Operating System: Chap7 Deadlocks

National Tsing Hua University

2022, Fall Semester

Overview

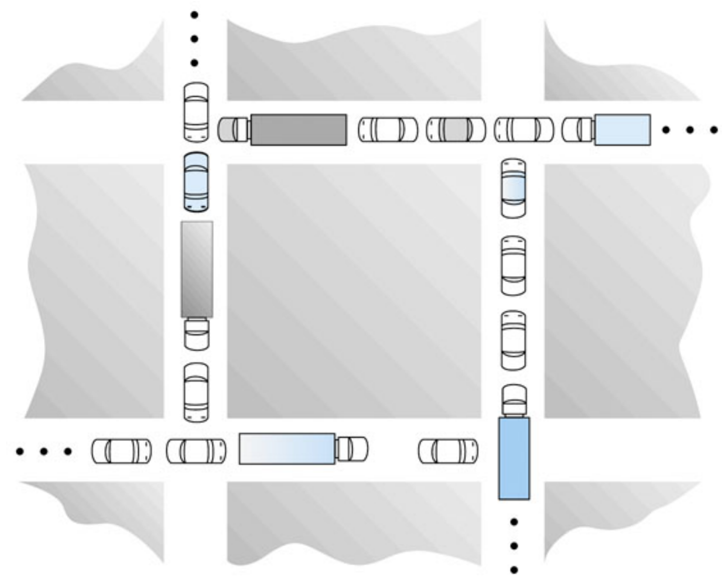
- System Model
- Deadlock Characterization
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlock Problem

- A set of blocked processes each holding some resources and waiting to acquire a resource held by another process in the set
- Ex1: 2 processes and 2 tape drivers
  - Each process holds a tape drive
  - Each process requests another tape drive
- Ex2: 2 processes, and semaphores A & B
  - P1 (hold B, wait A): wait(A), signal(B)
  - P2 (hold A, wait B): wait(B) , signal(A)

Necessary Conditions

- Mutual exclusion:
  - only 1 process at a time can use a resource
- Hold & Wait:
  - a process holding some resources and is waiting for another resource
- No preemption:
  - a resource can be only released by a process voluntarily
- Circular wait:
  - there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n \rightarrow P_0$



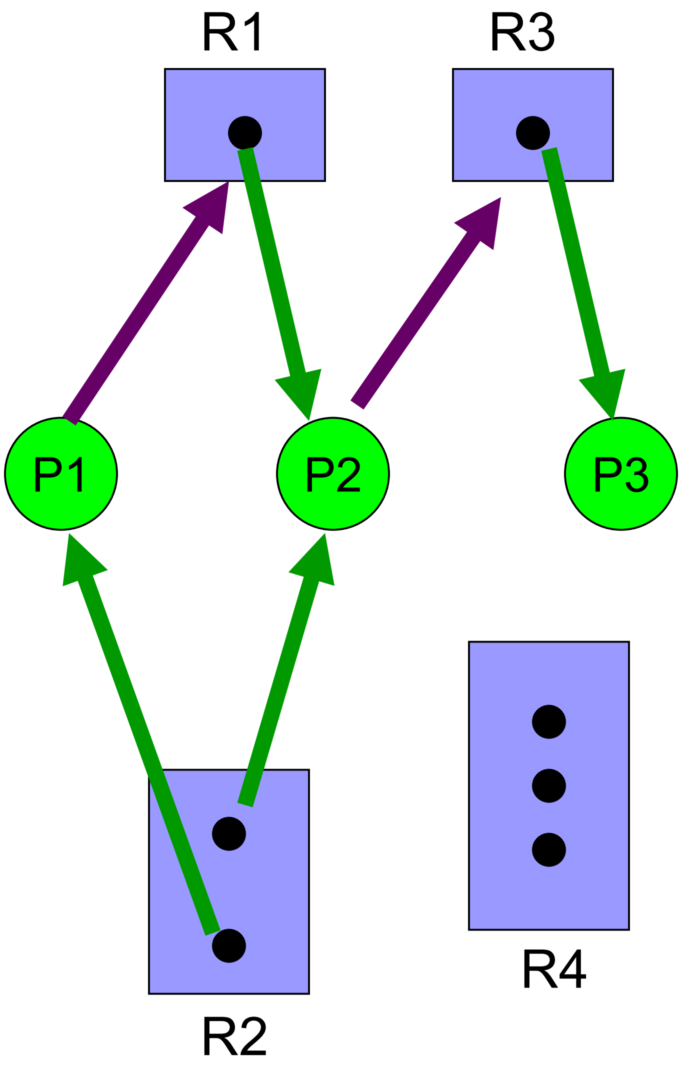
➔ All four conditions must hold for **possible** deadlock!

