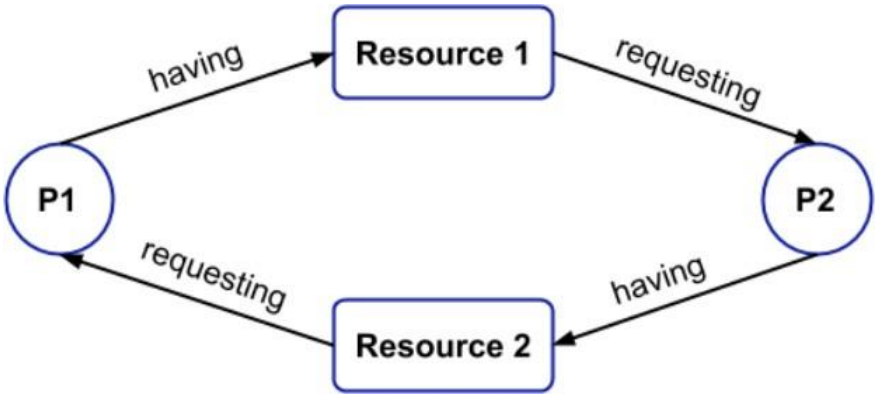
Chapter 5: Deadlock

Deadlock:

- A deadlock is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process. Therefore, none of the processes gets executed.
- Deadlock occurs when every process in a set of processes are in simultaneous wait state and each of them is waiting for the release of a resource held exclusively by one of the waiting processes in the set.
- None of the process can proceed until at least one of the process release the acquired resources.



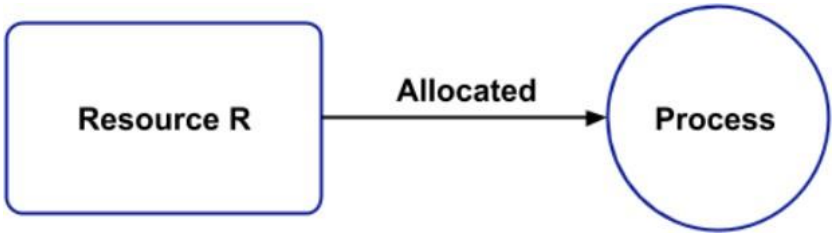
- For example, let us assume, we have two processes P1 and P2. Now, process P1 is holding the resource R1 and is waiting for the resource R2. At the same time, the process P2 is having the resource R2 and is waiting for the resource R1. So, the process P1 is waiting for process P2 to release its resource and at the same time, the process P2 is waiting for process P1 to release its resource. And no one is releasing any resource. So, both are waiting for each other to release the resource. This leads to infinite waiting and no work is done here. This is called Deadlock.
- A system may consists of various types of resources like input/output devices, memory space, processors, disks etc.
- A process may need multiple resources to accomplish its task. However to use, it must follow following steps:
  - **Request for the required resources:** Firstly, the process requests the resource. In a case, if the request cannot be granted immediately (e.g.: resource is being used by any other process), then the requesting process must wait until it can acquire the resource.

- **Use the allocated resources**: The Process can operate on the resource (e.g.: if the resource is a printer then in that case process can print on the printer).
- **Release the resources:** Process releases the resource after completing its task.

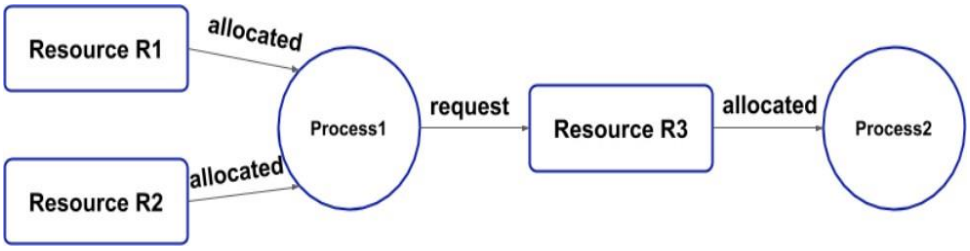
Deadlock Conditions:

There are four different conditions that result in Deadlock. These four conditions are also known as Coffman conditions. They are as follows:

- **Mutual Exclusion:**
  - A resource can be held by only one process at a time.
  - In other words, if a process P1 is using some resource R at a particular instant of time, then some other process P2 can't hold or use the same resource R at that particular instant of time.
  - The process P2 can make a request for that resource R but it can't use that resource simultaneously with process P1.

