Chapter -3

Dead Lock

Date:
Page:

System Model: System model or structure consist of fixed number of resources to be circulated among some opposing processes. The resources are then partitioned into numerous types, each consisting of some specified quantity of identical instances. Memory space, CPU cycle, directories and files I/O devices, printers, etc are the examples of resource type.

Under the standard mode of operation, any process may use a resource in only the below mentioned sequence:

- Request:
When the request can't be approved immediately, then the requesting job must remain waited until it can obtain the resources.
- Use:
The process can run on the resources when possible.
- Release:
The process releases the resource when the process is completely executed or its time slice is finished.

Introduction to Dead lock:

→ Dead lock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming towards each other on same track and there is only one track. In, none of the trains can move once they are infront of each other.

Deadlock situation occurs in operating system when there are two or more processes holding some resources and waiting for resources held by other processes.

A diagram showing deadlock is shown below:-

→

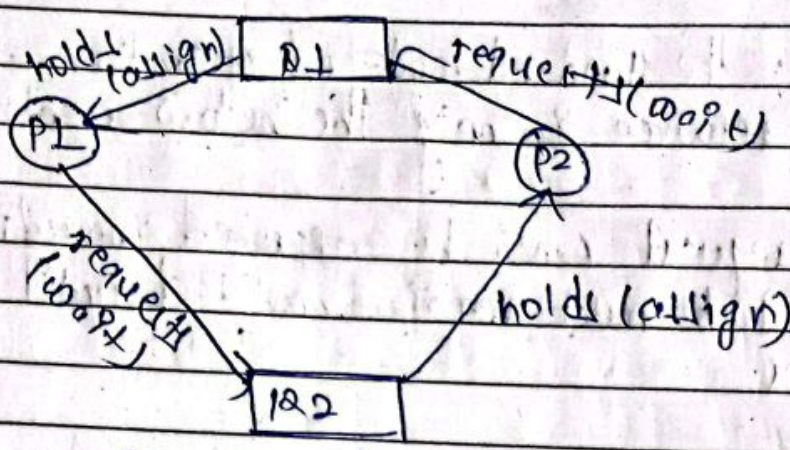


Fig: Deadlock

□ → represents Resource.

○ → represents process

In above diagram, process P1 is holding resource R1 and waiting for R2 which is acquired by process P2 and process P2 is waiting for resource R1 which is acquired by process P1. Now, neither of the process can proceed until other process releases a resources. So, the operating system doesn't know what action to take and at this point, only alternative is to abort (stop) one of the process.

Date:
Page:

System Resources → Resources come in two flavours in the system namely preemptable and non-preemptable.

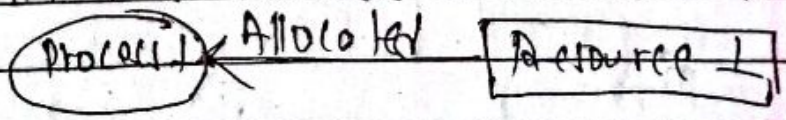
A preemptable resources is one that can be taken away from the process with no ill effects. For example Memory on the other hand, a non-preemptable resources is one that cannot be taken away from the process without causing ill effect. For example: CD resources are not preemptable at all arbitrary moment.

Deallocating resources can resolve deadlock that involve preemptable resources. But the deadlocks that involve non-preemptable resources are difficult to deal with.

Conditions / Criteria for deadlock

(a) Mutual exclusion (c) Non-preemptive condition
(b) Hold and wait (d) Circular wait

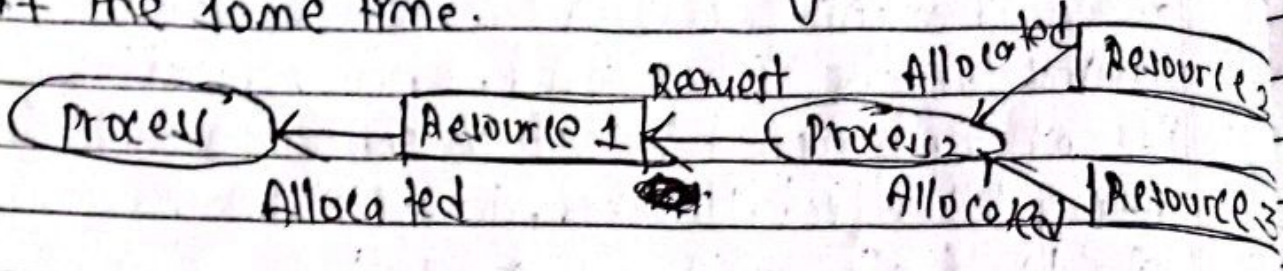
(a) Mutual Exclusion → At least one resources must be held in non-sharable mode for only one process at a time claims exclusive control of the resources. If another process request that resources the requesting process must be delayed until the resource has been released.