System Model

- Resources types $R_1, R_2, \ldots, R_m$
  - E.g. CPU, memory pages, I/O devices
- Each resource type R_i has W_i instances
  - E.g. a computer has 2 CPUs
- Each process utilizes a resource as follows:
  - Request → use → release

Resource-Allocation Graph

- 3 processes, P1 ~ *P3*
- 4 resources, R1 ~ *R4*
  - *R1* and *R3* each has one instance
  - *R2* has two instances
  - *R4* has three instances



- **Request edges**:
  - *P1*→*R1*: *P1* requests R1
- **Assignment edges**:
  - R2→P1: One instance of R2 is allocated to P1

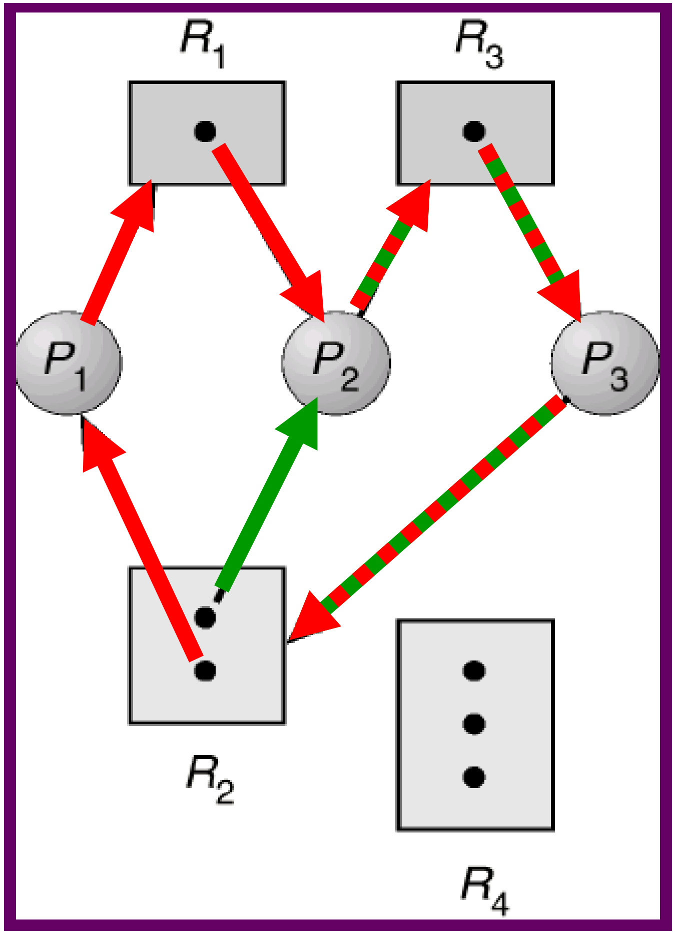
→P1 is hold on an instance of R2 and waiting for an instance of R1

Resource-Allocation Graph w/ Deadlock

- If the graph contains a cycle, a deadlock **may exist**
- In the example:
  - P1 is waiting for P2
  - P2 is waiting for P3

    ➔P1 is also waiting for P3
  - Since P3 is waiting for P1 or P2, and they both waiting for P3

    ➔deadlock!

