Deadlock

Deadlock

- In Multiprogramming environment, processes compete for finite number of resources.
- If resource requested by process is not available, process enters waiting state.
- Sometimes, a waiting process is not able to change the state, because resources it has requested are held by other waiting process --- **Deadlock**.

- Example
  – System has 2 tape drives.
  – P1 and P2 each hold one tape drive and each needs another one.

System Model

- Resources are partitioned into several types:
  – CPU cycles, Memory Space, I/O devices.

- Each consisting of some number of identical instances.
  – Ex. – System having 2 CPU or 5 Printers

- Process utilize resource in following sequence :
  1. Request
  2. Use
  3. Release

Deadlock – Necessary Conditions

- Deadlock can arise if four conditions hold simultaneously.

1. Mutual Exclusion
   - At least one resource must be held in a non-sharable mode.
2. Hold and Wait
   - A process holding at least one resource is waiting to acquire additional resources held by other processes.
3. No Preemption
   - A resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. Circular Wait
   - There exists a set {P0, P1, …, Pn} of waiting processes such that P0 is waiting for a resource held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

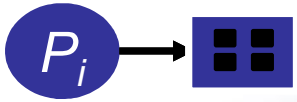
Resource-Allocation Graph

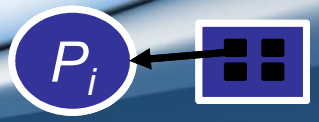
- Consist of set of vertices V & a set of edges E.
- V is partitioned into 2 different types of nodes:
  1. P = {P1, P2, …, Pn}, the set consisting of all the processes in the system.

  2. R = {R1, R2, …, Rm}, the set consisting of all resource types in the system.

- Request edge – directed edge P1 → Rj
- Assignment edge – directed edge Rj → Pi

Resource-Allocation Graph

- Process 
- Resource Type with 4 instances 
- Pi requests instance of Rj 

- Pi is holding an instance of Rj 

Example of a Resource Allocation Graph

P = { P1, P2, P3 }

R = {R1, R2, R3, R4 }

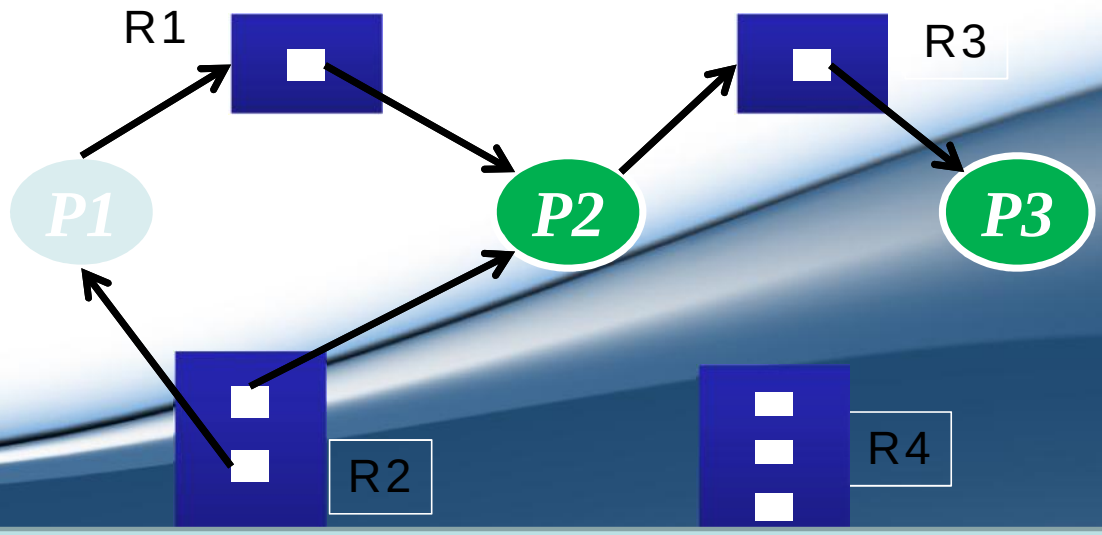
E = {P1→R1, P2→R3, R1→P2, R2→P2, R2→P1, R3→P3 }

