Distributed Deadlock Detection

- Assumptions:
  - System has only reusable resources
  - Only exclusive access to resources
  - Only one copy of each resource
  - States of a process: running or blocked
  - Running state: process has all the resources
  - Blocked state: waiting on one or more resource

Deadlocks

- Resource Deadlocks
  - A process needs multiple resources for an activity.
  - Deadlock occurs if each process in a set request resources held by another process in the same set, and it must receive all the requested resources to move further.

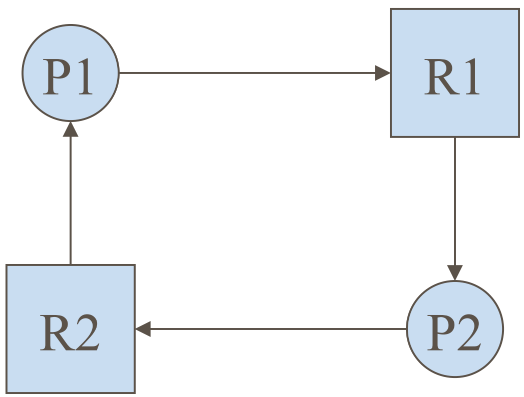
- Communication Deadlocks
  - Processes wait to communicate with other processes in a set.
  - Each process in the set is waiting on another process's message, and no process in the set initiates a message until it receives a message for which it is waiting.

Graph Models

- Nodes of a graph are processes. Edges of a graph the pending requests or assignment of resources.
- Wait-for Graphs (WFG): P1 -> P2 implies P1 is waiting for a resource from P2.
- Transaction-wait-for Graphs (TWF): WFG in databases.
- Deadlock: directed cycle in the graph.
- Cycle example:

Graph Models

- Wait-for Graphs (WFG): P1 -> P2 implies P1 is waiting for a resource from P2.



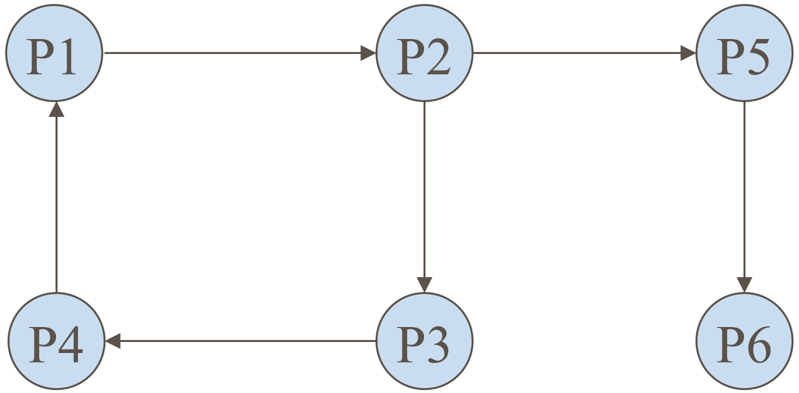
AND, OR Models

- AND Model
  - A process/transaction can simultaneously request for multiple resources.
  - Remains blocked until it is granted *all* of the requested resources.

- OR Model
  - A process/transaction can simultaneously request for multiple resources.
  - Remains blocked till *any one* of the requested resource is granted.

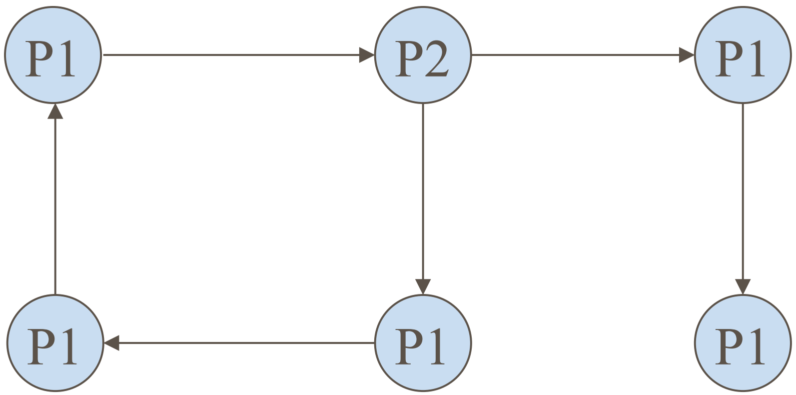
Sufficient Condition

Deadlock ??