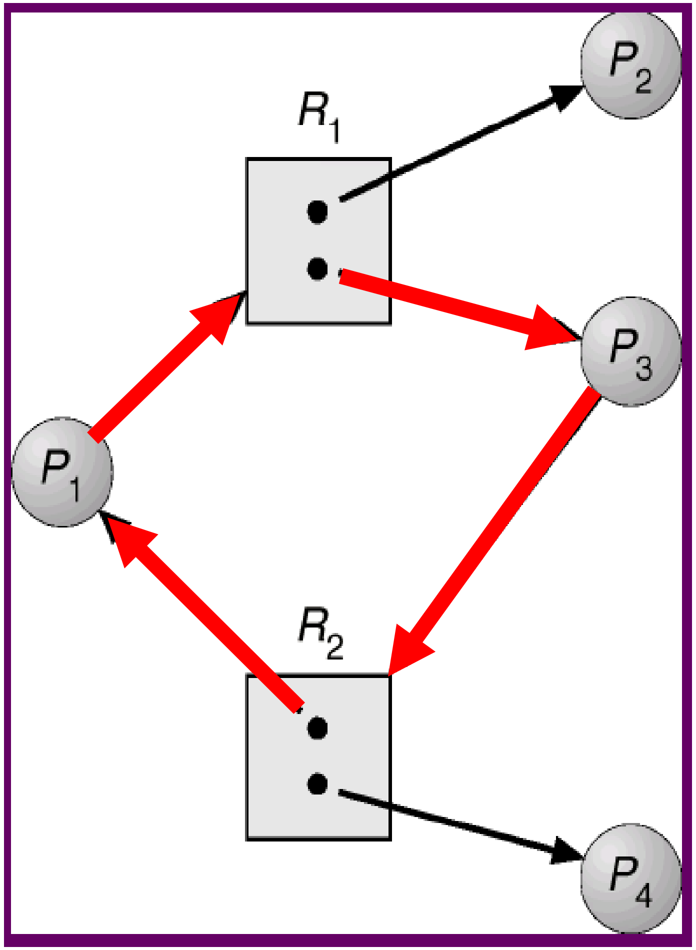


RA Graph w/ Cycle but NO Deadlock

- If the graph contains a cycle, a deadlock **may exist**
- In the example:
  - P1 is waiting for P2 or P3
  - P3 is waiting for P1 or P4
  - Since P2 and P4 wait no one

    → no deadlock

    between P1 & P3!



Deadlock Detection

- If graph contains **no cycle → no deadlock**
  - **Circular wait** cannot be held
- If graph contains a cycle:
  - if one instance per resource type → deadlock
  - if multiple instances per resource type → **possibility** of deadlock

Handling Deadlocks

- Ensure the system will ***never*** enter a **deadlock state**
  - **deadlock prevention:** ensure that at least one of the 4 necessary conditions cannot hold
  - **deadlock avoidance:** dynamically examines the resource-allocation state before allocation
- Allow to enter a deadlock state and ***then* recover**
  - **deadlock detection**
  - **deadlock recovery**
- **Ignore the problem** and pretend that deadlocks never occur in the system
  - used by most operating systems, including UNIX.

Review Slides (I)

- deadlock necessary conditions?
  - mutual exclusion
  - hold & wait
  - no preemption
  - circular wait
- resource-allocation graph?
  - cycle in RAG → deadlock?
- deadlock handling types?
  - deadlock prevention
  - deadlock avoidance
  - deadlock recovery
  - ignore the problem

Deadlock Prevention & Deadlock Avoidance

Deadlock Prevention

- Mutual exclusion (ME): do not require ME on sharable resources
  - e.g. there is no need to ensure ME on read-only files
  - Some resources are not shareable, however (e.g. printer)
- Hold & Wait:
  - When a process requests a resource, it does not hold any resource
  - Pre-allocate all resources before executing

☹ resource utilization is low; starvation is possible

Deadlock Prevention (con't)

- No preemption
  - When a process is waiting on a resource, all its holding resources are preempted
    - e.g. P1 request R1, which is allocated to P2, which in turn is waiting on R2. $(P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2)$
    - R1 can be preempted and reallocated to P1
  - Applied to resources whose states can be easily saved and restored later
    - e.g. CPU registers & memory
  - It cannot easily be applied to other resources
    - e.g. printers & tape drives

Deadlock Prevention (con't)

- Circular wait
  - impose a total ordering of all resources types
  - a process requests resources in an increasing order
    - Let $R=\{R_0, R_1, \ldots, R_N\}$ be the set of resource types
    - When request R_k, should release all R_i, $i \geq k$
  - Example:
    - F(*tape drive*) = 1, F(*disk drive*) = 5, F(*printer*) = 12
    - A process must request tape and disk drive before printer
  - proof: counter-example does not exist
    - $P_0\ (R_0) \rightarrow R_1,\ P_1\ (R_1) \rightarrow R_2, \ldots, \underline{P_N(R_N) \rightarrow R_0}$
    - Conflict: $R_0 < R_1 < R_2 < \ldots < R_N < R_0$