R1 – 1 Instance

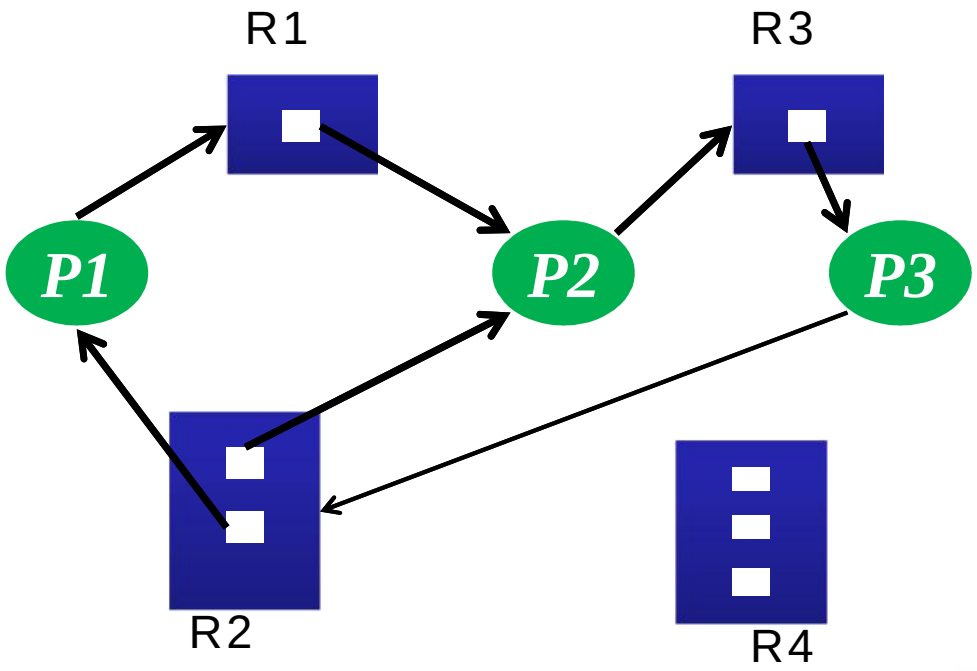
R2 - 2 Instances

R3 - 1 Instance

R4 – 3 Instance



Resource Allocation Graph With A Deadlock



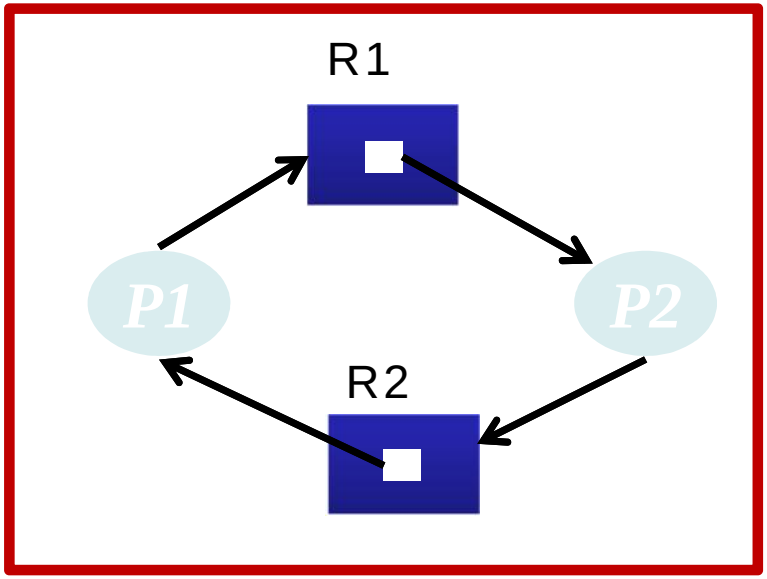
- For example in the fig. on the left two minimal cycles exist in the system
  - P1 →R1 →P2→ R3 → P3 → R2 → P1
  - P2 → R3 → P3 → R2 → P2
- Process P1, P2 and P3 are deadlocked

