Deadlock

A deadlock is a state where a set of processes request resources that are held by other processes in the set.

DME.ppt

Deadlock Detection and Recovery

A deadlock detection algorithm must satisfy the following two conditions:
(i)
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.

(ii)
Safety (No false deadlocks):
The algorithm should not report deadlocks which do not
exist (called phantom or false deadlocks).

Resolution of 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.

Deadlock Conditions in WFG

Cycle is always a *necessary* condition for deadlock

- single-unit -- cycle is sufficient
- AND request model -- cycle is sufficient

- OR request model -- knot is sufficient
- AND-OR request model -- knot is sufficient
- P-out-of-Q request model -- knot is sufficient

Knapp's Classification

Distributed deadlock detection algorithms can be divided into four classes:

- path pushing
- edge pushing
- diffusion computation
- global state detection

Path Pushing

- In path-pushing algorithms, distributed deadlocks are detected by maintaining an explicit global WFG.
- The basic idea is to build a global WFG for each site of the distributed system.
- In this class of algorithms, at each site whenever deadlock computation is performed, it sends its local WFG to all the neighboring 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 WFG
has led to the term path-pushing algorithms.

COP 5611 SPRING 2003
These notes were used in a subsititute lecture, for Xiuwen Liu's class in Spring 2003.
https://www.cs.fsu.edu/~baker/cop5611.S03/index.html

Edge Chasing

- In an edge-chasing algorithm, 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.

Example

Chandy Mishra Haas's AND model

(#,#,#) ⇒ (initiator, sender, receiver)

Data Structure

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

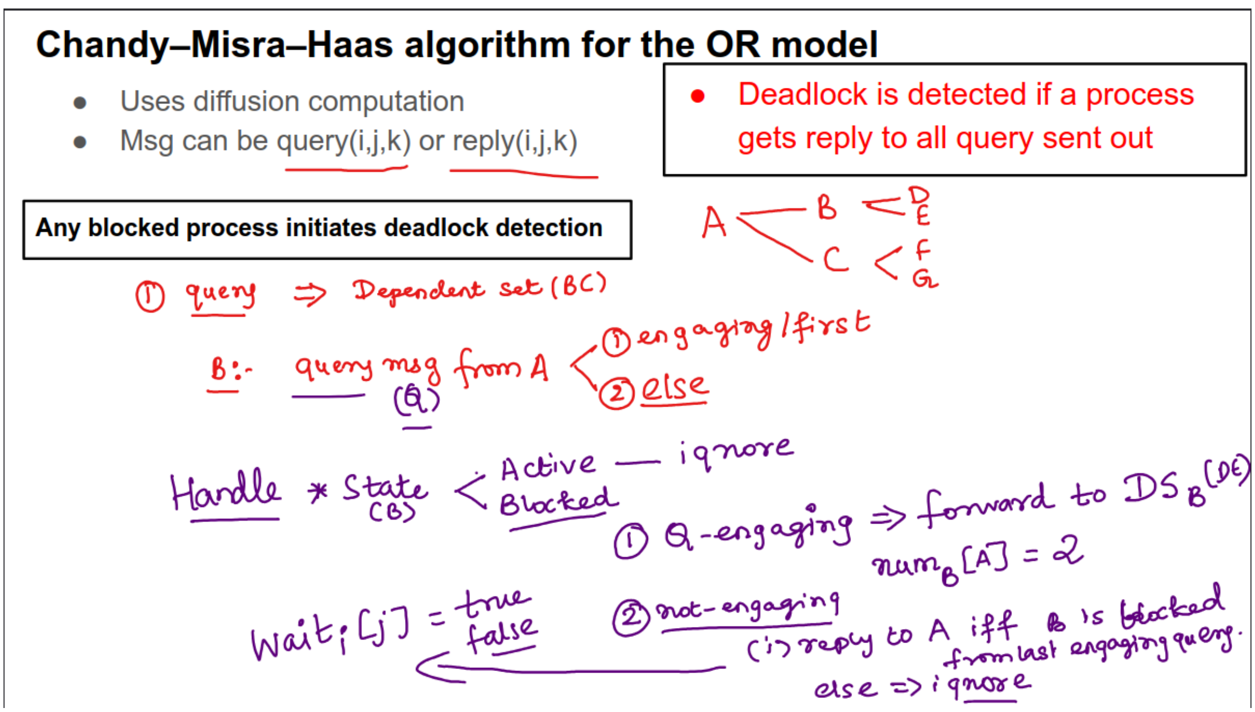
Diffusion Computation

Chandy Mishra Haas's for OR Model

- Message 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. Any blocked process sends a query message to all the nodes in its dependent set
2. The receiving process/node let’s say B, will receive either engaging query(first message) or else (already received message).
3. If the receiving process B is in
   - Active state - ignores the message
   - Blocked state
     - If the received message is engaging, it forwards the message to Dependent set of B. Additionally, B maintains an array called **num**b[A] = 2 (2 is the no of nodes, it has forwarded message)

…



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.

Algorithm

① **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;
$num_i(i) := |DS_i|$; $wait_i(i) := true$;

② **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.

③ **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].

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.