DEADLOCKS

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. If process A is waiting for a resource that process B has, and process B is waiting for a resource that process A has, then both the processes are deadlocked.

Resources are of two types

1. Preemptable
2. Non-preemptable

A preemptable resource can be taken away from the process that owns it.
A non-preemptable resource can not be taken from its current owner.

Deadlocks generally occur in non-preemptable resources. Deadlocks that involve preemptable resources can be avoided on resolved by the reallocation of resources. The sequence in which resources are required is:

i. Requested the resource
ii. Use the resource
iii. Release the resource

If the resource is not available, when a process requested for it, then the process will start waiting, so that wherever that resource becomes available, the requesting process can use it. Resource can be hardware device (type drive) or source information (a record in a file).

Deadlock Characterization

A deadlock can occur if following four conditions hold simultaneously.

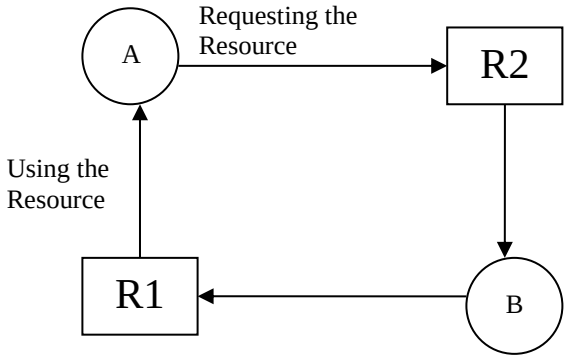
i) **Mutual Exclusion** Only one process should be using the resource and if another process requests that resource, then the requesting process should be delayed until the resource is released.

ii) **Hold and Wait** There should be a process that is holding a resource and is waiting for one or more resources that are in use by other process.

iii) **Non-Preemption** Resources can not be preempted. A process will release the resource only when it has completed its task.

iv) **Circular Wait** There should a circular chain of at least two or more process where each of them is waiting for a resource held by the next process in the chain, e.g. a set { P_0 , P_l, ... , P_n } of waiting processes must exist such that P_0 is waiting for a

resource held by P_1, P_1 is waiting for a resource held by P_2, ... , P_{n-1} is waiting for a resource held by P_n and P_n is waiting for a resource held by P_0.



A deadlock is not possible, if any of these four conditions described earlier are absent. So all the four conditions must hold simultaneously for a deadlock to occur.

Deadlock Prevention

For a deadlock to occur, 4 necessary conditions must hold. If we try to overcome any of the condition out of four, we can prevent the occurrence of a deadlock.

i) Mutual Exclusion

If a resource is not assigned separately to a process, then we will not have deadlocks, but there are some resources that cannot be assigned to more than once process simultaneously. So, to avoid assigning the resources to the process unless that resource is not very necessary and only a few processes should request for such kind of resources.

ii) Hold and Wait

We may overcome on Hold and wait problem if a process request for all the resources that it needs before starting execution. If all the requested resources are available, then the process will be executed otherwise process will wait for those resources to become available.

Although, instead of efficient, this way is time consuming and wastes resources but a deadlock can be prevented like this.

iii) Non-Preemption

If a process that is holding some resources requests another resource that cannot be allocated to it then it will release its resources for use by other processes. Other processes that could have waited for those resources will use these preempted resources and after completion returns them to the system. Now the process that sacrificed its resources for other processes will restart again and will have all the resources available.