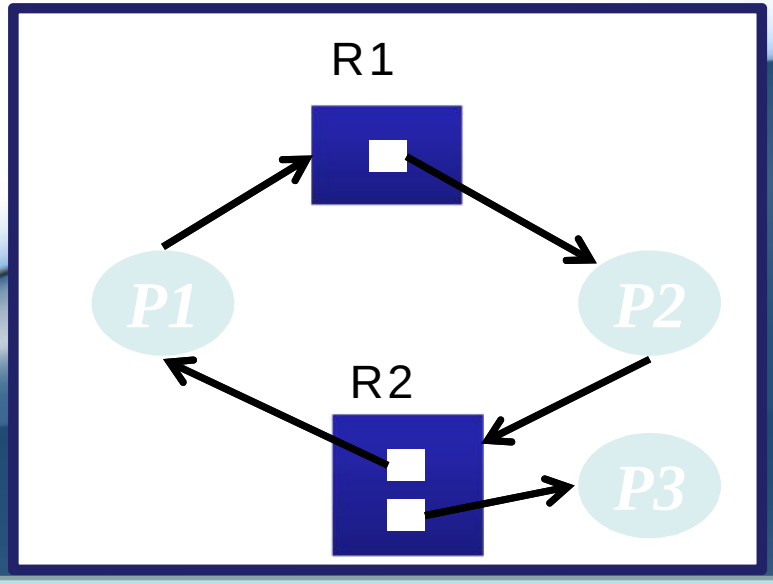
P_2 is waiting for P_3 to release R_3, P_3 is waiting for P_1 or P_2 to release R_2 and P_1 is waiting for P_2 to release R_1

- If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred and each process in the cycle are deadlocked

- If each resource type has several instances then a cycle in the graph does not necessarily imply that a deadlock has occurred

- Hence cycle in graph is a necessary but not a sufficient condition for the existence of deadlock





Basic Facts

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Deadlock Prevention

- Prevent any one of the necessary conditions from occurring.
- Mutual Exclusion
  - Not required for sharable resources – several processes can read a file.
  - Some resources are intrinsically non sharable.

Deadlock Prevention

- Hold and Wait
  - Must guarantee that whenever a process requests a resource, it does not hold any other resources.
  - 2 approaches
    1. Process requests all required resources at the beginning & starts execution only after getting all the required resources
    2. Allow process to request resource only when it has none.
  - 2 Problems :
    - Low resource utilization
    - Starvation possible

Deadlock Prevention

- 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 released.
  - 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.
  - Alternatively, if requested resources are being held by another waiting process, we preempt the desired resources from the waiting process and allocate them to requesting process.

Deadlock Prevention

- Circular wait
  - Number the resources & make sure that each process request the resource in increasing order of numbers.
  - Ex. We have 3 resources tape drive, disk drive & printer numbered 1,2 & 6 respectively. If a process wants to access tape drive & printer then it should first request tape drive & then printer.
  - Once a process has some resources assigned to it, it can acquire a new resource only if number of all its acquired resources is less than the number of newly requested resource.
  - Ex. If process wants to access disk drive(2) but it already has a printer allotted to it, then it can not acquire disk drive until it releases the printer(6).

Deadlock Avoidance

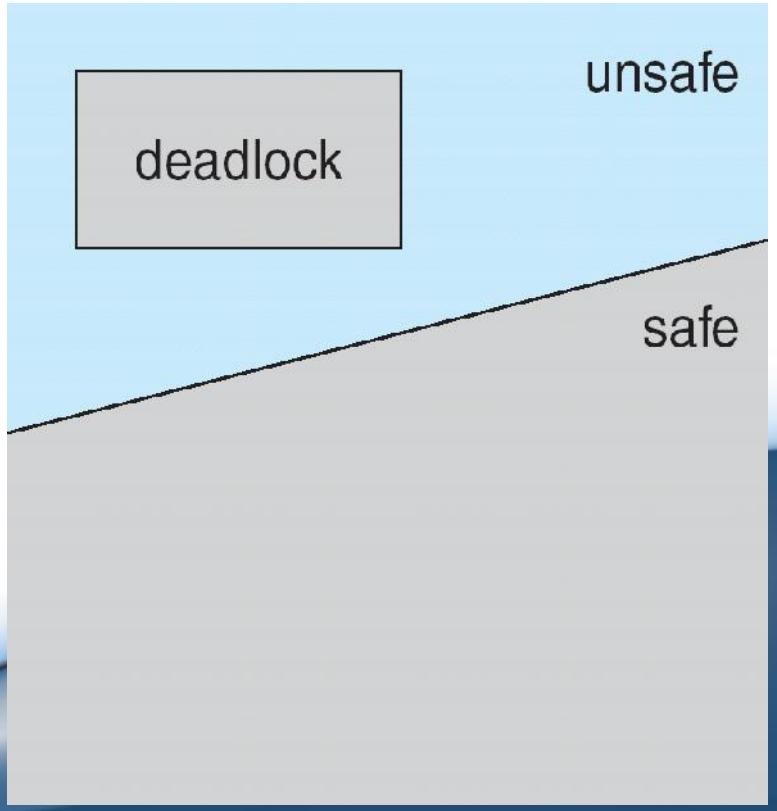
- Deadlock Prevention
  - Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design).
  - The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.
  - Resource allocation strategy for deadlock prevention is conservative, it under commits the resources.
- Deadlock Avoidance
  - The system dynamically considers every request and decides whether it is safe to grant it at this point
  - The system requires additional apriori information regarding the overall potential use of each resource for each process.

