Deadlock

A process is a program in execution state.

A resource is any hardware or software component present in the computer system.

Some example resources are: RAM, HD, CPU, compiler, file and so on.

A number of processes may be executing in the computer system at any particular time.

Each process requires some resources.

Operating system allocates the resources to the processes running in the computer system.

When a process requires a resource then the process makes a request to the operating system.

The operating system allocates the requested resource if that resource is free otherwise the process has to wait.

A computer system may have one or more instances of each resource type.

For example, a computer system may have 3 CPUs, 2 printers and so on.

There are two types of resources: sharable and non-sharable.

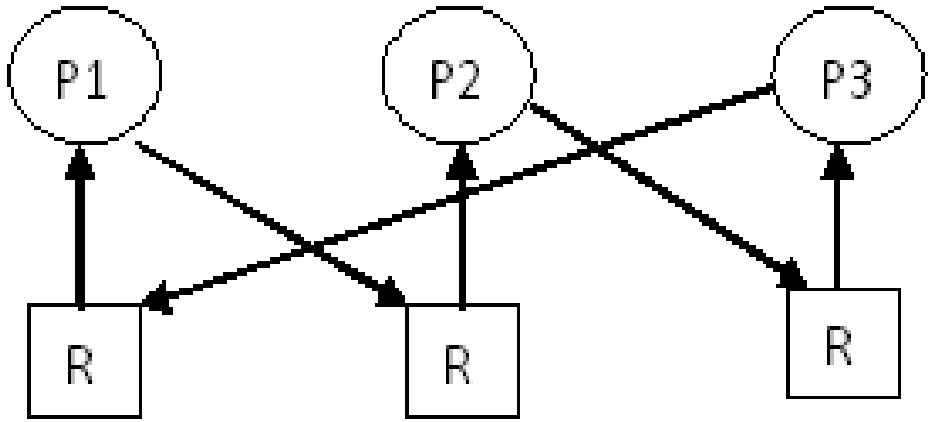
A sharable resource can be used by a number of processes at a time.

A non-sharable resource can be used by only one process at a time.

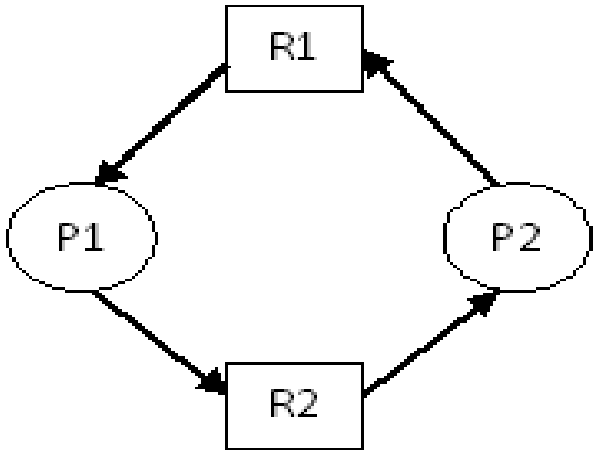
Deadlock

A set of processes is said to be in deadlock state if each process in the set is waiting for an event (releasing of resource) that can be caused by another process in the set.

Example for deadlock involving a single resource with multiple instances:



Example for deadlock involving multiple resource types:



Necessary conditions for deadlock

The following four conditions must hold for the occurrence of deadlock state.

(or)

In a deadlock state, the following four conditions are satisfied.

Mutual Exclusion

Mutual exclusion condition indicates that a non-sharable resource should be used by only one process at a time.

Hold and Wait

Each process is holding some resources and waiting for other resources.

No-Preemption

A resource cannot be released from a process before completion of the process.

Circular Wait

There is a set of processes (P1, P2, P3, … , Pn) such that

P1 is waiting for P2
P2 is waiting for P3
.
.
.
.
Pn is waiting for P1

Resource Allocation Graph

Resource allocation graph is used to describe the deadlock state.

Nodes in the resource allocation graph represent both processes and resources.

