4 Deadlock

4.1 BASICS OF DEADLOCK

In an environment of multi-programming, many processes compete for a finite number of resources. A process gets into a waiting state if the resource the process requested or needs is not available. A deadlock happens when such a process can never change its state to a ready state since the resource is held by other processes that are in a waiting state.

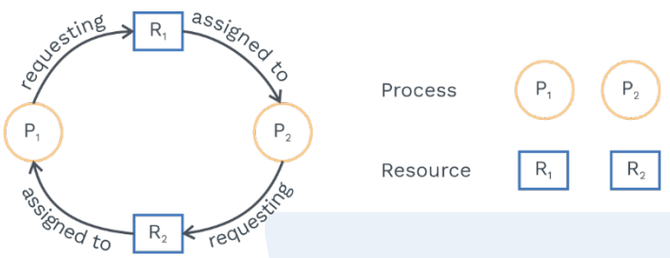


Fig. 4.1 Deadlock State

System model:

1) A system consists of a number of resources and processes.

2) The number of resources is finite and is to be shared between the processes.

3) Some resources are further divided into instances that are identical.

4) If a process requests for some type of resource any one of the instances of the same type of resource can be allocated.

Examples– CPU cycles, memory, files and I/O devices.

5) If the request is not granted, there is a problem with the definition of the resource type, and the instances are also not identical in the resource type.

Example– If there are two printers on the same floor of an office, suppose the 12th floor, employees working on the 12th floor can use any of the two printers.

6) Consider another case, if one printer is on the 12th floor and another is on the ground floor, both can't be treated equivalent and have to be defined with different resource classes.

7) A process needs to request if a resource/resources are needed to complete the assigned task. The process needs to release the same resources after its use. The number of resources a process requests is always equal or less than the total number of resources in the system.

Example– A process cannot ask for three printers if only two printers are available in the system.

8) A process under normal mode of operation utilizes a resource in the below sequence-

a) Request:
The process asks for a resource, if the resource asked for is not available at that time (maybe allocated to any other process), then the process gets into waiting state. It waits until it acquires the resource asked for.

b) Use:
The process makes use of resources to complete the task.
Example – A process uses the printer to print.

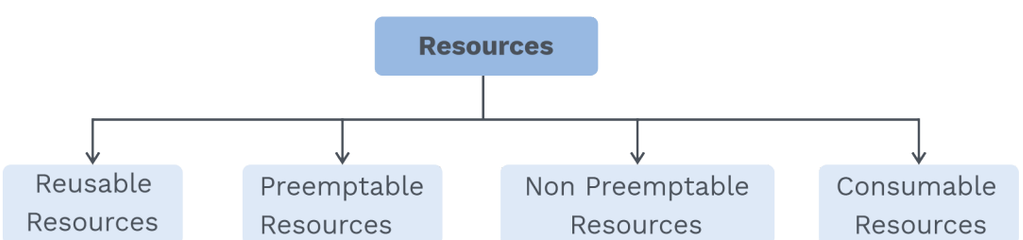
c) Release:
The process frees the resource.
The request() and release() function calls of the resource are system calls like open(), close(), allocate() and free() memory system calls.

Note:

Request and release of resources that are not managed by the operating system can be accomplished through wait() and signal() operations on semaphore or through acquisition and release of a mutex lock.

9) A set of processes are said to be in a deadlock state if every process is waiting for happening for an event which would never happen.

Types of resources:



a) Reusable resources:
It is used only by one process at a time and a does not get depleted by that use.
eg.: CPU, I/O, memory (physical/secondary).

b) Consumable resources:
It is created (produced) and destroyed (consumed).
eg.: Interrupts, signals, I/O buffers, etc.

c) Preemptable resources:
These are the resources which can be taken away from the process owning it, with no ill effect.
eg.: Memory, buffers, CPU, etc.

d) Non-preemptable resources:
These are the resources which cannot be taken away from the current owning process without causing the computation to fail.
e.g.: Tape drives, printers and optical scanners.

Deadlock characterization:
In a deadlock, a process never finishes execution and releases the resources required by another process which is in a deadlock state, thus preventing the execution of other processes.

Necessary conditions for deadlock:
Deadlock occurs if all the below four conditions occur together.

1) **Mutual exclusion**
   No two processes share the only instance of a resource. If the resource a process asked for is with another process, asking process has to wait till the resource gets freed.
2) **Hold and wait**
   A process holds atleast one resource before it requests for another resource which is already held by another process.
3) **No pre-emption**
   A resource cannot be preempted from a process all of a sudden until the process itself releases it.
4) **Circular wait**
   A set $\{P_0, P_1, \ldots\ldots\ldots, P_n\}$ 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, $\ldots\ldots\ldots\ldots$, P_{n-1} is waiting for a resource held by P_n, and P_n is waiting for a resource held by P_0.
   All the four conditions must hold for a deadlock to a occur.

Note:

The circular-wait condition implies the hold and wait condition, which means four conditions are not completely independent.

Grey Matter Alert!

Suppose all the four necessary conditions be represented as (1, 2, 3 and 4). So for a deadlock to happen, The propositional Logic (1 ∧ 2 ∧ 3 ∧ 4) must be true and if ¬ (1 1 ∧ 2 1 ∧ 3 1 ∧ 4) is true, then deadlock does not exist.