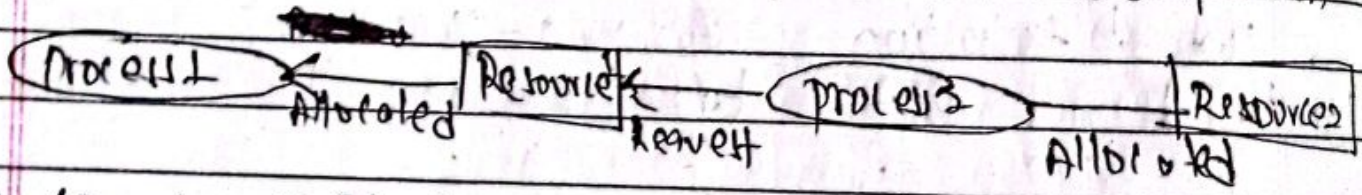
(b) Hold and wait → Requesting process hold already resources while waiting for requested resources, there must exist a process that is holding a resource already allocated

Date:
Page:

P1 P2 while waiting for additional resources that are currently being held by other process. The process waits for some resources while holding on another resources at the same time.



(c) No-preemptive condition → Resources already allocated to a process cannot be preempted in the non-preemptive approach. Resources cannot be removed from the process that are used for completion or released voluntarily by the process holding it. It means that the process once scheduled will be executed till the completion.



(d) Circular Wait: All the processes must wait for a resource in a cyclic manner where the last process waits for the resource held by the first process.

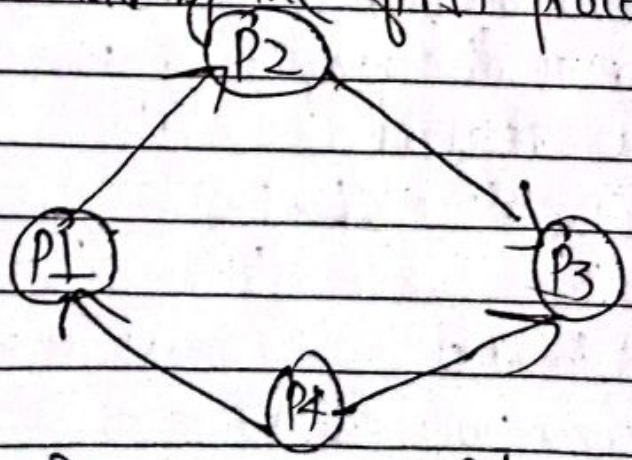


fig: Circular Wait

Date:
Page:

Methods for Handling Deadlock.

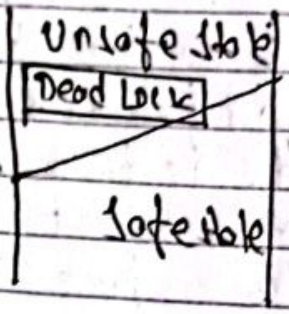
⇒ The methods that are used in order to handle the problem of deadlock are as follows :-

- Ignoring the deadlock (The Ostrich approach): According to this method, it is assumed that deadlock would never occur. This approach is used by many OS where they assumed that deadlock will never occur which means OS simply ignores the deadlock. It is also known as Ostrich approach because it defines to stick one's head in the sand and pretend that there is no problem". Thus, ignoring the deadlock method can be useful in many cases but it is not perfect approach in order to remove the deadlock from the operating system.

- Deadlock prevention ⇒ As we have discussed earlier that all four conditions: Mutual exclusion, Hold and wait, No preemption and circular wait if held by a system causes deadlock to occur. So, the main aim of deadlock prevention method is to violate any one condition among the four, because if any of the condition is violated then, the problem of deadlock will never occurs.

- Avoiding the deadlock ⇒ This method is used by the operating system in order to check whether the system is in safe state or in unsafe state. This method checks every step performed by Operating System. Any process continues its execution until the system is in safe

state. Once the system enters into an unsafe state, it has to take a step back. This enters into an unsafe state, the OS has to take a step back. This method differs from deadlock prevention which guarantees that deadlock cannot occur by denying one of the necessary conditions of deadlock. Banker's algorithm is an example of deadlock avoidance strategy.



Note:

Safe state → When a system can allocate the resources to the process in such a way so that they still avoid deadlock, then the state is called safe state.

Unsafe state → If a safe state doesn't exist, then the system is in unsafe state which will lead to deadlock.

- Deadlock detection and Recovery:

With this method, the deadlock is detected first by using some algorithm of the Resource Allocation Graph (RAG). This graph is mainly used to represent the allocations of various resources to different processes. Deadlock detection is the process of actually determining that a deadlock exists and identifying

the processes and resources involved in the deadlock. After the detection of deadlock, a number of methods can be used in order to recover from that deadlock. Some of the common recovery methods are given below:-

(i) For Resource:

(a) preempt the resource: We can snatch one of the resources from the owners of the resource (i.e. process) and give it to other process with the expectation that it will complete the execution and will release this resources sooner.