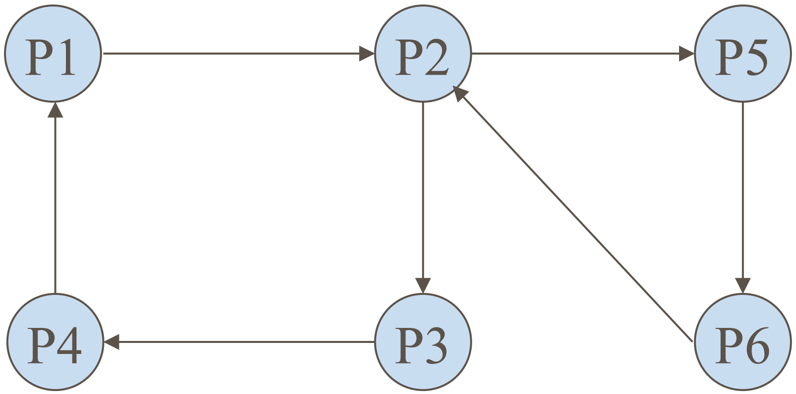
AND, OR Models

- AND Model
  - Presence of a cycle.



AND, OR Models

- OR Model
  - Presence of a knot.
  - Knot: Subset of a graph such that starting from any node in the subset, it is impossible to leave the knot by following the edges of the graph.



Deadlock Handling Strategies

- Deadlock Prevention: difficult
- Deadlock Avoidance: before allocation, check for possible deadlocks.
  - Difficult as it needs global state info in each site (that handles resources).
- Deadlock Detection: Find cycles. Focus of discussion.
- Deadlock detection algorithms must satisfy 2 conditions:
  - No undetected deadlocks.
  - No false deadlocks.

Distributed Deadlocks

- Centralized Control
  - A *control site* constructs wait-for graphs (WFGs) and checks for directed cycles.
  - WFG can be maintained continuously (or) built on-demand by requesting WFGs from individual sites.
- Distributed Control
  - WFG is spread over different sites.Any site can initiate the deadlock detection process.
- Hierarchical Control
  - Sites are arranged in a hierarchy.
  - A site checks for cycles only in descendents.

Centralized Algorithms

- Ho-Ramamoorthy 2-phase Algorithm
  - Each site maintains a status table of all processes initiated at that site: includes all resources locked & all resources being waited on.
  - Controller requests (periodically) the status table from each site.
  - Controller then constructs WFG from these tables, searches for cycle(s).
  - If no cycles, no deadlocks.
  - Otherwise, (cycle exists): Request for state tables again.
  - Construct WFG based *only* on common transactions in the 2 tables.
  - If the same cycle is detected again, system is in deadlock.
  - Later proved: cycles in 2 consecutive reports *need not* result in a deadlock. Hence, this algorithm detects false deadlocks.

Centralized Algorithms...

- Ho-Ramamoorthy 1-phase Algorithm
  - Each site maintains 2 status tables: *resource status* table and *process status* table.
  - Resource table: transactions that have locked or are waiting for resources.
  - Process table: resources locked by or waited on by transactions.
  - Controller periodically collects these tables from each site.
  - Constructs a WFG from transactions common to both the tables.
  - No cycle, no deadlocks.
  - A cycle means a deadlock.

Distributed Algorithms

- Path-pushing: resource dependency information disseminated through designated paths (in the graph).
- Edge-chasing: special messages or probes circulated along edges of WFG. Deadlock exists if the probe is received back by the initiator.
- Diffusion computation: queries on status sent to process in WFG.
- Global state detection: get a snapshot of the distributed system. Not discussed further in class.

Edge-Chasing Algorithm

- Chandy-Misra-Haas's Algorithm:
  - A probe(i, j, k) is used by a deadlock detection process Pi. This probe is sent by the home site of Pj to Pk.
  - This probe message is circulated via the edges of the graph. Probe returning to Pi implies deadlock detection.
  - Terms used:
    - Pj is *dependent* on Pk, if a sequence of Pj, Pi1,.., Pim, Pk exists.
    - Pj is *locally dependent* on Pk, if above condition + Pj,Pk on same site.
    - Each process maintains an array *dependenti: dependenti(j)* is true if Pi knows that Pj is dependent on it. (initially set to false for all i & j).

Chandy-Misra-Haas's Algorithm

Sending the probe:

if Pi is locally dependent on itself then deadlock.

else for all Pj and Pk such that

(a) Pi is locally dependent upon Pj, and

(b) Pj is waiting on Pk, and

(c) Pj and Pk are on different sites, send probe(i,j,k) to the home site of Pk.

Receiving the probe:

if (d) Pk is blocked, and

(e) *dependentk(i)* is false, and

(f) Pk has not replied to all requests of Pj,

then begin

dependentk(i) := true;

if k = i then Pi is deadlocked

else ...

Chandy-Misra-Haas's Algorithm

Receiving the probe:

…….

else for all Pm and Pn such that

(a') Pk is locally dependent upon Pm, and

(b') Pm is waiting on Pn, and

(c') Pm and Pn are on different sites, send probe(i,m,n) to the home site of Pn.

end.

