CS 492: Operating Systems

Deadlocks (2)

Instructor: Iraklis Tsekourakis

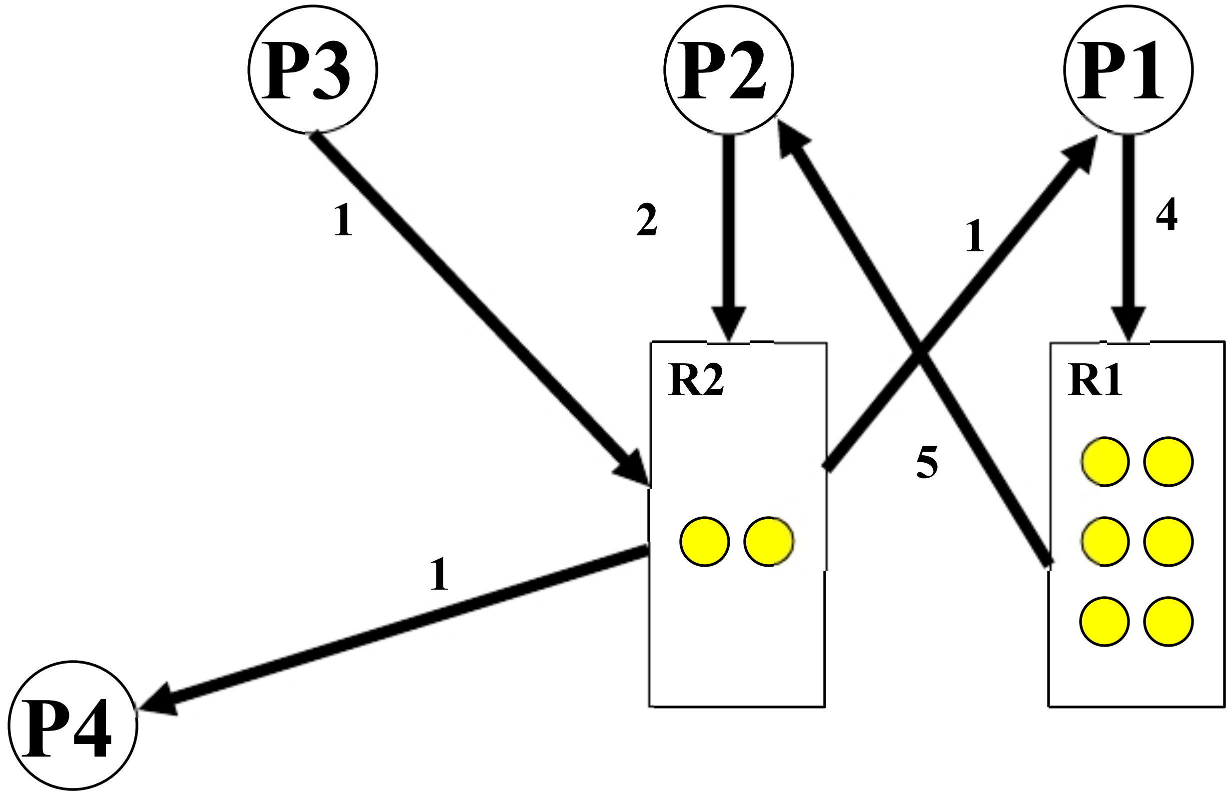
Email: itsekour@stevens.edu

Recap

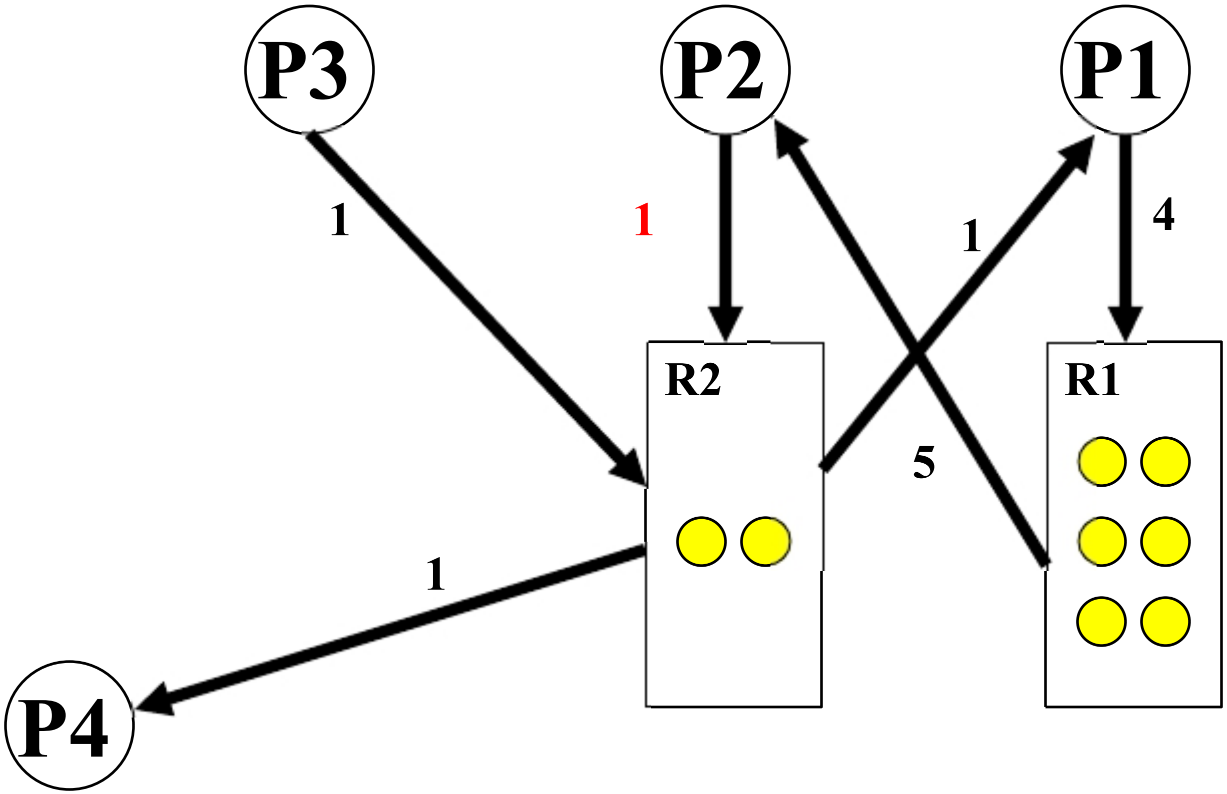
- Deadlock
  - Concepts
  - How to deal with deadlocks
    - Ignoring them: ostrich algorithm
    - Detecting & recovering from deadlock
    - Preventing deadlock
    - Avoiding deadlock

Conditions for Resource Deadlocks

- Mutual exclusion
  - Only one thread at a time can use a resource
- Hold and wait
  - Thread holding at least one resource is waiting to acquire additional resources held by other threads
- No preemption
  - Resources are released only voluntarily by the thread holding the resource, after thread is finished with it
- Circular wait
  - There exists a set $\{T_1, \ldots, T_n\}$ of waiting threads
    - T_1 is waiting for a resource that is held by T_2
    - T_2 is waiting for a resource that is held by T_3
    - …
    - T_n is waiting for a resource that is held by T_1



Is there a deadlock?



Is there a deadlock?

Question

Suppose that there is a resource deadlock in a system. Give an example to show that the set of processes deadlocked can include processes that are not in the circular chain in the corresponding resource allocation graph.

Solution

Consider three processes, A, B and C and three resources R, I and S. Suppose A is waiting for I that is held by B, B is waiting for S held by A, and C is waiting for R held by A. All three processes, A, B and C are deadlocked. However, only A and B belong to the circular chain.

Goals for Today

- Deadlock
  - How to deal with deadlocks
    - Ignoring them: ostrich algorithm
    - Detecting & recovering from deadlock
    - Preventing deadlock
    - Avoiding deadlock

Recap: Graph Analysis VS Deadlock Detection

- Facts:
  - No cycle in resource allocation graph $\Rightarrow$ NO deadlock.