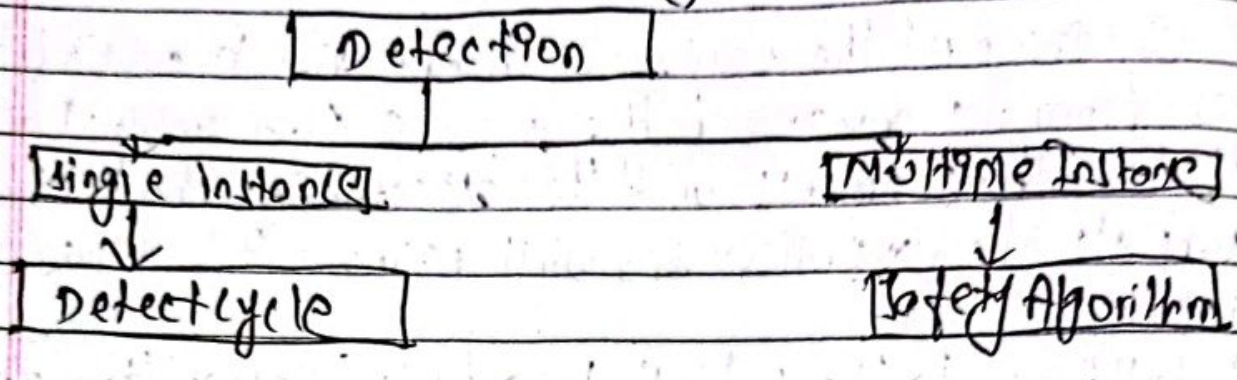
(b) Rollback to the safe state → system passes through various states to get into the deadlock state. The operating system can rollback the system to the previous state only for this purposes it needs to implement checkpoint at every step.

(ii) For process → (a) Kill a process: Killing a process can solve the problem of deadlock but the main concern is to decide which process to keep. Generally, it kills a process which has done least amount of work until now.

(b) Kill all the processes → This is not suggestable approach but can be implemented if the problem becomes very serious. Killing all the processes will lead to inefficiency in the system because all the process will execute again from the beginning.

(#) Deadlock Detection using RAG

⇒ The main task of OS is detecting the deadlock. The OS can detect the deadlocks with the help of resource allocating graph (RAG)



⇒ In single instance resource type, if a cycle is in the system then there will definitely a deadlock, on the other hand in multiple instance resource type graph, detecting a cycle is not enough we have to apply safety algorithm on the system by converting the RAG into the allocation matrix and request matrix.

Resource Allocation Graph (RAG)

⇒ The resource Allocation graph is the pictorial representation of the state of a system. As its name suggests, RAG is the complete information packet about all the processes which are holding some resources and waiting for some resources. In RAG the process is represented by a circle while the resource is represented by a rectangle. We know

Date:
Page:

that any graph contains vertices and edges. So, RAG also contains vertices and edges.

(iv) Components of resource Allocating Graph:

⇒ Given below are the components of RAG:

(i) Vertices

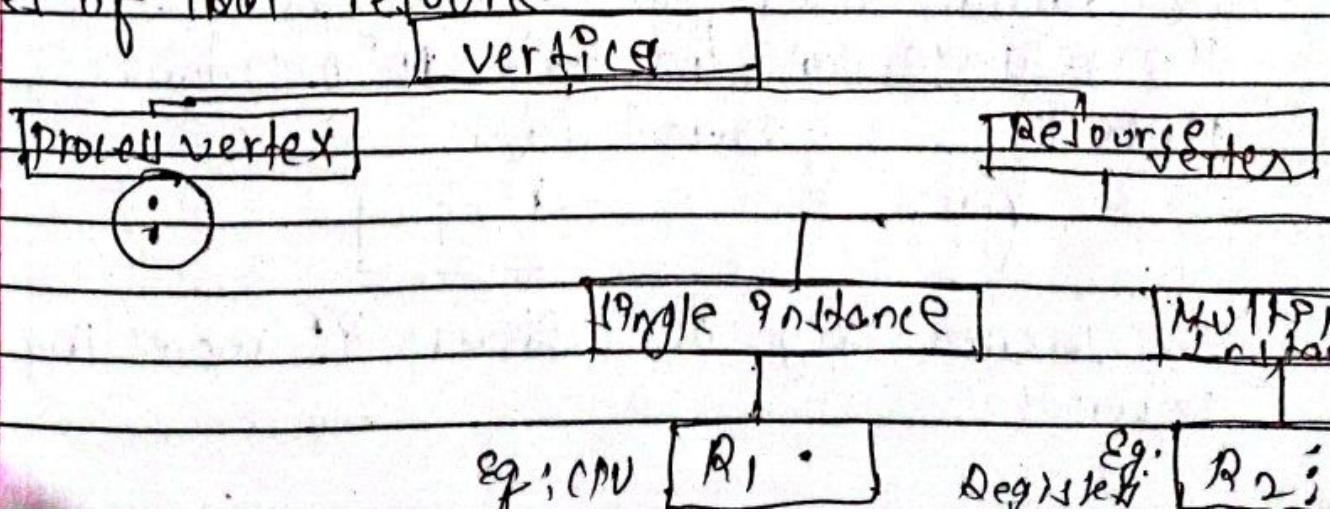
(ii) Edges

(i) Vertices: There are 2 kinds of vertices used in RAG and they are:

(a) Process vertices (b) Resource vertices.