This technique can be applied to the resources like memory etc but cannot be used for resources like Printer or Tape drives.

iv) Circular Wait

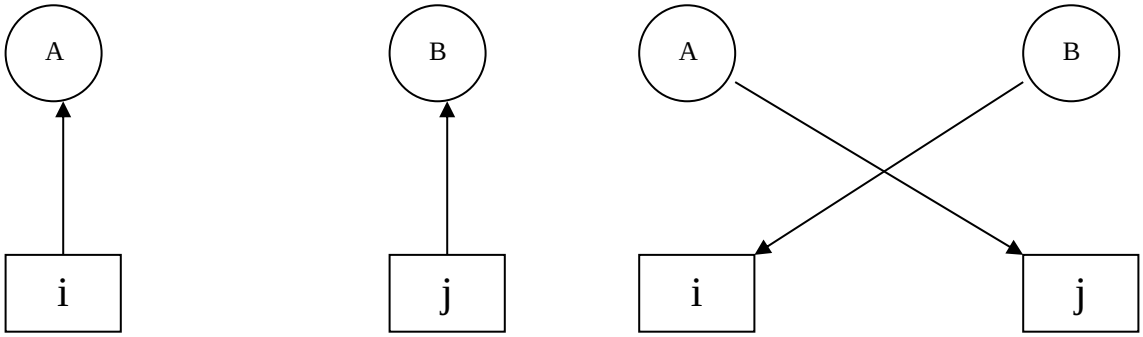
One way of avoiding the circular wait condition is that one process can use only resource at any moment. It if needs another, then it should release the first resource, only then it can use the other resource. This technique cannot be used if we want to print a huge file from tape to printer.

Another technique to avoid circular wait is if we provide a global number for the resources i.e.

1. **Printer**
2. **Plotter**
3. **Tape Drive**
4. **CD-ROM Drive**

And process can request the resources only in numerical order e.g. a process can first request the printer and then the plotter, and it cannot request first a plotter then a printer.

We will never have cycles in this case e.g. "*i*" and "*j*" are two resources with different numbers, Process A requested resource "*i*" and process B requested resource "*j*". Now dead lock can occur if process A requests resource "*j*" and process B requests resource "*i*".



Holding the Resource **Requesting the Resource**

As both resources "*i*" and "*j*" have different number, so if "*i*" > "*j*" then A will not be allowed to request "*j*" or if "*i*" < "*j*" then B ill not be allowed to request "*i*". So deadlock will not occur.

Deadlock Avoidance

In deadlock prevention we try to overcome any of the 4 conditions that are responsible for the occurrence of deadlock. Deadlocks can be prevented by this way but the device utilization is reduced in this method.

Another method for avoiding deadlocks is to get information about the resources that how they are requested. If we know the sequence of resource requests and releases, then we can decide that for which resource the process should wait. So, whenever, a process requests for a resource then Operating System checks the available resources and the resources allocated to other processes. This checking enables the Operating System to

decide whether the request can be satisfied or put to wait in order to avoid the deadlocks, possible in future.

Different algorithms can be used that avoid deadlocks to occur. A simple method is that all the processes declare the maximum number of resources that it may need during execution. If we have the information about all the processes that which can need which resource then we can develop an algorithm that ensures that the system will never enter into deadlock state. The deadlock avoidance algorithm checks the resource allocation state and ensure that there can never be a circular wait-condition.

A state is safe if the system can allocate resources to each process (up to its maximum) through same way and still avoid a deadlock, so safe state is not a deadlock state.

Similarly, when the Operating System is unable to prevent the process from requesting resources then a deadlock occurs and is in unsafe state. Although not all-unsafe state lead to deadlocks but chances are that the unsafe state will result a deadlock to occur.

A system has 12 Tape drives and three processes P1, P2 and P3. Maximum tape drive needed by P1 is 10, 4 for P2 and 9 to P3. At time T0, process P1 is holding 5, P2 is holding 2 and P3 is possessing 2 Tape drives as shown in the following table.

	Maximum Needs	**Current Needs**
P1	10	5
P2	4	2
P3	9	2

The sequence **<P2, P1, P3>** is satisfying the safety condition so the state of the system is safe. Out of 12 tape drives 3 are free as 5 are in use by P1, 2 are in use by P2 and 2 are in use by P3. Process P2 can use 2 more tape drives although it is already using 2 tape drives and will be allotted the 2 tape drives as 3 are free. After use all the tape drives will be returned and the free tape drives will now be 5, that can be used by P1 and then by P3. So at all the time of allocating the tape drives to processes our system remains in a safe state.

