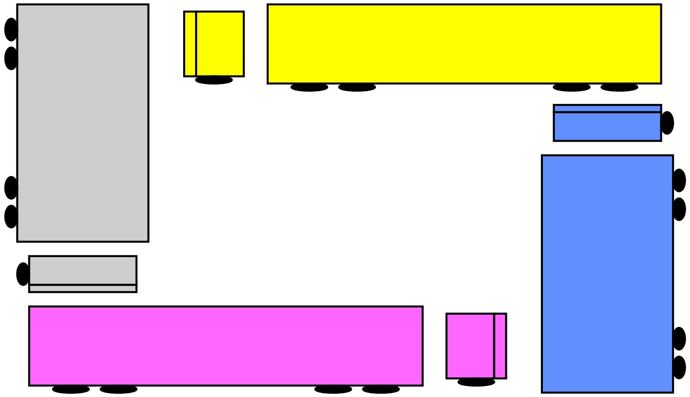


CS310 Operating Systems

Lecture 30: Deadlock Avoidance, Banker’s Algorithm

Ravi Mittal

IIT Goa

Acknowledgements !

- Contents of this class presentation has been taken from various sources. Thanks are due to the original content creators:
    - CS162, Operating System and Systems Programming, University of California, Berkeley
    - Book: Modern Operating System, Andrew Tanenbaum and Herbert Bos – Chapter 6
    - Book: Operating Systems: Principles and Practice: Thomas Anderson and Michael Dahlin, Volume II
        - Chapter 5.6

Reading

- Book: Operating Systems: Principles and Practice: Thomas Anderson and Michael Dahlin, Volume II
- Book: Operating System Concepts, 10th Edition, by Silberschatz, Galvin, and Gagne
- Book: Modern Operating System, Andrew Tanenbaum and Herbert Bos – Chapter 6

Previous Classes

Necessary Conditions for Deadlock

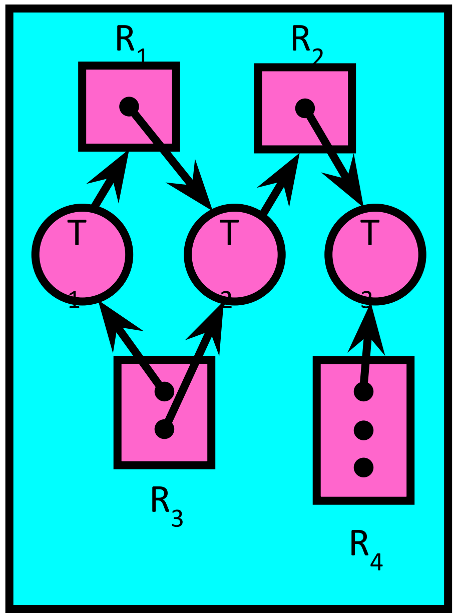
- There are 4 necessary conditions for deadlock to occur
- If you can prevent any one of these conditions, you can eliminate the possibility of deadlock

1. Bounded Resources
   - There are a finite number of threads that can simultaneously use a resource
2. No preemption
   - Once a thread acquires a resource, its ownership cannot be revoked until the thread acts to release it
3. Wait while Holding
   - A thread holds one resource while waiting for another
     - Also called multiple independent requests
4. Circular Waiting
   - A set of waiting threads such that each thread is waiting for a resource held by another

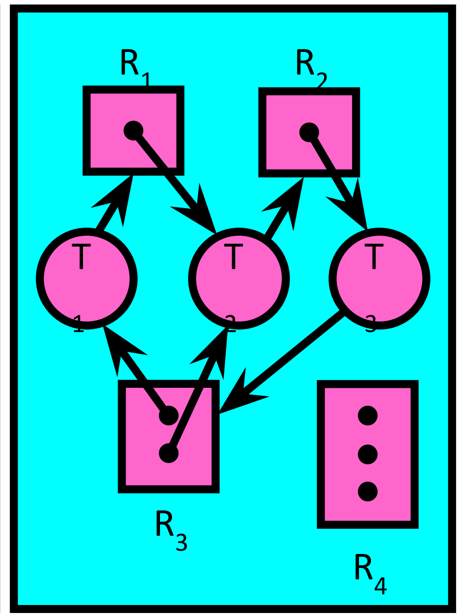
Resource-Allocation Graph (RAG) Examples