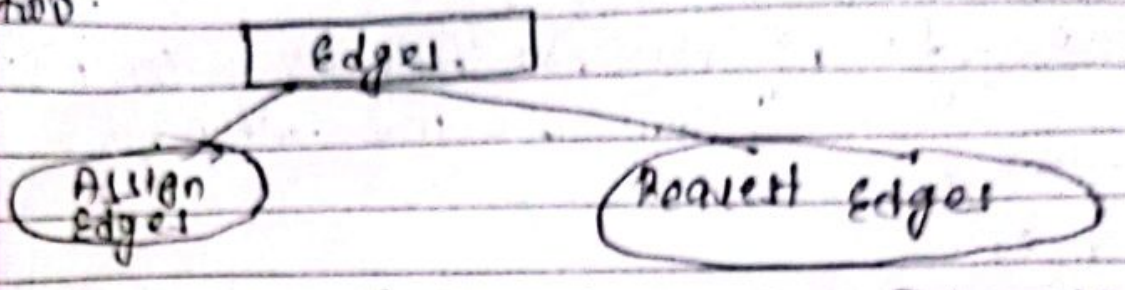
(a) Process vertices ⇒ It represents process. The circle is used in order to draw process vertex and name of the process is mentioned inside the circle.

(b) Resource vertices ⇒ Resource vertices are used to represent vertices. The rectangle is used in order to draw resource vertices and we use dots inside the rectangle in order to mention the number of instances of that resource

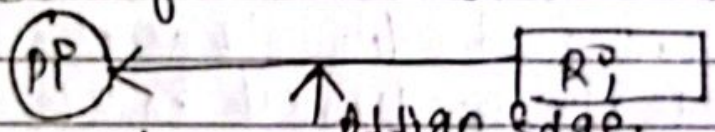


(ii) Edges:

In RAG, edges are further categorized as two.



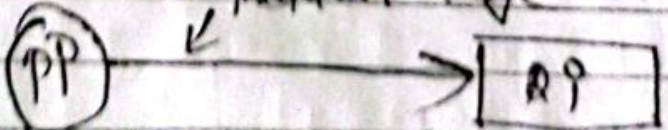
- Assign Edges → Assign edges are mainly used to represent the allocation of the resource to the process. We can draw assign edges with the help of an arrow in which the arrow head points towards the process and the process tail points towards the instance of the resource.



In above figure, the resource Rj is assigned to the process.

- Request Edge:

→ Request Edges is mainly used to signify the waiting state of the process. In request edges the arrow head points towards the instance of the resource and the tail points towards the process.

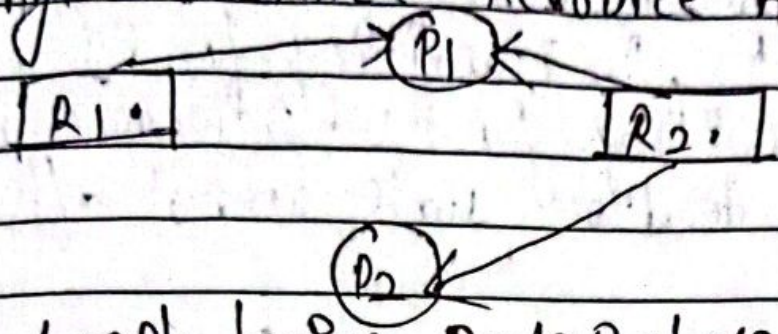


In above fig, the process Pi requesting the resources.

Date
Page

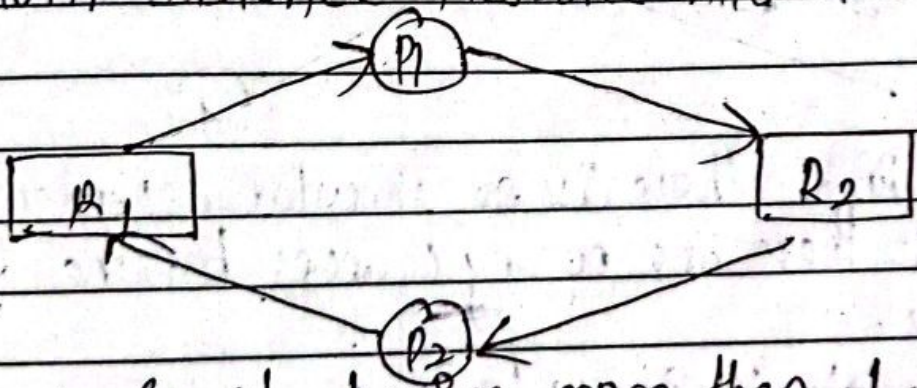
(*) On the basis of number of instance, the graph is of two types & they are as follows:-