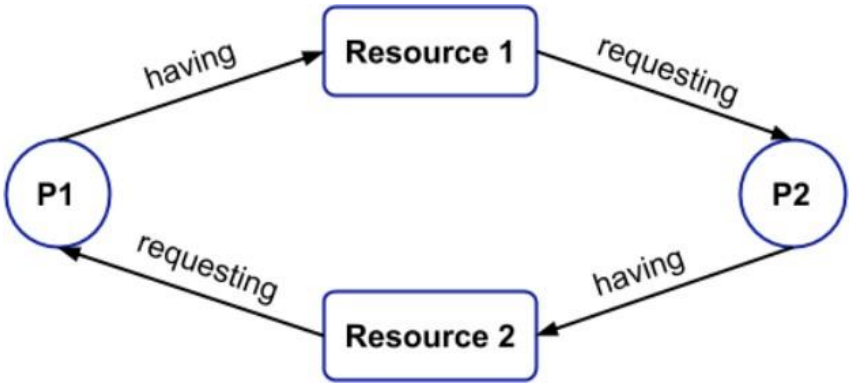


- **Hold and Wait:**
  - A process can hold a number of resources at a time and at the same time, it can request for other resources that are being held by some other process.
  - For example, a process P1 can hold two resources R1 and R2 and at the same time, it can request some resource R3 that is currently held by process P2.



- **No preemption:**
  - A resource can't be preempted from the process by another process, forcefully.
  - For example, if a process P1 is using some resource R, then some other process P2 can't forcefully take that resource.
  - If it is so, then what's the need for various scheduling algorithm. The process P2 can request for the resource R and can wait for that resource to be freed by the process P1.

- **Circular Wait:**
  - Circular wait is a condition when the first process is waiting for the resource held by the second process, the second process is waiting for the resource held by the third process, and so on. At last, the last process is waiting for the resource held by the first process.
  - So, every process is waiting for each other to release the resource and no one is releasing their own resource. Everyone is waiting here for getting the resource. This is called a circular wait.



Deadlock Handling

Following are the various deadlock handling strategies:
1) Deadlock Ignorance.
2) Deadlock Prevention.
3) Deadlock Avoidance.
4) Deadlock Detection and Recovery

1. Deadlock Ignorance:

- Deadlock Ignorance is the most widely used approach among all the mechanism. This is being used by many operating systems mainly for end user uses.
- In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock.
- This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff.

Note: The above approach is an example of Ostrich Algorithm. It is a strategy of ignoring potential problems on the basis that they are extremely rare.

Ostrich Algorithm

- Pretend that there is no problem.
- This is the easiest way to deal with problem.
- This algorithm says that stick your head in the sand and pretend that there is no problem at all.

- This strategy suggests to ignore the deadlock because deadlock occur rarely, but system crashes due to hardware failures, compilers errors and operating system bugs frequently, then not to pay a large penalty in performance or convenience to eliminate deadlocks.

2. Deadlock Prevention

Deadlocks can be prevented by preventing at least one of the four required conditions:

a) Mutual Exclusion
b) Hold and wait
c) No preemption
d) Circular wait

a) Mutual Exclusion:

- Deadlock can be prevented by preventing mutual exclusion (making all resources shareable).
- But it is not always possible to prevent deadlock by preventing mutual exclusion as certain resources are cannot be shared safely.

b) Hold and wait:

- To prevent this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.
- One approach is that the processes holding resources must release them before requesting new resources, and then re-acquire the released resources along with the new ones in a single new request.

c) No preemption:

- Preemption is temporarily interrupting an executing task and later resuming it.
- Preemption of process resource allocations can prevent this condition of deadlocks, when it is possible.
- Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.
- One approach is that if a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, (preempted), forcing this process to re-acquire the old resources along with the new resources in a single request, similar to the previous discussion.

d) Circular wait:

- Deadlock can be prevented if circular wait condition is avoided.
- One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order.
- In other words, in order to request resource Rj, a process must first release all Ri such that i>=j.
- To prevent circular wait, we impose ordering of all resource types i.e., a unique integer number is assigned to each resource. This concept is applied only for resources of higher numbers or higher "id". If a process is requesting resources of a higher "id", it must be released.