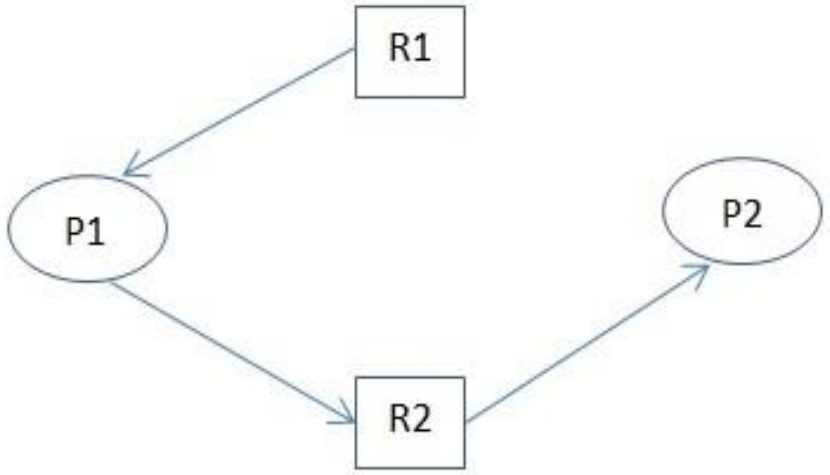
Edges indicate both allocation of resources to processes and request to resources by the processes.

Processes are indicated using circles, resources are indicated using boxes.

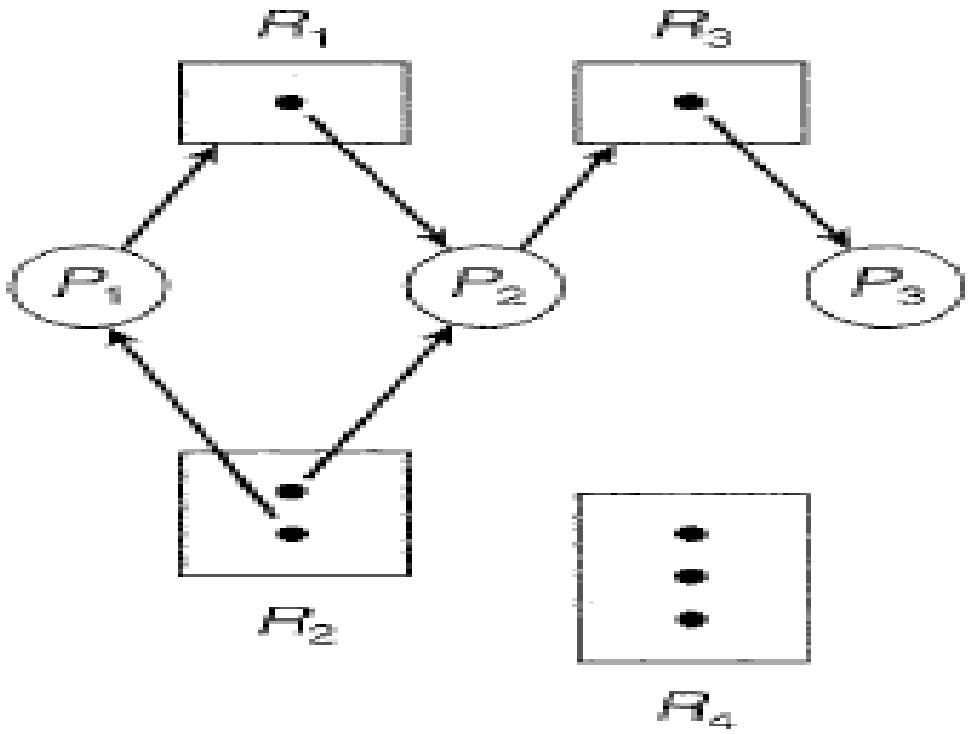
A request edge is from a process to a resource.

An allocation edge is from a resource to a process.

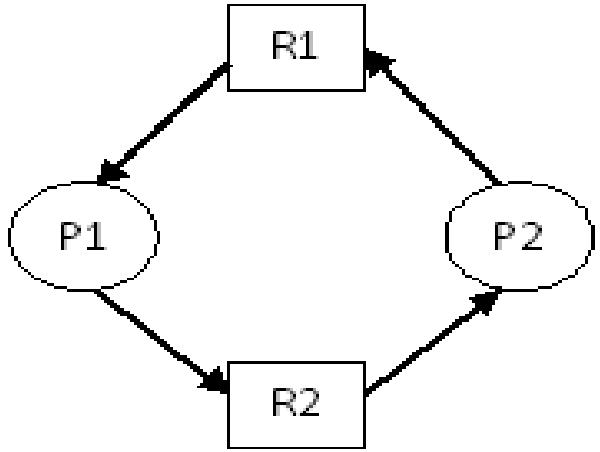
If a resource has number of instances then that number of instances is indicated using number of dot symbols inside the box representing that resource.