- Single Instance Resource Allocation Graph



∴ Graph having single instance of the resource

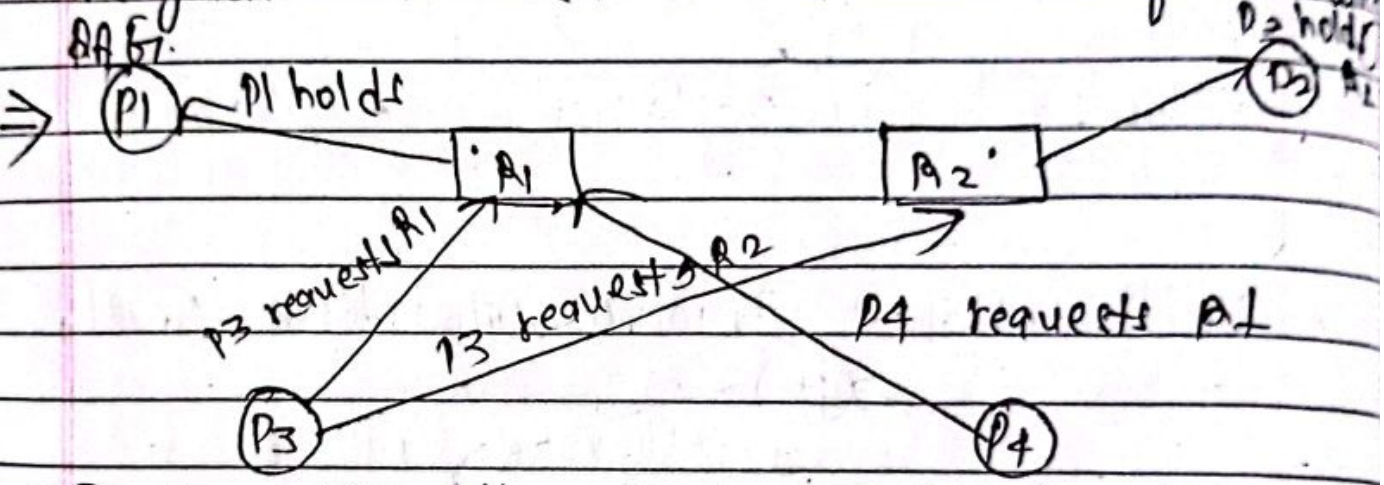
- Multi Instance Resource Allocation Graph



∴ Graph having more than 1 instance of the resource.

(#) Single Instance RAG Example:

(1) Suppose there are 4 processes P1, P2, P3 and P4 and two resources R1 and R2 where P1 is holding R1 and P2 is holding R2, P3 is waiting for R1 and R2 while P4 is waiting for R1. Determine whether the system is in deadlock state or not after drawing RAG.



→ In above RAG, there is no circular dependency i.e. no cycle, so there are no any chances for the occurrence of deadlock.

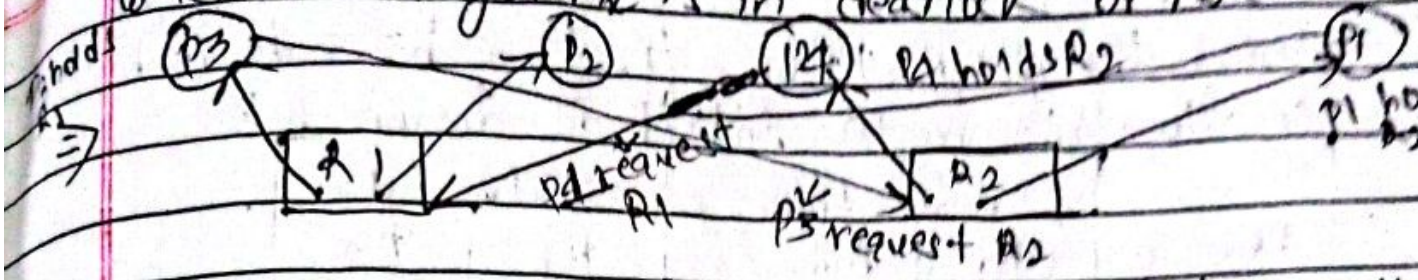
Note ⇒ ~~In the~~ Having cycle in single instance resource type must be the sufficient condition for deadlock.