- Consider that the resources have been assigned positive integers as, Printer → 1, Tap drive → 2, Card punch → 3, Card reader → 4, Plotter → 5. Now, the process must request the resources in numerical order. For example, the process can request the printer first followed by the card punch and card reader (1, 3, 4). It cannot request a card reader first and then the printer.

3. Deadlock Avoidance:

- A deadlock can be prevented by eliminating any one of the four conditions necessary for deadlock.
- Preventing deadlock by this method results in inefficient use of resources. Thus, instead of preventing deadlock, it is better to avoid it by never allowing allocation of a resource to a process if it will lead to a deadlock.
- This can be achieved when some additional information is available about how the process are going to request for a resources in future.
- Deadlock avoidance can be achieved by being careful at the time of resource allocation.
- The system must be able to decide whether granting a resource is safe or not, and only make allocation when it is safe.

Avoidance Algorithms:

- **Single instance of a resource type:**
  - Use resource allocation graph
- **Multiple Instance of a resource type**
  - Use Banker's Algorithm

Safe and Unsafe State:

- A state is safe if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state.
- Precisely a system is in safe state if there is a safe sequence.
- A safe sequence is a sequences of process execution such that each and every process executes till its completion.
- If a safe sequence does not exist, then the system is in an unsafe state, which MAY lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks).

Example: Consider three process P1, P2 and P3, that are currently executing and there are 10 instances of resource type. The maximum number of resources required by process, the number of resources already allocated and available resources are given as below:

Process	Maximum resources	Allocated
P1	9	3
P2	4	2
P3	7	2

Available Resources = 3

- Here, since process P1 needs 6 instances of resources but available resource is only 3. So process P1 should not be given available resources, otherwise it leads to deadlock.

- Now process P2 needs only 2 instances of resources, so 2 instances can be given to process P2 so that process P2 can execute. So after resources are allocated to P2,

Process	Maximum resources	Allocated
P1	9	3
P2	4	4
P3	7	2

Available Resources = 1

- After P2 completely executes, all the resources allocated to P2 are released, so new available resources becomes 5.

Process	Maximum resources	Allocated
P1	9	3
P2	0	0
P3	7	2

Available Resources = 5

- Now available resources can be given to process P3

Process	Maximum resources	Allocated
P1	9	3
P2	0	0
P3	7	7

Available Resources = 0

- After P3 completely executes, all the resources allocated to P3 are released, so new available resources becomes 7.

Process	Maximum resources	Allocated
P1	9	3
P2	0	0
P3	0	0

Available Resources = 7

- Now process P1 needs only 6 instances of resources, so 6 instances can be given to process P1 so that process P1 can execute. So, after resources are allocated to P1,

Process	Maximum resources	Allocated
P1	9	9
P2	0	0
P3	0	0

Available Resources = 1

- After P1 completely executes, all the resources allocated to P1 are released, so new available resources becomes 10,

Process	Maximum resources	Allocated
P1	0	0
P2	0	0
P3	0	0

Available Resources = 10

- Since all process execute completely with the sequence P2, P3 and P1. The safe sequence is P2, P3 and P1.

Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Algorithm

1) Let Work and Finish be two matrix of size m and n respectively. Initialize the Work matrix with available resources and Finish [i] = false, for i = 1 to n.

2) Find an i such that both

a) Finish [i] = false

b) Need [i] <= Work [I]

If no such I exist go to step 4.

3) Work = Work + Allocation

Finish [i] = true

Goto step 2.

4) If Finish [I] = true for all I, then the system is in safe state.