Model: Directed Graph

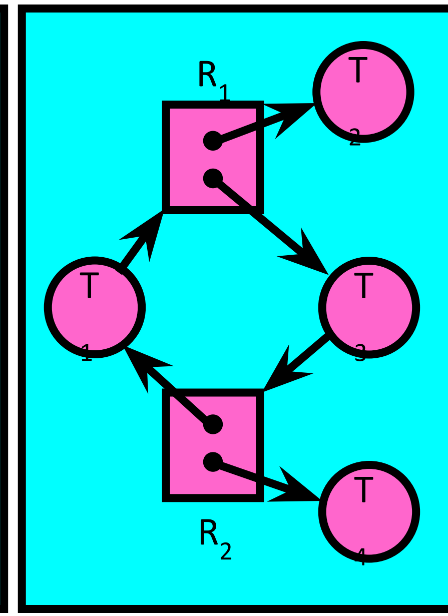
- request edge
  - $T_i \rightarrow R_j$
- assignment edge
  - $R_j \rightarrow T_i$



Simple Resource Allocation Graph



Allocation Graph With Deadlock

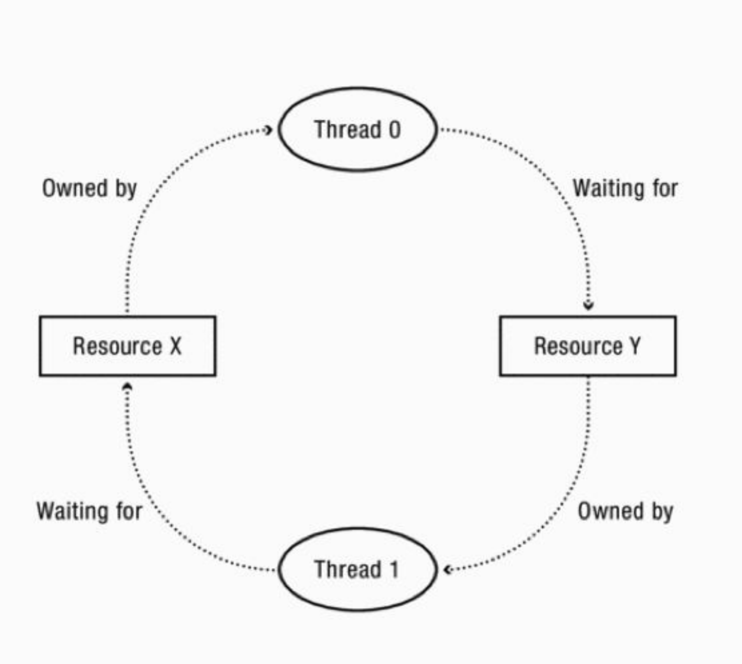


Allocation Graph With Cycle, but No Deadlock

Instead of thread T, we can also represent a process with a circle

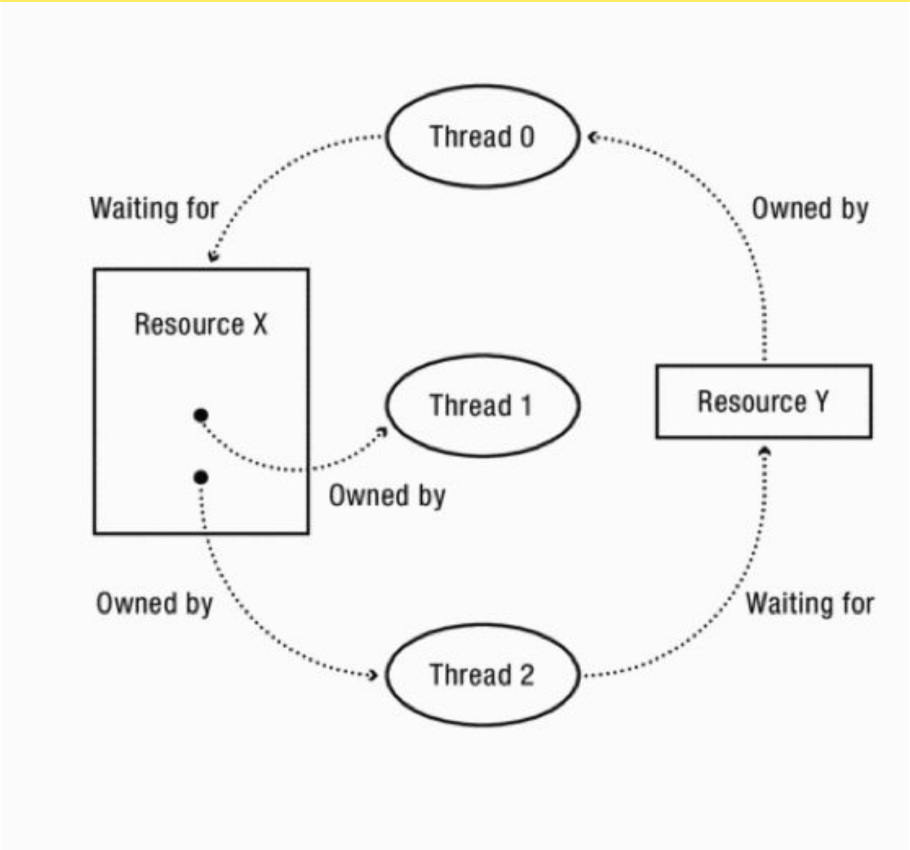
Analyzing - Resource Allocation Graph (RAG)

