Unit - 3

Part - 1

Deadlocks

By

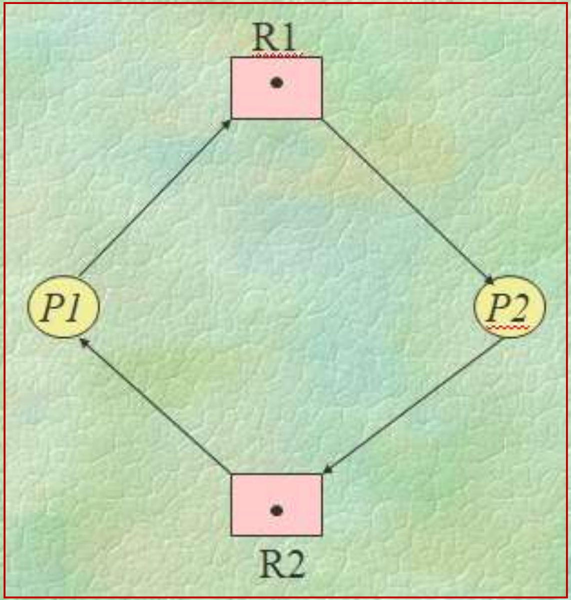
Prof. RB Kallam

Topics to be covered

- Deadlocks
- System Model
- Deadlocks Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlock :

- In multiprogramming environment several processes may compete for the same resource.
- If a process request for resources and if the resources are not available at that time then the process enters into a wait state.
- It may happen that waiting process will never again change the state, because the resources they have requested are held with other waiting processes.
- This situation is called a **deadlock**.
- **Deadlock** is defined as the permanent blocking of a set of processes that compete for system resources.



Resource allocation graph with a dead lock

System Model:

- A system consists of a finite number of resources to be distributed among a number of competing processes.
- The resources are partitioned into several types, each consisting of some number of identical instances.
- Reusable: A resource can be safely used by one process at a time and is not consumed by that use. Processes obtain resources that they later release for reuse by others (processors, memory, files, devices, databases, and semaphores).

- Consumable: these can be created and destroyed. When a resource is acquired by a process, it is consumed (interrupts, signals, messages, etc).
- A preemptable resource: is one that can be taken away from the process owning it with no ill effects. Memory (also CPU) is an example of a preemptable resource.
- A nonpreemptable resource, in contrast, is one that cannot be taken away from its current owner without causing the computation to fail (printer, CD-R(W)floppy disk).

- **A process must request a resource before using it and must release the resource after using it.**
  - A process may request as many resources as it requires to carry out its designated task.
  - Obviously, the number of resources requested may not exceed the total number of resources available in the system.
- **Under the normal mode of operation, a process may utilize a resource in only the following sequence:**

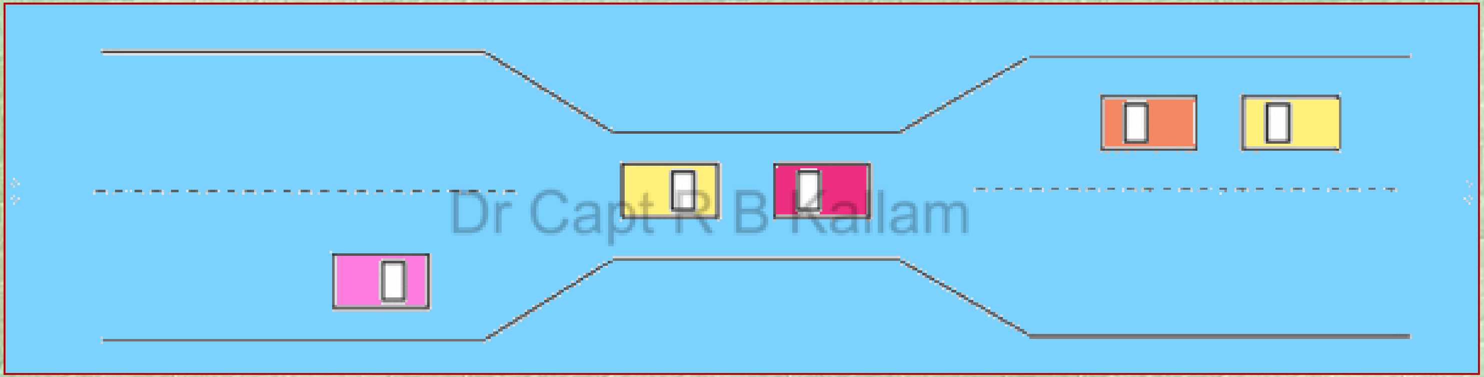
  Request

  Use

  Release
- The request and release of resources are system calls. Examples are the request() and release() device, open() and close() file, and allocate() and free() memory system calls.

