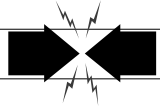


Deadlocks



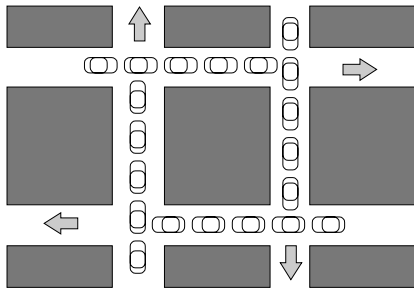
• Set of processes

- Each waiting for resource
- Resource held by another

• No process can continue

- Mutual waiting
 - “Dining philosophers”

Traffic Deadlock



Deadlock Conditions

• Mutual exclusion

• “Hold and wait”

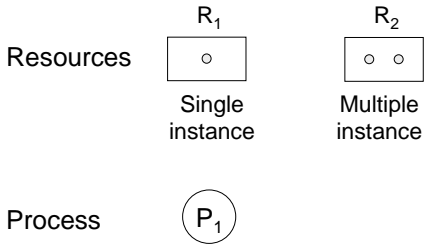
- Multiple independent requests

• No preemption

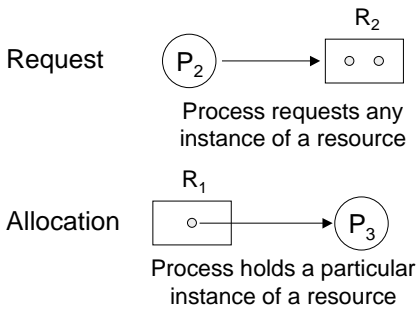
- No process forced to release resource

• Circular wait

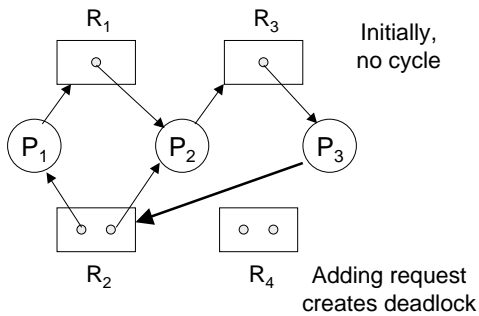
Allocation Graph Notation (1)



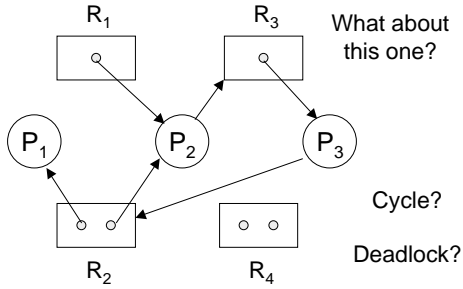
Allocation Graph Notation (2)



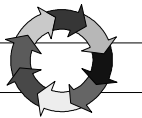
Allocation Graph (1)



Allocation Graph (2)



Allocation Graph Cycles



- **Single-instance resources**
 - Cycle implies deadlock exists
- **Multiple-instance resources**
 - Deadlock requires a cycle
 - But cycle may not be deadlock

Deadlock Handling

- **Prevention**
 - Remove a necessary condition
- **Avoidance**
 - Use information to manage
- **Recovery**
 - Detect and "fix"
 - Automatic or manual
- **Ignore**
 - Reboot?

Prevention - Mutual Exclusion

- **Remove “mutual exclusion”**
- **Make all resources sharable**
 - E.g., read-only files
- **Generally not practical**
 - Some resources do not permit simultaneous access

Prevention - Hold and Wait

- **Never “hold and wait”**
- **Don’t request a new resource while holding another**
 - Request all at once (beginning)?
 - Request only when holding none?
- **Starvation still possible**

Prevention - Preemption

- **Deadlock condition**
 - Held resources cannot be taken away
- **Solution: permit preemption**
 - Take away resources already held
 - When new request cannot be granted
 - Problems
 - Rollback infeasible, starvation?

Prevention - Circular Wait

- **Deadlock needs circular wait**
- **Solution: order all resources**
 - Common sequence
 - E.g., memory, then file, then printer
- **Always request in order**
 - Circular request no longer possible

Deadlock Prevention Summary

- **Ideal if it works**
 - Guarantees freedom from deadlock
 - No overhead
- **Problems**
 - Often not practical
 - System inflexibility
 - Poor resource utilization

Avoidance

- **Use information to avoid**
- **Process must forecast needs**
 - Binding contract
- **Maintain safe states**
 - Resource needs can be satisfied
 - Deadlock cannot result

Safe Sequence

- **Ordered requests**
- **Guaranteed to succeed**
 - “First” process can get all it needs
 - While others wait if necessary
 - Process releases all resources
 - Providing enough for “next”
- **Successful completion for all**

Unsafe Sequence

P₀ tries to allocate what it needs, but blocks
 Neither P₁ nor P₂ can get their “needs” either
 The result is deadlock

Process	Allocated	Need	Done	Unused
P ₀	5	5	No	3
P ₁	2	2	No	
P ₂	2	7	No	

Testing for Safe Sequence

Neither P₀ or P₂ can finish
 But P₁ can get all it needs
 Then P₁ terminates, freeing resources
 Continue for P₀ and P₂

Try again?

Process	Allocated	Need	Done	Unused
P ₀	0	--	Yes	12
P ₁	0	--	Yes	
P ₂	0	--	Yes	

Safe Algorithm

Banker's algorithm,
page 220

- **When request is made**
 - System must have safe sequence
- **Tentatively grant request**
 - Check resulting state for safety
- **Force process to wait**
 - If cannot be granted safely

Detection

- **Detect when deadlock occurs**
 - Monitor process delays?
 - Explicit algorithm
 - Similar to banker's algorithm
- **Overhead and delay**
 - Monitoring and algorithm

Recovery

- **Terminate process(es)?**
- **Preempt resource and notify**
 - Victim choice?
 - Rollback to prior consistent state?
- **Starvation**
 - What if repeated preemption?

⋮
⋮
Deadlock Summary

• **Available theory**

- Prevention, detection, avoidance

• **Seldom used in practice**

- Low probability, high overhead?
- Common: abort process/transaction
- May be different in the future
 - More intense resource sharing?
