CSCE 455/855
Distributed Operating Systems

Deadlocks

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE855

1

Deadlocks

- Definition of deadlock
  - » each process in set is waiting for a resource to be released by another process in set
    - ❖ the set is some subset of all processes
    - ❖ deadlock only involves the processes in the set

2

Deadlocks

- Necessary conditions for deadlock
  - » mutual exclusion
    - process has exclusive use of resource allocated to it
  - » hold and wait
    - process can hold one resource while waiting for another
  - » no preemption
    - resources are released only by explicit action by controlling process
    - requests cannot be withdrawn (i.e. request results in eventual allocation *or* deadlock)
  - » circular wait
    - given a set of processes $\{p_0, p_1, ..., p_n\}$, p_0 is waiting for a resource held by p_1, is waiting for a resource held by p_2, [...], and p_n is waiting for a resource held by p_1.

3

Deadlock Handling Strategies

- No strategy
- Avoidance
  - » allocate resources so deadlock can't occur
- Detection
  - » let deadlock occur, find deadlocked processes, recover
- Prevention
  - » make it structurally impossible to have a deadlock

4

No strategy
The "ostrich algorithm"

- Most popular approach
- Assumes deadlock rarely occurs
  - » Becomes more probable with more processes
- Catastrophic consequences when it does occur
  - » may need to re-boot all or some machines in system

5

Deadlock Avoidance

- General idea: refuse states that may lead to deadlock
  - » method for keeping track of states
  - » need to know resources required by a process
  - » requires some advance knowledge of resource usage
- Banker's algorithm
  - » must know maximum number allocated to p_i
  - » keep track of # of resources available
  - » for each request, make sure max need will not exceed total available
  - » under utilizes resources (algorithm assumes max claim will be requested)
- Never used
  - » advance knowledge not available and CPU-intensive

6

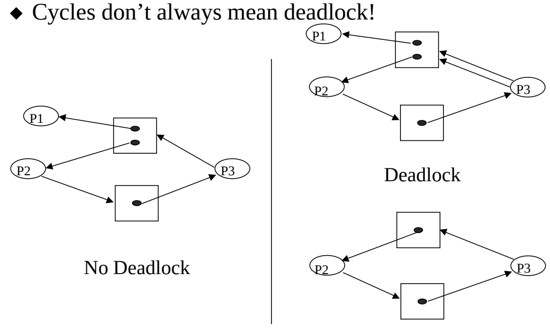
Centralized Deadlock Detection

- General method: construct a resource graph and analyze it
  - » analyze through resource reductions
  - » if cycle exists after analysis, deadlock has occurred
    - ❖ processes in cycle are deadlocked
- Local graphs
  - » P1 requests R1
    - ❖ R1's site places request in local graph
  - » if cycle exists in local graph, perform reductions to detect deadlock
- Need to calculate union of all graphs
  - » deadlock cycle may transcend machine boundaries

7

Graph Reduction

- Cycles don't always mean deadlock!



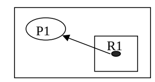
8

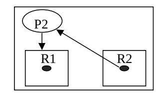
Centralized Deadlock Detection (cont.)

- All hosts communicate resource state to coordinator
  - » construct resource graph on coordinator
  - » coordinator must be reliable, fast
- When to construct the graph
  - » report <u>every</u> request, acquisition, release
  - » periodically send set of operations
  - » whenever cycle detection is called for

False Deadlock

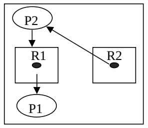
- problem: messages may not arrive in a timely fashion
  - » in particular, may arrive out-of-order
  - » given below, assume
    - P2 releases R2 (message A)
    - P1 requests instance of R2 (message B)



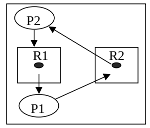


False Deadlock (cont.)

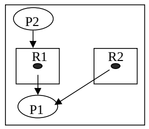
Initial coordinator representation:



After receiving message B:



After receiving message A:

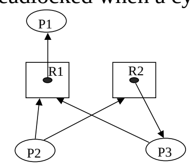


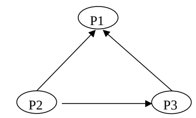
- problem: will detect deadlock after message B
  - » even though no deadlock exists

11

Waits-For Graphs (WFGs)

- Based on Resource Allocation Graph (RAG)
- An edge from P_i to P_j
  - » means P_i is waiting for P_j to release a resource
  - » replaces two edges
    - $P_i \rightarrow R$
    - $R \rightarrow P_j$
- deadlocked when a cycle is found





12

Distributed Deadlock Detection

- Chandry-Misra-Haas algorithm
  - » use waits-for graph
  - » send probe messages to processes you are waiting on
  - » if message gets back, deadlock has occurred
- Invoke algorithm when process has to wait
  - » send message to process holding resources
    - process just blocked
    - process sending the message
    - receiving process
  - » recipient forwards message to all processes it is waiting on
  - » if message gets back to original sender, deadlock has occurred
    - note that first field of message will always be the initiator

13

Distributed Deadlock Detection
An Example

- p0 gets blocked, resource held by p1
  - » initial message from p0 to p1: (0, 0, 1)
- p1 waiting on p2
  - » p1 sends message (0, 1, 2) to p2
- p2 waiting on p3: (0, 2, 3)
- p3 waiting on p4 and p5: (0, 3, 4) and (0, 4, 5)
- eventually message gets to p8, which is waiting on p0
  - » p0 gets message, sees itself as the initiator: (0, 8, 0)
    - a cycle exists
    - p0 knows there is deadlock

14

Distributed Deadlock Prevention

- Prevention
  - » make deadlocks *structurally* impossible
  - » make sure 4 necessary conditions don't hold
    - process can only hold one resource at a time
    - process releases all resources before requesting one
    - resource ordering

15

Distributed Deadlock Prevention
Timestamp-ordering approaches

- Arbitrarily order requests - prevents cycles
  - » two requirements:
    - global time (Lamport's will do)
    - atomic transactions
  - » Transaction assigned timestamp when it starts
    - wait for resource only if timestamp is lower (older) than the transaction waited for
      - can do the vice-versa...
      - makes more sense to kill off younger processes
    - otherwise abort

16

Timestamp-Based Prevention

- wait-die scheme
  - » Pi requests resource held by Pj
  - » if $TS_i < TS_j$, Pi can wait (Pi is older)
  - » otherwise Pi is rolled back
  - » example: $TS_1 = 5$, $TS_2 = 10$
    - P_1 requests resource held by P_2
    - P_2 requests resource held by P_1

17

Timestamp-Based Prevention (cont.)

- wound-wait scheme
  - » same as wait-die...
  - » but allow preemption of a resource
    - old process preempts young one
  - » suppose a process wants a resource held by a younger one
    - older one preempts younger
    - younger transaction is aborted
    - immediately re-starts
    - assigned new (younger) timestamp
    - waits for older
  - » contrast with wait-die

18