Deadlock Characterization:

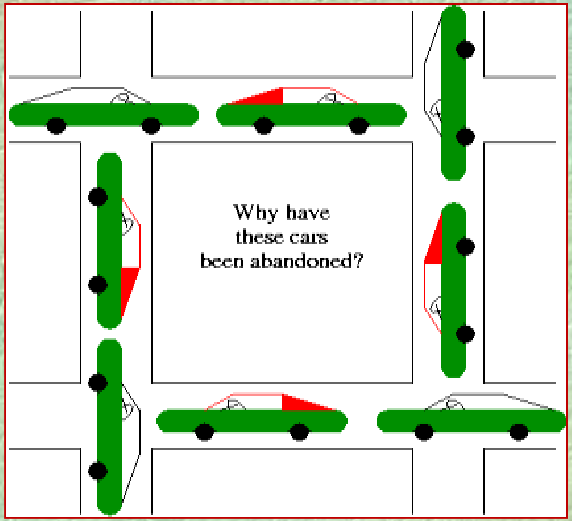
- **Necessary Conditions:**
  - **Mutual exclusion:** **At least one resource must be held in a non sharable mode; that is only one process at a time can use the resource.**

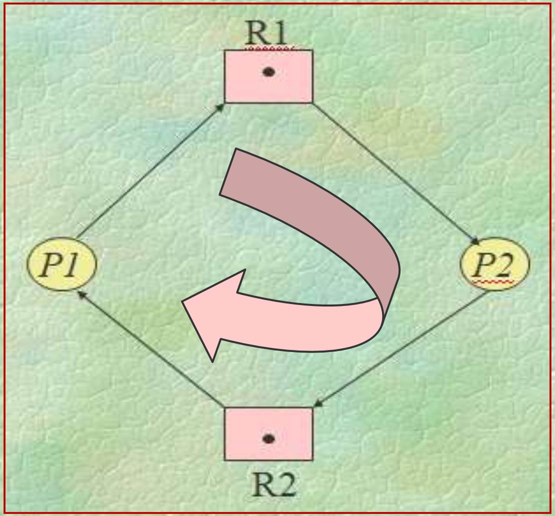


  - **Hold and wait:** **There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.**

- **No preemption:** Resources cannot be preempted; that is a resource can be released only voluntarily by the process holding it, after that process has completed its task.

- **Circular wait:** There must exist a set {P0, P1,.., Pn } of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2,…. Pn, and Pn is waiting for a resource that is held by P0.





Resource Allocation graph:

- Deadlocks can be described more precisely in terms of a **directed graph** called a **system resource allocation graph.**

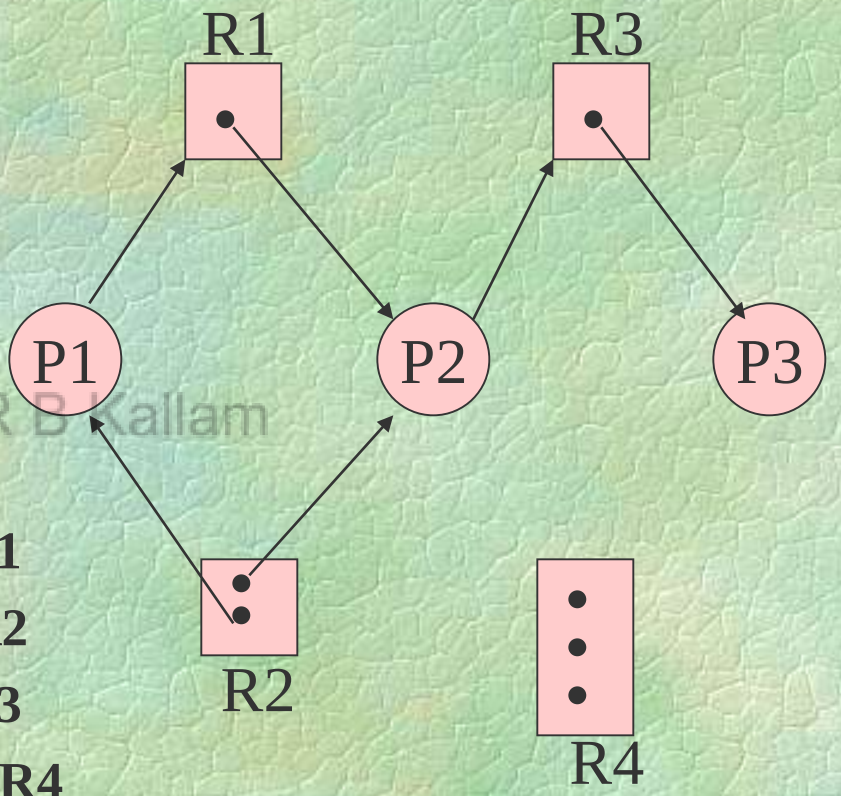
- This graph consists of a set of vertices V and a set of edges E.