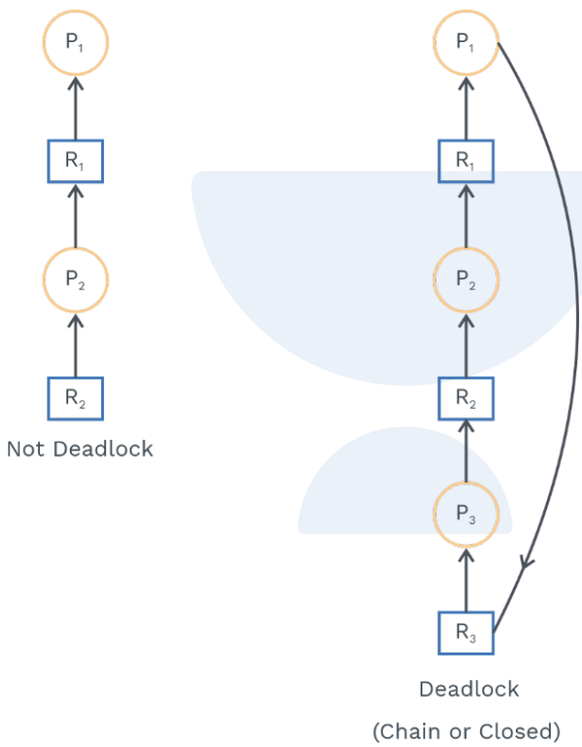


Fig. 4.2 Representation of Deadlock

4.2 RESOURCE ALLOCATION GRAPH (RAG)

1) Deadlocks can be represented more precisely using a resource allocation graph. The resource allocation graph contains two types of vertices, the active processes in the system and the resource type.

2) The graph used to represent the resource allocation graph is a directed graph. The arrow from process to resource denotes the request edge, and the arrow from resource to process denotes the assignment edge.

Fig. 4.3 Request Edge and Assignment Edge

Note:

In resource allocation graph, the process node is denoted by a circle and the resource node is denoted by a rectangle.

3) When a process P_i requests for a resource R_j, a request edge is added between the P_i and R_j; this edge is made into an assignment edge after checking if the resource R_j is free (checked by OS). The process P_i has to wait until the resource is released by the assigned process and their assignment edge is deleted.

Example:

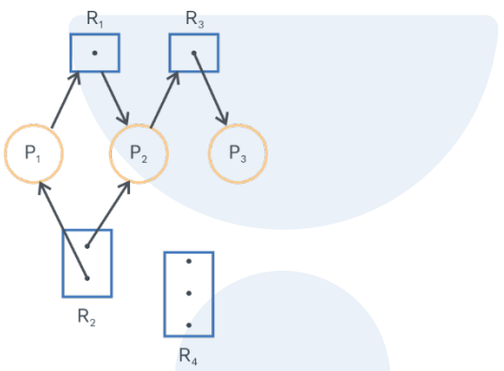


Fig. 4.4 Resource Allocation Graph

Graph elements:

1) P (Processes) = $\{P_1, P_2, P_3\}$

2) R (Resources) = $\{R_1, R_2, R_3, R_4\}$

3) E (Edges) = $\{P_1 \rightarrow R_1, P_2 \rightarrow R_3$
$R_1 \rightarrow P_2, R_2 \rightarrow P_2$
$R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

Instances of resources:

1) R_1 has one instance

2) R_2 has two instances

3) R_3 has one instance

4) R_4 has three instances

Note:

The instances of a resource are denoted by dots inside the rectangular box (resource).

Process states:

1) Process P_1 is having an instance of resource type R_2 and waiting on R_1.

2) Process P_2 is having an instance of resource type R_1 and R_2, and waiting on R_3.

3) Process P_3 is having an instance of R_3.

4) It can be shown that if a graph has no cycles, then no process is in a deadlock state, but if a graph has a cycle, then a deadlock may or may not exist.

Note:

1) Cycle in a resource allocation graph (RAG) is a necessary but not sufficient condition for the existence of deadlock in the case of multi-instance resources.

2) Cycle in a resource allocation graph (RAG) with only single instance resources is necessary as well as sufficient condition for the existence of deadlock.

Grey Matter Alert!

Sufficient and necessary:

Suppose we have two events A and B. Now "If A then B" condition is TRUE then the following two statements can be concluded.

1) A is a sufficient condition for B whenever the occurrence of A is all that is needed for the occurrence of B.

2) B is a necessary condition for A whenever A cannot occur without the occurrence of B.

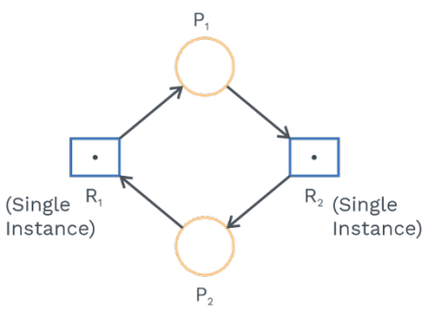


Fig. 4.5 Cycle in RAG with Single Instances

In above figure, cycle exist

$$P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$$

So in RAG with single instance resources, we know the cycle is necessary as well as sufficient, so above RAG has a deadlock.

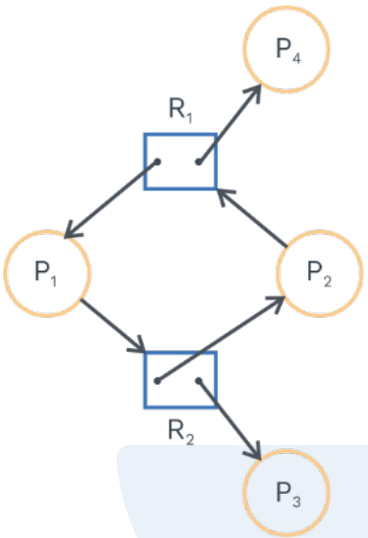


Fig. 4.6 Cycle in RAG of Multi Instance Resources (Cycle but no Deadlock).

