different segments existed in the system at different times. This is the main reason why many deadlock detection algorithms reported in the literature are incorrect.



Issues in Deadlock - Resolution of Deadlocks



- 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.
- Note that several deadlock detection algorithms propagate information regarding wait-for dependencies along the edges of the wait-for graph.
- Therefore, 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.

LIKE

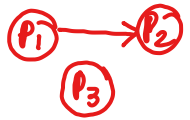
COMMENT

SHARE

SUBSCRIBE

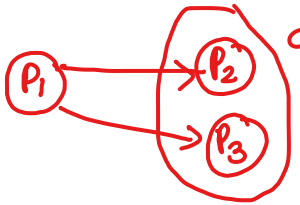
Models of Deadlock

1. Single Resource Model → @most 1 outstanding request for single resource
   out degree of node = 1



2. AND Model → R1, R2 cycle → deadlock

3. OR Model → R1, R2



dependent set
P1 – permanent block
set S ⇒ ∀a ∈ S ⇒ a – permanently block

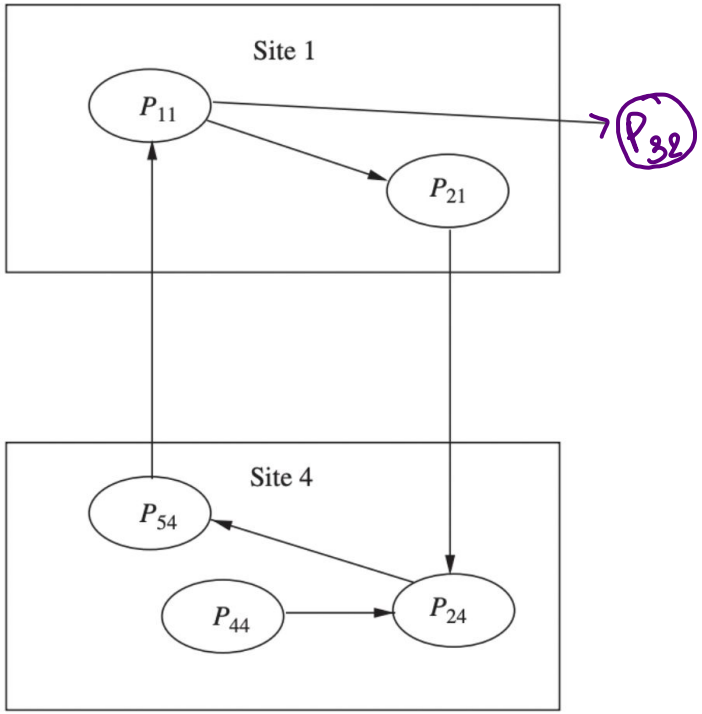
4. AND-OR Model → R1 R2 or R3 ⇒ graph X → OR-model

knot
A, B, C

5. $\binom{p}{q}$ model → req p resources from a pool
   $\binom{p}{p}$ – AND $\binom{1}{p}$ – OR of q resources

6. Unrestricted Model

1. Single Resource Model
   a. a process can have at most one outstanding request for only one unit of a resource.
   b. 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. AND Model
   a. 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.
   b. The requested resources may exist at different locations.
   c. The out degree of a node in the WFG for AND model can be more than 1.
   d. 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.