  - Cycle in resource allocation graph $\Rightarrow$
    - if only one instance per resource type, then deadlock.
    - if several instances per resource type, possibility of deadlock.
- Detect deadlocks (for single instance per resource type) = detect the cycles in the resource allocation graph.

Detection with One Resource of Each Type

- We need a formal algorithm for detecting deadlocks
  - i.e., to detect the cycles in the resource graph
- A simple one to detect cycles:
  - Use resource graph
  - Perform DFS (depth first search) starting at **each** node, looking for cycles (recurrence of starting node)
  - If it comes to a node it has encountered in this run, then there exists a cycle.

Pseudocode for Deadlock Detection

```
func detectCycle()
  for node in graph:
    visited = bool[N]
    set all visited to false
    detectCycle(n, n, visited)

func detectCycle(n, origin, visited)
  for neighbour in graph[n]
    if neighbour == origin
       cycle detected
    if not visited[neighbour]
       visited[neighbour] = true
       detectCycle(neighbour, visited)
       visited[neighbour] = false
```

Detection: Time Complexity

- N nodes, E edges
- Time complexity for DFS is O (N + E)
- Need to run DFS N times (once for each node)
  - O (N (N + E))
- Complexity
  - $O(N(N+E)) = O(N^2 + NE)$
  - N << E, so $N^2 << N E$, total complexity is O (NE)

Recovery from Deadlock (1)

- Recovery through preemption
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback
  - checkpoint a process periodically
  - use this saved state
  - restart the process if it is found deadlocked

Recovery from Deadlock (2)

- Recovery through killing processes
  - crudest but simplest way to break a deadlock
  - kill one of the processes in the deadlock cycle
  - the other processes get its resources
  - choose process that can be rerun from the beginning

What's Next?

