Universidade Estadual de Campinas
Instituto de Computação
MC504 Sistemas Operacionais

T23b Common Concurrency Problems

Referência principal
Ch.32 of *Operating Systems: Three Easy Pieces* by Remzi and Andrea Arpaci-Dusseau (pages.cs.wisc.edu/~remzi/OSTEP/)

Discutido em classe em 22 de outubro de 2018

Arthur João Catto, PhD

2º semestre de 2018

Deadlock avoidance: the Safe State concept

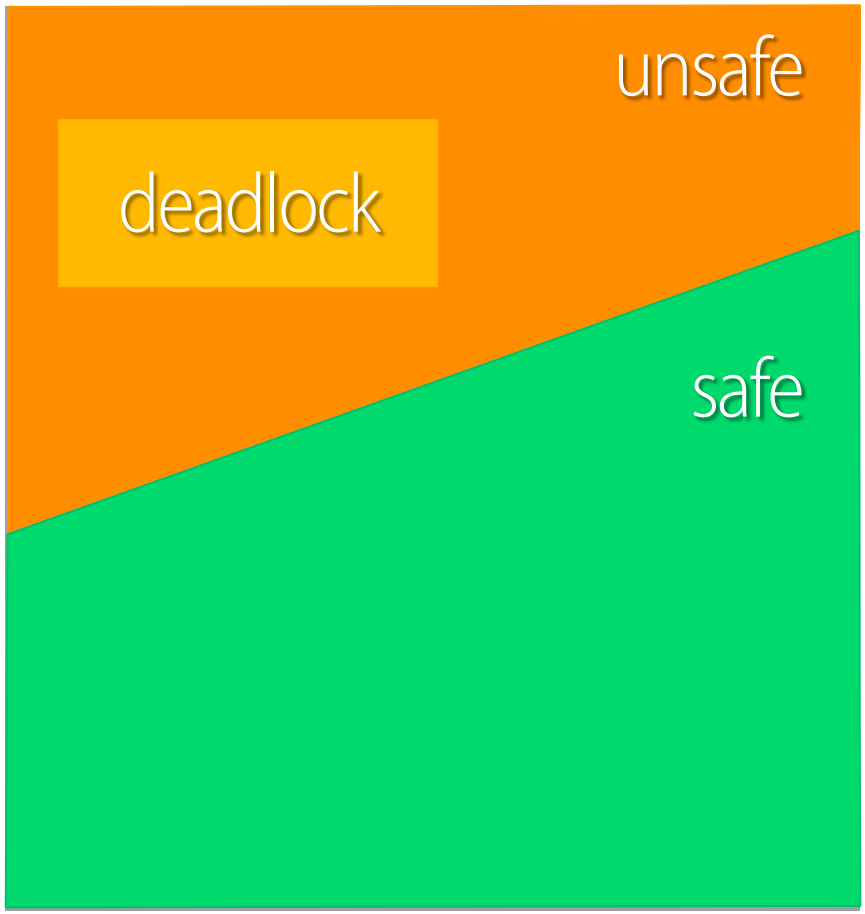
- When a process requests an available resource, the resource manager must decide if immediate allocation leaves the system in a safe state.
- A system with n processes, $P_1, \ldots P_n$, is said to be in a ***safe state*** if there exists a sequence of execution $\langle P_{i_1}, P_{i_2}, \ldots P_{i_n} \rangle$ of all the processes in the system such that
  for each P_{i_j}, the resources that P_{i_j} may still need can be satisfied by currently available resources plus resources held by some P_{i_k}, with $k < j$.

Safe State

- In other words…
  - If P_{i_j} needs a resource that is not immediately available, then P_{i_j} can wait until all P_{i_k}, $k < j$, have finished.
  - When P_{i_k}, $k < j$, has finished, P_{i_j} can obtain the resources that it needs, execute, return its allocated resources and terminate.
  - When P_{i_j} terminates, $P_{i_{j+1}}$ can obtain the resources that it needs, and so on.

Basic Facts about Safe and Unsafe States

- System is in a safe state
  - No deadlocks.
- System is in an unsafe state
  - Possibility of deadlock.
- To avoid deadlocks
  - Ensure that the system will never enter an unsafe state.