Deadlock Avoidance

- System consider the resources currently available, resources currently allocated to each process & future request & release of each process.
- Various algorithms are used
- Simplest & most useful model 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

- A state is safe if the system can allocate resources to each process in some order & still avoid a deadlock.
- Safe sequence – sequence of processes <P1, P2…Pn> for current allocation state if, for each Pi, resources requests that Pi makes can be satisfied by currently available resources plus resources held by all Pj, j<i.
- A system is in safe state only if there exists such a safe sequence.
- An unsafe state may lead to deadlock state.



Consider a system with 12 magnetic tape drives & 3 processes.

	Maximum Needs	Current Allocation (time t0)	Remaining Need
P0	10	5	5
P1	4	2	2
P2	9	2	7

- Available no. of resources at t0 = 3 [12-(5+2+2)]
- Sequence <P1, P0, P2> satisfies safety condition.

	Maximum Needs	Current Allocation (time t0)	Remaining Need
P0	10	5	5
P1	4	2	2
P2	9	2	7

	Maximum Needs	Current Allocation (time t1)	Remaining Need
P0	10	5	5
P1	4	2	2
P2	9	3	6

Available = 2

- System can enter into unsafe state if at time t1, P2 request & is allocated 1 more tape drive.
- Whenever a process request a resource that is currently available, the system must decide whether the resource can be allocated immediately or it must wait.
- The request must granted only if allocation leaves the system in safe state.

Deadlock Avoidance Algorithms

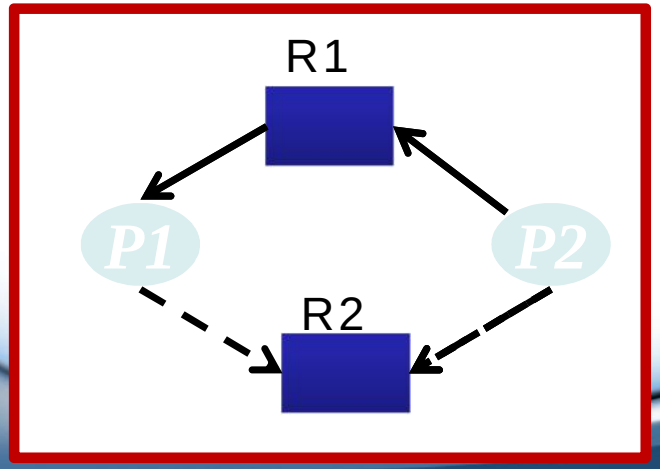
- Resource Allocation Graph Algorithm
- Banker’s Algorithm

Resource Allocation Graph Algorithm

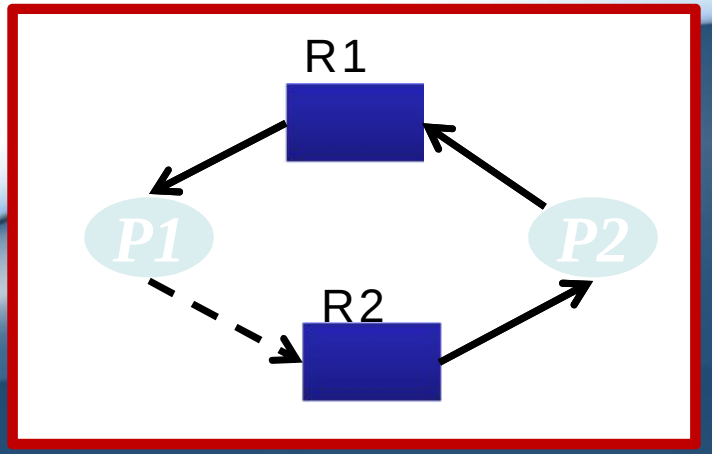
- Claim edge: Pi - → Rj indicates that process Pi may request resource Rj at some time in the future.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by process, assignment edge reconverts to claim edge.
- Resources must be claimed a priori in the system.
- Before process Pi starts executing, all its claim edges must appear in the graph
- Resource Allocation graph is not applicable to a resource allocation system with multiple instances of each resource type.

