

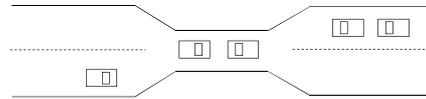
Deadlocks

Fan Wu

Department of Computer Science and Engineering
Shanghai Jiao Tong University
Spring 2020



Bridge Crossing Example



- Traffic only in one direction
- Each section of a bridge can be viewed as a resource
- A deadlock occurs when two cars get on the bridge from different directions at the same time



The Problem of Deadlock

- Example
 - System has 2 disk drives
 - P_1 and P_2 each hold one disk drive and each needs another one
- Example
 - semaphores S and Q, initialized to 1

P_0	P_1
① wait (S);	② wait (Q);
③ wait (Q);	④ wait (S);
- **Deadlock:** A set of blocked processes each holding some resources and waiting to acquire the resources held by another process in the set



Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously.
 - **Mutual exclusion:** only one process at a time can use a resource
 - **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
 - **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
 - **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .



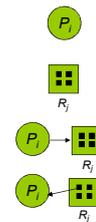
System Model

- Processes P_1, P_2, \dots, P_n
- Resource types R_1, R_2, \dots, R_m
e.g., CPU, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - request
 - use
 - release



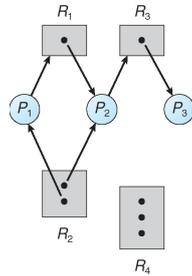
Resource-Allocation Graph

- Deadlocks can be identified with system resource-allocation graph.
 - A set of vertices V and a set of edges E .
 - V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system
 - E has two types:
 - **request edge** – directed edge $P_i \rightarrow R_j$
 - **assignment edge** – directed edge $R_j \rightarrow P_i$



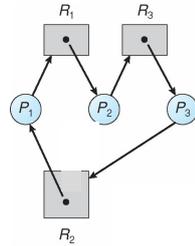
Example of a Resource Allocation Graph

- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- Resource instances:
 - $W_1=W_3=1$
 - $W_2=2$
 - $W_4=3$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$



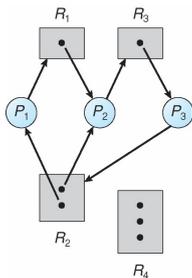
Resource Allocation Graph With A Deadlock

- A circle
 - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

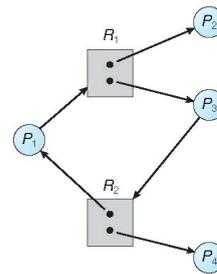


Resource Allocation Graph With A Deadlock

- Two circles
 - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
 - $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no circle \Rightarrow no deadlock
- If graph contains a circle \Rightarrow
 - if only one instance per resource type, then deadlock
 - if several instances per resource type, possibility of deadlock
- Question:
 - Can you find a way to determine whether there is a deadlock, given a resource allocation graph with several instances per resource type?

Methods for Handling Deadlocks

- Ensure that the system will **never** enter a deadlock state
 - Deadlock prevention
 - Deadlock avoidance
- **Allow** the system to enter a deadlock state and then recover
 - Deadlock detection
 - Deadlock recovery

Deadlock Prevention

Restrain the ways request can be made

- **Mutual Exclusion** – not required for sharable resources; must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
 - Require process to request and be allocated all its resources before it begins execution
 - Or allow process to request resources only when the process has none (has released all its resources)
 - Low resource utilization; starvation possible

Deadlock Prevention (Cont.)

- **No Preemption**
 - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are preempted
 - Preempted resources are added to the list of resources for which the process is waiting
 - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

Requires that the system has some additional *a priori* information available

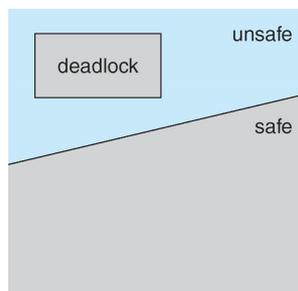
- Requires that each process declare the *maximum number* of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- **Resource-allocation state** is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a **safe state**
- System is in **safe state** if there exists a **safe sequence** $\langle P_1, P_2, \dots, P_n \rangle$ of ALL the processes in the systems such that for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j , with $j < i$
- That is:
 - If P_i 's resource needs are not immediately available, then P_i can wait until all P_j have finished
 - When all P_j are finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
 - When P_i terminates, P_{i+1} can obtain its needed resources, and so on
- Otherwise, system is in **unsafe state**

Safe, Unsafe, Deadlock State

- If a system is in safe state
⇒ no deadlocks
- If a system is in unsafe state
⇒ possibility of deadlock
- Avoidance
⇒ ensure that a system will never enter an unsafe state.



Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P_0	10	5	5	3
P_1	4	2	2	
P_2	9	2	7	

Safe sequence: ?

Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P ₀	10	5	5	1
P ₁	4	4	0	
P ₂	9	2	7	

Safe sequence: P₁

Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P ₀	10	5	5	5
P ₁	4	--	--	
P ₂	9	2	7	

Safe sequence: P₁

Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P ₀	10	10	0	0
P ₁	4	--	--	
P ₂	9	2	7	

Safe sequence: P₁ → P₀

Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P ₀	10	--	--	10
P ₁	4	--	--	
P ₂	9	2	7	

Safe sequence: P₁ → P₀

Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P ₀	10	--	--	3
P ₁	4	--	--	
P ₂	9	9	0	

Safe sequence: P₁ → P₀ → P₂

Safe & Unsafe States

	Maximum Needs	Holds	Needs	Available
P ₀	10	--	--	12
P ₁	4	--	--	
P ₂	9	--	--	

Safe sequence: P₁ → P₀ → P₂

Safe & Unsafe States

	Maximum Needs	Holds	Needs
P ₀	10	5	5
P ₁	4	2	2
P ₂	9	3	6

Available
2

Safe sequence: ?

Safe & Unsafe States

	Maximum Needs	Holds	Needs
P ₀	10	5	5
P ₁	4	--	--
P ₂	9	3	6

Available
4

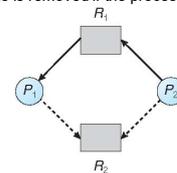
Safe sequence: P₁ → ?

Avoidance Algorithms

- Avoidance algorithms ensure that the system will never deadlock.
 - Whenever a process requests a resource, the request is granted only if the allocation leaves the system in a safe state.
- Two avoidance algorithms
 - Single instance of a resource type
 - ▶ Use a resource-allocation graph
 - Multiple instances of a resource type
 - ▶ Use the banker's algorithm

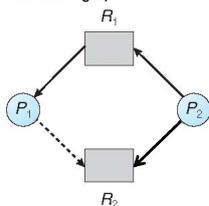
Resource-Allocation-Graph Algorithm

- **Claim edge** P_i → R_j indicates that process P_i may request resource R_j; represented by a **directed dashed line**
- Resources must be claimed *a priori* in the system
- Claim edge converts to request edge when a process requests a resource
- Request edge converts to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge (the edge is removed if the process finishes)

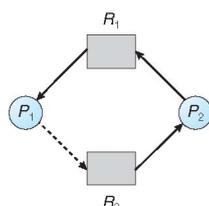


Resource-Allocation Graph Algorithm

- Suppose that process P₁ requests a resource R₁
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a circle in the resource allocation graph



Can we grant P₂'s request for R₂?



Circle! Therefore, P₂'s request cannot be granted, and P₂ needs to wait.

Banker's Algorithm

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- **Available:** Vector of length m . If $available[j] = k$, there are k instances of resource type R_j available
- **Max:** $n \times m$ matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation:** $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need:** $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

Safety Algorithm

1. Let $Work$ and $Finish$ be vectors of length m and n , respectively. Initialize:
 $Work = Available$
 $Finish[j] = false$, for $i = 0, 1, \dots, n-1$

2. Find an i such that both:
 (a) $Finish[i] = false$
 (b) $Need_i \leq Work$
 If no such i exists, go to step 4

3. $Work = Work + Allocation_i$
 $Finish[i] = true$
 go to step 2

4. If $Finish[i] == true$ for all i , then the system is in a safe state

Resource-Request Algorithm for Process P_i

$Request_i$ = request vector for process P_i . If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- If **safe** \Rightarrow the resources are allocated to P_i
- If **unsafe** $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm

- 5 processes P_0 through P_4 ;

3 resource types:

A (10 instances), B (5 instances), and C (7 instances)

Snapshot at time T_0 :

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P_0	7 5 3	0 1 0	7 4 3	3 3 2
P_1	3 2 2	2 0 0	1 2 2	
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

- Is the system in safe state?