In the above figure of resource allocation graph with multiple instances resources,

$$\text{Cycle exist : } P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$$

So, the necessary condition is met, so deadlock may or may not occur but to confirm the existence of deadlock, we need to check whether this cycle is sufficient in the above graph or not as cycle is not sufficient in multi-instance RAG. In the figure, we can see one instance of R_1 and one instance of R_2 is allocated to P_4 and P_3, respectively. So when P_3 and P_4 are done with their work, they will release their instances, and other processes will acquire them, thus breaking the cycle.

Grey Matter Alert!

If a resource-allocation graph does not have a cycle, then the system is not in a deadlocked state. If there is a cycle, then the system may or may not be in a deadlocked state. This observation is crucial when we deal with the deadlock problem.

SOLVED EXAMPLES

Q1 **Suppose a system with only one type of resource and three processes and each process request a maximum of two instances of resource. Find, maximum number of resource instances so that system goes into deadlock. Also, find the number of resource instances so that system doesn't go into deadlock. What is the sum of both the numbers found?**

Sol: **Range: 7–7**

i) Let processes be P_1, P_2, P_3 each acquired one copy of resource each, so every process needs one copy more, but if the system has only three copies of resource, then deadlock happens.

P1 → One copy of resource

P2 → One copy of resource

P3 → One copy of resource

Three copies of resource

So maximum three copies for deadlock.

ii) Now for minimum resources for deadlock prevention, we can provide just one more copy to the system so that one process satisfies with its need and, after execution, deallocate its resource, and some other process acquires. So four copies of resources for deadlock prevention.

So, the sum of both the numbers is 3 + 4 = 7.

Q2 **Consider the system with 'N' processes and '10' tape drives (resources). If each process requires '2' tape drives to complete their execution, then what is the maximum value of 'N' which ensures deadlock-free operation.**

a) 3

b) 6

c) 9

d) 5

Sol: **Option: c)**

There are total '10' tape drives. Each process requires = '2' tape drives. First, we will allocate one resource to each process, leaving one resource aside.

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
1	1	1	1	1	1	1	1	1

⟶ All processes are waiting for second resource

Now add that last resource to any of the processes then it will guarantee deadlock-

free operation.

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
1	1	1	1	1	1	1	1	1
1								

P_1 will execute and deallocate resources after completion.

Q3 **Consider a system with five processes. Where each process requires two tape drives to complete their execution then what is the minimum number of resources required to ensure deadlock-free operation.**

a) 2

b) 4

c) 6

d) 10

Sol: **Option: c)**

Number of resources for 'n' processes for deadlock-free operation where each process requires 'k' instances of resources of resource z

$n (k - 1) + 1$

$5 (2 - 1) + 1 = 6 \quad [n = 5, k = 2]$

4.3 METHODS FOR HANDLING DEADLOCKS

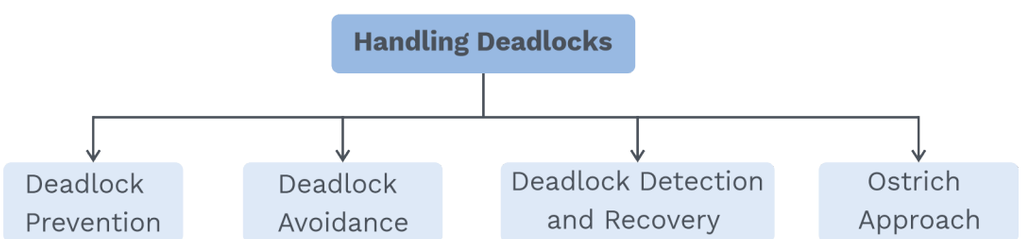
There are many ways to solve deadlocks in your system.

1) One approach involves assuming that deadlock will never happen; this is known as ignorance (Ostrich method).

2) One approach puts attention well before occurring, which is known as prevention.

3) One approach allows the system to get into deadlock and then solves it; this is known as deadlock detection and recovery.

4) One other approach involves checking whether the system gets into deadlock a step ahead and avoiding it. This approach is named as deadlock avoidance.



If a deadlock has to occur, there are four necessary conditions that have to be satisfied. Deadlock can be removed by making sure that atleast one of these conditions doesn't hold. The four necessary conditions are:

a) Mutual exclusion:

1) Sharable resources do not get into deadlock since they need not be accessed mutually exclusive. We need atleast one resource that must be non–sharable so that mutual exclusiveness can apply.

2) Read–only files are a good example of a sharable resource, i.e., if several processes try to read the read–only file, permission is always granted.

Note:

We cannot always prevent deadlock by denying the mutual-exclusion condition because some resources are intrinsically non-sharable. For example, mutex lock.

b) Hold and wait:

The problem arises when a process is holding resources and waiting for the resources currently held by other processes.

This can be violated in three ways :

1) **Conservative approach:** The process has to acquire all the resources before starting its task. But, it is very less efficient and cannot be implemented because the resources needed cannot be known well ahead.

2) **Violating hold:** If a process holds some resources, it has to release the resources held to make a fresh request.

3) **Violating wait:** After a particular waiting period, if the process is not getting alloted the requesting resources, it has to release the currently held resources.

Examples:

1) Let's consider a process that copies data from CD–drive to a file on disk and then prints the file. If all resources must be requested at the beginning of the process, then CD–drive, disk file and printer will be allocated to the process, so it will hold the printer for the entire duration of its execution, even though it needs the printer in the end. Here, wait is not satisfying.

2) Let's consider another process, which does the same job as the above process but in a different manner as it requests initially only the CD–drive and disk file. It copies from the drive to file and then releases both the resources. Then the process request the disk file and printer to print, and when done, it again releases both resources and terminates.

c) No preemption

No resource can be preempted from the process it is alloted. To violate this, a process will release all the resources if it is not alloted the resources it has requested for. The process is restarted only when it can access both old and new resources.