System states

Deadlock Dynamics

- Safe state
  - For any possible sequence of future resource requests, it is possible to eventually grant all requests.
  - May require waiting even when resources are available!
- Unsafe state
  - Some sequence of resource requests can result in deadlock.
- Doomed state
  - All possible computations lead to deadlock.

Food for thought...

- What are the doomed states for Dining Philosophers?
- What are the unsafe states?
- What are the safe states?

Deadlock Avoidance algorithms

Single instance of a resource type

Use a resource-allocation graph scheme.

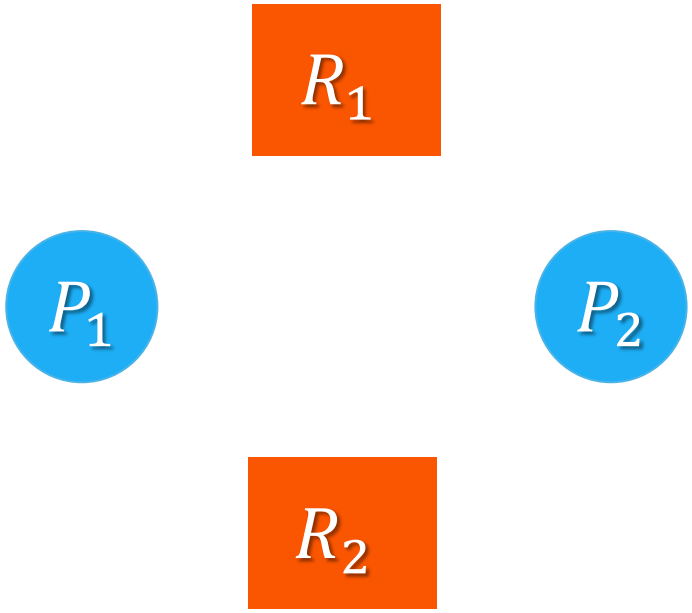
Multiple instances of a resource type

Use the banker's algorithm.

Deadlock avoidance with single instance of each resource type

Resource-Allocation Graph Scheme

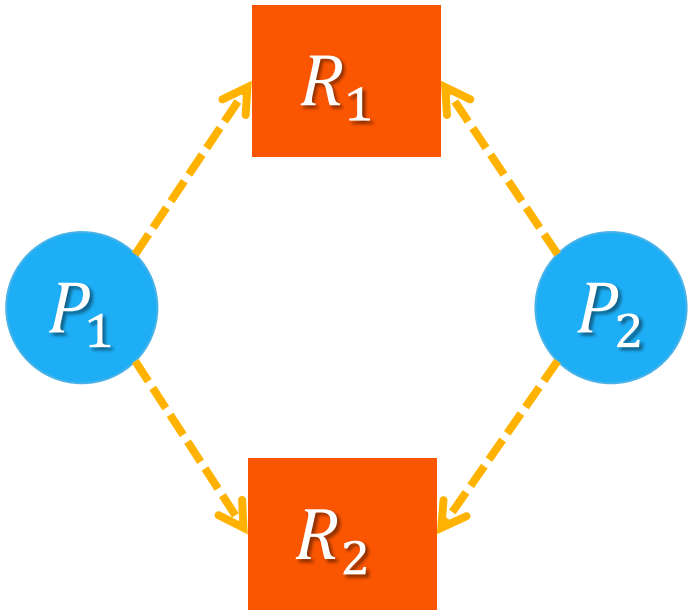
- Create a resource allocation graph.