There is no cycle in above resource allocation graph. There is no deadlock.



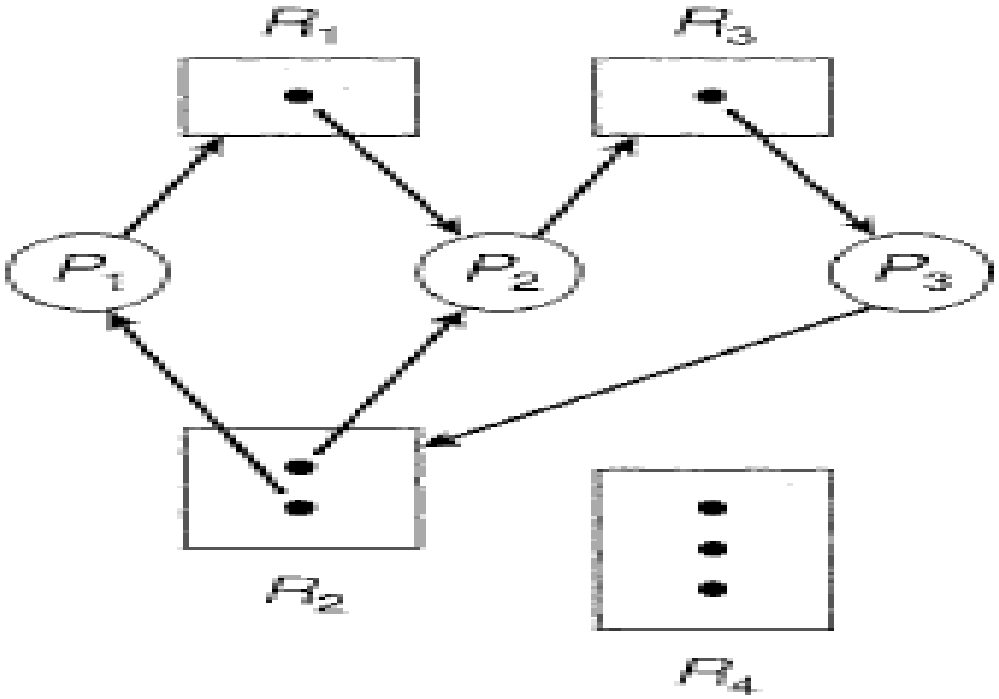
There is no cycle in above resource allocation graph. There is no deadlock.



There is a cycle in the above graph.

P1 -> R2 -> P2 -> R1 -> P1

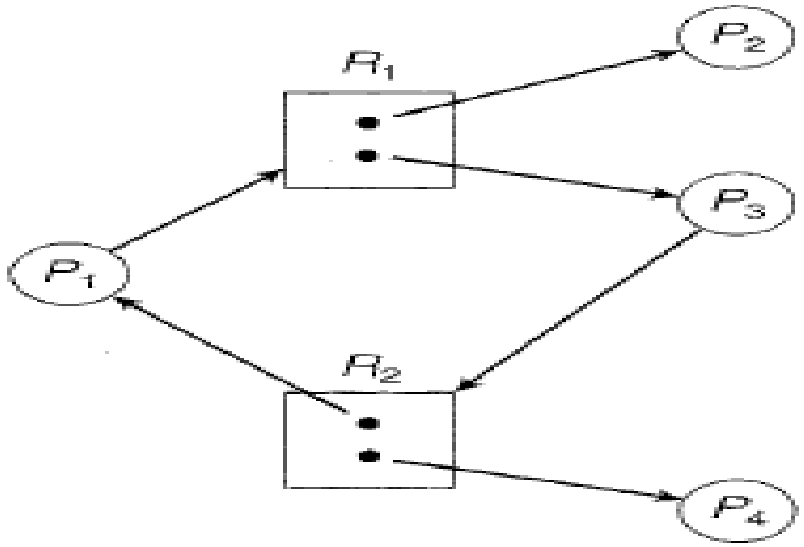
Processes P1 and P2 are in deadlock state.