- The set of vertices V is partitioned into two different types of nodes P = {P_1, P_2,…., P_n }, the set consisting of all the active processes in the system, and R= { R_1, R_2,…, R_m}, the set consisting of all resource types in the system.

- A directed edge $P_i \longrightarrow R_j$ is called a **request edge**; and a directed edge $R_j \longrightarrow P_i$ is called an ***assignment** edge.*

The resource allocation graph shown below have the following situations:

- **The sets P,R,and E:**
  - **P={P1,P2,P3}**
  - **R={ R1,R2,R3,R4}**
  - **E={P1→R1, P2→R3, R1→P2, R2→P2, R2→P1,R3→P3}**
- **Resource instances:**
  - **One instance of resource type R1**
  - **Two instance of resource type R2**
  - **One instance of resource type R3**
  - **Three instance of resource type R4**

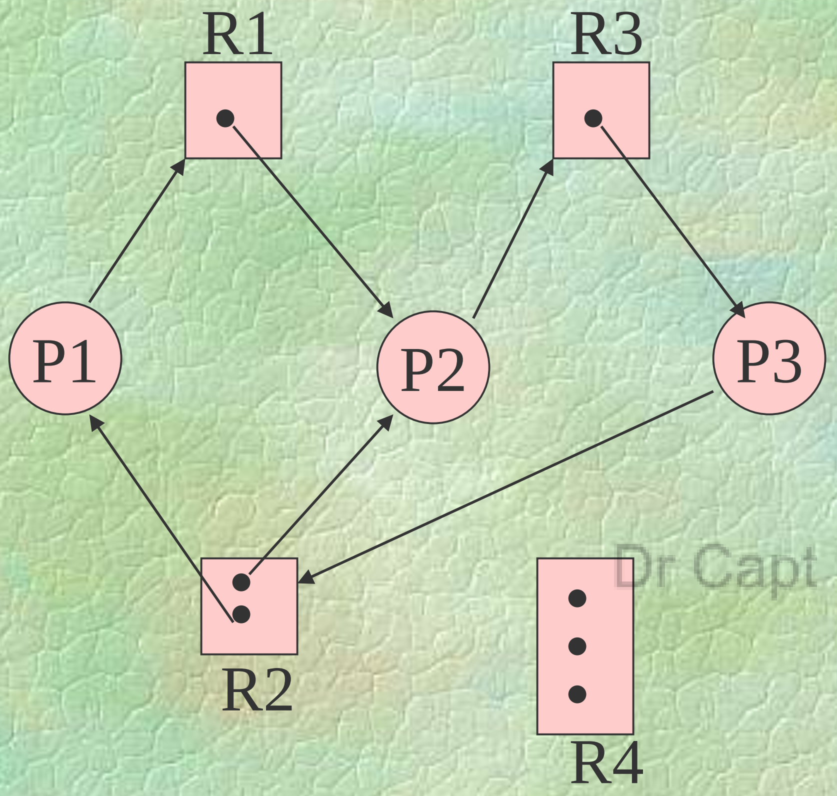


Resource allocation graph

Process states:

- Process P1 is holding an instance of resource type R2, and is waiting for an instance of resource type R1.

- Process P2 is holding an instance of resource type R1and R2, and is waiting for an instance of resource type R3.
- Process P3 is holding an instance of R3.



$P1 \rightarrow R1 \rightarrow P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P1$

$P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P2$

Resource allocation graph with a dead lock

Methods for handling deadlocks:

- Principally there are three different methods
  - We can use a protocol to ensure that the system will never enter a deadlock state.
  - We can allow the system to enter a deadlock state and then recover.
  - We can ignore the problem all together, and pretend that deadlock never occur in the system.