Deadlock avoidance with single instance of each resource type

Resource-Allocation Graph Scheme

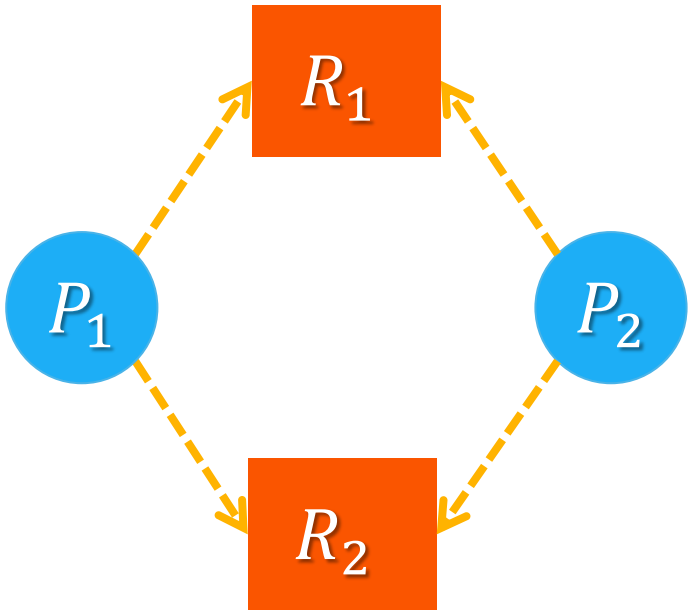
- Introduce a **claim edge** $P_i \dashrightarrow R_j$ to indicate that process P_i may request resource R_j sometime.
  - A claim edge is denoted by a dashed line.
  - Resources must be claimed *a priori*.



Deadlock avoidance with single instance of each resource type

Resource-Allocation Graph Scheme

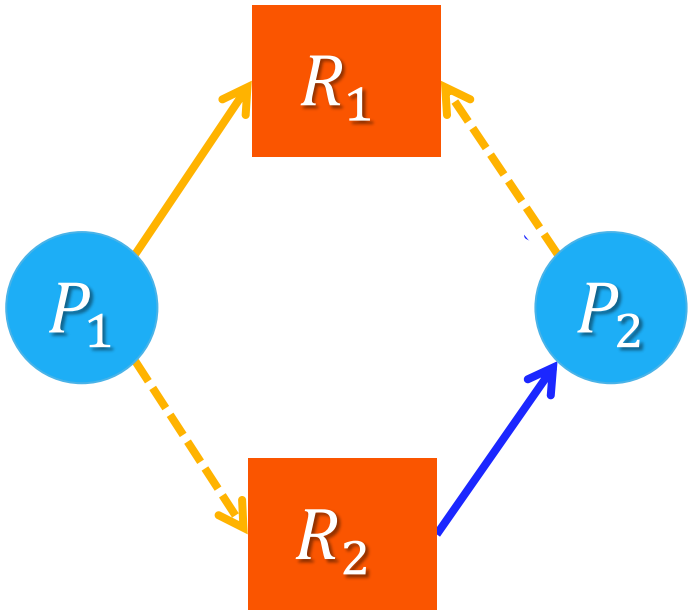
- A claim edge is converted to a **request edge** when a process requests a resource.
- A request edge is converted to an **assignment edge** when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.



Deadlock avoidance with single instance of each resource type

Resource-Allocation Graph Scheme

- Suppose that process P_i requests a resource R_j.
- The request can only be granted if converting the request edge to an assignment edge does not create a cycle in the graph, which would lead the system to an unsafe state.



Deadlock avoidance with multiple instances of each resource type

Predict the Future: Banker's Algorithm

- Grant request if and only if result is a safe state.
- Sum of maximum resource needs of current threads can be greater than the total resources…
  - … provided there is some way for all the threads to finish without getting into deadlock.
- Example
  - Proceed if and only if
    - **total available resources − number allocated ≥ max remaining** that might be needed by this thread in order to finish.
    - It can be guaranteed that this thread can finish.

Deadlock avoidance with multiple instances of each resource type

Banker's Algorithm

- Can deal with multiple instances of resources.
- Each process must claim maximum use a priori.
- 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.

Banker's Algorithm

Data Structures

- Let $P_i, 0 \leq i < n,$ be a process and $R_k, 0 \leq k < m,$ a resource type.
- $Available_k$ (vector of length m)
  - Number of available instances of resource type R_k.
- $Max_{i,k}$ ($n \times m$ matrix)
  - Maximum number of instances of R_k that P_i may require.
- $Allocation_{i,k}$ ($n \times m$ matrix)
  - Number of instances of R_k currently allocated to P_i.
- $Need_{i,k}$($n \times m$ matrix)
  - Number of instances of R_k that P_i may still require.
  - $Need_{i,k} = Max_{i,k} - Allocation_{i,k}$.

