Handling deadlocks
- definition, wait-for graphs
- fundamental causes of deadlocks
- resource allocation graphs and conditions for deadlock existence
- approaches to handling deadlocks
  - deadlock prevention
  - deadlock avoidance
  - deadlock detection/resolution
    - centralized algorithms
    - distributed algorithms
    - hierarchical algorithms

What is a deadlock?
- deadlock – a set of processes is blocked waiting for requirements that cannot be satisfied
- illustrated by a wait-for graph (WFG)
  - nodes – processes in the system
  - directed edges – wait-for blocking relation
  - a cycle in WFG indicates a deadlock
- starvation – a process is indefinitely prevented from making progress
- deadlock implies starvation, is the converse true?

Fundamental causes of deadlocks
- Mutual exclusion — if one process holds a resource, other processes requesting that resource must wait until the process releases it (only one can use it at a time)
- Hold and wait — processes are allowed to hold one (or more) resource and be waiting to acquire additional resources that are being held by other processes
- No preemption — resources are released voluntarily; neither another process nor the OS can force a process to release a resource
- Circular wait — there must exist a set of waiting processes such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, … Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held P0

Resource allocation graph
- The deadlock conditions can be modeled using a directed graph called a resource-allocation graph (RAG)
  - 2 kinds of nodes:
    - Boxes — represent resources
      - Instances of the resource are represented as dots within the box
    - Circles — represent processes
  - 2 kinds of (directed) edges:
    - Request edge — from thread to resource — indicates the thread has requested the resource, and is waiting to acquire it
    - Assignment edge — from resource instance to thread — indicates the thread is holding the resource instance
  - When a request is made, a request edge is added
    - When request is fulfilled, the request edge is transformed into an assignment edge
    - When process releases the resource, the assignment edge is deleted

RAG with single resource instances

* a cycle in RAG with single resource instances is necessary and sufficient for deadlock

RAG with multiple resource instances

* cycle does not indicate deadlock
* knot — strongly connected subgraph (no sinks) with no outgoing edges
* a knot in RAG is necessary and sufficient for deadlock
Deadlock prevention and avoidance

- Deadlock prevention — eliminate one of the 4 deadlock conditions
  - examples
    - acquire all resources before proceeding (no wait while hold)
    - allow preemption (eliminate 3rd condition)
    - prioritize processes and assign resources in the order of priorities (no circular wait)
  - may be inefficient
- Deadlock avoidance — consider each resource request, and only fulfill those that will not lead to deadlock
  - Stay in a safe state — a state with no deadlock where resource requests can be granted in some order such that all processes will complete
  - may be inefficient
    - Must know resource requirements of all processes in advance
    - Resource request set is known and fixed, resources are known and fixed
    - Complex analysis for every request

Deadlock avoidance

- Consider each resource request, and only fulfill those that will not lead to deadlock

Deadlock detection

- Deadlock detection and resolution — detect, then break the deadlock

- Issues
  - Maintenance of WFG
  - Search of WFG for deadlocks

- Requirements
  - Progress — no undetected deadlocks
  - Safety — no false (phantom) deadlocks

- Resolution
  - Roll back one or more processes to break dependencies in WFG and resolve deadlocks

Distributed deadlock detection algorithms

- Centralized algorithm - coordinator maintains global WFG and searches it for cycles
  - Simple algorithm
  - Ho and Ramamoorthy’s one- and two-phase algorithms

- Distributed algorithms - Global WFG, with responsibility for detection spread over many sites
  - Obermack’s path-pushing
  - Chandy, Misra, and Haas’s edge-chasing
  - Diffusion

- Hierarchical algorithms - hierarchical organization, site detects deadlocks involving only its descendants
  - Menasce and Muntz’s algorithm
  - Ho and Ramamoorthy’s algorithm

Problem of False Deadlock

Now assume process p1 releases resource p3 is waiting on
Slightly thereafter, p2 requests resource p3 is holding
However, first message reaches coordinator after second message
The global WFG now has a false cycle, which leads to a report of false deadlock

Ho and Ramamoorthy two phase centralized deadlock detection

- Every site maintains a status table, containing status of all local processes
  - Resources held, resources waiting on

- Periodically, coordinator requests all status tables, builds a WFG, and searches it for cycles
  - No cycles — no deadlock

- If cycle is found, coordinator again requests all status tables, again builds a WFG, but this time uses only those edges common to both sets of status tables

- Rationale was that by using information from two consecutive reports, coordinator would get a consistent view of the state
  - However, it was later shown that a deadlock in this WFG does not imply a deadlock exists (see 1 phase alg.)
  - So, the HR-two-phase algorithm may reduce the possibility of reporting false deadlocks, but doesn’t eliminate it

Simple centralized deadlock detection

- A central coordinator maintains a global wait-for graph (WFG) for the system

- All sites request and release resources (even local resources) by sending request and release messages to the coordinator

- When coordinator receives a request/release, it
  - Updates the global WFG
  - Checks for deadlocks

- Problems
  - Large communication overhead, coordinator is a performance bottleneck and single point of failure, etc.
  - May report phantom deadlock
Ho and Ramamoorthy one-phase centralized deadlock detection

- Every site maintains two tables
  - all local processes and resources locked
  - resources locked at this site (by both local and non-local processes)
- one site periodically requests both tables (once) constructs WFG; WFG only includes the info on non local processes if this info is matched by the process site and resource site
  - if cycle — deadlock
  - if not — no deadlock
- correctly detects deadlocks by eliminating inconsistency of reporting due to message propagation delay
- more space overhead than 2-phase H&R