(2) Multi Instance RAG Example:

(i) Suppose there are four processes P1, P2, P3, P4 and there are 2 instances of resources R1 and 2 instances of Resource R2 and 1 instance of R2 is assigned to P1 and another instance of R2 is assigned to P4, P3 is waiting for R1. One instance of R1 is assigned to P2 while another instance of R1 is assigned to P3. and P3 is waiting for R2. Draw RAG and de

P_2 holds R1

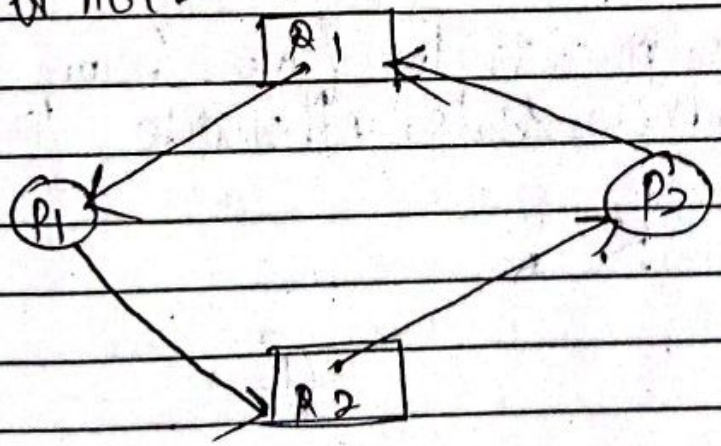
whether the system is in deadlock or not.



In above graph, the cycle is formed but there is no deadlock. because the resources are allocated in such a way that, there is safe sequence for safe state.

(#) Practice problem based on Detecting deadlock using RAG.

(1) Consider the resource allocation graph as shown in the figure below and state whether the system is in deadlock or not.



Solution:

Method I:

⇒ The above given RAG is single instance with cycle contained in it. Hence, the system is definitely in a deadlock state.

Method II

⇒ Using the given RAG, first we have to make resource allocation matrix, which is drawn below:

	Allocation		Need	
Process	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

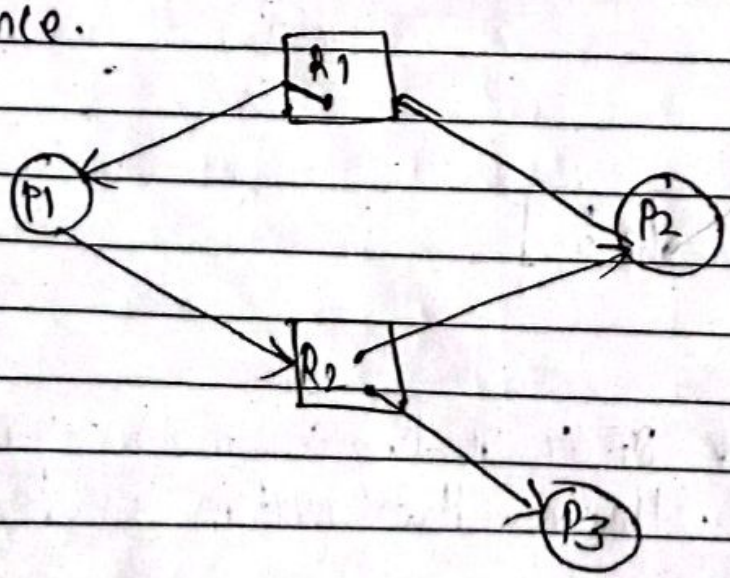
Here,

Available = $[R_1\ R_2] = [0, 0]$

Now, there are no instances currently available and both the process requires a resource to execute. Therefore, none of the process can be executed and both keep waiting infinitely. Thus, the system is in deadlock state.

(#) Consider a RAG given in the figure below. Find if the system is in deadlock state, otherwise find a safe sequence.

⇒



Date:
Page:

Solution

⇒ The above given RAG is multiple instance cycle so it is, the system may or may not be in dead lock state. Leaving given RAG first make resource allocation matrix which is drawn below:

	Allocation		Need	
Process	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

Here,

Available = $[R_1, R_2] = [0 \ 0]$

Step (i)

⇒ Since process P3 doesn't need any resource so it need is [0, 0], so it executes and after execution process P3 releases it resources. Then,

Available = [0 0] + [0 1]

= [0 1] (New)

Step2: With the instances currently available = [0 1], only the requirement of the process P1 can be satisfied so process P1 is allocated with the requested resources. It completes its execution and then frees up (releases). The instance of the resources held by it. Then. Available = [0 1] + [1 0]