Safety Algorithm

- Let $Work$ and $Finish$ be vectors of length m and n, respectively.

1. Initially

$$Work = Available$$
$$Finish_i = false, 0 \leq i < n$$

2. Find an i such that $Finish_i = false$ and $Need_i \leq Work$. If no such i exists, go to step 4.
3. Let

$$Work \leftarrow Work + Allocation_i$$
$$Finish_i \leftarrow true$$

   and go back to step 2.

4. If $Finish_i = true$ for all $0 \leq i < n$, the system is in a safe state.

Resource-Request Algorithm for Process P_i

1. Let $\boldsymbol{Request}$ be the request vector for process $\boldsymbol{P_i}$.
   - $\boldsymbol{Request_k = x}$, means that $\boldsymbol{P_i}$ wants $\boldsymbol{x}$ instances more of resource $\boldsymbol{R_k}$.
2. If $\boldsymbol{Request > Need_i}$, raise error condition, since $\boldsymbol{P_i}$ has exceeded its maximum claim.
3. If $\boldsymbol{Request > Available}$, $\boldsymbol{P_i}$ must wait, since resources are not immediately available.
4. …

Resource-Request Algorithm for Process P_i

4. Pretend to allocate the requested resources to P_i:
   - $Available \leftarrow Available - Request$
   - $Allocation_i \leftarrow Allocation_i + Request$
   - $Need_i \leftarrow Need_i - Request$
5. Run the safety algorithm.
   - If state is safe, allocate the requested resources to P_i.
   - If state is unsafe, P_i must wait and the original state of resource allocation is restored.

Example of Banker's Algorithm

- 5 processes P_0 through P_4
- 3 resource types,
  A (10 instances), B (5), C (7).

Total		
A	B	C
10	5	7

System 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									
P_1	3	2	2									
P_2	9	0	2									
P_3	2	2	2									
P_4	4	3	3									

Example of Banker's Algorithm

- 5 processes P_0 through P_4
- 3 resource types,
  A (10 instances), B (5), C (7).

Total		
A	B	C
10	5	7

System 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						
P_1	3	2	2	2	0	0						
P_2	9	0	2	3	0	2						
P_3	2	2	2	2	1	1						
P_4	4	3	3	0	0	2						

Example of Banker's Algorithm

- 5 processes P_0 through P_4
- 3 resource types,
  A (10 instances), B (5), C (7).

Total		
A	B	C
10	5	7

System 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				3	3	2
P_1	3	2	2	2	0	0						
P_2	9	0	2	3	0	2						
P_3	2	2	2	2	1	1						
P_4	4	3	3	0	0	2						

Example of Banker's Algorithm

- 5 processes P_0 through P_4
- 3 resource types,
  A (10 instances), B (5), C (7).

Total		
A	B	C
10	5	7

System 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						
P_2	9	0	2	3	0	2						
P_3	2	2	2	2	1	1						
P_4	4	3	3	0	0	2						

Example of Banker's Algorithm

- 5 processes P_0 through P_4
- 3 resource types,
  A (10 instances), B (5), C (7).

Total		
A	B	C
10	5	7

System 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						
P_3	2	2	2	2	1	1						
P_4	4	3	3	0	0	2						

Example of Banker's Algorithm

- 5 processes P_0 through P_4
- 3 resource types,
  A (10 instances), B (5), C (7).

Total		
A	B	C
10	5	7

System 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			

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - To find the answer we must run the safety algorithm.

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
3	3	2

Work		
A	B	C

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - First, create $Work$ and $Finish$ vectors.

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
3	3	2

Work		
A	B	C
3	3	2

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - Can any process terminate?
    - Yes. P_1 can. Retrieve the resources that it holds.

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
P_1	3	2	2	2	0	0	1	2	2	T
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	

Available		
A	B	C
3	3	2

Work		
A	B	C
5	3	2

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - Can another process terminate?
    - Yes. P_3 can. Retrieve the resources that it holds.