Grey Matter Alert!

1) Another method could also be that if the process requests some resources, we first check whether they are available; if yes, they are allocated.

2) If resources are not available, then we check whether they are allocated to some other process that is also waiting for some resources. If yes, we preempt the resources from the waiting process to requesting process.

3) If the resources are neither held by the waiting process nor available, the requesting process must wait and while it is waiting, some of the allocated processes could also be preempted explained in the above point, if another process requests them. The process will only restart when it regains all the previously allocated resources and the ones it is waiting for, from the other process.

d) Circular wait

If process and resources dependency makes a circle resulting in deadlock, it is known as a circular wait. This can be violated by allowing a process to take resources in increasing enumeration order strictly.

1) All the resources are identified uniquely.

2) Never allow a process to request a lower number of resources than the last one requested and allocated.

Example: Let's suppose we have eight resources with resource ID as follows.

Resource	Resource ID
R_1	10
R_2	5
R_3	3
R_4	8
R_5	9
R_6	12

Resource	Resource ID
R_7	15
R_8	11

Now a process 'P' requests resources in the following sequences of resource ID – 3, 9, 10, 11, 12, 15, 8, 3. So, if circular wait does not hold, then process can request the resources in increasingly linear order. So, resource ID –

When process request resource with ID 3 again, It will not allow us to break the cycle (circular wait).

Previous Years' Question (GATE-2018)

Consider a system with three processes that share four instances of the same resource type. Each process can request a maximum of K instances. Resources can be requested and released only one at a time. The largest value of K that will always avoid deadlock is .....

Sol: 2.0

4.4 DEADLOCK AVOIDANCE

All the above-discussed approaches are used to avoid deadlock, but they bring disadvantages with them. These disadvantages include low device utilization and reduced overall system throughput.

Another approach to overcome these disadvantages needs more information about how resources are requested.

Example:

Let's consider there are two resources R1 and R2 in the system. The system needs to know the order in which the resources are held before releasing them. With this knowledge, the system can decide whether a process should wait for its resources (if they are currently available or not) at a particular time.

Grey Matter Alert!

Various algorithms that use the above approach differ in the amount and type of information needed. The most algorithm requires that each process declare the maximum number of resources of each type that it might need. Given this information, algorithm ensures that the system never enters a deadlocked state. A deadlock avoidance algorithm dynamically examines the RAG state to ensure that circular wait conditions can never exist.

Note:

The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

4.5 SAFE AND UNSAFE STATES

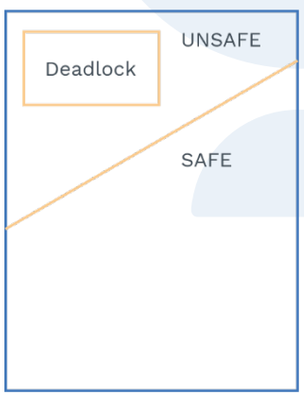


Fig. 4.7 Safe, unsafe and Deadlock State

Safe state:

Definition

There is atleast one sequence of processes that does not result in deadlock.

1) No process is deadlocked, and there exists no possible sequence of future requests in which deadlock could occur.
2) No process is deadlocked, and the current state will not lead to a deadlock state.
3) A system is in a safe state only if there exists a safe sequence.

Definition

Safe sequence: A sequence of processes <P_1,P_2,P_3, ... P"> is a safe s quence for the current allocation state if, for each P_1, the resource that P; can still request, can be satisfied by the currently available resources.

Unsafe state:

It is not a safe state. There is a possibility of deadlock with some sequence.

Note:

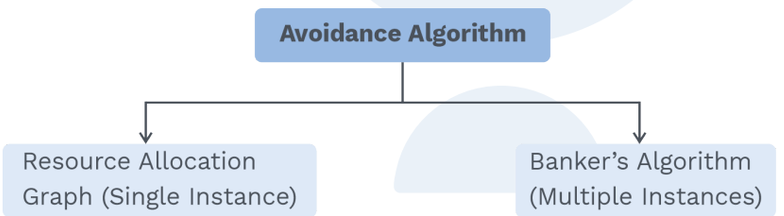
1) If a system is in a safe state, then the system has no deadlock.

2) If a system is in a unsafe state, then there is a possibility of deadlock.

3) Do not start a process if its maximum requirement might lead to deadlock.

4) Deadlock avoidance ensures that a system will never enter

Deadlock avoidance algorithm:



Resource allocation graph avoidance:

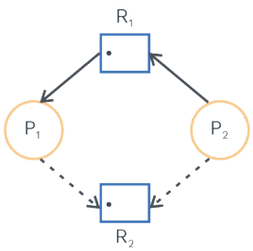
1) This algorithm can be used if we have only one instance of each resource type.

2) **Claim edge:** An edge P---->R represents that the process may request the resource R in the near future.

3) **Assignment edge:** When the process requests the resource then claim edge is changed to assignment edge i.e., P->R
The assignment edge is changed to the claim edge right after the process releases the resource.

4) Resources have to be claimed well before the process starts executing, and the respective claim edges have to appear in the resource allocation graph.

5) When a process P_i requests a resource R_j, the request should only be granted if the conversion of the request edge $P_i \rightarrow R_j$ to an assignment edge wouldn't result in a cycle in RAG.
For that, when a cycle detection algorithm is used on RAG if no cycle exists, then the allocation of the requested resource will leave the system in a safe state. If a cycle is found, then the process P_i will have to wait.



Resource-allocation graph for deadlock avoidance.