A system very easily can go from a safe state to unsafe state if resources are not allocated carefully. At time T1, process P3 requests 1 tape drive although it is already using 2. As free tape drive are 3 so the request of P3 is fulfilled and 1 tape drive is allocated to P3. The moment system deviates from its defined safety sequence of **<P2, P1, P3>**, its state is changed from safe to unsafe state. At this time, if process P2 is allocated all its tape drives and when it returns them then the system will have only 4 tape drives available. Now process P1 requests for 5 tape drives and only 4 are free so process P1 starts waiting. Similarly process P3 request 6 tape drives but only 4 are available. So process P3 also starts waiting, both will be waiting forever as the deadlock has occurred.

The problem in above example started by granting the request of 1 tape drive to process P3. Had P3 been put to wait for the request of 1 tape drive until either of the other processes had finished and released their resources, then we could have avoided the deadlock.

Banker's Algorithm (Deadlock Avoidance)

Banker's algorithm is used for deadlock avoidance and determines the safe state for a given set of processes. When a new process enters the system, it declares the maximum number of instances of each resource type that it may need during execution. This number should not exceed the total number of resources in the system. When a process requests for resources, the system must determine whether the allocation of these resources will leave the system in a safe state or not. If it will leave the system in a safe state, the resources are allocated; otherwise, the process waits until some other process releases enough resources.

Data structures required for implementation of banker's algorithm are:

i) Resource **Allocation** matrix
ii) **Maximum** resource request matrix
iii) **Available** resource vector
iv) **Need**ed resource matrix

In order to understand banker's algorithm, consider a system with 5 Processes <P1, P2, P3, P4, P5> and 3 Resource types <R1, R2, R3>. Resource R1 has 10 instances, Resource R2 has 5 instances and Resource R3 has 7 instances. The following snapshot of the system is given below:

	Resource **Allocation** Matrix					**Maximum** Resource Request Matrix		
	R1	**R2**	**R3**			**R1**	**R2**	**R3**
P1	0	1	0		**P1**	7	5	3
P2	2	0	0		**P2**	3	2	2
P3	3	0	2		**P3**	9	0	2
P4	2	1	1		**P4**	2	2	2
P5	0	0	2		**P5**	4	3	3

i) Find Available Resource Vector
ii) Find the Need Matrix
iii) Find whether the system is in SAFE STATE or not
iv) If the system is in SAFE STATE, find the SAFETY SEQUENCE

Solution:

i) Available Resource Vector = Total - Allocation

R1 = 10 – 7 >> 3
R2 = 5 – 2 >> 2
R3 = 7 – 5 >> 2

Available Resource Vector		
R1	**R2**	**R3**
3	3	2

ii) Need Matrix = Maximum - Allocation

	Need Matrix (Max – Allocation)		
	R1	**R2**	**R3**
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

iii) To find whether the system is in SAFE STATE or not, we'll have to apply Safety Algorithm, e.g.

if (need of process$_i$ ≤ available resource vector)
{ execute process$_i$
new available = available + allocation
}
else
{ do not execute process$_i$
move on to next process$_{i+1}$
}

P1 if [need P1 ≤ available] (False)
[7 4 3 ≤ 3 3 2]

P2 if [need P2 ≤ available] **(True)**
[1 2 2 ≤ 3 3 2]
then
new available = available + allocation
= 3 3 2 + 2 0 0
= 5 3 2

New Available Resource Vector		
R1	**R2**	**R3**
5	3	2

P3 if [need P3 ≤ available] (False)
[6 0 0 ≤ 5 3 2]