P_N hold R_N, wait R_0

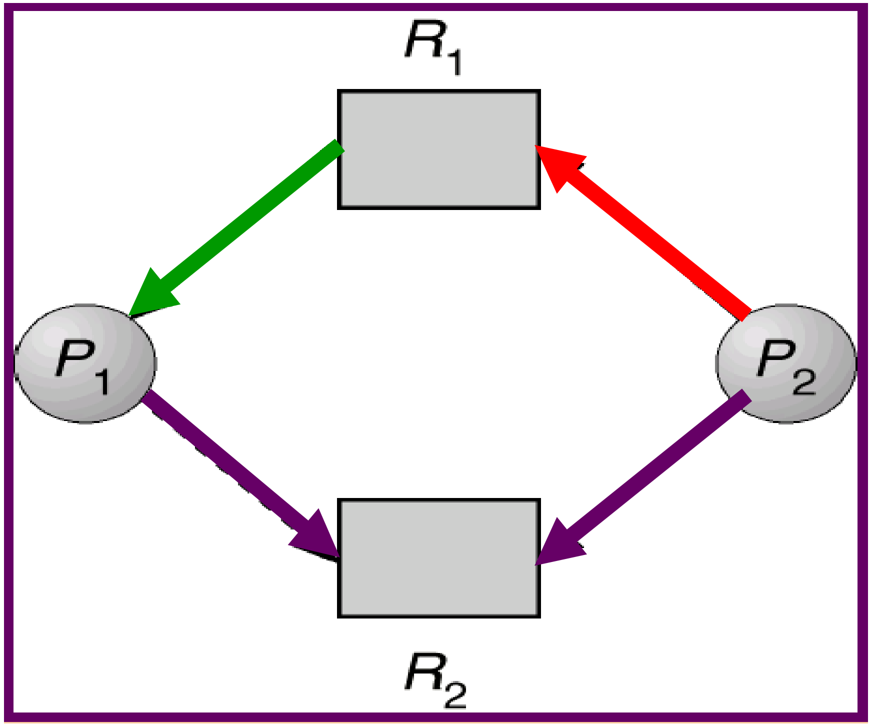
Avoidance Algorithms

- Single instance of a resource type
  - **resource-allocation graph (RAG) algorithm** based on circle detection

- Multiple instances of a resource type
  - **banker's algorithm** based on safe sequence detection

Resource-Allocation Graph (RAG) Algorithm

- **Request edge**: Pi→Rj
  - Process Pi is **waiting** for resource Rj
- **Assignment edge**: Rj→Pi
  - Resource Rj is **allocated** and held by process Pi
- **Claim edge**: Pi→Rj
  - process Pi may **request** Rj **in the future**

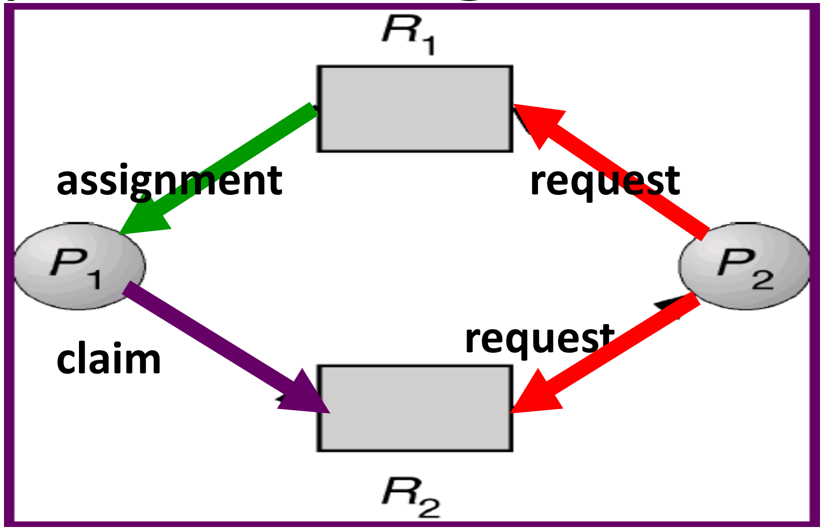


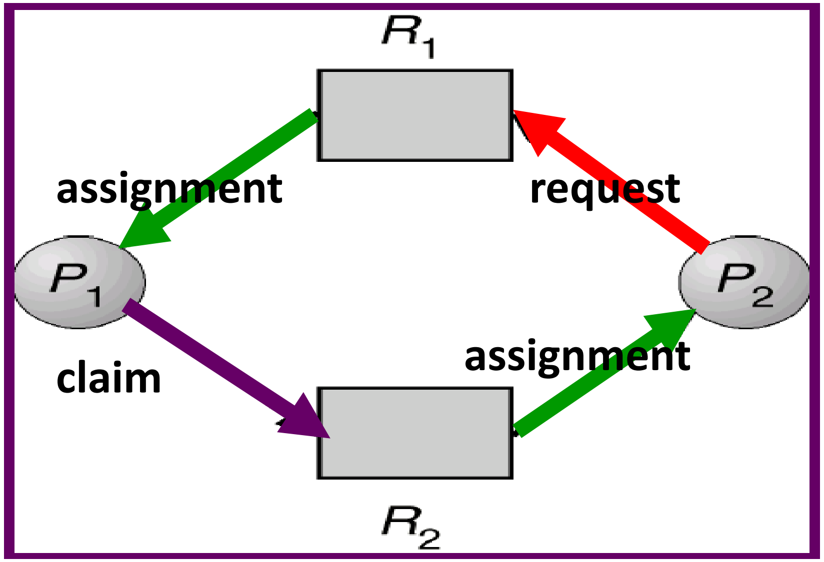
- Claim edge converts to request edge
  - When a resource is requested by process
- Assignment edge converts back to a claim edge
  - When a resource is released by a process