- Consider several resources and only one thread can hold each resource at a time
  - Example: resources – one printer, one keyboard, one speaker
- We can detect a deadlock by analyzing simple graph

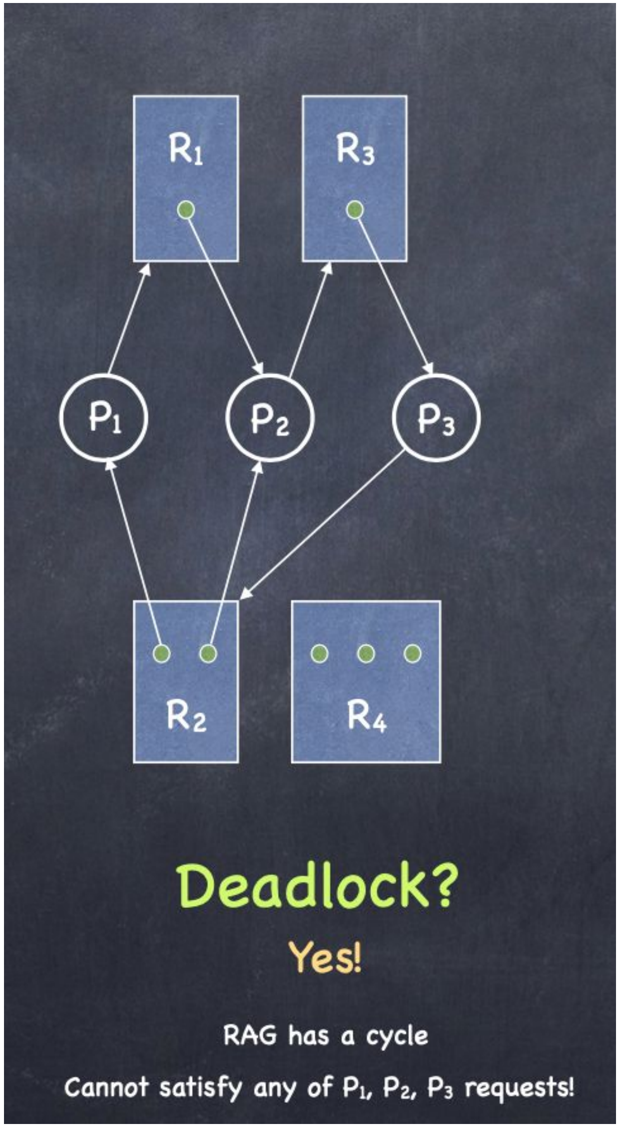


Multiple instances of one resource

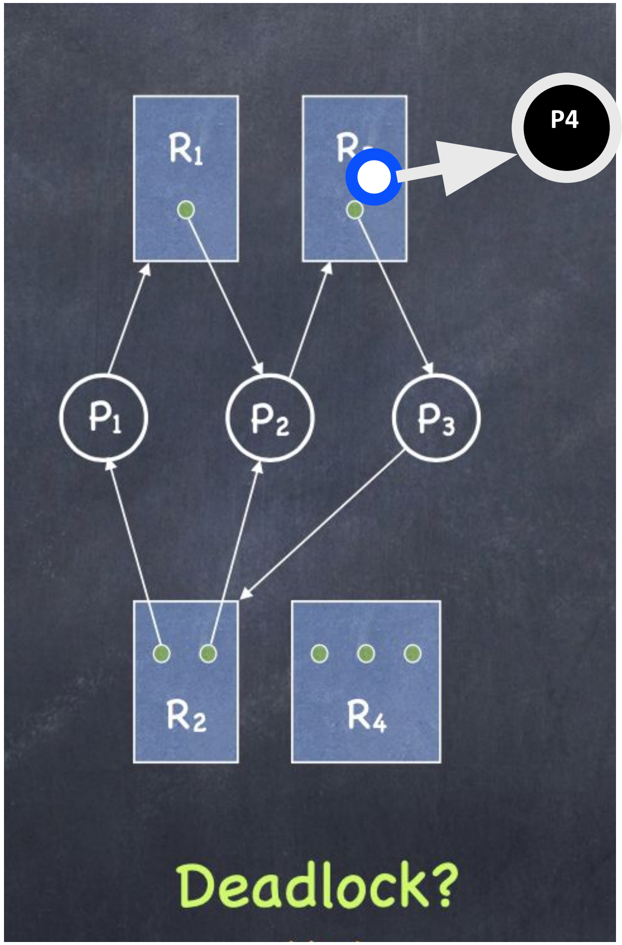
- Multiple instances of a resource can be represented as a resource with k interchangeable instances
  - Eg K equivalent printers as a node with k connections
- Cycle is necessary but not sufficient condition for deadlock



Example 2: RAG Reduction



Example 2: RAG Reduction



Today we will study

- Deadlock handling approaches
- Deadlock Avoidance
- Banker’s Algorithm

Deadlock Handling

Deadlock can be deadly!

- Does a deadlock disappear on it’s own?
  - No. Unless a process is killed or forced to release a resource, deadlock exists

- If a system is not deadlock at time T, is it guaranteed to be deadlock-free at T+1?

Deadlock can be deadly!

- Does a deadlock disappear on it's own?
  - No. Unless a process is killed or forced to release a resource, deadlock exists

- If a system is not deadlock at time T, is it guaranteed to be deadlock-free at T+1?