P4 if [need P4 ≤ available] **(True)**
[0 1 1 ≤ 5 3 2]
then

new available = available + allocation
= 5 3 2 + 2 1 1
= 7 4 3

New Available Resource Vector		
R1	**R2**	**R3**
7	4	3

P5 if [need P5 ≤ available] **(True)**
[4 3 1 ≤ 7 4 3]
then
new available = available + allocation
= 7 4 3 + 0 0 2
= 7 4 5

New Available Resource Vector		
R1	**R2**	**R3**
7	4	5

P1 if [need P1 ≤ available] **(True)**
[7 4 3 ≤ 7 4 5]
then
new available = available + allocation
= 7 4 5 + 0 1 0
= 7 5 5

New Available Resource Vector		
R1	**R2**	**R3**
7	5	5

P3 if [need P3 ≤ available] **(True)**
[6 0 0 ≤ 7 5 5]
then
new available = available + allocation
= 7 5 5 + 3 0 2
= 10 5 7

New Available Resource Vector		
R1	**R2**	**R3**
10	5	7

As all the processes executed successfully so the system is in a SAFE STATE.

iv) SAFETY SEQUENCE identified during execution of the processes is:

SAFETY SEQUENCE < P2, P4, P5, P1, P3 >

Java based Banker's Algorithm Implementation: SAMPLE OUTPUT

Enter no. of processes and resources: 5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
1 3 3
Enter available matrix:
3 3 2

SAFETY SEQUENCE - <P2, P4, P5, P1, P3>
All processes allocated safely

Java based Banker's Algorithm Implementation: SAMPLE OUTPUT

Enter no. of processes and resources: 3 1
Enter allocation matrix:
5
2
2
Enter max matrix:
10
4
9
Enter available matrix:
3

SAFETY SEQUENCE - <P2, P1, P3>
All processes allocated safely

Deadlock Detection

In deadlock detection we do not try to prevent deadlocks from occurring, so when a deadlock occurs, the system detects that the deadlock has occurred and takes action to recover from the Deadlock State. There are different ways through which we detect the deadlocks in the system and then try to recover the system from Deadlock State.

Different techniques can be used for detecting a deadlock, for example when we have a single resource of each resource type, then deadlock detection method will be different and when we have multiple resources of each resource type the deadlock detection method will be different. So, the basic function in deadlock detection is that check the system to find whether a deadlock has occurred or not. If the deadlock is detected then we use the techniques to recover from the detected deadlock.

Deadlock Detection in Single Resource of Each Resource Type

In this case we detect the deadlock from the system that has only one resource of each resource type, i.e. one tape drive, one printer one plotter etc.

A resource graph is constructed in order to detect the deadlocks from a system having single resource of each resource type. If the graph contains one or more cycles it means that a deadlock exists. Any process or processes that are the part of the cycle are deadlock.

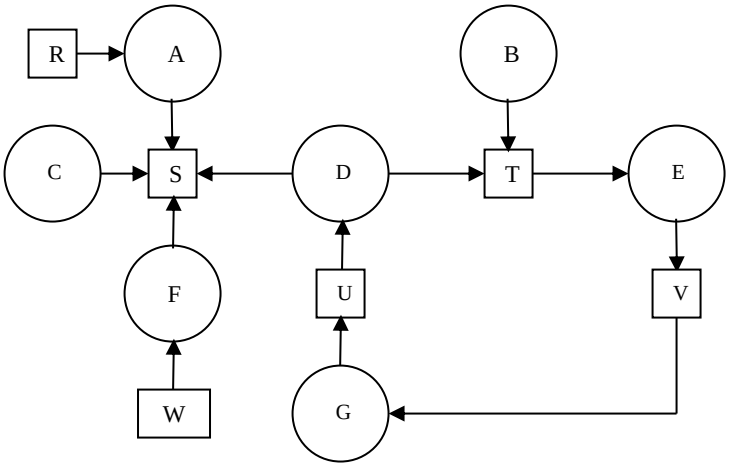
A system has 7 processes A – G, and 6 resources R – W. Resource ownership for this system is

1. Process A holds R and wants S.
2. Process B holds nothing and wants T.
3. Process C holds nothing but wants S.
4. Process D holds U and also wants T & S.
5. Process E holds T and wants V.
6. Process F holds W and wants S.
7. Process G holds V and requests U.