Resource-Allocation Graph (RAG) Algorithm

- Resources must be claimed *a priori* in the system
- Grant a request only if **NO** cycle created
  - Check for safety using a cycle-detection algorithm, O(n^2)
- If a request changes to assignment could cause circle, it will be rejected or delayed
  - E.g.: request from P2 to R2 cannot be approved

    ➔Deadline occurs if P2 requests for R2 happens





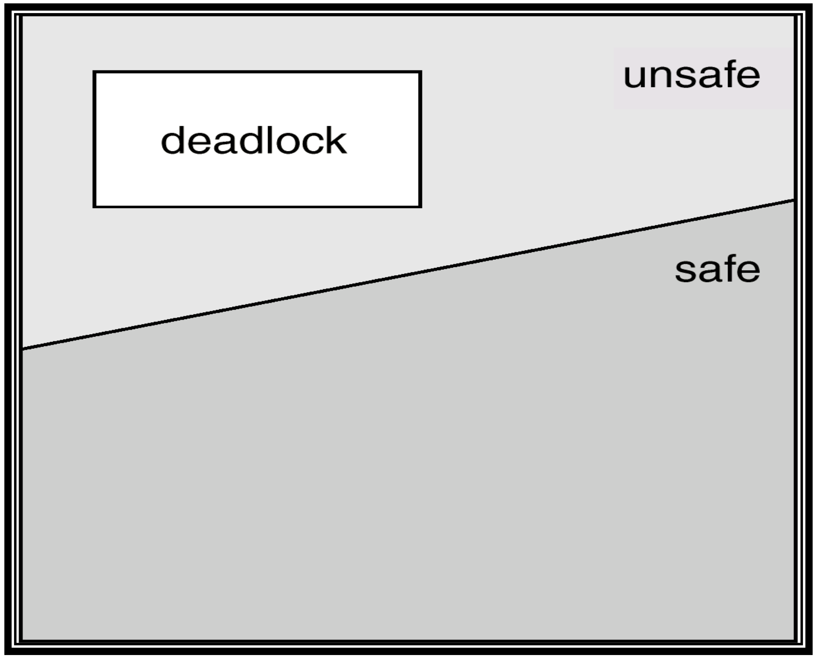
Avoidance Algorithms

- Single instance of a resource type
  - **resource-allocation graph (RAG) algorithm** based on circle detection

- Multiple instances of a resource type
  - **banker's algorithm** based on safe sequence detection

Deadlock Avoidance

- ***safe state***: a system is in a safe state if there exists **a sequence of allocations** to satisfy requests by all processes
  - This sequence of allocations is called ***safe sequence***
- safe state → no deadlock
- unsafe state → possibility of deadlock
- deadlock avoidance → ensure that a system never enters an **unsafe state**



Safe State with Safe Sequence

- There are 12 tape drives
- Assuming at t0:

Hint from processes →

	Max Needs	Current Holding
P0	10	5
P1	4	2
P2	9	2

➔ <P1, P0, P2> is a safe sequence

Safe State with Safe Sequence

- There are 12 tape drives
- Assuming at t0:

	Max Needs	Current Holding	Available
P0	10	5	
P1	4	2	3
P2	9	2	

➔ <P1, P0, P2> is a safe sequence

1. P1 satisfies its allocation with 3 available resources

Safe State with Safe Sequence

- There are 12 tape drives
- Assuming at t0:

	Max Needs	Current Holding	Available
P0	10	5	5
P1	4	0	
P2	9	2	

➔ <P1, P0, P2> is a safe sequence

1. P1 satisfies its allocation with 3 available resources
2. P0 satisfies its allocation with 5 available resources

Safe State with Safe Sequence

- There are 12 tape drives
- Assuming at t0:

	Max Needs	Current Holding	Available
P0	10	0	
P1	4	0	
P2	9	2	10

➔ <P1, P0, P2> is a safe sequence

1. P1 satisfies its allocation with 3 available resources
2. P0 satisfies its allocation with 5 available resources
3. P2 satisfies its allocation with 10 available resources

Un-Safe State w/o Safe Sequence

- Assuming at t1:

	Max Needs	Current Holding	Available
P0	10	5	
P1	4	2	2
P2	9	~~2~~ 3	

if P2 requests & is allocated 1 more tape drive

→No safe sequence exist…

→ this allocation enters the system into an unsafe state

- A request is only granted if the allocation leaves the system in a safe state

Banker's Algorithm

- Use for multiple instances of each resource type
- Banker algorithm:
  - Use a general **safety algorithm** to pre-determine if any safe sequence exists after allocation
  - Only proceed the allocation if safe sequence exists
- Safety algorithm:
  1. Assume processes need maximum resources
  2. Find a process that can be satisfied by free resources
  3. Free the resource usage of the process
  4. Repeat to step 2 until all processes are satisfied

Banker's Algorithm Example1

- Total instances: A:10, B:5, C:7
- Available instances: A:3, B:3, C:2

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- Is the system currently in safe state?