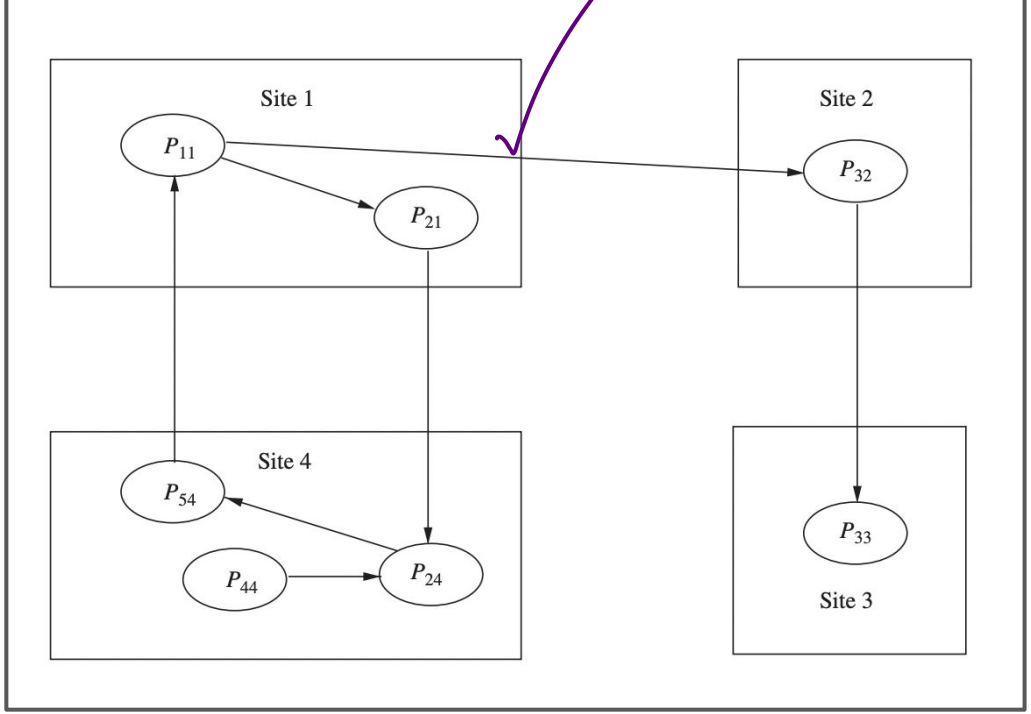


Cycle => Deadlock but not vice versa

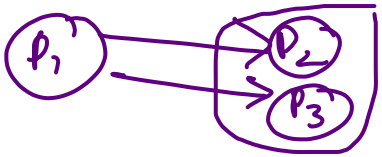
$P_{11} \rightarrow P_{21} \rightarrow P_{24} \rightarrow P_{54} \rightarrow P_{11}$,

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.
- In the OR model, the presence of a knot indicates a deadlock . In a WFG, a vertex v is in a knot if for all u :: u is reachable from v : v is reachable from u.



When deadlock occurs 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.
- Intuitively, a set of processes S is deadlocked if all the processes in S are permanently blocked.
- To formally state that a set of processes is deadlocked, the following conditions hold true:
  - Each of the process is the set S is blocked.
  - The dependent set for each process in S is a subset of S.
  - No grant message is in transit between any two processes in set S

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 or z).
- The requested resources may exist at different locations.
- To detect the presence of deadlocks in such a model, there is no familiar construct of graph theory using WFG.
- Since a deadlock is a stable property (i.e., once it exists, it does not go away by itself), this property can be exploited and a deadlock in the AND-OR model can be detected by repeated application of the test for OR-model deadlock.

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. However, $\binom{p}{q}$ model lends itself to a much more compact formation of a request.

Every request in the $\binom{p}{q}$ model can be expressed in the AND-OR model and vice-versa. Note that AND requests for p resources can be stated as $\binom{p}{p}$ and OR requests for p resources can be stated as $\binom{p}{1}$.

Unrestricted Model

- In this model, only one assumption that the deadlock is stable is made and hence it is the most general model.
- concerns about properties of the problem (stability and deadlock) are separated from underlying distributed systems computations (e.g., message passing versus synchronous communication).
- Hence, these algorithms can be used to detect other stable properties as they deal with this general model.
- But, these algorithms are of more theoretical value for distributed systems since no further assumptions are made about the underlying distributed systems computations which leads to a great deal of overhead

LIKE

COMMENT

SHARE

SUBSCRIBE

CS 3551 DISTRIBUTED COMPUTING

UNIT III DISTRIBUTED MUTEX AND DEADLOCK **10**

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.