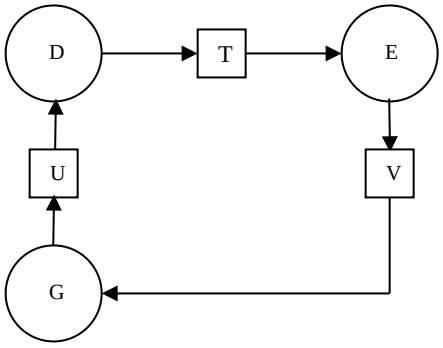
On drawing the resource graph for these processes and resources it become clear that a cycle exits in this graph. So, Process D, E and G are deadlocked. Process A, B, C & F are not deadlocked. Resource S can be allocated to Process A, C and F and after completion it will be returned so that other processes can use it.

To implement this idea in actual systems we use some algorithms for the detection of deadlocks. One ways is that each node is taken in its turn as a root and perform a depth-first search on it. If it comes back to a node that is already checked, it means that a cycle exists, i.e. like this we have detected the deadlocked, same process is repeated for all nodes and if there is no cycle then it means there is no deadlock.

L = [R, A]
L = [B, T, E, V, G, U, D, T]



Resource Graph



Cycle extracted from Resource Graph

Deadlock Detection in
Multiple Resources of Each Resource Type

For detecting a deadlock when we have multiple resources of each resource type, there is a matrix based algorithm.

Let there are n processes i.e. P1 through Pn. Let the number of resource class be "m", with E1 resources of class 1, E2 resources of class 2 and generally Ei resources of class "i" ($1 \leq i \leq m$).

E is the existing resource vector. It gives the total number of instances of each resource in existence i.e. if class 1 is Tape drives then E1=2 means that the system has 2 Tape drives.

At any instant, some of the resources are assigned and are not available.

Let A be the available resource vector, with Ai giving the number of instances of resource "i" that are currently available i.e. unassigned. If both of the Tape drives are assigned than A1 will be 0.

Now we need two arrays, C, the current allocation matrix, and R, the request matrix. The "i"th row of C tells how many instances of each resource class Pi currently holds. So, Cij is the number of instances of resource "j" that are held by process "i". Similarly, Rij is the number of instances of resource "j" that Pi wants.

So, the 4 data structures are

i)

Resources in Existence

(E1, E2, E3, …..Em)

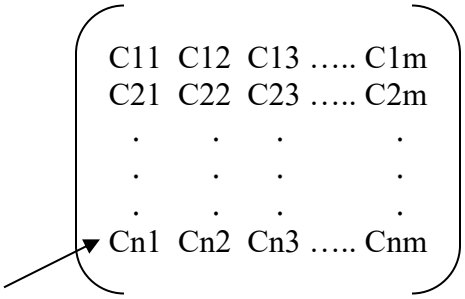
ii)

Resources Available

(A1, A2, A3, …..Am)

iii)

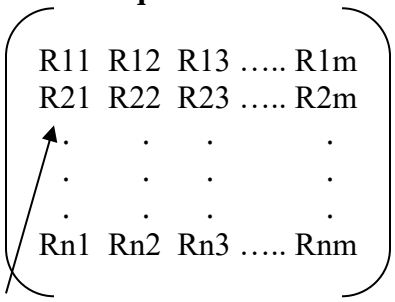
Current Allocation Matrix



Row "n" is the current allocation to Process n.

iv)

Request Matrix



Row "2" is what Process 2 needs.

So, these 4 data structures are needed for detecting a deadlock when we have multiple resources of each resource type.

Every resource that we have will either be allocated or will be available. So,

$$\sum_{i=1}^{m} Cij + Aj = Ej$$

i.e. if we add all the instances of resource "j" that have been allocated and all its instances that are available, then the result will be the number of instance of that resource class.