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

Available		
A	B	C
3	3	2

Work		
A	B	C
7	4	3

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - Now, P_0 can terminate. Let's retrieve its resources.

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

Available		
A	B	C
3	3	2

Work		
A	B	C
7	5	3

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - Now, P_2 can terminate.

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

Available		
A	B	C
3	3	2

Work		
A	B	C
10	5	5

Example of Banker's Algorithm

- Is the system state at time T_0 safe?
  - And finally P_4 can terminate.
  - YES!

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

Available		
A	B	C
3	3	2

Work		
A	B	C
10	5	7

Example of Banker's Algorithm

- Can a (1, 0, 2) request by P_1 be granted at T_0?
  - Pretend to grant the request

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
3	3	2

Work		
A	B	C

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
2	3	0

Work		
A	B	C

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
2	3	0

Work		
A	B	C
2	3	0

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
P_1	3	2	2	3	0	2	0	2	0	T
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	

Available		
A	B	C
2	3	0

Work		
A	B	C
5	3	2

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
2	3	0

Work		
A	B	C
7	4	3

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
2	3	0

Work		
A	B	C
7	5	3

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
2	3	0

Work		
A	B	C
10	5	5

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
2	3	0

Work		
A	B	C
10	5	7

Example of Banker's Algorithm

- At T_0 can a (1, 0, 2) request by P_1 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

→ Yes!

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

Available		
A	B	C
2	3	0

Work		
A	B	C
10	5	7

Example of Banker's Algorithm

- At T_0 can a (3, 3, 0) request by P_4 be granted?
  - Pretend to grant the request

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
3	3	2

Work		
A	B	C
3	3	2

Example of Banker's Algorithm

- At T_0 can a (3, 3, 0) request by P_4 be granted?
  - Pretend to grant the request

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	3	3	2	1	0	1	

Available		
A	B	C
0	0	2

Work		
A	B	C
0	0	2

Example of Banker's Algorithm

- At T_0 can a (3, 3, 0) request by P_4 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

→ No!

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	3	3	2	1	0	1	

Available		
A	B	C
0	0	2

Work		
A	B	C
0	0	2

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
3	3	2

Work		
A	B	C

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	1	0	7	4	3	
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	

Available		
A	B	C
3	3	2

Work		
A	B	C

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	3	0	7	2	3	
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	

Available		
A	B	C
3	1	2

Work		
A	B	C

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	3	0	7	2	3	
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	

Available		
A	B	C
3	1	2

Work		
A	B	C
3	1	2

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

	Max			Allocation			Need			Finish
	A	B	C	A	B	C	A	B	C	
P_0	7	5	3	0	3	0	7	2	3	
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	T
P_4	4	3	3	0	0	2	4	3	1	

Available		
A	B	C
3	1	2

Work		
A	B	C
5	2	3

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
3	1	2

Work		
A	B	C
7	2	3

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
3	1	2

Work		
A	B	C
7	5	3

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

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

Available		
A	B	C
3	1	2

Work		
A	B	C
10	5	5

Example of Banker's Algorithm

- At T_0 can a (0, 2, 0) request by P_0 be granted?
  - Pretend to grant the request
  - Run the safety algorithm

→ Yes!

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

Available		
A	B	C
3	1	2

Work		
A	B	C
10	5	7

Deadlock Detection

- Allow system to enter deadlock state

- Run a detection algorithm
  - Resource allocation graphs are used to detect the existence or possibility of deadlock.

- Apply a recovery scheme

Detect and Repair

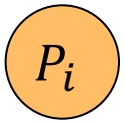
- Algorithm
  - Scan wait-for graph
  - Detect cycles
  - Fix cycles
- Proceed without the resource
  - Requires robust exception handling code
- Roll back and retry
  - Transaction: all operations are provisional until have all required resources to complete operation

Resource-Allocation Graph (RAG)

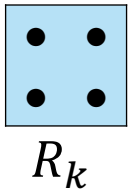
- A RAG is a directed graph given by a set of vertices V and a set of edges E such that
  - V is partitioned into two subsets:
    - $P = \{P_1, P_2, \ldots P_n\}$, the set consisting of all the processes in the system
    - $R = \{R_1, R_2, \ldots R_m\}$, the set consisting of all resource types in the system.
  - A directed edge $P_i \rightarrow R_k$ is called a *request edge*.
  - A directed edge $R_k \rightarrow P_i$ is called an *assignment edge*.