Resource-Allocation Graph For Deadlock Avoidance

- Request can be granted only if converting request edge Pi → Rj to an assignment edge Rj → Pi does not result into formation of cycle.
- If no cycle exist , then allocation of the resource will leave the system in a safe state.
- If cycle found, then allocation will put the system in unsafe state.



Though R2 is free but if allocated to P2, forms a cycle in graph.



Banker's Algorithm

- When a new process enters a system, it must declare the maximum no. of instances of each resource type it may need.
- This maximum need < total no. of resources in the system.
- When a process requests a resource , the system must determine whether the allocation of these resources will leave the system in a safe state, If it will ,the resources are allocated ; otherwise, 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

- n = number of processes
- m = number of resources types.
- Define 4 matrices – Available, Max, Allocation, Need
- Available
  - Vector of length m indicates number of available resources of each type.
  - If available [j] = k, there are k instances of resource type Rj available.
- Max
  - n x m matrix defines max. demand of each process.
  - If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj.

Banker's Algorithm

- Allocation
  - n x m matrix defines no. of resource of each type currently allocated to process.
  - If Allocation[i,j] = k then Pi is currently allocated k instances of Rj.

- Need
  - n x m matrix indicates remaining resource need of each process
  - If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task.
  - Need [i, j] = Max[i, j] – Allocation [i, j]

Safety Algorithm

Used to find out whether or not a system is in safe state.

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   Work := Available

   Finish [i] = false for i = 1,2,3, …, n.
2. Find an i such that both: *// find a process which can finish its work now*

   (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

Used to determine whether request can be safely granted.

- Let $Request_i$ = 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 Pi.
- If unsafe $\Rightarrow$ Pi must wait, and the old resource-allocation state is restored

Banker's Algorithm Example

– 5 processes P0 through P4; 3 resource types A (10 instances), B (5 instances) and C (7 instances).

- Snapshot at time T0:

Processes	**Allocation**			**Max**			**Available**		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

a) What is the content of matrix need ?
b) Is the system in safe state ? Give the sequence.
c) If request from P1 arrives for (1,0,2), can it be granted immediately?

Example (Need Matrix)

a) The content of the matrix Need is defined to be Max – Allocation.

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

Example (Safe State?)

- Look for a process whose need of resource <=available resources.
- Assume that process runs to completion. Mark it as terminated and add all its assigned resources to available.
- If no such process exists and some non terminated processes exist, then state is unsafe
- Repeat step 1-3. If all processes are marked terminated then system is in safe state, else unsafe.

Example (Safe State?)

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

Available		
A (3)	B (3)	C (2)

+

Run P1 to completion	**5**	**3**	**2**
Run P3 to completion	**7**	**4**	**3**
Run P4 to completion	**7**	**4**	**5**
Run P0 to completion	**7**	**5**	**5**
Run P2 to completion	**10**	**5**	**7**

< P1, P3, P4, P0, P2> satisfies safety criteria.

Example (Cont.): P1 request (1,0,2)

- Check that Request ≤ Need [(1,0,2) ≤ (1,2,2)] → true
- Check that Request ≤ Available [(1,0,2) ≤ (3,3,2)] →true.
- Pretend that the request has been fulfilled.

Processes	**Allocation**			**Need (Max-Allocation)**			**Available**		
	A	**B**	**C**	**A**	**B**	**C**	A	B	C
P0	0	1	0	7	4	3	~~3~~ 2	3	~~2~~ 0
P1	~~2~~ 3	0	~~0~~ 2	~~1~~ 0	2	~~2~~ 0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

- Executing safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement.

- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?