The deadlock detection algorithm with multiple resource of each resource type is based on comparing the vectors.

Each process is initially unmarked and as the algorithm progresses, processes will be marked indicating that they are able to complete, so, they are not deadlocked. When the algorithm terminates, any unmarked processes are known to be deadlocked.

The deadlock detection algorithm is

1. Look for an unmarked process, Pi for which the "i"th row of R is less than A.
2. If such a process is found, add the "I"th row of C to A, mark the process and go back to step 1
3. If no such process exists, the algorithm terminates.

When algorithm finishes, all the unmarked processes, if any are deadlocked.

Example

We have 3 Processes and 4 Resource classes which are labeled as

1. Tape drives
2. Plotters
3. Printers
4. CD-ROM

Process 1 has 1 Printer
Process 2 has 2 Tape drives and 1 CD-ROM
Process 3 has 1 Plotter and 2 Printers

Now, each process needs additional resource that is shown in matrix R.

Resources in Existence

E = (4 2 3 1)

Tape drives Plotters Printers CD-ROM

Resources Available

A = (2 1 0 0)

Tape drives Plotters Printers CD-ROM

Current Allocation Matrix

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$

Request Matrix

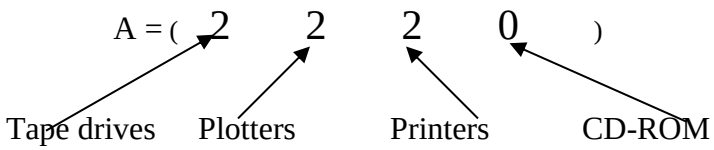
$$R = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix}$$

To run the deadlock detection algorithm, we look for a process whose resource request can be satisfied.

The request of 1st Process can not be satisfied because there is no CD-ROM available. Similarly, the request of 2nd process can also not be satisfied because there is no Printer free, but the request of 3rd Process can be satisfied, So, Process 3 runs successfully.

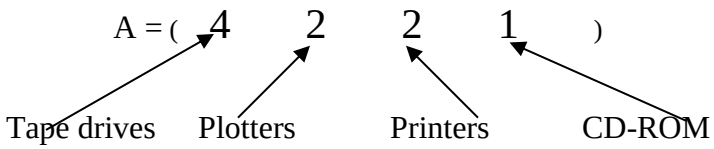
When Process 3 terminates, it returns all its resources. So, resources available now are

Resources Available



Now, Process 2 can also run and after its completion the resources available are

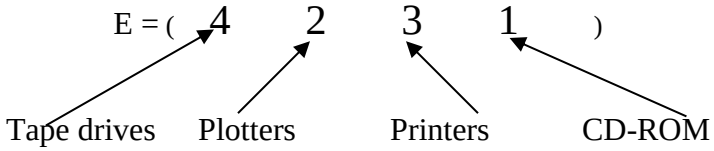
Resources Available



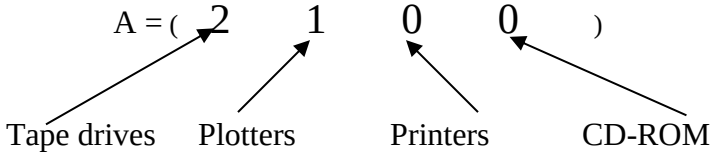
and Process 1 can also run now. So, there is no deadlock in the system.