There are two cycles in the above graph

P1 -> R1 -> P2 -> R3 -> P3 -> R2 -> P1
P2 -> R3 -> P3 -> R2 -> P2

Processes *P1, P2,* and *P3* are in deadlock state.



There is a cycle

P1 -> R1 -> P3 -> R2 -> P1

There is no deadlock.

Conclusions from the above examples are:

1) If there is no cycle in the Resource Allocation Graph then there is no deadlock state.

2) If each resource has only one instance then the presence of cycle in the resource allocation graph indicates the presence of deadlock state.

3) If the resources have multiple instances then the presence of cycles in the resource allocation graph may or may not indicate the presence of deadlock state.

Methods for handling deadlocks

1) Deadlock prevention
2) Deadlock avoidance
3) Deadlock detection
4) Deadlock recovery

Deadlock prevention: completely avoiding the occurrence of deadlock state.

Ex: controlling the traffic using traffic signal.

Deadlock avoidance: avoiding the deadlock state whenever it is going to occur.

Ex: controlling the traffic using traffic police.

Deadlock detection: detecting the occurrence of deadlock state.

Deadlock recovery: recovering from the deadlock state when it occurs.

Deadlock prevention

deadlock is prevented by avoiding the occurrence of any of the 4 necessary conditions for deadlock.

Mutual Exclusion

Avoiding the occurrence of mutual exclusion condition means allocating a resource to a number of processes at a time.

This is possible only with sharable resources.

We cannot avoid the occurrence of mutual exclusion condition for non-sharable resources.

Hold and Wait

Any one of the following two mechanisms is used to avoid the occurrence of hold and wait condition.

1) Allocate all required resources to the process before starting the execution of the process.

2) Allocate a subset of required resources to the process.

Release this subset of resources from the process after using them and allocate another subset of resources.

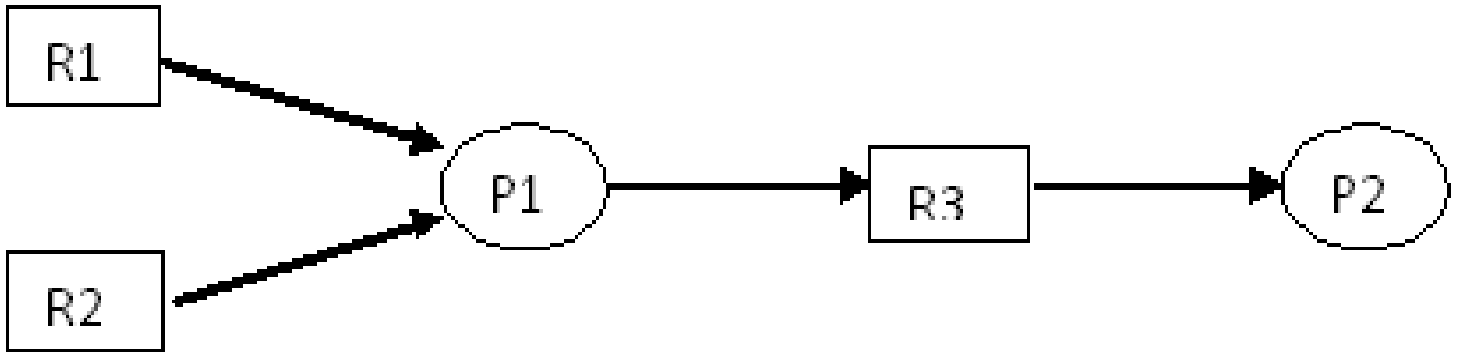
Repeat this procedure till the completion of process.