  - No. Just by requesting a resource (never mind being granted one) a process can create a circular wait!

Handling Deadlocks

- Ignore the problem
  - Pretend that deadlock never occurs in the system
    - This solution is used by most Operating Systems
      - Example: Linux, Windows
- We can use a protocol to prevent or avoid deadlocks ensuring that system will never enter a deadlocked state
  - It is up to kernel and application developers to write programs that handle deadlocks
- We allow the system to enter deadlocked state, detect it and recover from it

How Should a System Deal With Deadlock?

Three different approaches:

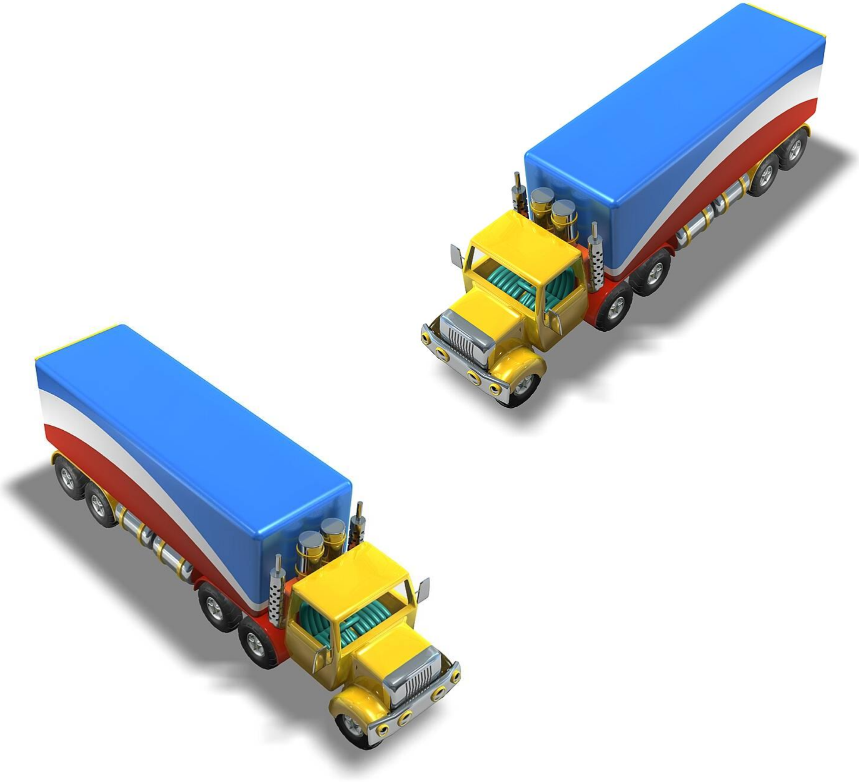
1. Deadlock avoidance
   - Dynamically delay resource requests so deadlock doesn't happen
   - Thread must have enough information about resource requests and use during its lifetime
   - This knowledge can be used by OS to grant resources at appropriate time (now or later)
2. Deadlock prevention
   - Write your code in a way that it isn't prone to deadlock
3. Deadlock recovery
   - Let deadlock happen, and then figure out how to recover from it