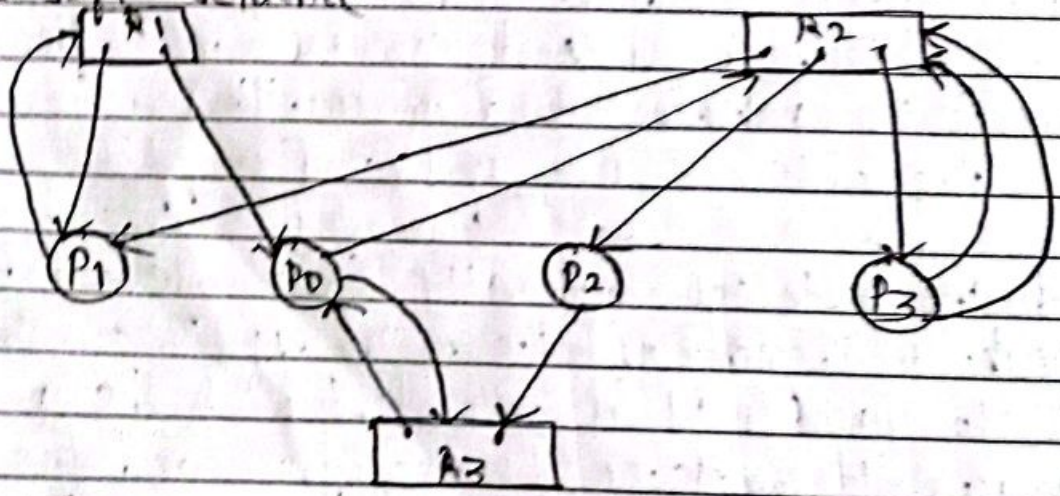
= [1 1] (new)

Step 3: With the resources currently available [1 1], the requirement of the process P2 can be satisfied. So process P2 is allocated with the required resources. Finally it completes its execution and releases the instances of resources held by it.

Available = [1 1] + [0 1]
= [1 2] (new)

Thus, there exist a safe sequence of (P0, P1, P2) in which all the ~~resources~~ processes get executed. So, we can say that the system is in safe state.

(1) Considered the RAG as shown in the figure below. Find if the system is in deadlock state. Otherwise find a safe sequence.



⇒ Solution:

The above given Resource Allocation graph is multi instance cycle graph. So the system may or may not be in deadlock state. So using given RAG we make resource

allocation matrix which is given below:

Process	Allocation			Need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	0	2	0

So, Available = $[R_1 R_2 R_3]$ = [0 0 1]

Step 1: With the instances currently available i.e. [0 0 1] the requirement of process P_2 can be satisfied. So process P_2 is allocated with the required resources. Then it completes its execution & then releases the instances of the resource held by it.

Then,

available = [0 0 1] + [0 1 0]

= [0 1 1] (new)

Step 2: With the instances currently available [0 1 1] only requirement of process P_0 can be satisfied so process P_0 is allocated with the required resources. It completes its execution and then frees up instance of it resources.

Available = [0 1 1] + [1 0 1]

= [1 1 2] (new)

Step 3: With the instances currently available i.e. [1 1 2], the requirement of process P_1 can be satisfied

Available = [1 1 2] + [1 1 0]

= [2 2 2] (new)

in which all the processes get executed. So we can conclude that the system is in safe state.

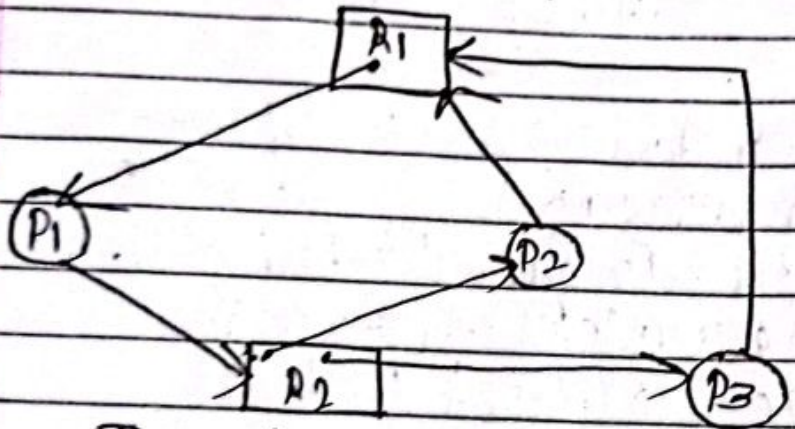
Step 4: At last P3 can be satisfied so,

Available = [2 2 2] + [0 1 0]

= [2 3 2]

Thus there exists a safe sequence of (P2, P0, P3)

(4) Considered the RAG as shown in the figure below find if the system is in deadlock state. Otherwise find a safe sequence.



⇒ The above given RAG is multiple instance cycle so the system may or may not be in deadlock state. So using given RAG first make resource allocation as per which is drawn below:

Process	Allocation		Need	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

Here, Available = [R1 R2] = [0 0]

Now, there are no instances currently available and all the processes required resource to execute. Therefore, none of the process can be executed and all

keep waiting infinitely. Thus, the system is in deadlock state.

Deadlock Avoidance

⇒ Mainly Banker's algorithm is used for deadlock avoidance.

Banker's Algorithm:

⇒ Banker's algorithm is resource allocation and deadlock avoidance algorith which tests all the request made by the processes for Resources. It checks for safe state if after granting request system remains in the safe state, it allows the request and if there is no safe state, it doesn't allow the request made by the process.

(*) Inputs to Banker's Algorithm.

⇒ Maximum need of resource by each process.
⇒ Currently allocated resource by each process.
⇒ Maximum free available resources in the system.