$R_1 \rightarrow P_1$ [Assignment edge]
$P_2 \rightarrow R_1$ [Request edge]

$P_1 \rightarrow R_2$
$P_2 \rightarrow R_2$ [Claim edges]

Banker's algorithm:

1) It is used for a system with multiple instances of resources.
2) Each process has to claim the resources required.
3) If a process is allocated all the resources it requested, it has to complete its task and release them in a finite time.

Note:

Banker's algorithm runs each time

- A process requests resource: Is it safe?
- A process terminates: Can I allocate released resources to a suspended process waiting for them.

A new state is safe if and only if every process can complete after allocation is made. Make allocation and then check system state and deallocate if unsafe.

Banker's algorithm implementation:

1) Let 'n' is the number of processes and 'm' is the number of resource types.
2) Available: If available [j] = K, it means there are K instances of resource type R_j available.
3) Max: If Max [i, j] = K means that process P_i may request atmost K instances of R_j.
4) Allocation: If allocation [i, j] = K then P_i is currently allocated K instances of R_j.
5) **Need:** If need [i, j] = K, then it depicts that P_i may need K more instances of R_j to complete its task.
6) Need [i, j] = Max [i, j] – Allocation [i, j].

Resource request algorithm working (banker's algorithm):

Assume $request_i$ be the list of resources a process P_i requested and $request_i[j]=k$ means process P_i needs k instances of R_j.

1) Check if the sum of values in $Request_i \leq$ the sum of values in $Need_i$, continue if true, else an error is raised since the requested number of resources exceeded the number of needed (claimed before).

2) Check if the sum of values in $request_i \leq$ the sum of values in available, continue if true, else the process must wait as the resources are not available.

3) Algorithm must calculate the below process states by updating values according to
Available = Available – $Request_i$;
$Allocation_i = Allocation_i + Request_i$;
$Need_i = Need_i – Request_i$;
If the resulting state is safe, i.e., if a process can be allocated with resources required, then the process is allocated resources in that order.
If the resulting state is unsafe, the process has to wait until the resources in the old state are restored.

Previous Years' Question (GATE-2007)

A single processor system has three resource types X, Y and Z, which are shared by three processes. There are five units of each resource type. Consider the following scenario, where the column "alloc" denotes the number of units of each resource type allocated to each process, and the column "request" denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish LAST?

	alloc			request		
	X	Y	Z	X	Y	Z
P0	1	2	1	1	0	3
P1	2	0	1	0	1	2
P2	2	2	1	1	2	0

a) P0
b) P1
c) P3
d) None of the above, since the system is in a deadlock

Sol: c)

SOLVED EXAMPLES

Q4 **The following system state given defines how four (A,B,C,D) types of resources are allocated to five running processes.**

$$\text{Available} = \begin{matrix} A & B & C & D \\ \{2, & 1, & 0, & 0\} \end{matrix}$$

$$\text{Allocation} = \begin{matrix} & A & B & C & D \\ P_0 & 0 & 0 & 1 & 2 \\ P_1 & 2 & 0 & 0 & 0 \\ P_2 & 0 & 0 & 3 & 4 \\ P_3 & 2 & 3 & 5 & 4 \\ P_4 & 0 & 3 & 3 & 2 \end{matrix}$$

$$\text{Max} = \begin{matrix} & A & B & C & D \\ P_0 & 0 & 0 & 1 & 2 \\ P_1 & 2 & 7 & 5 & 0 \\ P_2 & 6 & 6 & 5 & 6 \\ P_3 & 4 & 3 & 5 & 6 \\ P_4 & 0 & 6 & 5 & 2 \end{matrix}$$

Compute the need matrix and determine whether the system is in a safe state? Answer 1 if any safe state exists, else answer 0.

Sol: **Range: 1–1**

Need = Max – Allocation.

$$\text{Need} = \begin{matrix} & A & B & C & D \\ P_0 & 0 & 0 & 0 & 0 \\ P_1 & 0 & 7 & 5 & 0 \\ P_2 & 6 & 6 & 2 & 2 \\ P_3 & 2 & 0 & 0 & 2 \\ P_4 & 0 & 3 & 2 & 0 \end{matrix}$$

As we can see, P_0 doesn't need additional resources in the above need matrix, so after P_0 done its execution.

Available = {2, 1, 1, 2}

So now, P_3 can acquire A and D resources available to complete its execution.

Similarly P_4, P_1 and P_2 also get completed in that sequence.

Safe sequence: $<P_0, P_3, P_4, P_1, P_2>$

So, answer is 1.

Previous Years' Question (GATE-2010)

A system has n resources R_0, ... Rn_{n-1}, and k processes P_0, ..., Pk_{k-1},. The
implementation of the resource request logic of each process P_i is as follows:

if(i%2 == 0) {
if(i < n) request R_i;
if(i + 2 < n) request R_{i+2};}
else{
if(i < n) request R_{n-1};
if(i + 2 < n) request R_{n-i-2};}

In which of the following situations is a deadlock possible?

a) n = 40, k = 26
b) n = 21, k = 12
c) n = 20, k = 10
d) n = 41, k = 19

Sol: b)

Deadlock detection and recovery

If the system did not employ both the deadlock prevention and avoidance algorithms, then a deadlock may happen. Therefore the system should provide the following:

1) An algorithm to detect the deadlock.
2) An algorithm to recover it.

Note:

The detection and recovery scheme requires overhead that includes not only the run-time costs of maintaining the necessary information and executing the detection algorithm but also losses from recovering from a deadlock.

SOLVED EXAMPLES

Q5 **There are four processes and three types of resources. The number of copies of each resource type are : (r1, r2, r3) = (3, 2, 2).**
Given table shows the process ids, number of allotted resources of each type for every process and the resource requests for every process.