No preemption:

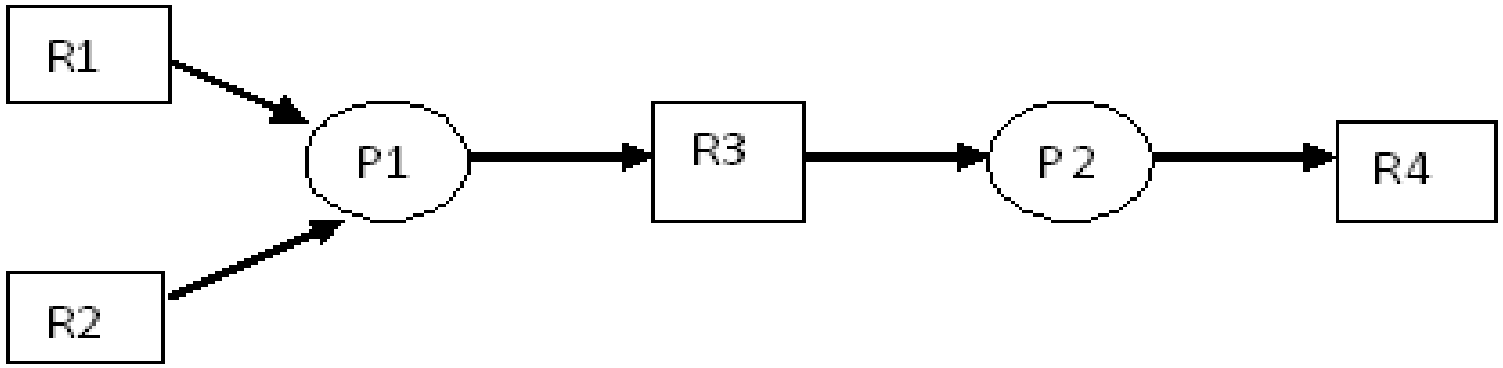
The resources are released from the process before completion of the process to avoid the occurrence of no preemption condition.

The following two mechanisms are used to avoid the occurrence of no preemption:

1) When a process requests for any resource which is currently allocated to a running process then the requesting process will go the waiting state and the resources that are currently allocated to the requesting process are released.



2) When a process request for a resource which is currently allocated to a waiting process then the resource from the waiting process is released and allocated to the requesting process.



Circular wait

The following mechanism is used to avoid the occurrence of circular wait condition:

An ordering like R1, R2, R3 ….. is given to the resources and the processes are allowed to request the resources in an increasing order.

A process is allowed to request a resource whose number is greater than the numbers of resources that are currently allocated to the process.

When a process request for a resource whose number is less than the number of any resource allocated to the process then that request is rejected by the operating system.

For example, if a process P1 is holding resources R2 and R5 and if it requests for resource R7 then the request is accepted.

If P1 request for R3 then that request is rejected.

Deadlock Avoidance

To avoid the occurrence of deadlock, the following information must be known in advance:

1) the number of processes in the computer system
2) the resources required by each process

Safe state

If all processes can be completed in any order with available resources then the system is said to be in safe state.

Ex:

	Max	Allocated	Required
P0	10	5	5
P1	4	2	2
P2	9	2	7

Total resources = 12 copies
Available = 3 copies

The above state is safe state as all processes can be completed in the order P1, P0, P2.

Ex:

	Max	Allocated	Required
P0	10	5	5
P1	4	2	2
P2	9	3	6

Total resources = 12 copies
Available =2 copies

The above state is unsafe as it is not possible to complete P0 and P2.

The following procedure is used to avoid the occurrence of deadlock state:

When any process request for any resource then the operating system checks whether the allocation of requested resource will lead to (or) result in the safe state or not.

If the allocation of requested resource to the process will lead to safe state, then the operating system accepts the request. Otherwise, rejects the request.

Banker's Algorithm for Deadlock Avoidance

Four data structures are used in this algorithm.

1) Available: is a vector of size m. Where m is the number of resource types.

Available vector indicates the number of copies of each resource type available in the system.

2) Max: is a matrix of size nxm. Where n is the number of processes.

Max matrix indicates the total number of copies of each resource type required by each process.

3) Allocation: is a matrix of size nxm.

Allocation matrix indicates the number of copies of each resource type allocated to each process.

4) Need: is a matrix of size nxm.

Need matrix indicates the number of copies of each resource type still required by each process.

Need matrix is calculated as

Need = Max - Allocation

Ex: n=4, m=2

	Max		Allocation		Available		Need	
	A	B	A	B	A	B	A	B
P0	7	5	0	1	3	3	7	4
P1	3	2	2	0			1	2
P2	9	0	3	0			6	0
P3	2	2	2	1			0	1

Safety algorithm

This algorithm is used to check whether the system is in safe state or not.