Banker's Algorithm Example1: Step1

- Total instances: A:10, B:5, C:7
- Available instances: A:3, B:3, C:2

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- Is the system currently in safe state?
  - Safe sequence: P1

Banker's Algorithm Example1: Step2

- Total instances: A:10, B:5, C:7
- Available instances: A:5, B:3, C:2

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- Is the system currently in safe state?
  - Safe sequence: P1, P3

Banker's Algorithm Example1: Step3

- Total instances: A:10, B:5, C:7
- Available instances: A:7, B:4, C:3

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- Is the system currently in safe state?
  - Safe sequence: P1, P3, P4

Banker's Algorithm Example1: Step4

- Total instances: A:10, B:5, C:7
- Available instances: A:7, B:4, C:5

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- Is the system currently in safe state?
  - Safe sequence: P1, P3, P4, P2

Banker's Algorithm Example1: Step5

- Total instances: A:10, B:5, C:7
- Available instances: A:10, B:4, C:7

	Max			Allocation			Need(Max.-Alloc.)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- Is the system currently in safe state?
  - Safe sequence: P1, P3, P4, P2, P0

Yes, a safe sequence is found

Banker's Algorithm Example2

- Total instances: A:10, B:5, C:7
- Available instances: A:3, B:3, C:2

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- If Request (P1) = (1, 0, 2) arrives, can we grant the request?

Banker's Algorithm Example2: Step1

- Total instances: A:10, B:5, C:7
- Available instances: A:2, B:3, C:0

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	3	0	2	0	2	0
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- If Request (P1) = (1, 0, 2) arrives, can we grant the request?
  - Assume we grant the request of P1, and change the system states accordingly:
    - The allocation and need of P1, and the total available instances

Banker's Algorithm Example2: Step2

- Total instances: A:10, B:5, C:7
- Available instances: A:2, B:3, C:0

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	3	0	2	0	2	0
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- If Request (P1) = (1, 0, 2) arrives, can we grant the request?
  - Find safe sequence: P1, P3, P4, P0, P2

Yes, the system can remain in safe state, so the request of P1 can be granted

Banker's Algorithm Example3

- Total instances: A:10, B:5, C:7
- Available instances: A:3, B:3, C:2

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	0	0	2	4	3	1

- If Request (P4) = (3, 3, 0) arrives, can we grant the request?

Banker's Algorithm Example3: Step1

- Total instances: A:10, B:5, C:7
- Available instances: A:0, B:0, C:2

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	3	3	2	1	0	1

- If Request (P4) = (3, 3, 0) arrives, can we grant the request?
  - Assume we grant the request of P4, and change the system states accordingly:
    - The allocation and need of P4, and the total available instances

Banker's Algorithm Example3: Step2

- Total instances: A:10, B:5, C:7
- Available instances: A:0, B:0, C:2

	Max			Allocation			Need(Max-Alloc)		
	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3
P1	3	2	2	2	0	0	1	2	2
P2	9	0	2	3	0	2	6	0	0
P3	2	2	2	2	1	1	0	1	1
P4	4	3	3	3	3	2	1	0	1

- If Request (P4) = (3, 3, 0) arrives, can we grant the request?
  - enter into an unsafe state (no safe sequence can be found!)

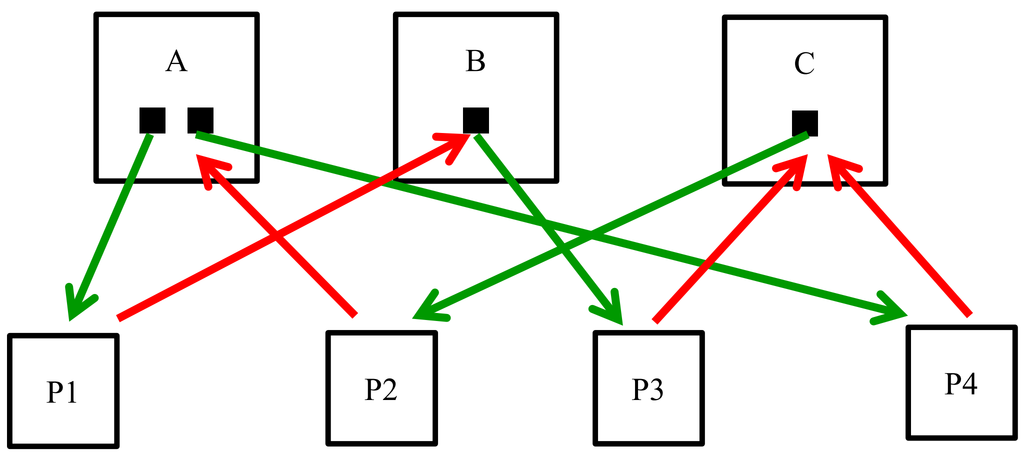
No, the system will enter unsafe state, if the request of P1 is granted
The request must wait until the system can remain in safe state