Process id	Allotted			Request		
	r1	r2	r3	r1	r2	r3
1	1	0	0	0	1	2
2	0	0	1	1	2	1
3	0	1	0	1	0	1
4	1	0	0	0	2	2

Which of the following is a safe sequence?
a) P4, P2, P3, P1
b) P1, P2, P4, P3
c) P2, P1, P4, P3
d) P3, P2, P1, P4

Sol: **Option: d)**

Process id	Allotted r1 r2 r3	Request r1 r2 r3	Available r1 r2 r3
1	1 0 0	0 1 2	1 1 1 1 2 1 1 2 2 2 2 2 3 2 2
2	0 0 1	1 2 1	
3	0 1 0	1 0 1	
4	1 0 0	0 2 2	

Therefore the following is a safe sequence:
P3 –> P2 –> P1 –> P4

Q6 **The system has five process and three resources (A B C). The maximum count of resources are (10 5 7). Consider the following table of resources allocation.**

	MAX			Allocated		
	(A	B	C)	(A	B	C)
P_0	7	5	3	0	1	0
P_1	3	2	2	2	0	0
P_2	9	0	2	3	0	2
P_3	2	2	2	2	1	1
P_4	4	3	3	0	0	2

What will be a safe sequence?

a) P_1, P_2, P_3, P_4, P_o

b) P_1, P_o, P_3, P_4, P_2

c) P_1, P_3, P_4, P_0, P_2

d) Unsafe Sequence

Sol: **Option: c)**

	MAX Allocation			Allocated			Current Need		
	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3
P_1	3	2	2	2	0	0	1	2	2
P_2	9	0	2	3	0	2	6	0	0
P_3	2	2	2	2	1	1	0	1	1
P_4	4	3	3	0	0	2	4	3	1

After P_1 available resources are (5, 3, 2).
Now, P_3 and P_4 both can be served. After serving both available resources (7,4,5).
Now, similarly, P_0 and P_2 can be served.
So possible safe sequence is an option (C) which is P_1, P_3, P_4, P_0, P_2

Deadlock detection:

If all resources are with only one instance, then for deadlock detection, a variation of resource–allocation graph named as wait–for graph can be obtained from the RAG by removing the resource nodes and corresponding edges.

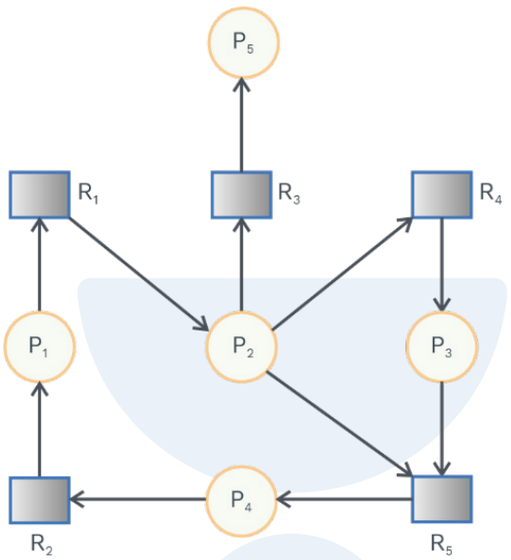


Fig. 4.8 Resource Allocation Graph

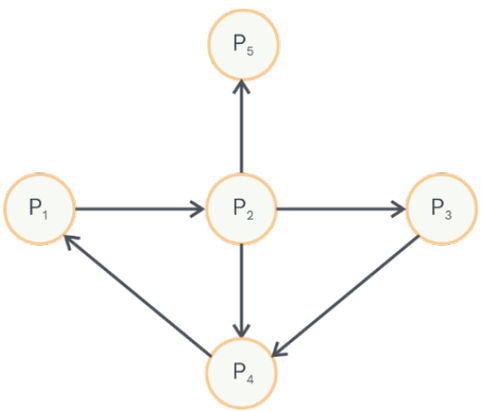


Fig. 4.9 Corresponding Wait–For Graph

So, deadlock exists in the system if and only if the wait–for graph contains a cycle.

Note:

Wait-for graph is a graph sharing the dependencies between processes waiting for one another.

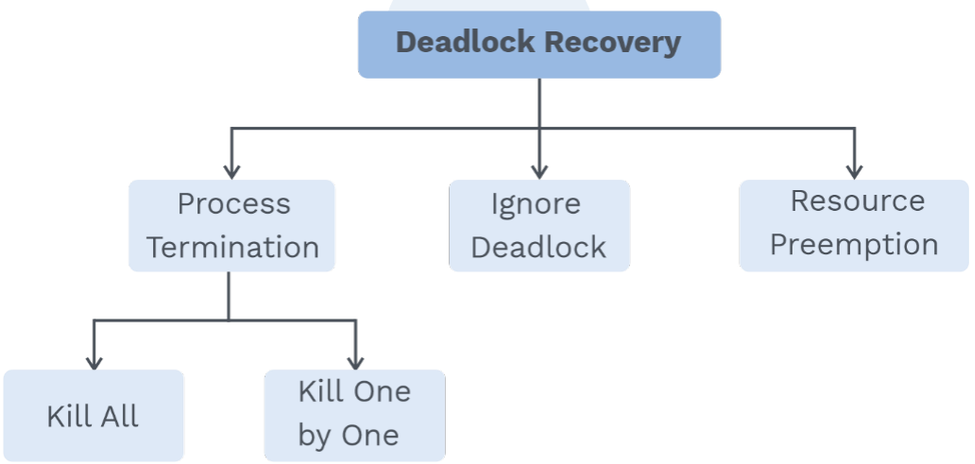
The system needs to maintain the wait–for graph and periodically calls an algorithm that searches for a cycle.

Rack Your Brain

- What would be the time complexity of banker's algorithm?
- What would be the time complexity to detect a cycle in a graph of 'n' vertices?