Steps in the algorithm are:

1. Find a process P_i which is not marked as finished and **$Need_i$ <= Available**. If no such process exists then go to step 3.

2. Mark the process P_i as finished and update the Available vector as

Available = Available + $Allocation_i$

goto step 1.

3. If all processes are marked as finished, then indicate that the system is in safe state. Otherwise, indicate that the system is in unsafe state.

Ex: n=4, m=2

	Max		Allocation		Available		Need	
	A	B	A	B	A	B	A	B
P0	7	5	0	1	3	3	7	4
P1	3	2	2	0			1	2
P2	9	0	3	0			6	0
P3	2	2	2	1			0	1

Above state is safe state as all processes can be completed in the order: P1, P3, P0, P2.

At first, process P1 is selected as its **need$_1$<=Avialable** ((1,2)<=(3,3)).

After completing the process P1, the resources allocated to process P1 are released and then added to the available vector.

Available
5 3

Next, process P3 is selected as its **need$_3$<=Available** ((0,1)<=(5,3)).

After completing process P3, the resources allocated to process P3 are released and then added to the available vector.

Available
7 4

Next, process P0 is selected as its **need$_0$<=Available** ((7,4)<=(7,4)).

After completing process P0, the resources allocated to process P0 are released and then added to available vector.

Available
7 5

Next, process P2 is selected as its **need$_2$<=Available** ((6,0)<=(7,5)).

After completing process P2, the resources allocated to process P2 are released and then added to available vector.

Available
10 5

Resource request algorithm

When a process P_i makes a $Request_i$ for resources then the operating system calls the **resource request algorithm** in order to decide whether to accept or reject the request.

The steps in the algorithm are:

1. If **$Request_i$ <= Available** then go to step 2. Otherwise, the process P_i has to wait.

2. Pretend that the requested resources are allocated to the process and update the values of matrices as

   $Allocation_i = Allocation_i + Request_i$
   $Need_i = Need_i - Request_i$
   $Available = Available – Request_i$

3. Check whether the resulting state is safe or not by calling the safety algorithm. If the resulting state is safe, then accept the request. Otherwise, reject the request and restore the old state.

Ex: n=4, m=2

	Max		Allocation		Available		Need	
	A	B	A	B	A	B	A	B
P0	7	5	0	1	3	3	7	4
P1	3	2	2	0			1	2
P2	9	0	3	0			6	0
P3	2	2	2	1			0	1

The above state is safe as the processes can be completed in the order: P1, P3, P0, P2.

If the process P1 makes a request

$$\textbf{Request}_1 = (1,0)$$

Then the operating system calls the **resource request algorithm** to decide whether the request can be accepted or not.

The **resource request algorithm** checks whether **$Request_1$<=Available** or not.

As **$Request_1$<=Available ((1,0)<=(3,3))**, the matrices are modified as

	Allocation		Need		Available	
	A	B	A	B	A	B
P0	0	1	7	4	2	3
P1	3	0	0	2		
P2	3	0	6	0		
P3	2	1	0	1		

Now the safety algorithm is called to determine whether the resulting state is safe or not.

The safety algorithm returns that the state is safe as the processes can be completed in the order: P1, P3, P0, P2.

So the operating system accepts the request.

Ex: n=4, m=2

	Max		Allocation		Available		Need	
	A	B	A	B	A	B	A	B
P0	7	6	0	1	2	3	7	5
P1	3	2	3	0			0	2
P2	9	3	3	0			6	3
P3	2	2	2	1			0	1

In the above state, if process P0 makes a request

$Request_0$ = (0,2)

Then the operating system calls the resource request algorithm to decide whether the request can be accepted or not.

The resource request algorithm checks whether **$Request_0$<=Available** or not.

As **$Request_0$<=Available ((0,2)<=(2,3))**, the matrices are modified as

	Allocation		Need		Available	
	A	B		A	B	A	B
P0	0	3	P0	7	3	2	1
P1	3	0	P1	0	2		
P2	3	0	P2	6	3		
P3	2	1	P3	0	1		

Now the safety algorithm is called to determine whether the resulting state is safe or not.

The safety algorithm returns that the state is unsafe as it is not possible to complete all processes.

So the operating system rejects the request.

Deadlock detection

Deadlock detection method is used to check the occurrence of deadlock state in the computer system.

Two methods are used for detecting the occurrence of deadlock state.