Deadlock Prevention:

- Deadlock prevention is same as take the preventive methods before attacking the deadlock.
- For a deadlock to occur, each of the four necessary conditions must hold., by ensuring that at least one of these conditions can’t hold, we can prevent the occurrence of the deadlock.

- Mutual exclusion:
  - Mutual exclusion means only one process can use the resource at a time, it means resources are not sharable by the number of processes at a time.
  - We can deny this condition with simple protocol i.e. “Convert the all non sharable resources to sharable resource". So this condition is not satisfied by the deadlock, hence we can prevent the deadlock.
  - But it is not possible all the time.

Hold and wait:

- It means each and every process in the dead lock state, must hold at least one resource and wait for at least one resource.

- We can deny this condition with a small protocol. That is " A process request the resources only when the process has none"

- But it is not possible all the time.

No preemption:

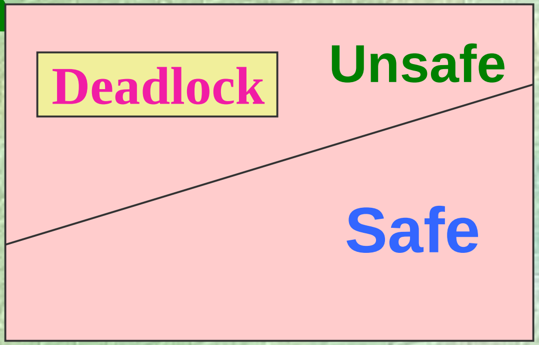
- **It means resources are not released in the middle of the processing of a resource.**
- **To ensure that this condition does not hold, we can use the following protocol.**
  - **If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are preempted.**
  - **That is these resources are released and added to the list of resources for which the process is waiting.**
  - **The process will be restarted only when it regains its old resources, as well as the new once that it is requesting**

Circular wait:

- One way to ensure that the circular wait condition never holds is to impose a total ordering of all resource types, and to require that each process request resources in an increasing order.
- For this we have two protocols:
  - **Whenever " a processes request an instance of resource type Rj, it has released any resource Ri such that F(Ri)>F(Rj).**
  - **A process can initially request any number of instances of a resource type, say Ri. After that, the process can request the instances of resource type Rj, if and only if**
  - **F(Rj)>F(Ri)".**
  - **If these two protocols are used, then the circular wait condition cannot hold.**

Deadlock avoidance:

- It is a dynamic approach to escape from deadlock .
- With this, if a process request for resources the avoidance algorithm checks before the allocation of resources about the state of the system.
- If the state is safe, the system allocates the resources to the requesting process otherwise do not allocates the resources.
- So taking care before the allocation is said to be deadlock avoidance.



Safe, unsafe, and deadlock state space

Banker's algorithm:

- It is a dead lock avoidance algorithm, the name was chosen because the bank never allocates more than available cash.
- Available:
  - A Vector length m indicates the number of available resources of each type.
  - If *Available[j] = k,* there are k instances of resource type Rj available.
- **Max:**
  - An n x m matrix defines the maximum demand of each process.
  - If *Max [i, j] =k,* then Pi may request at most k instances of resource type Rj

- **Allocation:**
  - **An n x m matrix defines the number of resources of each type currently allocated to each process.**
  - If ***Allocation [i , j] = k, then process Pi is currently allocated k instances of resource type Rj.***
- **Need:**

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

Safety Algorithm:

- 1. Let Work and Finish be vectors of length m and n, respectively. Initialize work:= Available. and Finish [i]:=false; For i= 0,1,2,..n-1.
- 2. Find an i such that both
  - Finish[i] == false
  - Need i <= **Work.** If
    If no such exist, go to step4.
- 3. Work:= Work + Allocation $_{\mathbf{i}}$
  Finish[i] := true
  go to step 2
- 4. If Finish [i] == True, for all i, then the system is in safe state.

- *Note:* This requires an order of ***mxn***^{2} operations to detect whether the system is in safe state.

Algorithm – Example:

- Consider a system with 5 processes P0,P1,P2,P3 and P4, and 3 resources A,B, C. Resource type A has 10 instances, Resources type B has 5 instances and Resources C has 7 instances. Suppose that, at time, the following snap shot of the system has been taken.