If resources have multiple instances, then for deadlock detection, variation of banker's algorithm is used; we satisfy the process request of resource one by one if it leads to deadlock, we roll back else find a safe sequence.

Deadlock recovery:



a) Process termination:

To eliminate deadlock by killing a process, we use two methods.

1) **Abort all deadlock processes:**
   All the process gets killed, and this method will clearly break from the deadlock cycle.
2) **Abort one process at a time:** until the deadlock cycle is eliminated.

Note:

Aborting a process may not be easy if the process was in a middle of an important task like printing or updating a database.

Killing processes one by one is more useful than killing every process in the system. Various factors which decide which process is chosen to be killed are:

1) What is the current priority of the process?
2) How long the process has already been executed?
3) How many resources are still required for the process to get completed?
4) What is the nature of the process, i.e., whether it is an interactive or batch process?

b) Resource preemption

Definition

To eliminate deadlocks using resource preemption, we successively preempt some resources from the process and given these to other processes until the deadlock is broken.

Various issues while preempting resources to deal with deadlocks,

1) Victim selection:
Which resources and processes are to be preempted? So that cost will be minimum, such as a number of resources a particular process is holding.

2) Rollback:
Now after preemption, we have to determine whether to continue the preempted process execution or rollback it to some safe state.

3) Starvation:
In a system where a selection of victim is based on the cost factor, it may very well happen that the same process get victimized everytime, so starvation occurs for some process in the system, so we have to put an upper bound for a process for preemption.

c) Ignore the deadlock:
It is also known as ostrich algorithm where the system ignores the deadlock and acts like the deadlock never happened. It's widely regarded as the most effective way to deal with a deadlock.

SOLVED EXAMPLES

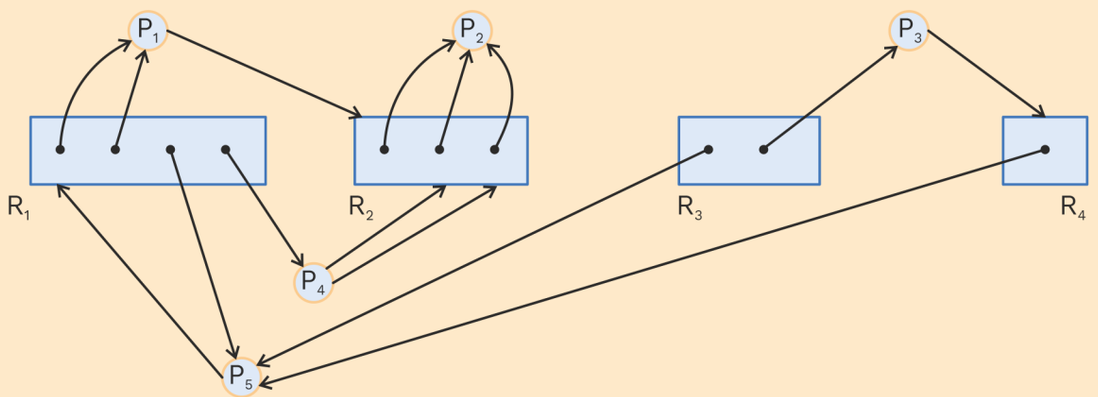
Q7 **Consider the following statements**
S1: Deadlock avoidance is seldom used as a practical solution to the deadlock problem
S2: Banker's algorithm is sufficient for a practical system.
a) S1 is true, S2 is false
b) S1 is false, S2 are true
c) Both S1 and S2 are false
d) Both S1 and S2 are true

Sol: **Option: a)**
S1 is true: Deadlock avoidance needs lots of overhead and prior information on resources need a program.
S2 is false: Banker's algorithm has certain limitations in its implementation. It should know the number of instances of each resource a process will request. In most of the current systems, it is impossible to have such information prior.

Q8 **Which of the following statements is/are incorrect?**
a) Processes may lead to deadlock as a result of contending for the processor.
b) Only hardware resources can be non-preemptible.
c) An unsafe state is a deadlocked state.
d) A system cannot transition from an unsafe state to a safe state.

Sol: **Options: b), c), d)**
a) TRUE. The processor is a preemptible resource
b) FALSE. Some software resources are also non-preemptible like monitors.
C,D) Both FALSE. When processes release resources, the number of available resources can be enough for the state of the system to make a transition from unsafe to safe.

Q9 **Consider the following resource allocation graph with resources R_1, R_2z, R_3, R_4 with 4, 3, 2, 1 instances of the same resource, respectively.**



Which of the following execution sequences of processes is/are safe?

a) <P2$_2$, P_1, P_4, P_5, P_3>

b) <P_2, P_4, P_1, P_5, P_3>

c) <P_2, P_1, P_5, P_3, P_4>

d) <P_2, P_4, P_5, P_1, P_3>

Sol: **Options: a), b), c), d)**

From the given resource allocation graph, current allocation and need can be seen as follows:

Resources (Instances)	Allocated					Need				
	P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_5
R_1(4)	2	0	0	1	1	0	0	0	0	1
R_2(3)	0	3	0	0	0	1	0	0	2	0
R_3(2)	0	0	1	0	1	0	0	0	0	0
R_4(1)	0	0	0	0	1	0	0	1	0	0

Currently available resource instances can be seen as <R_1, R_2, R_3, R_4> = <0, 0, 0, 0>, i.e., no resource is free. From the above table, it is clear that only P_2 can complete its execution as it doesn't need any resources.