Applying Safety Algorithm

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P_0	7 5 3	0 1 0	7 4 3	5 3 2
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

Safe sequence: P_1

Applying Safety Algorithm

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P_0	7 5 3	0 1 0	7 4 3	7 4 3
P_2	9 0 2	3 0 2	6 0 0	
P_4	4 3 3	0 0 2	4 3 1	

Safe sequence: $P_1 \rightarrow P_3$

Applying Safety Algorithm

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
				7 5 3
P_2	9 0 2	3 0 2	6 0 0	
P_4	4 3 3	0 0 2	4 3 1	

Safe sequence: $P_1 \rightarrow P_3 \rightarrow P_0$

Applying Safety Algorithm

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
				10 5 5
P_4	4 3 3	0 0 2	4 3 1	

Safe sequence: $P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2$

Applying Safety Algorithm

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
				10 5 7

Safe sequence: $P_1 \rightarrow P_3 \rightarrow P_0 \rightarrow P_2 \rightarrow P_4$

Example: P_1 Request (1,0,2)

- Check that Request \leq Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true)

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P_0	7 5 3	0 1 0	7 4 3	2 3 0
P_1	3 2 2	3 0 2	0 2 0	
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence $\langle P_1, P_3, P_0, P_2, P_4 \rangle$ satisfies safety requirement

Example: P_0 Request (0,2,0)

- Check that Request \leq Available (that is, $(0,2,0) \leq (2,3,0) \Rightarrow$ true)

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P_0	7 5 3	0 3 0	7 2 3	2 1 0
P_1	3 2 2	3 0 2	0 2 0	
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

- Does there a safe sequence exist?
 - No

Pop Quiz

- 5 processes P_0 through P_4 ;
- 3 resource types:
A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0 :

	Max	Allocation	Need	Available
	A B C	A B C	A B C	A B C
P_0	7 5 3	0 1 0	7 4 3	3 3 2
P_1	3 2 2	2 0 0	1 2 2	
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

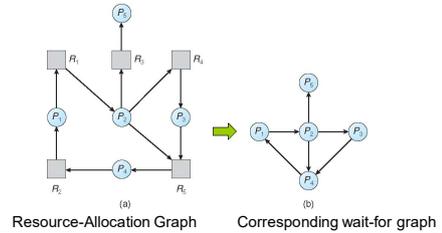
- Can P_4 's request (2, 1, 0) be granted?
- Can P_4 's request (2, 1, 2) be granted?

Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

- Maintain wait-for graph
 - Nodes are processes
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock



Several Instances of a Resource Type

- Available:** A vector of length m indicates the number of available resources of each type.
- Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- Request:** An $n \times m$ matrix indicates the current request of each process. If $Request[i][j] = k$, then process P_i is requesting k more instances of resource type R_j .

Detection Algorithm

- Let *Work* and *Finish* be vectors of length m and n , and initialize:
 - $Work = Available$
 - For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then $Finish[i] = false$; otherwise, $Finish[i] = true$
- Find an index i such that both:
 - $Finish[i] == false$
 - $Request_i \leq Work$
 If no such i exists, go to step 4
- $Work = Work + Allocation_i$, $Finish[i] = true$, go to step 2
- If $Finish[i] == false$, for some $i, 1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked

Example of Detection Algorithm

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances)
- Snapshot at time T_0 :

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = true$ for all i

Example (Cont.)

- P_2 requests an additional instance of type C

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- 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-Algorithm Usage

- 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

Recovery from Deadlock

- Process Termination
 - abort one or more processes to break the circular wait
- Resource Preemption
 - preempt some resources from one or more of the deadlocked processes

Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
 - Priority of the process
 - How long process has computed, and how much longer to completion
 - Resources the process has used
 - Resources process needs to complete
 - How many processes will need to be terminated
 - Is process interactive or batch?

Resource Preemption

- Selecting a victim – minimize cost
- Rollback – return to some safe state, restart process from that state
- Starvation – same process may always be picked as victim, include number of rollback in cost factor

Homework

- Reading
 - Chapter 7
- Exercise
 - See course website