Examples:

1) Consider following matrices and calculate a) Need Matrix b) Is the system is in safe state? c) If P1 arrive with request (0, 3, 2, 0), can it be granted immediately?

Process	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	2	0	6	5	2				

Soln:

Here, No. of process (n) = 5 and No. of resources (m) = 4.

At first we need to calculate Need Matrix:

Need Matrix = Maximum Required– Allocation

Process	Allocation			
	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	0

Since Need of resources for P0 is less than that of Available resources, P0 can be successfully executed. So after completion of P0, work (New Available) matrix becomes as follows:

Work Matrix			
A	B	C	D
1	5	2	0
0	0	1	2
1	5	3	2

Now, P1 cannot be executed, since Need matrix of P1 is greater than available Resources.
But here P2 can be executed, since Need matrix of P2 is less than that of Available Matrix. So after completion of P2, work (New Available) matrix becomes as follows:

New Available Matrix			
A	B	C	D
1	5	3	2
1	3	5	4
2	8	8	6

Now here P3 can be executed, since Need matrix of P3 is less than that of Available Matrix. So after completion of P3, work (New Available) matrix becomes as follows:

New Available Matrix			
A	B	C	D
2	8	8	6
0	6	3	2
2	14	12	10

Now here P4 can be executed, since Need matrix of P4 is less than that of Available Matrix. So after completion of P4, work (New Available) matrix becomes as follows:

New Available Matrix			
A	B	C	D
2	14	12	10
1	0	0	0
3	14	12	10

Since, all the processes are successfully executed with the sequence P0, P2, P3, P4, P1. Hence it is in safe state.
Since available matrix is (1, 5, 2, 0), when P1 arrives with request P1 (0, 3, 2, 0), it can be immediately executed.

4. Deadlock Detection and Recovery

Deadlock Detection

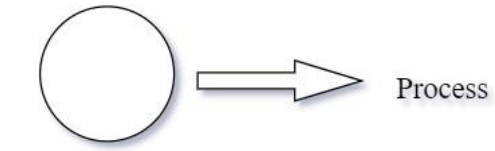
- There is possibility of deadlock if neither of prevention or avoidance method is applied in a system.
- In such case algorithm must be provided for recovering the system from deadlock.

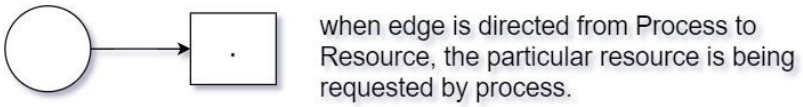
a) Deadlock detection for single instance of each resource type

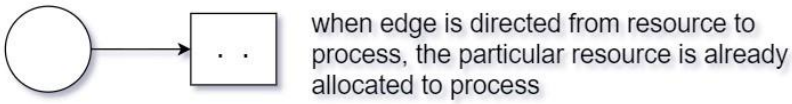
- For single instance of each resource type, deadlock can be detected by constructing Resource allocation graph.
- Resource allocation graph gives the information of whether deadlock exist or not.
- If cycle exist in resource allocation graph, then deadlock may exist, else deadlock does not exist.
- If cycle exist in resource allocation graph, then wait-for graph is constructed from resource allocation graph.
- If cycle exist in wait-for graph then there exist deadlock, otherwise deadlock does not exist.

Resource Allocation Graph:

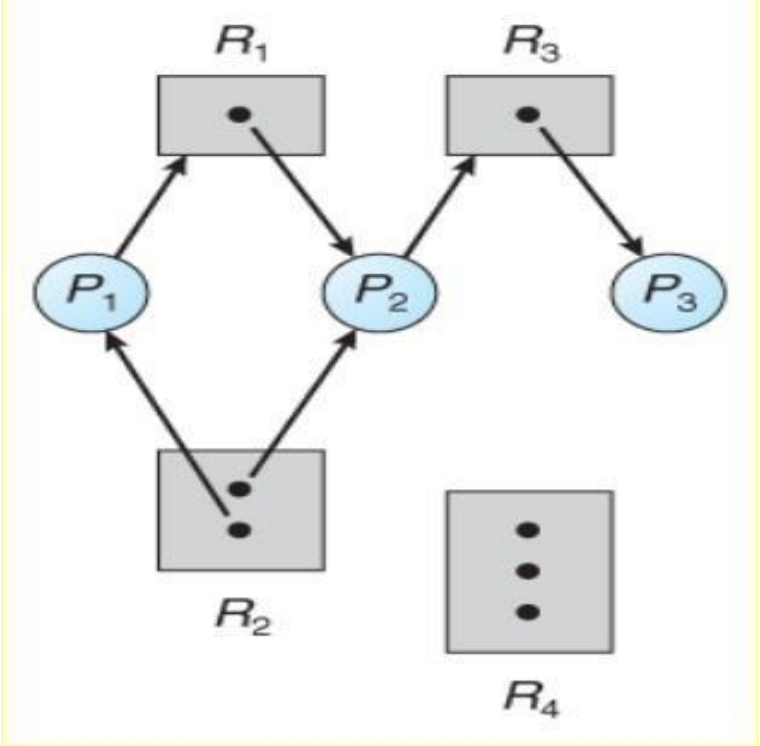
- It is the pictorial representation of the state of a system.
- It consists of all the information which is related to all the instances of the resources means the information about available resources and the resources which the process is being using.
- Resource Allocation graph consist of following Notation:







- A set of resource categories, {R1, R2, R3, RN}, which appear as square nodes on the graph. Dots inside the resource nodes indicate specific instances of the resource. (E.g. two dots might represent two laser printers).
- A set of processes, { P1, P2, P3, . . ., PN }
- Request Edges - A set of directed arcs from Pi to Rj, indicating that process Pi has requested Rj, and is currently waiting for that resource to become available.
- Assignment Edges - A set of directed arcs from Rj to Pi indicating that resource Rj has been allocated to process Pi, and that Pi is currently holding resource Rj.
- Note that a request edge can be converted into an assignment edge by reversing the direction of the arc when the request is granted.



Example of a Resource Allocation Graph:

- P = {P 1, P 2, P 3}
- R = {R 1, R 2, R 3, R 4}
- E = {P 1 →R 1, P 2 →R 3, R 1 →P 2, R 2 →P 1, R 3 →P 3}

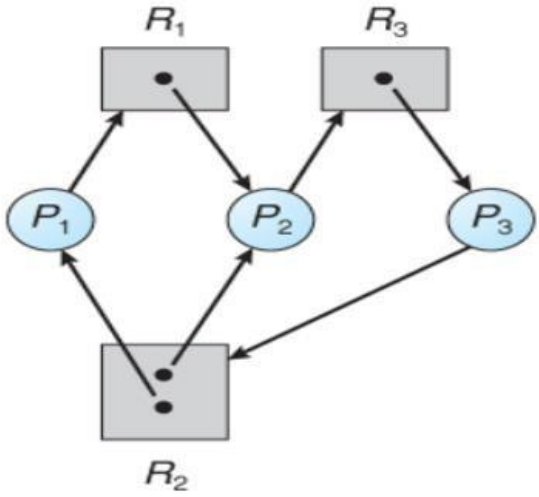
Resource instances:

- One instance of resource type R 1,
- Two instances of resource type R 2,
- One instance of resource type R 3,
- Two instances of resource type R 4

Process states

- P1 is holding an instance of R 2 and waiting for an R 1
- P 2 is holding an R 1 and an R 2 and is waiting for an R 3
- P 3 is holding an R 3

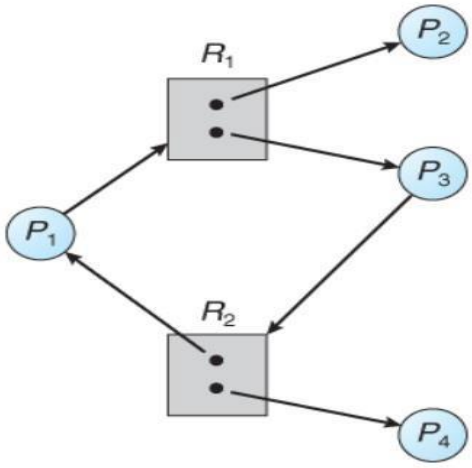
Resource Allocation Graph with a Deadlock



- Suppose P 3 requests an instance of resource type R 2.
- Two cycles exist in the system:
  - P 1 →R 1 P 2 →R 3 →P 3 →R 2 →P 1
  - P 2 →R 3 →P 3 →R 2 →P 2
- Processes P 1, P 2, P 3 are deadlocked.