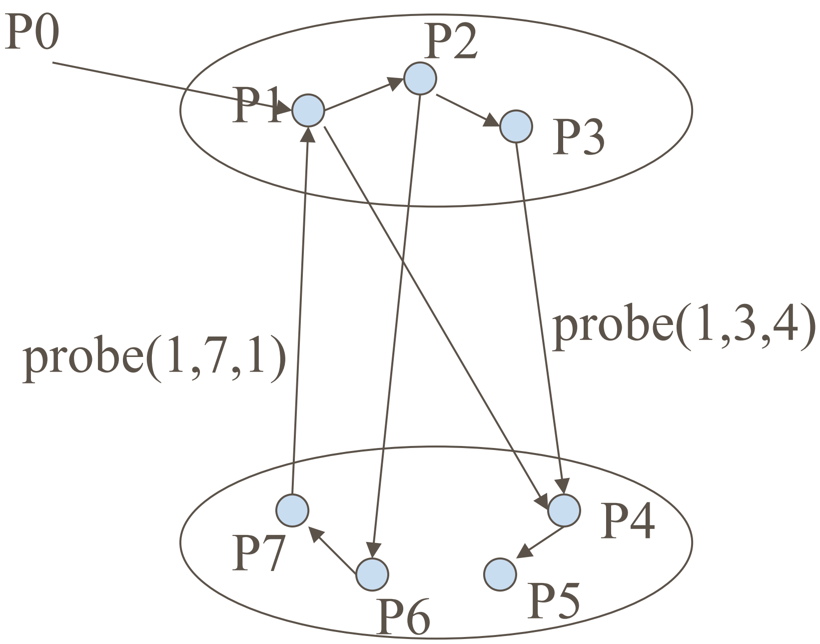
Performance:

For a deadlock that spans m processes over n sites, m(n-1)/2 messages are needed.

Size of the message 3 words.

Delay in deadlock detection O(n).

C-M-H Algorithm: Example



Diffusion-based Algorithm

Initiation by a blocked process Pi:

send query(i,i,j) to all processes Pj in the dependent set DSi of Pi;

num(i) := |DSi|; waiti(i) := true;

Blocked process Pk receiving query(i,j,k):

if this is *engaging* query for process Pk /* first query from Pi */

then send query(i,k,m) to all Pm in DSk;

numk(i) := |DSk|; waitk(i) := true;

else if waitk(i) then send a reply(i,k,j) to Pj.

Process Pk receiving reply(i,j,k)

if waitk(i) then

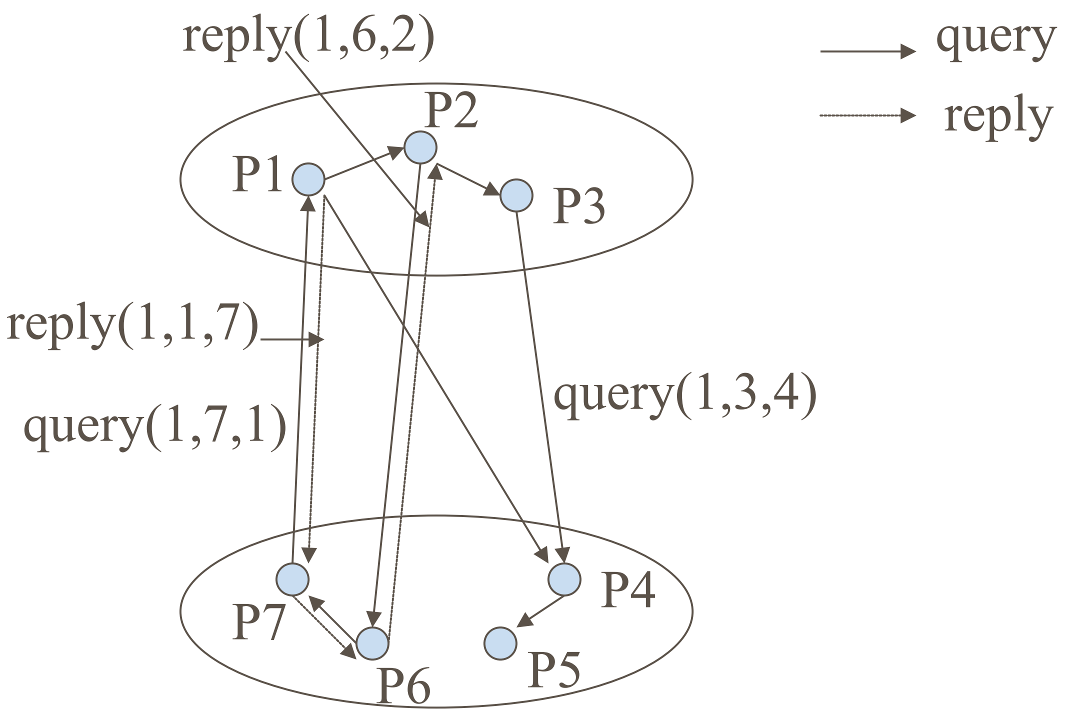
numk(i) := numk(i) - 1;

if numk(i) = 0 then

if i = k then declare a deadlock.

else send reply(i, k, m) to Pm, which sent the engaging query.