Question: How to detect deadlocks with multiple resources of each type?

Detection with Multiple Resources of Each Type

- Basic idea: Allocate resources to a process that can be executed to completion
- Data structure

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

Detection with Multiple Resources of Each Type

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

Detection with Multiple Resources of Each Type: Algorithm

- Deadlock detection is based on comparing vectors
- Algorithm
  - Look for an unmarked process, P_i, for which R[i] (i.e. the i-th row of R) <= A (i.e., P_i ‘s request can be granted by available resources)
  - If such a process is found, update A = A + C[i] (i.e. add the i-th row of C to A), mark the process and go back to step 1 (i.e., P_i releases its resources)
  - If no such process exists, the algorithm terminates

Output of Algorithm

- If the algorithm outputs a sequence of processes, then no deadlock!
- Otherwise, deadlock exists

Example of Detection Algorithm

- Five processes P_0 through P_4;
- Three resource types: A (7 instances), *B* (2 instances), and *C* (6 instances). Existence matrix: E = (7, 2, 6)
- Snapshot at time T_0:

	Allocation C	*Request R*	*Available A*	
	ABC	*ABC*	*ABC*	
			0 0 0	P_0 gets resource,
P_0	010	000		
P_1	200	202		
P_2	303	000		
P_3	211	100		
P_4	002	002		

Example of Detection Algorithm

- Five processes P_0 through P_4;
- Three resource types: A (7 instances), *B* (2 instances), and *C* (6 instances). E = (7, 2, 6)

	Allocation C	*Request R*	*Available A*
	ABC	*ABC*	*ABC*
P_0	010	000	0 1 0
P_1	200	202	
P_2	303	000	
P_3	211	100	
P_4	002	002	

P_0 finishes and releases resources

P_2 gets resource,

Example of Detection Algorithm

- Five processes P_0 through P_4;
- Three resource types: A (7 instances), *B* (2 instances), and *C* (6 instances). E = (7, 2, 6)

	Allocation C	*Request R*	*Available A*	
	ABC	*ABC*	*ABC*	
P_0	010	000	0 1 0	
P_1	200	202		
P_2	303	000	3 1 3	P_2 finishes and releases resources P_3 gets resource
P_3	211	100		
P_4	002	002		

Example of Detection Algorithm

- Five processes P_0 through P_4;
- Three resource types: A (7 instances), *B* (2 instances), and *C* (6 instances). E = (7, 2, 6)

	Allocation C	*Request R*	*Available A*	
	ABC	*ABC*	*ABC*	
P_0	010	000		
P_1	200	202		P1 gets resource
P_2	303	000	3 1 3	
P_3	211	100	5 2 4	P3 finishes and releases resources
P_4	002	002		

Example of Detection Algorithm

- Five processes P_0 through P_4;
- Three resource types: A (7 instances), *B* (2 instances), and *C* (6 instances). E = (7, 2, 6)

	Allocation C	*Request R*	*Available A*	
	ABC	*ABC*	*ABC*	
P_0	010	000		
P_1	200	202	7 2 4	P_1 finishes and releases resources
P_2	303	000		
P_3	211	100	5 2 4	
P_4	002	002		P_4 gets resource

Example of Detection Algorithm

- Five processes P_0 through P_4;
- Three resource types: A (7 instances), *B* (2 instances), and *C* (6 instances). E = (7, 2, 6)

	Allocation C	*Request R*	*Available A*	
	ABC	*ABC*	*ABC*	
P_0	010	000		✓
P_1	200	202	7 2 4	✓
P_2	303	000		✓
P_3	211	100		✓
P_4	002	002	7 2 6	P_4 finishes and releases resources

Sequence <P_0, P_2, P_3, P_1, P_4> will result in *all* processes finished
No deadlock!

Example (Cont.)

- P_2 requests an additional instance of type C.

	Allocation C	*Request R*	*Available A*
	ABC	*ABC*	*ABC*
P_0	010	000	0 0 0
P_1	200	202	
P_2	303	00 1	
P_3	211	100	
P_4	002	002	

Example (Cont.)

- P_2 requests an additional instance of type C.

	Allocation C	*Request R*	*Available A*
	ABC	*ABC*	*ABC*
P_0	010	000	0 1 0
P_1	200	202	
P_2	303	00 1	
P_3	211	100	
P_4	002	002	

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes requests.
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Detection: How often to run?

- Possibilities:
  - After every allocation of resource
    - Too expensive
  - Every few minutes
  - When CPU is idle – indication of blocked processes waiting for resources

Detection: How often to run? (Cont.)

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - One for each disjoint cycle