Question 1

Processes	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	2	1	6	4	2	4	2	4
P1	0	0	1	2	2	1			
P2	2	1	0	3	2	1			
P3	2	0	0	6	0	3			
P4	3	1	1	4	2	2			
P5	1	1	1	2	2	2			

a) What is the content of matrix need ?
b) Is the system in safe state ? Give the sequence.
c) If request from P0 arrives for (0, 1, 0), can it be granted immediately?

Question 1 (Need Matrix)

a) The content of the matrix Need is defined to be Max – Allocation.

Processes	Allocation			Max			Need (Max-Allocation)		
	A	B	C	A	B	C	A	B	C
P0	0	2	1	6	4	2	6	2	1
P1	0	0	1	2	2	1	2	2	0
P2	2	1	0	3	2	1	1	1	1
P3	2	0	0	6	0	3	4	0	3
P4	3	1	1	4	2	2	1	1	1
P5	1	1	1	2	2	2	1	1	1

Question1(Safe State?)

Processes	Allocation			Max			Need (Max-Allocation)		
	A	B	C	A	B	C	A	B	C
P0	0	2	1	6	4	2	6	2	1
P1	0	0	1	2	2	1	2	2	0
P2	2	1	0	3	2	1	1	1	1
P3	2	0	0	6	0	3	4	0	3
P4	3	1	1	4	2	2	1	1	1
P5	1	1	1	2	2	2	1	1	1

Available		
A (4)	B (2)	C (4)

+

Run P1 to completion	4	2	5
Run P2 to	6	3	5
Run P3 to	8	3	5
Run P4 to	11	4	6
Run P5 to completion	12	5	7
Run P0 to	12	7	8

< P1, P2, P3, P4, P5, P0> satisfies safety criteria.

Question1 (Cont.): P0 request (0,1,0)

- Check that Request ≤ Need [(0,1,0) ≤ (6, 2, 1)] → true
- Check that Request ≤ Available [(0,1,0) ≤ (4, 2, 4)] →true.
- Pretend that the request has been fulfilled.

Available		
A (4)	B (1)	C (4)

Executing safety algorithm shows that

- sequence <P2, P3, P4, P5,P0, P1>
- satisfies safety requirement.
- Hence we can immediately
- grant the request of P1.

Processes	Allocation			Max			Available			Need (Max-Allocation)		
	A	B	C	A	B	C	A	B	C	**A**	**B**	**C**
P0	0	~~2~~ 3	1	6	4	2	4	~~2~~ 1	4	6	~~2~~ 1	1
P1	0	0	1	2	2	1				2	2	0
P2	2	1	0	3	2	1				1	1	1
P3	2	0	0	6	0	3				4	0	3
P4	3	1	1	4	2	2				1	1	1
P5	1	1	1	2	2	2				1	1	1

Question 2

Processes	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	2	1	0	2	1	1	0	2	2
P2	1	2	0	2	5	2			
P3	0	1	1	1	4	2			
P4	0	0	1	2	0	1			

a) What is the content of matrix need ?
b) Is the system in safe state ? Give the sequence.
c) If request from P2 arrives for (1,0,0), can it be granted immediately?

Question 3

Processes	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	4	5	0	0	4	5	4	8	5	0
P1	3	0	0	0	4	10	8	0				
P2	3	6	8	7	5	6	8	9				
P3	0	9	6	5	0	9	8	5				
P4	0	0	4	7	0	9	8	9				

a) What is the content of matrix need ?
b) Is the system in safe state ? Give the sequence.
c) If request from P1 arrives for (0, 4, 2, 0), can it be granted immediately?

Question 3 (Need Matrix)

a) The content of the matrix Need is defined to be Max – Allocation.

Processes	**Allocation**				**Max**				**Need**			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	4	5	0	0	4	5	0	0	0	0
P1	3	0	0	0	4	10	8	0	1	10	8	0
P2	3	6	8	7	5	6	8	9	2	0	0	2
P3	0	9	6	5	0	9	8	5	0	0	2	0
P4	0	0	4	7	0	9	8	9	0	9	4	2

Question3 (Safe State?)

Processes	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	4	5	0	0	4	5	0	0	0	0
P1	3	0	0	0	4	10	8	0	1	10	8	0
P2	3	6	8	7	5	6	8	9	2	0	0	2
P3	0	9	6	5	0	9	8	5	0	0	2	0
P4	0	0	4	7	0	9	8	9	0	9	4	2

Available			
A (4)	B (8)	C (5)	D(0)

+

P0 already complete	**4**	**8**	**9**	**5**
Run P2 to completion	**7**	**14**	**17**	**12**
Run P3 to completion	**7**	**23**	**23**	**17**
Run P4 to completion	**7**	**23**	**27**	**24**
Run P1 to	**10**	**23**	**27**	**24**

< P0, P2, P3, P4, P1> satisfies safety criteria.