Calculate :

- **Available matrix**
- **Need matrix**
- **And find whether the system is in the safe state or not.**

	Allocation			MAX		
	A	B	C	A	B	C
P0	0	1	0	7	5	3
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

Process ID	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
	7	**2**	**5**							

	A	**B**	**C**
Total	10	5	7
Allocated	-7	-2	-5
Available	**3**	**3**	**2**

Execution sequence for the system to be in safe state:
P1, P3,P4,P0,P2

	A	**B**	**C**
Available (X)	3	3	2
Resource allocated to P1(Y)	2	0	0
Available (X+Y)	**5**	**3**	**2**
Resource allocated to P3	2	1	1
Available	**7**	**4**	**3**
Resource allocated to P4	0	0	2
Available	**7**	**4**	**5**
Resource allocated to P0	0	1	0
Available	**7**	**5**	**5**
Resource allocated to P2	3	0	2
Available	**10**	**5**	**7**

Ex Q: Consider the following snapshot of a system:

Process ID	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					

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

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

Process ID	Allocation				Max					Need Matrix			
	A	B	C	D	A	B	C	D		A	B	C	D
P0	0	0	1	2	0	0	1	2		**0**	**0**	**0**	**0**
P1	1	0	0	0	1	7	5	0		0	7	5	0
P2	1	3	5	4	2	3	5	6		1	0	0	2
P3	0	6	3	2	0	6	5	2		0	0	2	0
P4	0	0	1	4	0	6	5	6		0	6	4	2
	2	9	10	12									

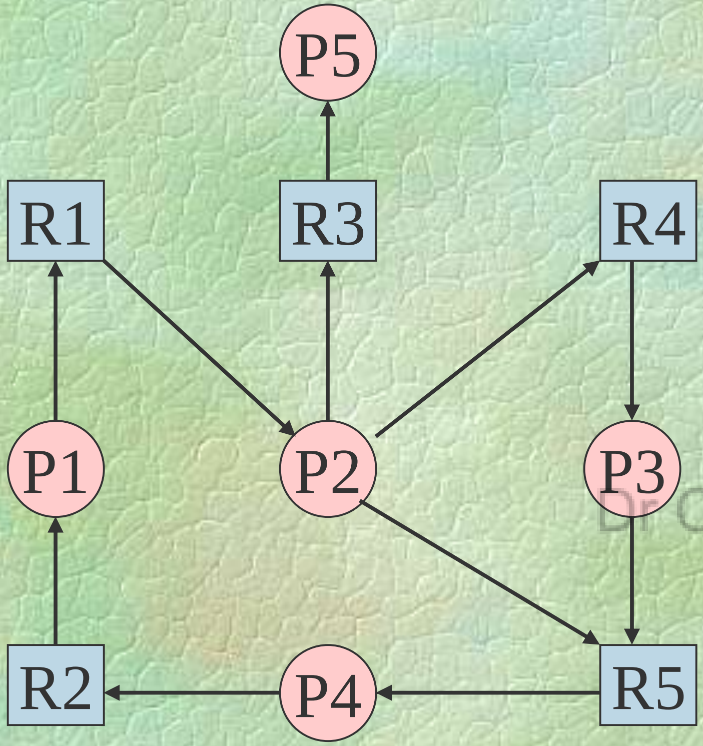
	A	**B**	**C**	**D**
Available	1	5	2	0
Allocated	2	9	10	12
Total Resources	**3**	**14**	**12**	**12**

	A	B	C	D
Available (X)	1	5	2	0
Resource allocated to P0(Y)	0	0	1	2
Available (X+Y)	**1**	**5**	**3**	**2**
Resource allocated to P2	1	3	5	4
Available	**2**	**8**	**8**	**6**
Resource allocated to P3	0	6	3	2
Available	**2**	**14**	**11**	**8**
Resource allocated to P4	0	0	1	4
Available	**2**	**14**	**12**	**12**
Resource allocated to P1	1	0	0	0
Available	**3**	**14**	**12**	**12**

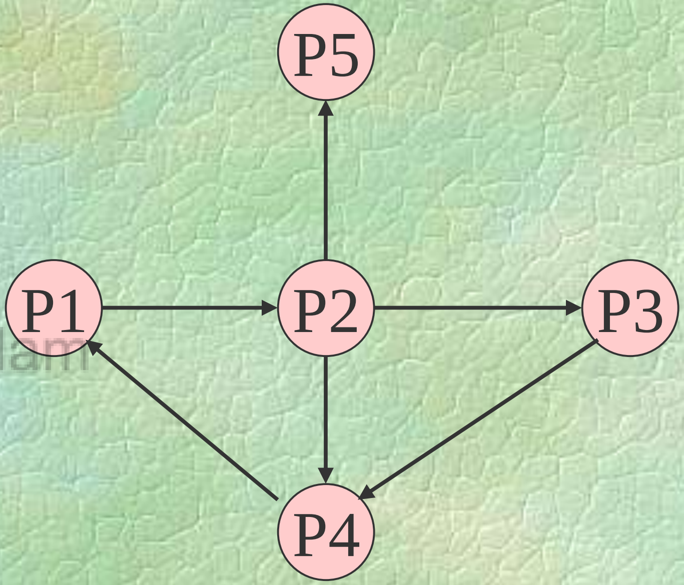
- Execution sequence: P0, P2,P3,P4 and P1
- System is in the safe state
- If a request from the process P1 arrives for (0,4,2,0), the request can be granted immediately.