Programming Example

- A, B, C are semaphores initialized with the values of 2, 1, 1,

//P1	//P2	//P3	//P4
wait(A)	**wait(C)**	**wait(B)**	**wait(A)**
wait(B)	**wait(A)**	**wait(C)**	**wait(C)**
signal(B)	signal(A)	signal(C)	signal(C)
signal(A)	signal(C)	signal(B)	signal(A)

If no avoidance applies, Deadlock Occurs!



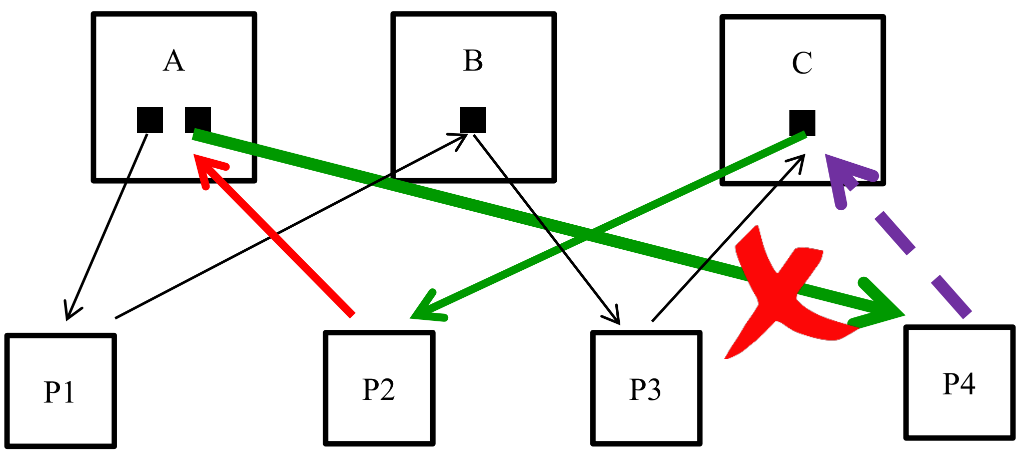
Programming Example

- A, B, C are semaphores initialized with the values of 2, 1, 1,

//P1	//P2	//P3	//P4
wait(A)	**wait(C)**	**wait(B)**	**wait(A)**
wait(B)	**wait(A)**	**wait(C)**	**wait(C)**
signal(B)	signal(A)	signal(C)	signal(C)
signal(A)	signal(C)	signal(B)	signal(A)

If avoidance algorithm is applied, the call will be blocked
P4 can't receive lock A, even if A is available

Deadlock Avoid!



Banker's algo: avail: 0 0 0

	Max A	Max B	Max C	Alloc A	Alloc B	Alloc C
P1	1	1	0	1	0	0
P2	1	0	1	0	0	1
P3	0	1	1	0	1	0
P4	1	0	1	1	0	0

After P4.wait(A)

No safe sequence!