Question3 (Cont.): P1 request (0,4,2,0)

- Check that Request ≤ Need [(0,4,2,0) ≤ (1,10,8,0)] → true
- Check that Request ≤ Available [(0,4,2,0) ≤ (4,8,5,0)] →true.

Available			
A (4)	B (8)	C (5)	D(0)
4	**4**	**3**	**0**

Processes	**Allocation**				**Max**				**Need**			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	4	5	0	0	4	5	0	0	0	0
P1	3	~~0~~ 4	~~0~~ 2	0	4	10	8	0	1	~~10~~ 6	~~8~~ 6	0
P2	3	6	8	7	5	6	8	9	2	0	0	2
P3	0	9	6	5	0	9	8	5	0	0	2	0
P4	0	0	4	7	0	9	8	9	0	9	4	2

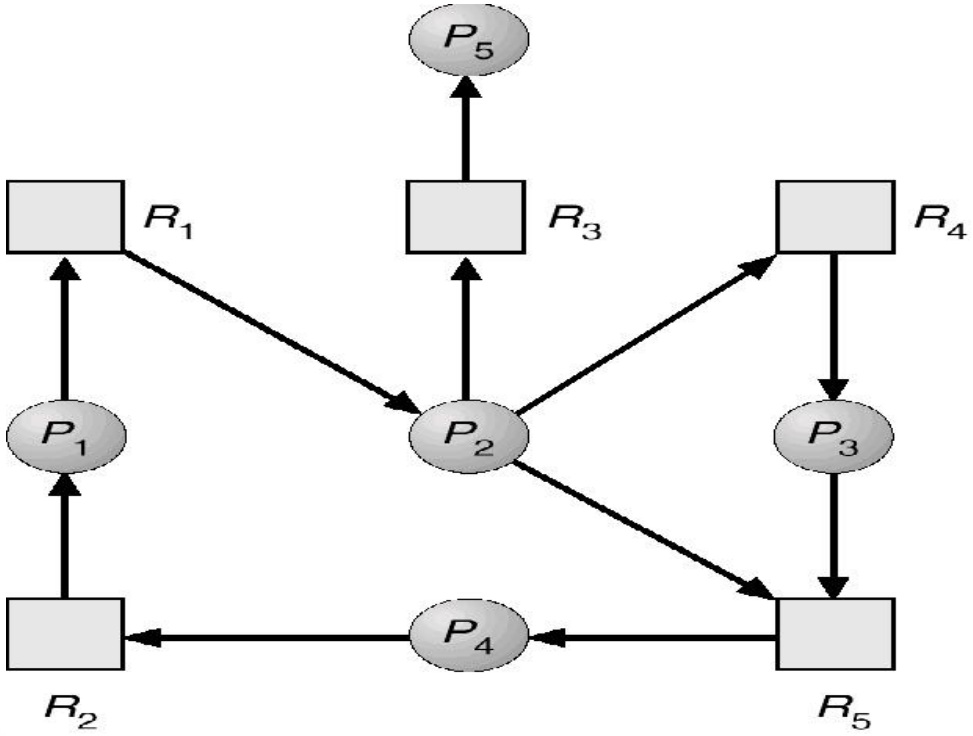
Deadlock Detection

- Algorithm that examine the state of the system to determine whether a deadlock has occurred.
- Algorithm to recover from deadlock