- Modern operating systems:
  - Make sure the *system* isn't involved in any deadlock
  - Ignore deadlock in applications
    - "Ostrich Algorithm" or deadlock denial

Deadlock Avoidance

Deadlocks Avoidance

Deadlocks Avoidance



- How do cars do it?
  - Never block an intersection
  - Must back up if you find yourself doing so

Deadlock Avoidance

- Idea: When a thread requests a resource, OS checks if it would result in deadlock
  - If not, it grants the resource right away
  - If so, it waits for other threads to release resources

THIS DOES NOT WORK AT ALL TIME!!!!

- Example:

	Thread A	Thread B	
	`x.Acquire();`	`y.Acquire();`	**Wait**
Blocks	`y.Acquire();`	`x.Acquire();`	
...	`...`	`...`	**But it's too**
	`y.Release();`	`x.Release();`	**late...**
	`x.Release();`	`y.Release();`	

A priori information Requirement

- Additional a priori information availability
  - Example: Process P will request R1 and then R2 before releasing them; Process Q will require R2 and then R1
  - System can now take decision whether to allow the request, now, or later
- Simplest and most useful model
  - Let each process declare the maximum number of resources of each type that it may need
- Dynamically examine 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

Deadlock Avoidance: Three States

- Safe state
  - System can delay resource acquisition to prevent deadlock

Deadlock avoidance: prevent system from reaching an *unsafe* state

- Unsafe state
  - No deadlock yet…
  - But threads can request resources in a pattern that ***unavoidably*** leads to deadlock
- Deadlocked state
  - There exists a deadlock in the system

Safe State

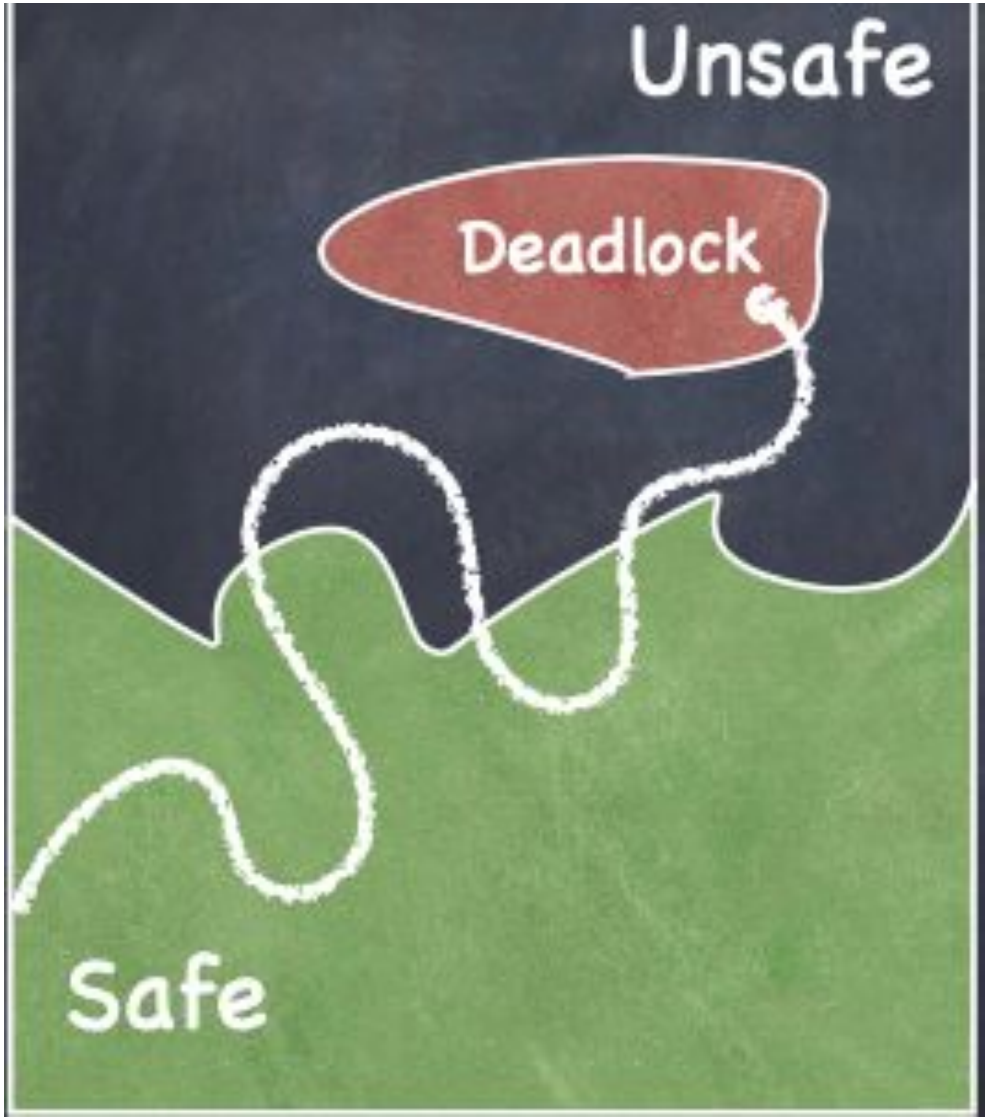
- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in safe state if there exists a sequence <P_1, P_2, …, P_n> of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$
- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is 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
- If no such sequence exists, then the system state is said to be unsafe