Deadlock Detection:

- Single instance of each resource type:
  - **If all resources have a single instance, then we can define a Deadlock algorithm that uses a variant of the resource allocation graph, called a *wait - for* graph.**
  - **We obtain this graph from the resource allocation graph by removing the nodes of type resource and collapsing the edges.**
  - **More precisely, an edge from Pi to Pj in a wait - for graph implies that Pi is waiting for process Pj to release a resource that Pi needs.**
  - **A Deadlock exist in the system if and only if the wait - for graph contains a cycle.**
  - **To detect the deadlock, the system needs to maintain the wait - for graph and periodically to invoke an algorithm that searches for a cycle in the graph.**



Resource allocation graph



Corresponding wait - for graph

- Several instances of a resource type:
  - We need a deadlock detection algorithm with several time varying data structures that are similar to those used in banker's algorithm.
  - **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 Pi is requesting k more instances of resource type Rj.

Banker's algorithm for Deadlock detection

- 1. Let Work and Finish be vectors of length m and n, respectively. Initialize work:= Available. For i= 1,2,..n, if Allocation = 0, then Finish [i]:=false; otherwise, Finish[i]:= true.
- 2. Find an index i such that both
  - Finish[i] == false
  - Request i <= **Work.**

    If no such exist, go to step4.

- 3. Work:= Work + Allocation $_i$

  Finish[i] := true

  go to step 2
- 4. If Finish [i] = false, for some i, 1<= i <= n, then the system is in a deadlock state. Moreover, Finish [i]:=false, then process Pi is deadlocked.

- *Note:* This requires an order of ***mxn***^{2} operations to detect whether the system is in deadlocked state.

Q: Consider a system with 5 processes P0,P1,P2,P3 and P4, and 3 resources A,B, C. Resource type A has 7 instances, Resources type B has 2 instances and Resources C has 6 instances. Suppose that, at time T0, we have the following resource allocation state.

	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	

Answer the following questions:

a. What is the content of a matrix Max?

b. Is the system is in safe state?

c. If a request from process P2 arrives for (0,0,1) is the system is in safe state?

	Allocation	Request	Available	Max (Allocation+ Req)
	A B C	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0	0 1 0
P1	2 0 0	2 0 2		4 0 2
P2	3 0 3	0 0 0		3 0 3
P3	2 1 1	1 0 0		3 1 1
P4	0 0 2	0 0 2		0 0 4
	7 2 6			

	A	B	C
Available	0	0	0
Allocated	7	2	6
Total Resources	**7**	**2**	**6**

	A	B	C
Available (X)	0	0	0
Resource allocated to P0(Y)	0	1	0
Available (X+Y)	**0**	**1**	**0**
Resource allocated to P2	3	0	3
Available	**3**	**1**	**3**
Resource allocated to P3	2	1	1
Available	**5**	**2**	**4**
Resource allocated to P4	0	0	2
Available	**5**	**2**	**6**
Resource allocated to P1	2	0	0
Available	**7**	**2**	**6**

✓System is in the safe state and the execution sequence is P0, P2, P3, P4 and P1

✓If a request from process P2 arrives for (0,0,1); the system is not in safe state as the available is < request, i.e (0 1 0 < 0 0 1)

H/W Q: Consider a 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				

Answer the following questions using Bankers algorithm?

a. What is the content of a matrix Need?

b. Is the system is in safe state?

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

Recovery from Deadlock:

- Process Termination
  - Abort all deadlocked processes: This method clearly will break the deadlock cycle, but at great expense.
  - Abort one process at a time until the deadlock cycle is eliminated: This method incurs considerable overhead, since, after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

- Resource Preemption
  - Selecting a victim: we must determine a victim and preempt it's all resources and restart.
  - Rollback: We must roll back the resources of a process to some safe state and restart it from that state.
- Starvation problem: In a system where victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. As a result, this process never completes its designated task and leads to the starvation.