Chandy–Misra–Haas algorithm for the AND model ⇒ deadlock detection

① Special msg (probe)

from – to

(i, j, P_k)

- to home site of P_k
- from home site of P_j
- deadlock detection for P_i

deadlock if ⇒ probe reaches the sender

② A dependent on G_2 if

③ A locally dependent on G_2 if

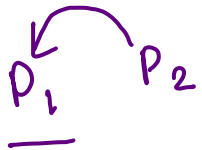
R_1 R_2 R_3 R_4

A B C D E F G_2

blocked

Same-site

ABC



DS

dependent$_i$[j]

dependent$_1$[2] = True

Algorithm

P_i, P_j, P_k

(1)

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

(i, j, k)

(2)

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

if
(1) (d) P_k is blocked, and
(2) (e) $dependent_k(i)$ is false, and
(3) (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.

(k, m, n)

Algorithm 10.1 Chandy–Misra–Haas algorithm for the AND model [6].

Key Points

The 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 P_i and it is being sent by the home site of process P_j to the home site of process P_k. 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.

Data structures

Each process P_i maintains a boolean array, $dependent_i$, where $dependent_i(j)$ is true only if P_i knows that P_j is dependent on it. Initially, $dependent_i(j)$ is false for all i and j.

Performance

- In the algorithm, one probe message (per deadlock detection initiation) is sent on every edge of the WFG which connects processes on two sites.
- Thus, 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).

LIKE

COMMENT

SHARE

SUBSCRIBE

CS 3551 DISTRIBUTED COMPUTING

UNIT III DISTRIBUTED MUTEX AND DEADLOCK **10**

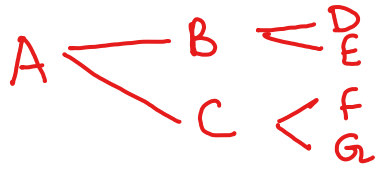
Distributed Mutual exclusion Algorithms: Introduction – Preliminaries – Lamport's algorithm – Ricart-Agrawala's Algorithm — Token-Based Algorithms – Suzuki-Kasami's Broadcast Algorithm; Deadlock Detection in Distributed Systems: Introduction – System Model – Preliminaries – Models of Deadlocks – Chandy-Misra-Haas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

Chandy–Misra–Haas algorithm for the OR model

- Uses diffusion computation
- Msg can be query(i,j,k) or reply(i,j,k)

- Deadlock is detected if a process gets reply to all query sent out

Any blocked process initiates deadlock detection



① query ⇒ Dependent set (BC)

B:- query msg (Q) from A
① engaging / first
② else

Handle * State (B) < Active — ignore
Blocked

① Q-engaging ⇒ forward to $DS_B^{(DE)}$
$num_B[A] = 2$

Wait$_i$[j] = true / false

② not-engaging
(i) reply to A iff B is blocked from last engaging query.
else ⇒ ignore

Algorithm

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.

Algorithm 10.2 Chandy–Misra–Haas algorithm for the OR model [6].

Key Points

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) and *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.

Basic idea

A blocked process initiates deadlock detection by sending query messages to all processes in its dependent set (i.e., processes from which it is waiting to receive a message). If an active process receives a *query* or *reply* message, it discards it. When a blocked process P_k receives a *query*(i, j, k) message, it takes the following actions:

1. If this is the first *query* message received by P_k for the deadlock detection initiated by P_i (called the *engaging query*), 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 P_k returns a *reply* message to it immediately provided P_k has been continuously blocked since it received the corresponding engaging *query*. Otherwise, it discards the *query*.

Process P_k 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 P_i. When a blocked process P_k 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.

Performance

Performance analysis

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

LIKE

COMMENT

SHARE

SUBSCRIBE