Resource-Allocation Graphs

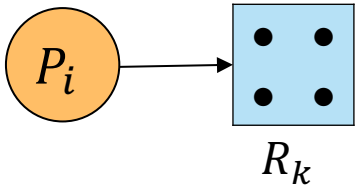
- Process P_i



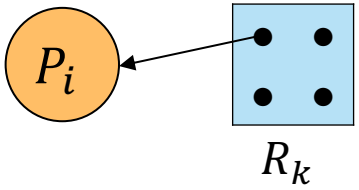
- Resource Type R_k with 4 instances

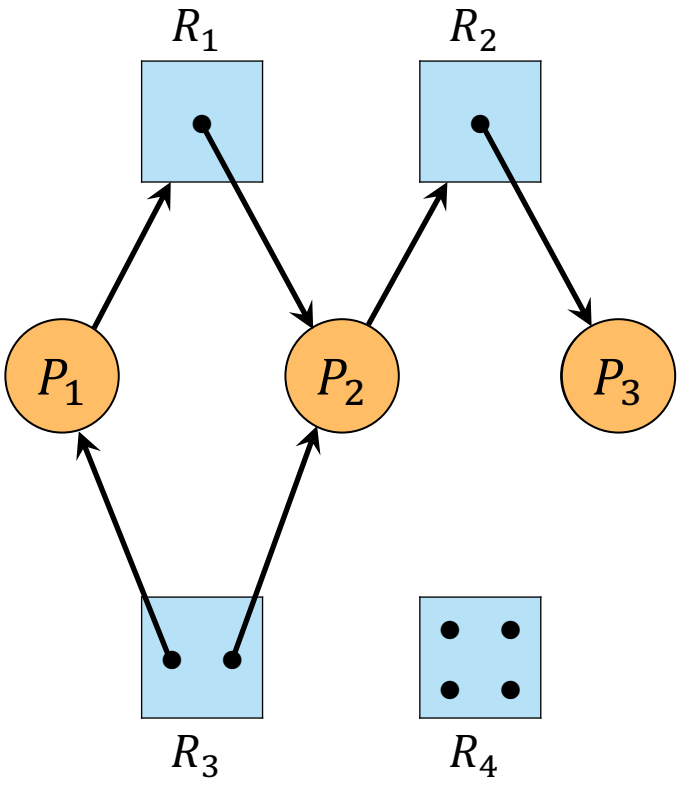


- P_i requests an instance of R_k

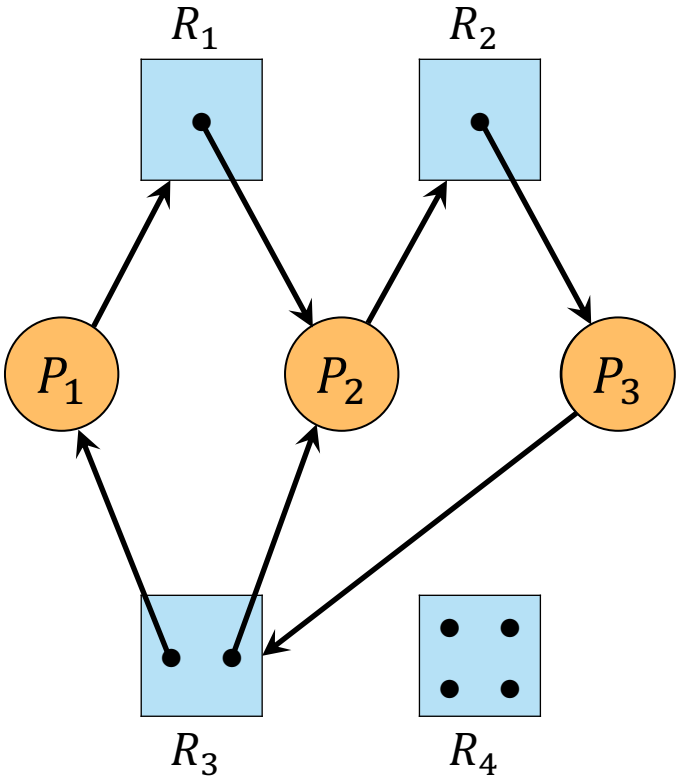