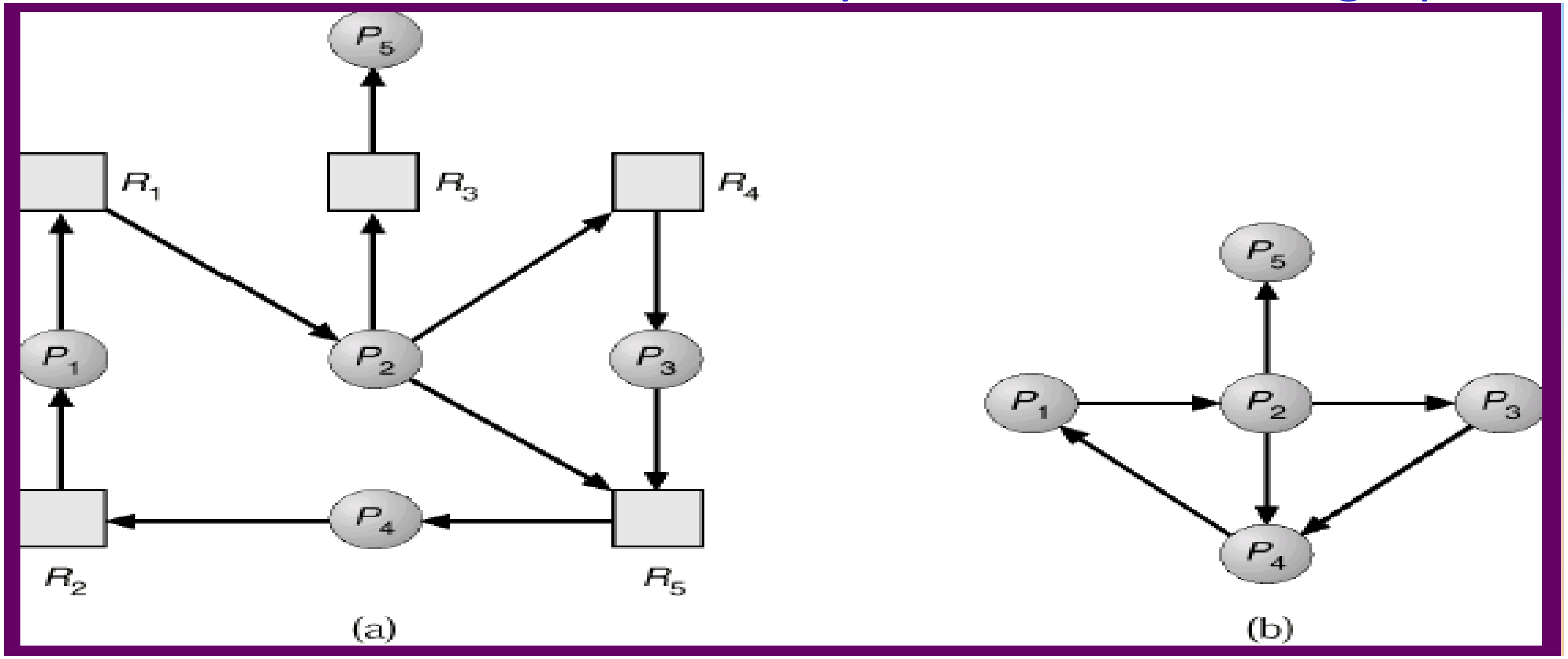
Review Slides (II)

- deadlock prevention methods?
  - mutual exclusion
  - hold & wait
  - no preemption
  - circular wait
- deadlock avoidance methods?
  - safe state definition?
  - safe sequence?
  - claim edge?

Deadlock Detection & Deadlock Recovery

Deadlock Detection

- Single instance of each resource type
  - convert request/assignment edges into wait-for graph
  - deadlock exists if there is a cycle in the wait-for graph



Resource-Allocation Graph   Corresponding wait-for graph

Multiple-Instance for Each Resource Type

- Total instances: A:7, B:2, C:6
- Available instances: A:0, B:0, C:0

	Allocation			Request		
	A	B	C	A	B	C
P0	0	1	0	0	0	0
P1	2	0	0	2	0	2
P2	3	0	3	0	0	0
P3	2	1	1	1	0	0
P4	0	0	2	0	0	2

- The system is in a safe state ➔ <P0, P2, P3, P1, P4>
  ➔ no deadlock
- If P2 request = <0, 0, 1> ➔ no safe sequence can be found
  ➔the system is deadlocked

Deadlock Recovery

- Process termination
  - abort all deadlocked processes
  - abort 1 process at a time until the deadlock cycle is eliminated
    - which process should we abort first?
- Resource preemption
  - select a victim: which one to preempt?
  - rollback: partial rollback or total rollback?
  - starvation: can the same process be preempted always?

Textbook Problem Set

- 7.6: In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and underwhat circumstances?

  a. Increase Available (new resources added).

  b. Decrease Available (resource permanently removed from system)

  c. Increase Max for one process (the process needs more resources than allowed, it may want more)

  d. Decrease Max for one process (the process decides it does not need that many resources)

  e. Increase the number of processes.

  f. Decrease the number of processes.

Textbook Problem Set

- 7.7: Consider a system consisting of four resources of the same type that are shared by threeprocesses, each of which needs at most two resources. Show that the system is deadlock free.
- 7.8: Consider a system consisting of m resources of the same type, being shared by n processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if *both* the following two conditions hold:

  a. The maximum need of each process is between 1 and m resources

  b. The sum of all maximum needs is less than m + n
- 7.9: Consider the version of the dining-philosophers problem in which the chopsticks are placed at the center of the table and any two of them can be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request can be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

Textbook Problem Set

- 7.12: Consider the following snapshot of a system:

	Allocation	Max
	A B C D	A B C D
P0	3 0 1 4	5 1 1 7
P1	2 2 1 0	3 2 1 1
P2	3 1 2 1	3 3 2 1
P3	0 5 1 0	4 6 1 2
P4	4 2 1 2	6 3 2 5

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the processes may complete. Otherwise, illustrate why the state is unsafe.

a. Available = (0, 3, 0, 1)

b. Available = (1, 0, 0, 2)

Textbook Problem Set

- 7.13: Consider the following snapshot of a system:

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?