1. Wait-for-graph algorithm
2. Banker’s algorithm

Wait-for-graph algorithm

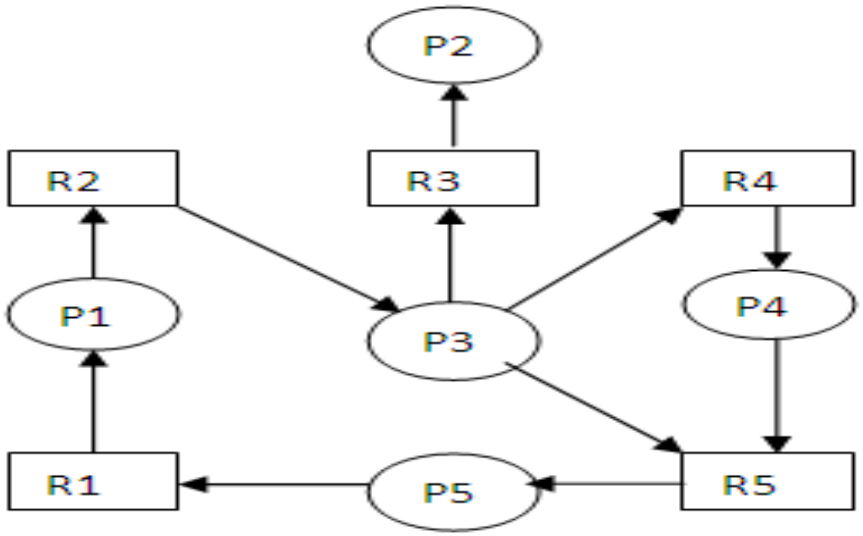
Wait-for-graph algorithm is used when each resource has only one instance.

In this algorithm, a wait-for-graph is constructed based on the resource allocation graph.

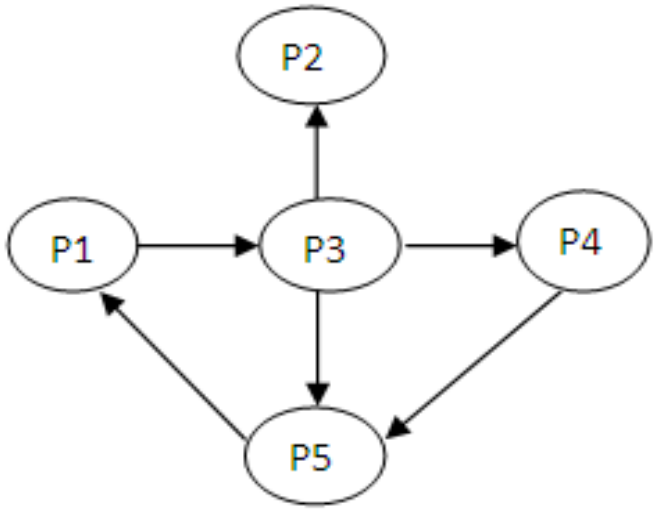
Wait-for-graph contains only the process nodes and they are indicated by circles.

If a process P_i is waiting for a resource held by process P_j, then an edge is drawn from process P_i to process P_j in the wait-for-graph.

Presence of cycles in the wait-for-graph indicates occurrence of the deadlock state.



Resource allocation graph



Wait-for-graph

There are 2 cycles in the wait-for-graph

Cycle 1: P1->P3->P5->P1
Cycle 2: P1->P3->P4->P5->P1

So, deadlock has occurred in the system. The processes P1, P3, P4 and P5 are in deadlock state. Process P2 is not involved in any cycle. P2 is not in deadlock state.

Banker’s algorithm for Deadlock Detection

Banker’s algorithm is used when the resources have number of instances or copies.

The following data structures are used in this algorithm.

Available: It is a vector of size m. Where m is number of resource types.

Available vector indicates the number of available copies of each resource type.

Allocation: It is a matrix of size nxm. Where n is number of processes.

Allocation matrix indicates the number of copies of each resource type that are currently allocated to each process.

Request: It is a matrix of size nxm.

Request matrix indicates the number of copies of each resource type that are required by each process.

Algorithm

1. Find a process P_i which is not marked as finished and **$Request_i$ <= Available**. If no such process exist then go to step 3.