Diffusion Algorithm: Example



Engaging Query

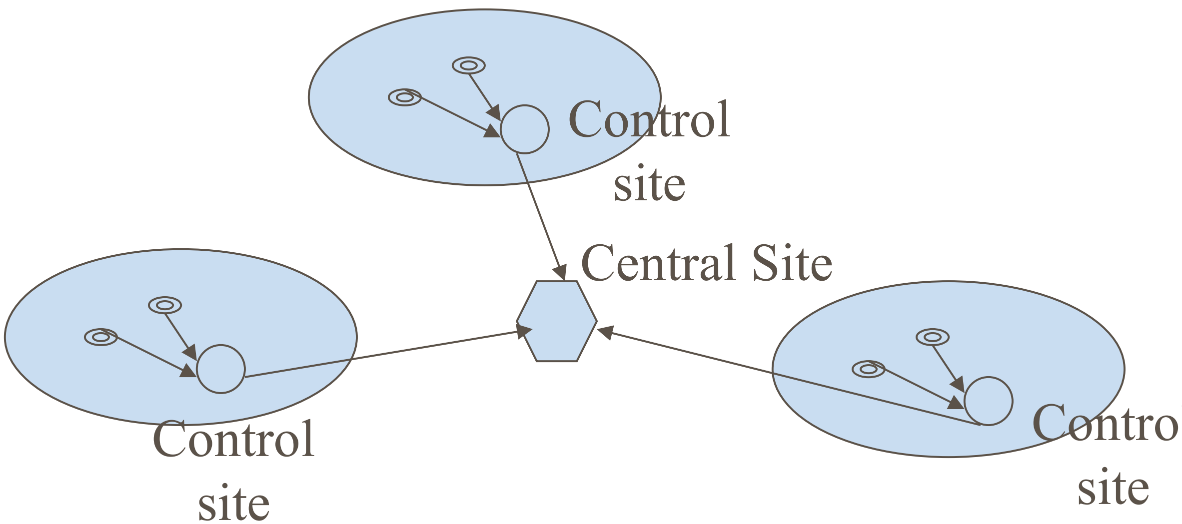
- How to distinguish an engaging query?
  - query(i,j,k) from the initiator contains a unique sequence number for the query apart from the tuple (i,j,k).
  - This sequence number is used to identify subsequent queries.
  - (e.g.,) when query(1,7,1) is received by P1 from P7, P1 checks the sequence number along with the tuple.
  - P1 understands that the query was initiated by itself and it is not an engaging query.
  - Hence, P1 sends a reply back to P7 instead of forwarding the query on all its outgoing links.

AND, OR Models

- AND Model
  - A process/transaction can simultaneously request for multiple resources.
  - Remains blocked until it is granted *all* of the requested resources.
  - Edge-chasing algorithm can be applied here.
- OR Model
  - A process/transaction can simultaneously request for multiple resources.
  - Remains blocked till *any one* of the requested resource is granted.
  - Diffusion based algorithm can be applied here.

Hierarchical Deadlock Detection

- Follows Ho-Ramamoorthy's 1-phase algorithm. More than 1 control site organized in hierarchical manner.
- Each control site applies 1-phase algorithm to detect (intracluster) deadlocks.
- Central site collects info from control sites, applies 1-phase algorithm to detect intracluster deadlocks.



Persistence & Resolution

- Deadlock persistence:
  - Average time a deadlock exists before it is resolved.
- Implication of persistence:
  - Resources unavailable for this period: affects utilization
  - Processes wait for this period unproductively: affects response time.
- Deadlock resolution:
  - Aborting at least one process/request involved in the deadlock.
  - Efficient resolution of deadlock requires knowledge of all processes and resources.
  - If every process detects a deadlock and tries to resolve it independently -> highly inefficient ! Several processes might be aborted.

Deadlock Resolution

- Priorities for processes/transactions can be useful for resolution.
  - Consider priorities introduced in Obermarck's algorithm.
  - Highest priority process initiates and detects deadlock (initiations by lower priority ones are suppressed).
  - When deadlock is detected, lowest priority process(es) can be aborted to resolve the deadlock.
- After identifying the processes/requests to be aborted,
  - All resources held by the victims must be released. State of released resources restored to previous states. Released resources granted to deadlocked processes.
  - All deadlock detection information concerning the victims must be removed at all the sites.