By making a minor change in the above defined situation and that is when Process 2 needs a CD-ROM and 2 Tape drives and 1 Plotter. i.e. the situation now is

Resources in Existence



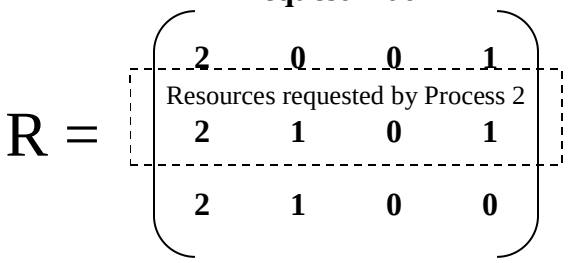
Resources Available



Current Allocation Matrix

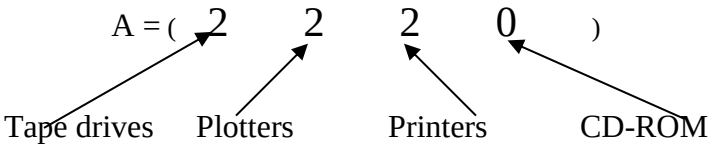
$$C = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{0} \end{pmatrix}$$

Request Matrix



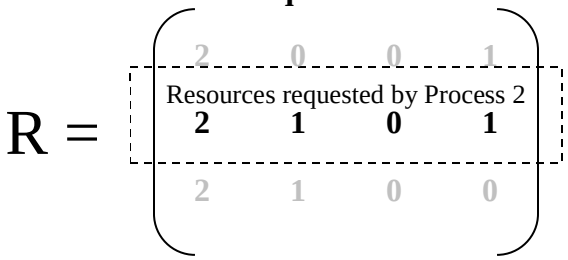
Now, when Process 3 completes first and returns its resources, then available resources becomes

Resources Available



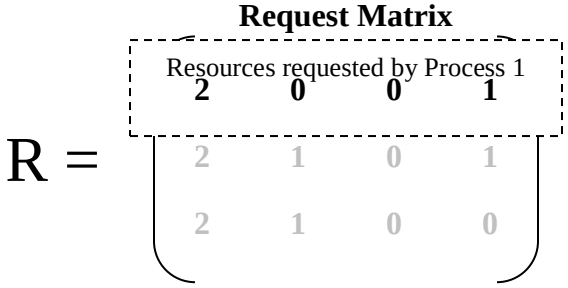
Whereas the request matrix of Process 2 is

Request Matrix



and request matrix of Process 1 is

Request Matrix



The request of both the processes i.e. Process 1 and Process 2 can not be completed within the available resources. So, a deadlock is obvious in Process 1 and 2.

Recovery from Deadlock

When the system has detected the deadlock then steps should be taken in order to recover the system from the Deadlock State. One way is to inform the operator that the deadlock has occurred, so that he can terminate the processes or process that are involved in the deadlock. So the operator can recover the system from the Deadlock State manually.

There are two ways through which a deadlock can be broken. One is the process termination and the other is resource preemption.

Process Termination

One way of breaking a deadlock is to terminate one or all the processes that are deadlocked. The following methods are used for this purpose

The method of **terminating all the deadlocked processes** will break the deadlock state of the system but this way will be wastage of time and resources as the processes were in execution for a long time and when they are terminated before completion, next time the same process will take more time to execute.

A better way in this situation is to **terminate one process at a time until a deadlock cycle is eliminated**. After terminating one process, the deadlock detection algorithm is used to detect whether the remaining processes are in deadlock or not. Process selected for termination should be the one that causes minimum ill effects. There are many factors on which a process should be selected for termination

i) The priority of the process.
ii) Time that a process has taken for computation
iii) Types and number of resources that a process has used and whether the resources are preempted.
iv) How many more resources are needed to complete the process.
v) How many processes will be terminated.
vi) The process is interactive or batch.

Resource Preemption

The second way of breaking the deadlock is to preempt some resources and give them to other processes till the deadlock is broken. The following three issues are involved in the resource preemption of deadlock process.

Process Selection The issues involved in this case are the number of resources, a deadlock process is holding and the amount of time a deadlock process has consumed during its execution.

Rollback If possible then after the preemption of resources from a process, that process should be rollback to a safe state so that later we can start that process from that state.

Starvation We ensure that the process selected for the preemption of resources will only be for a few time i.e. not always the same process will be selected.