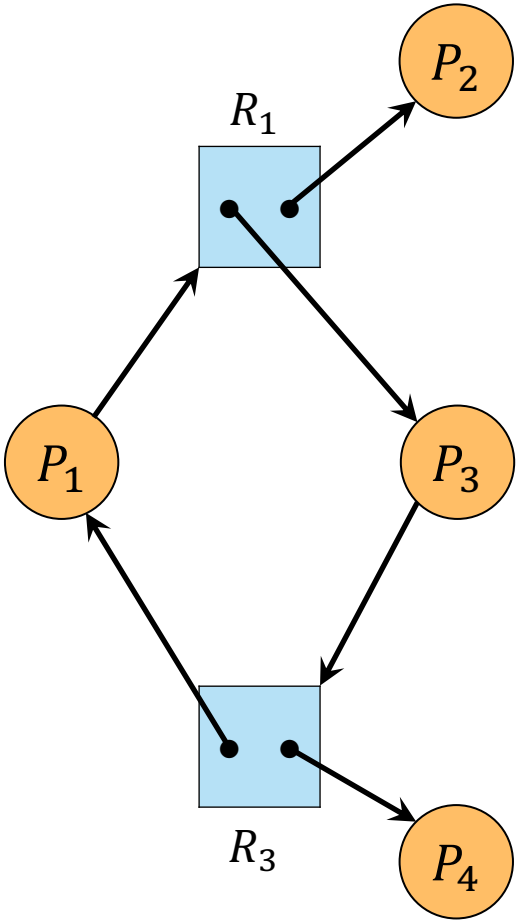
- P_i holds an instance of R_k





Do these Resource Allocation Graphs show deadlocks?





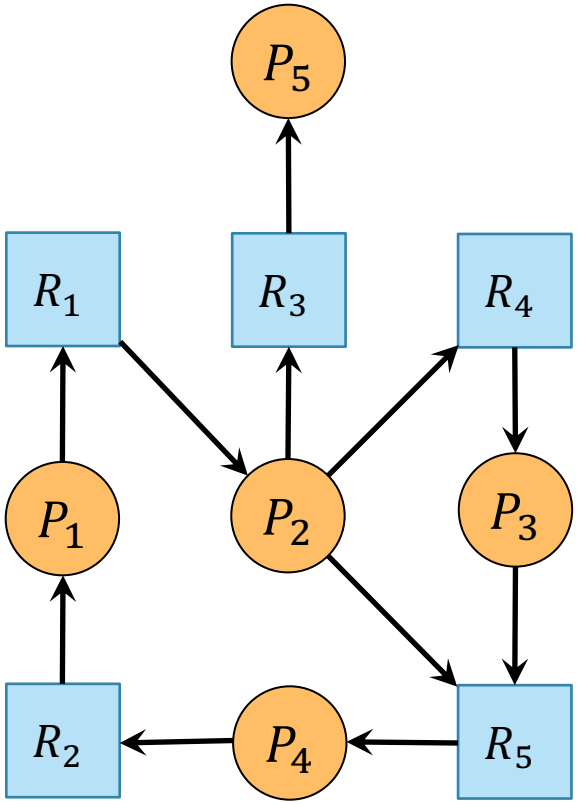
Basic facts about a Resource Allocation Graph

- If the graph contains no cycles, there is no deadlock.