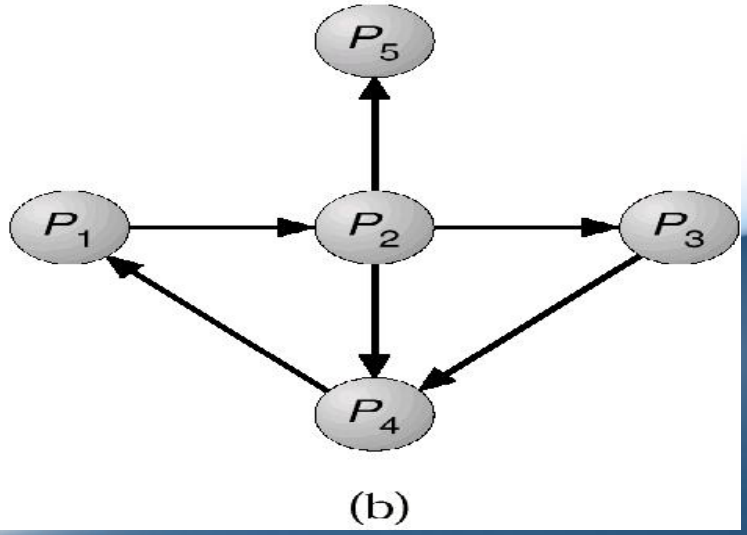
Single Instance of Each Resource Type

- Maintain wait-for graph
  - Nodes are processes.
  - Pi → Pj if Pi is waiting for Pj to release a resource that Pi needs.
- Cycle in wait-for graph → deadlock exist.
- Periodically invoke an algorithm that searches for a cycle in the graph.

Resource-Allocation Graph And Wait-for Graph



Resource-Allocation Graph



Corresponding Wait-for graph

Several Instances of a Resource Type

- Available: A vector of length m indicates the number of available resources of each type.
- Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.
- Request: An n x 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

1. Let Work and Finish be vectors of length m and n, respectively Initialize:

   (a) Work := Available

   (b) For i = 1,2, …, n, if $Allocation_i \neq 0$, then
   Finish[i] := false; otherwise, Finish[i] := true.

2. Find an index i such that both:

   (a) Finish[i] == false (b) $Request_i \leq Work$

   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] == false, for some i, $0 \leq i < n$, then the system is in deadlock state.

Example of Detection Algorithm

- Five processes P0 through P4; three resource types A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T0:

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	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			

Sequence <P0, P2, P3, P4, P1> will result in Finish[i] = true for all i. System is not deadlocked.

Example (Cont.)

- P2 requests an additional instance of type C.

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

- We can reclaim resources held by process P0, but insufficient resources to fulfil other processes requests. Deadlock exists, consisting of processes P1, P2, P3 & P4.

Recovery from Deadlock

- When deadlock is detected- different options are used to recover from deadlock :
  - Manual recovery – inform operator about deadlock occurrence & then operator recovers from deadlock manually.
  - Automatic recovery - 2 options for breaking deadlock
    1. Abort 1 or more processes to break circular wait
    2. Preempt some resources from one or more of the deadlocked processes.

Recovery from Deadlock: Process Termination

- One of two methods are used while aborting processes :
  1. Abort all deadlocked processes
  2. Abort one process at a time until the deadlock cycle is eliminated.

- In which order should we choose to abort depends on following factor :
  - 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?

Recovery from Deadlock ： Resource Preemption

- Preempt some resource from processes and give these resources to other process until the deadlock cycle is broken.
- 3 issues need to be addressed:-
  - Selecting a victim – minimize cost.
  - Rollback – rollback process to some safe state and then restart process from that state. Another option – total rollback
  - Starvation – same process may always be picked as victim

Reusable Resources

- Fixed number of units; Neither created nor destroyed
- Used by one process at a time and not depleted by that use
- Processes obtain resources that they later release for reuse by other processes
- Unit is either free or allocated; no sharing
- Process requests, acquires, releases units
- Deadlock occurs if each process holds one resource and requests the other
- Ex. Processors, I/O channels, main and secondary memory, files, databases, and semaphores

Consumable Resources

- Created (produced) by a process
- Destroyed (consumed) by a process
- Number of units varies at runtime
- Ex. Hardware interrupts, Unix signals, messages, and information in I/O buffers.

Questions

- What is deadlock? What are necessary conditions for occurrence of deadlock also mention methods of handling deadlock?
- Explain various methods of preventing deadlocks.
- What is deadlock? Explain bankers algorithm & explain how it can be used to avoid a deadlock.
- Explain reusable & consumable resources with help of an examples. What is deadlock? What is difference between deadlock avoidance, detection, prevention?
- Short note – resource allocation graph

Questions

Using banker's algorithm answers the following –

1. What is context of matrix need?
2. Is the system in safe state? Give sequence.
3. If a request from process P1 arrives for (1,0,2) can request be granted immediately?

Processes	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P0	0	1	1	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

THANK YOU