Basic Facts

- 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

- A deadlocked state is an unsafe state
- Not all unsafe states are deadlocks
- An unsafe state may lead to a deadlock
- As long as the state is safe, OS must avoid unsafe states

Safe, Unsafe, Deadlock State



Safe State Example

- Suppose there are 12 tape drives

	max need	current usage	could ask for
p0	10	5	5
p1	4	2	2
p2	9	2	7

3 drives remain

- Current state is safe because a safe sequence exists: <p1,p0,p2>

  p1 can complete with current resources

  p0 can complete with current+p1 released resources

  p2 can complete with current +p1+p0 released resources

Safe and Unsafe States (1)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

Is (a) safe?

Safe and Unsafe States (1)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Is (a) safe? YES

Safe and Unsafe States (2)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

Is state in (b) safe?

Safe and Unsafe States (2)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Demonstration that the sate in b is not safe

Banker’s Algorithm

E.W. Dijkstra & N. Habermann

Banker's Algorithm

- Suppose we know the worst case resource needs of processes in advance
  - A bit like knowing the credit limit on your credit cards. (This is why they call it the Banker's Algorithm)
- Observation: Suppose we just give some process ALL the resources it could need…
  - Then it will execute to completion.
  - After which it will give back the resources.
- Like a bank: If Visa just hands you all the money your credit lines permit, at the end of the month, you'll pay your entire bill

Banker's Algorithm

- So…
  - A process pre-declares its worst-case needs
  - Then it asks for what it "really" needs, a little at a time
  - The algorithm decides when to grant requests
- It delays a request unless:
  - It can find a sequence of processes…
  - such that it could grant their outstanding need…
  - so they would terminate…
  - letting it collect their resources…
  - and in this way it can execute everything to completion!

Banker's Algorithm for single resource

- The algorithm checks to see if granting the request leads to an unsafe state
  - If so, the request is denied
- If granting the request leads to a safe state, it is carried out
- Example: 4 customers: A, B, C, D
  - Each is granted credit limit of a certain number of units
    - Where 1 unit (= 1K USD)
  - Banker knows that not all customers will need max credit
    - So he kept only 10 units (instead of 22 units)

Banker’s Algorithm for single resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

Figure 6-11. Three resource allocation states: (a) Safe. (b) Safe

- In (b), system is safe
- Now, OS must allocate resources to C to complete

Banker’s Algorithm for single resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Figure 6-11. Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

- However, if it allocates resources to 1 resource to B
  - Unsafe state
- Banker must give resources only when it leads to safe state

Lecture Summary

- We have studied concepts of Safe state, Unsafe state and Deadlocked state
- Deadlock must be avoided by
  - keeping track of which states are safe and which are unsafe
  - A safe state is one in which there exists a sequence of events that guarantee that all processes can finish
- The banker's algorithm avoids deadlock by not granting a request if that request will put the system in an unsafe state