2. Mark the process P_i as finished and update the Available vector as

   Available = Available + $Allocation_i$

   goto step 1.

1. If all processes are marked as finished, then indicate that deadlock state has not occurred. If some of the processes are not marked as finished, then indicate that deadlock has occurred. The processes which are not marked as finished are in the deadlock sate.

Ex:

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

At first, process P0 is selected as it is not marked as finished and its

$Request_0$ <= Available

Process P0 is marked as finished and the available vector is updated as

Available	
0	1

Next, process P2 is selected as it is not marked as finished and its **$Request_2$ <= Available**. Process P2 is marked as finished and the available vector is updated as

Available	
3	1

Next, process P1 is selected as it is not marked as finished and its **$Request_1$ <= Available**. Process P1 is marked as finished and the available vector is updated as

Available	
5	1

Finally, process P3 is selected as it is not marked as finished and its **$Request_3$ <= Available**. Process P3 is marked as finished and the available vector is updated as

Available	
7	2

Deadlock state has not occurred in the system as all processes are marked as finished.

Ex:

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

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

Available		
A	B	C
0	0	0

First, process P0 is selected as it is not marked as finished and its **$Request_0$ <= Available**. Process P0 is marked as finished and the available vector is updated as

Available		
0	1	0

The request of other processes cannot be satisfied with available resources.

So, the processes P1, P2, P3 and P4 are not marked as finished and it indicates the occurrence of deadlock state in the system.

Deadlock Recovery

Any of the following two techniques is used to recover from the deadlock state:

1) Aborting the processes that are involved in the deadlock state.
2) Preempting the resources from the deadlocked processes.

Aborting the processes that are involved in the deadlock state

In this method, the processes which are involved in the deadlock state are terminated.

There are two ways for aborting the deadlocked processes.

1) Abort all processes at a time.

2) Abort the processes one by one until the deadlock cycle is eliminated.

In the first mechanism, all processes that are involved in the deadlock state are terminated or aborted.

Advantages

1. Deadlock can be recovered in less amount of time.
2. No need to apply the deadlock detection algorithm for number of times.

Disadvantage

1. Losing of computations of all processes.

In the second mechanism, first one of the processes in the deadlock state is terminated.

After that the presence of deadlock state is checked.

If the deadlock state still exist, then one more process in the deadlock state is terminated.

This procedure is repeated until the system is recovered from the deadlock state.

Advantage

1. Less wastage in computation time compared to the first mechanism as only some processes in the deadlock state are terminated.

Disadvantage

1. Need of applying the deadlock detection algorithm for number of times.
2. Some criteria have to be used to select a process for termination at each step.

The general criteria used to select a process for termination are

1) Execution time

The process that has been executed for less amount of time is selected for termination.

2) No of Resources

The process which is holding less number of resources is selected for termination.

3) Priority

The process with low priority is selected for termination.

Preempting the resources from the deadlocked processes

In this method, resources will be preempted successively from the deadlocked processes until the system is recovered to the normal state.

To use this mechanism for recovering from the deadlock state, the following factors are considered.

1. Selecting a victim (resource)
2. Rollback
3. Starvation

Selecting a victim: one of the deadlocked processes has to be selected and then one of the resources allocated to that process needs to be selected based on some criteria like execution time, number of resources allocated, priority and so on.

Rollback: When a resource is preempted from a deadlocked process then that process needs to be restated from the beginning.

Starvation: resources should not be preempted from the same process in each step.

Ex: n=5, m=4

Max

	A	B	C	D
P0	0	0	1	2
P1	1	7	5	0
P2	2	3	5	6
P3	0	6	5	2
P4	0	6	5	6

Allocation

	A	B	C	D
P0	0	0	1	2
P1	1	0	0	0
P2	1	3	5	4
P3	0	6	3	2
P4	0	0	1	4

Available

A	B	C	D
1	5	2	0

Need

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Process P1 makes a request

$Request_1$ = (0,4,2,0)

Can the request be accepted or rejected?

Allocation

	A	B	C	D
P0	0	0	1	2
P1	1	4	2	0
P2	1	3	5	4
P3	0	6	3	2
P4	0	0	1	4

Need

	A	B	C	D
P0	0	0	0	0
P1	0	3	3	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

Available

A	B	C	D
1	1	0	0