Distributed Coordination — Last Time

- Event ordering using the Happened-Before relationship
- Time stamping
- Centralized and decentralized semaphores

Distributed Coordination — Today

- Deadlock prevention and detection

Deadlock

Deadlock problem is the same in a distributed system as in a stand alone system: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can perform.

Solutions: Simple adaptations of previous solutions

1. Ordered resource policy:
   - Number the resources $r_1 \ldots r_n$
   - If a process holds resource r_j it can only request r_k where $j < k$.
   - Major drawback:
   - Major advantage:
2. Banker's Algorithm
   - One process is designated as the banker and all requests go to the banker.
   - Same algorithm as before to determine if satisfying the request results in a safe allocation.
   - Major drawback:
   - Major advantage:

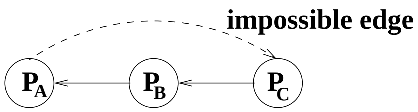
Deadlock Prevention using Wound and Wait

Intuition: Give older processes priority.

1. Processes get a unique timestamp identifying when their first resource request was made within a transaction:

$\Longrightarrow$ How are unique timestamps assigned?

2. When p_i requests a resource held by p_j
   - If starting timestamp $p_j < p_i$, p_i (the younger) waits.
   - Else OS kills p_i, i.e., releases any resources it holds, rolls it back, and restarts it (it must reacquire resources using transaction mechanisms).
   - No circular wait is possible:



timestamp(P_A) < timestamp(P_B) < timestamp(P_C)

   - Edge is p_i waiting on p_j for some resource.
   - When p_j frees resource, it gives it to the waiting process with the lowest timestamp.
   - How do processes decide which is younger?
   - Can starvation occur?

Wound and Wait Example

Initially:

- P_A has timestamp 3
- P_B has timestamp 5
- P_C has timestamp 10, holds R_1
- P_D has timestamp 15, holds R_2

1. P_C requests R_2

2. P_A requests R_1

3. P_B requests R_1

4.

5.

6.

7.

8.

9.

10.

11.

Deadlock Detection

Previous solutions for single processor:

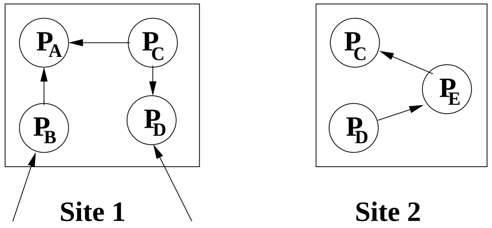
Wait-for Graph: processes are nodes and edge $P_A \rightarrow P_B$ exists if P_A is waiting for a resource process P_B has.

Theorem: A cycle in a wait-for graph implies deadlock.

Deadlock Detection

Wait-for Graph in a Distributed System:

- Each site maintains a *local* wait-for graph containing all processes in the system that are waiting for a local resource.



- A local cycle clearly implies deadlock, but no local cycle does not imply absence of deadlock!

$\Longrightarrow$ must compute a global graph by unioning the local graphs.

Real graph is unknowable since messages are not instantaneous.

Constructed graph approximates real graph.

$\Longrightarrow$ If a deadlock occurs, the constructed graph should be the same as the real graph, but there might be a time delay between when the deadlock occurs and when it is reflected in the graph.

Distributed Deadlock Detection

- Each site maintains a local waits-for graph that represents local processes that are waiting for resources local to the site.
- When a process P_i wants a resource from a remote machine S_j and the resource is not currently available:
  1. Add an edge from P_i to P_{ex} to the waits-for graph on the requester's machine.
  2. Add an edge from P_{ex} to P_j to the waits-for graph on P_j's machine where P_j currently holds the resource.

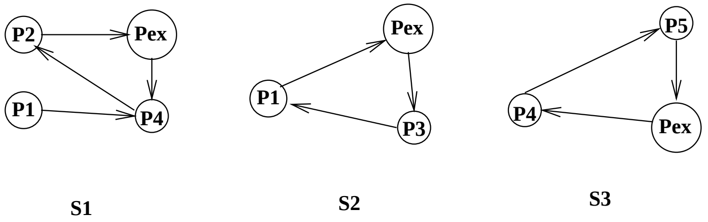
Distributed Deadlock Detection

- To check for deadlock:
  1. If any machine has a local waits-for graph with a cycle that does not include P_{ex}, there is deadlock.
  2. If no machine has a waits-for graph with a cycle, there is no deadlock.
  3. If a machine has a waits-for graph with a cycle that includes P_{ex}, deadlock might exist. We need to combine graphs to see if there really is deadlock.
     (a) Suppose there is an edge from P_i to P_{ex} because P_i is waiting for a resource held at remote site S_j and this link is part of a cycle.
     (b) P_i's machine sends a message to S_j identifying the cycle.
     (c) P_i removes the edges that it passes to S_j.
     (d) S_j adds these links to its waits-for graph.
     (e) If S_j now has a cycle not involving P_{ex}, there is deadlock. If it has no cycles, there is no deadlock (involving the cycle we identified, anyway). If it has a cycle involving P_{ex}, the algorithm repeats. Eventually, all cycle information gets propagated to all machines and we know whether deadlock is present or not.

Example: Distributed Deadlock Detection

Assume:

- P2 is waiting for P3.
- P1 is waiting for P4.
- P5 is waiting for P2.



Summary

- Distributed deadlock detection is even harder!
- Global knowledge on the state of the system is difficult to acquire, but sometimes we can do without it.
- Conservative algorithms may under utilize resources.