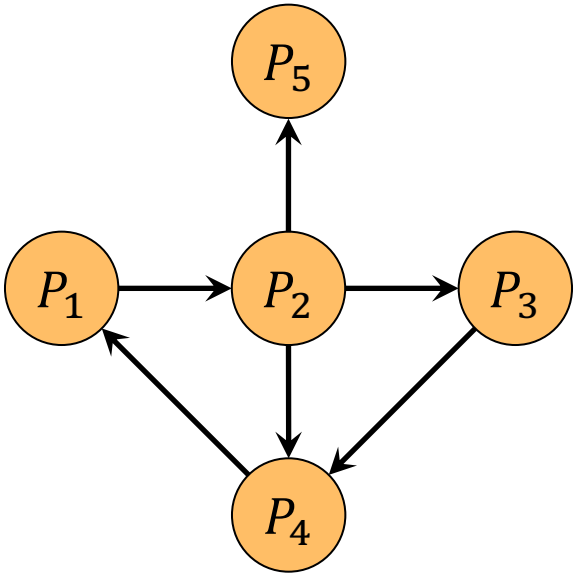
- If the graph contains a cycle, then
  - If there is only one instance per resource type, there is a deadlock.
  - If there are several instances per resource type, there is a possibility of deadlock.

Deadlock Detection

Single Instance of Each Resource Type



Resource-Allocation Graph



Corresponding wait-for graph

Single Instance of Each Resource Type

- Maintain a wait-for graph
  - Nodes are processes.
  - $P_i \rightarrow P_k$ if P_i is waiting for a resource that is held by P_k.
- Periodically invoke an algorithm to search for a cycle in the graph.
  - If there is a cycle, there exists a deadlock.
- An algorithm to detect a cycle in a graph with n vertices requires an order of n^2 operations.

Solution for multiple instances of a resource type

- $Available_{[m]}$
  - $Available_k = x$ means that x instances of resource type R_k are available.
- $Allocation_{[n][m]}$
  - $Allocation_{i,k} = x$ means that process P_i has x instances of resource type R_k currently allocated to it.
- $Request_{[n][m]}$
  - $Request_{i,k} = x$ means that process P_i is requesting x more instances of resource type R_k.

Detection Algorithm

- Let $Work_{[m]}$ and $Finish_{[n]}$ be initialized as
  - $Work \leftarrow Available$
  - $Finish_i \leftarrow (Allocation_i = 0)$
- While there is $i \mid (\neg Finish_i) \ \wedge (Request_i \leq Work)$ do
  - $Work \leftarrow Work + Allocation_i$
  - $Finish_i \leftarrow true$
- If $\neg Finish_i$ for some i, the system is in deadlock and every P_i for which $\neg Finish_i$ is deadlocked.
- This algorithm requires $O(m \times n^2)$ operations.

Example of Detection Algorithm

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

- No deadlock, because sequence $\langle P_0, P_2, P_1, P_3, P_4 \rangle$ results in $Finish_i$ being true for all i.

Example of Detection Algorithm

- Same problem, but P_2 now requests one additional instance of resource type C.
- 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	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Example of Detection Algorithm

- Same problem, but P_2 now requests one additional instance of resource type C.
- Snapshot at time t_0:

	Allocation			*Request*			*Finish*
	A	*B*	*C*	*A*	*B*	*C*	
P_0	0	1	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	

Available		
A	*B*	*C*
0	0	0

Work		
A	*B*	*C*
0	0	0

Example of Detection Algorithm

- Same problem, but P_2 now requests one additional instance of resource type C.
- Snapshot at time t_0:

	Allocation			*Request*			*Finish*
	A	*B*	*C*	*A*	*B*	*C*	
P_0	0	1	0	0	0	0	*T*
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	

Available		
A	*B*	*C*
0	0	0

Work		
A	*B*	*C*
0	1	0

- The system is now in a deadlock involving 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
- If the detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph.
  - It may be difficult or even impossible to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

- There are two options
  - Abort all deadlocked processes
  - Abort one process at a time until the deadlock ends.
- In which order should we choose to abort?
  - Priority of the process.
  - How long the process has run, and how long it will run.
  - Resources that the process has used.
  - Resources that the process needs to complete.
  - How many processes will have to be terminated.
  - Is the process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim
  - Minimize cost.
- Rollback
  - Return to some safe state, restart process from there.
- Starvation
  - Same process may always be picked as victim, so it is a good idea to include number of rollbacks in cost factor.