The request will only be granted under the below conditions

- If the request made by the process is less than or equal to max need of that process.
- If the request made by the process is less than or equal to freely available (currently available) resources in the system.

(*) Data structures is used to implement Banker's algorithm:

Some of the data structure used to implement Banker's are as follows:-

(1) Available → It is an array of length 'm' (one dimensional array) indicating the number of available resources of each types. If available [j] = k, then it means that there are 'k' instances of resource type 'Rj'.

(2) Max → It is two dimensional array of size 'n×m' where n is the number of processes and size 'n×m' defines the maximum number of instances of each resources type that a process can request. If Max [i][j] = k. Then process Pi can request at most 'k' instances of resource type Rj.

(3) Allocation → It is a 2 dimensional array of size 'n×m' which represents the number of resource of each type currently allocated to each process. If Allocation [i] [j] = k, then it means that process Pi is currently allocated with 'k' instances of resource type Rj ~~complete its execution~~

(4) Need: It is a 2 dimensional array of size 'n×m' which indicates the remaining resource needs of each process. If Need [i][j] = k, then it ~~requires~~ means that Pi may need more instances of resource type Rj to complete execution
Mathematically,

Need[i][j] = Max[i][j] - Allocation[i][j] #

Date:
Page:

(*) Banker's algorithm comprises of two Algorithms:
(a) Safety Algorithm
(b) Resource request algorithm

- Banker resource request Algorithm ⇒
Step 1 ⇒ If request ≤ Need go to step-2
else error
Step 2 ⇒ If request ≤ available Then go to step3
else wait
Step 3 ⇒ Available = Available - Request
Allocation = Allocation + Request
Need = Need - Request
Step 4 ⇒ Check new state is safe or not

- Safety Algorithm ⇒
Step 1: If need ≤ Available execute the process
Calculate New Available as,
Available = Available + Allocation
Step 2: Otherwise do not execute and go forward

Q⇒ Let us considered the following matrix table and perform the calculation using Banker's algorithm

Process/Resour	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	1	1	2	4	3	3	~~2~~	~~1~~	0
P1	2	1	2	3	2	2	~~4~~	~~2~~	2
P2	4	0	1	9	0	2	5	3	4

P3	0	2	0	7	5	3	6	4	6
P4	1	1	2	1	1	2	13	6	7

(a) Calculate the content of need matrix.
(b) Check if the system is in safe state.
(c) Determine the sum of each type of resource

a⇒ Calculating the need matrix we get;

Process	Request	Need matrix		
		A	B	C
P0		3	2	1
P1		1	1	0
P2		5	0	1
P3		7	3	3
P4		0	0	2

Need = Max - Allocated

Now, let us check for safe state i.e. safe sequence.

- For process P0, Need = [3 2 1] and available = [2 1 0] Therefore, Need ≤ Available (false)

So, the system will move to next process.

- For process P1, Need = [1 1 0] and available = [2 1 0]

Need ≤ Available (T)

So, the request of P1 is granted and kept in safe sequence.

Therefore, Available = Available + Allocation
= [2 1 0] + [2 1 2]
= [4 2 2]

- For process P2, Need = [1 0 1] and
Available = [4 2 2]
Therefore Need ≤ Available (false).
So the system will move to the next process

- For process P3, Need = [1 3 3] and
Available = [4 2 2]
Therefore Need ≤ Available (false)
So, the system will move to the next process.
- For process P4, Need = [0, 0 0] and
Available = [4 2 2]
Need ≤ Available
So, the request of P4 is granted and it is kept in safe sequence
Therefore, Available = Available + Allocation
= [4 2 2] + [1 1 2]
= [5 3 4] (new Available)
- Now for process P0, Need = [3 2 1] and available = [5 3 4]. Need ≤ Available (True)
So, the request of P0 is granted now and it kept in the safe sequence.
Therefore, Available = Available + Allocation
= [5 3 4] + [1 1 2]

Date:
Page:

= [6 4 6] (new)

Again, for process P_2, Need = [1 0 1]
and Available = [6 4 6]
∴ Need ≤ Available (True). So the request for P_2 is granted and kept in the safe sequence.
∴ Available = Available + Allocation
= [6 4 6] + [4 0 1]
= [10 4 7]

At last, for process P_3 Need = [7 3 3]
and Available = [10 4 7]
∴ Need ≤ Available (True). So, the request for P_3 is granted and kept in safe sequence
∴ Available = Available + Allocation
= [10 4 7] + [0 2 0]
= [10 6 7] (new Available)

Thus, there exists a safe sequence of P_1, P_4, P_0, P_2 and P_3 in which all the processes get executed so we can say that the system is in safe state.

We see that the system allocates all the needed resources to each process. So we can say that the system is in safe state.

Again, lastly the total number of resources = sum of colomn of allocation + Available

[8 5 7] + [2 1 0]

= [10 6 7]