Process	**Allocation**			**Request**			**Availability**		
	R1	**R2**	**R3**	**R1**	**R2**	**R3**	**R1**	**R2**	**R3**
P1	0	0	1	1	0	0	0	0	0
P2	1	0	1	0	1	0			
P3	0	1	0	0	0	1			

Resource Allocation Graph with a cycle but no Deadlock



- We have a cycle but no deadlock.
- Process P 4 may release its instance of resource type R 2 can then be allocated to P 3, breaking the cycle.

Process	Allocation		Request		Allocation	
	R1	R2	R1	R2	R1	R2

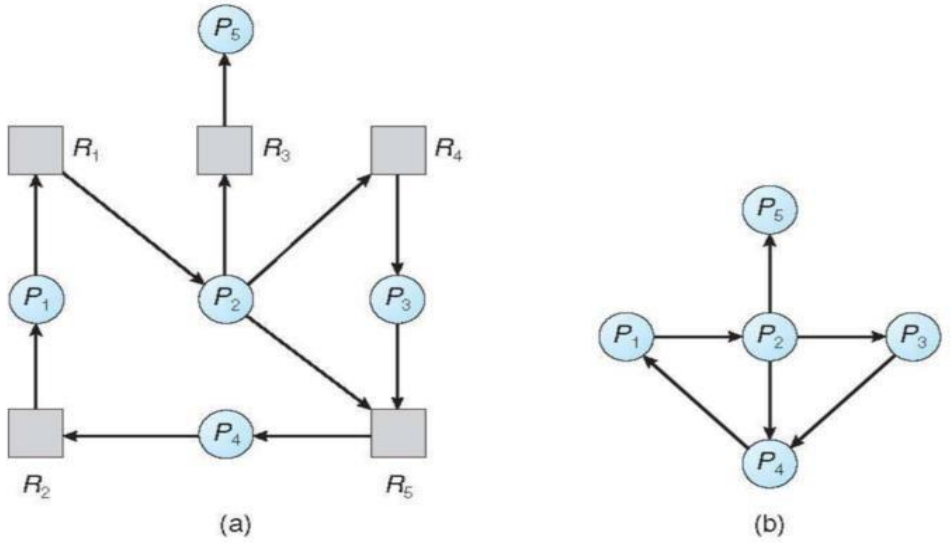
P1	0	1	1	0	0	0
P2	1	0	0	0		
P3	1	0	0	1		

- 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

Wait for graph

- If cycle exist in resource allocation graph, then wait-for graph is constructed from resource allocation graph.
- If cycle exist in wait-for graph then, there exist deadlock, otherwise deadlock does not exist.
- If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a wait-for graph.
- A wait-for graph can be constructed from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.
- An arc from P_i to P_j in a wait-for graph indicates that process Pi is waiting for a resource that process P_j is currently holding.

Example
Here cycle exist in wait-for graph so deadlock exist.



Resource-Allocation Graph Corresponding wait-for graph

Recovery from Deadlock

- Once the system has detected deadlock in the system, some method is needed to recover the system from the Deadlock and continue with processing.
- Methods for recovering from deadlock are:
    a) Recovery through Preemption.
    b) Recovery through Roll Back
    c) Recovery through killing Process

a) Recovery through Preemption:

- We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner.
- Well, choosing a resource which will be snatched is going to be a bit difficult.

b) Recovery through Roll Back

- System passes through various states to get into the deadlock state. The operating system can roll back the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.
- The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

c) Recovery through killing Process

- It is the crudest and simplest way to break deadlock
- Kill one of the process in the deadlock.
- The other process gets its resources.
- Choose process to kill that will yield no ill effect to entire system