Post P_2 completion resources available are <R_1, R_2, R_3, R_4> = <0, 3, 0, 0>, with 3 instances of R_2, either P_1 or P_4 can complete. Among P_3 and P_5, P_5 can complete its execution first as resource needs of P_5 are available post P_1 or P_4 completion. P_3 is waiting for one instance of resource R_4 which P_5 is holding, so P_3 will have to wait till P_5 completes and frees one instance of resource R_4.
So, all sequences are safe.

Q10 **Consider an operating system which implements Banker's Algorithm to avoid deadlock. Currently allocation table with three resources A, B and C to five processes P_0, P_1, P_2, P_3, P_4 can be seen below:**

Processes	Allocated			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

REQ 1: Process P_1 asks for one additional instance of resource A and two instances of resource C.

REQ 2: Process P_2 asks for two additional instance of resource A.

a) Only REQ 1 can be permitted

b) Only REQ 2 can be permitted

c) REQ 1 and REQ 2 both can be permitted

d) Neither REQ 1 nor REQ 2 can be permitted

Sol: **Option: c)**
REQ 1 (P1) <1, 0, 2> <= Available <3, 3, 2>
This request can be fulfilled, and the new state is as following:

Processes	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Now, by using safety algorithm

1) Need (P1) <0, 2, 0> is less than available <2, 3, 0> P1 gets the resources and after completion free the resources, updates available <5, 3, 2>

2) Need (P3) <01, 1> is less than available (5, 3, 2) P3 gets the resources and after completion free the resources, updates available (7, 4, 3)

3) Need (P4) <4, 3, 1> is less than available <7, 4, 3> P4 gets the resources and after completion free the resources, updates available <7, 4, 5>

4) Need (P2) <6, 0, 0> is less than available <7, 4, 5> P2 gets the resources and after completion free the resources, updates available <10, 4, 7>

5) Need (P0) <7, 4, 3> is less than available <10, 4, 7> P0 gets the resources and completes execution

6) All processes get completed, so REQ1 can be permitted. Safe sequence is <P1, P3, P4, P2, P0>

REQ 2 is in continuation after grant of REQ 1.
REQ2 (P2) <2, 0, 0> <= Available <2, 3, 0>.
This request can be fulfilled, and the new state is as following:

Processes	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	0	3	0
P_1	3	0	2	0	2	0			
P_2	5	0	2	4	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Now, by using safety algorithm,

1) Need (P1) <0, 2, 0> is less than available <0, 3, 0> P1 gets the resources completes execution and frees resources, updates available <3, 3, 2>
2) Need (P3) <0, 1, 1> is less than available <3, 3, 2> P3 gets the resources completes execution and frees resources, updates available <5, 4, 3>
3) Need (P4) <4, 3, 1> is less than available <5, 4, 3> P4 gets the resources completes execution and frees resources, updates available <5, 4, 5>
4) Need (P2) <4, 0, 0> is less than available <5, 4, 5> frees resources after completion and updates available <10, 4, 7>
5) Need (P0) <7, 4, 3> is less than available <10, 4, 7> P0 gets resources and completes its execution
6) REQ 2 can be permitted, safe sequence is <P1, P3, P4, P2, P0>

Q11 **CSuppose a system that uses banker's algorithm to deal with deadlocks. Currently allocation table with three resources A, B and C to five processes PO, P1, P2, P3, P4 can be seen below:**

Processes	Maximum			Allocated			Available		
	A	B	C	A	B	C	A	B	C
P_1	6	5	4	0	3	4	4	3	1
P_2	3	4	2	2	1	2			
P_3	1	0	4	0	0	2			
P_4	3	2	5	1	2	1			

Which of the following is invalid safe sequence(s)?

a) P_2, P_1, P_3, P_4 **b) P_2, P_4, P_1, P_3** **c) P_2, P_3, P_4, P_1** **d) P_2, P_1, P_4, P_3**

Sol: **Option: b)**

Need(P_2) = MAXIMUM(P_2) – ALLOCATED (P_2)
= (3 4 2) – (2 1 2) = (1 3 0)
CURRENT AVAILABLE >= Need(P_2)
[(4 3 1) >= (1 3 0)], so, P2 can execute.

After P2's execution current available = (4 3 1) + (2 1 2) = (6 4 3)

Need(P_1) = MAXIMUM (P_1) – ALLOCATED (P_1) = (6 5 4) – (0 3 4) = (6 2 0)

CURRENT AVAILABLE >= Need (P_1)
[(6 4 3) >= (6 2 0)], so, P1 can execute.

After, P_1's execution, Current Available = (6 4 3) + (0 3 4) = (6 7 7)
(6 7 7) is sufficient to process P3{NEED: (1 0 2)} and P4{NEED: (2 0 4)} in any order.
So, (A)[P2 P1 P3 P4] and (D) [P2 P1 P4 P3] are valid safe sequences.

(B) P_2 P_4 P_1 P_3
Similarly, Need(P_2) = (1 3 0) which can be satisfied by current available (4 3 1).
After P_2's execution:
Current Available = (4 3 1) + (2 1 2) = (6 4 3)
Need(P_4) = (2 0 4) which cannot be satisfied by current available (6 4 3)
Hence, (B) is an invalid safe sequence

(C) P_2 P_3 P_4 P_1
Need (P_2) = (1 3 0) which can be satisfied by current available (4 3 1)
After P_2's execution:
Current available = (6 4 3).
Need (P_3) = (1 0 2) can be satisfied by current available
After P_2's execution:
Current available = (6 4 5)
Need (P_4) = (2 0 4) can be satisfied by current available.
After P_4's sufficient to satisfy need of P_1 (6 2 0)
So, (C) is valid safe sequence.

Chapter Summary

1) Basics of deadlock – Process waiting for a resource which it won't get immediately and have a circular dependency of several processes to each other.

2) Types of resources –
   1) Reusable resources
   2) Preemptable resources
   3) Non-preemptable resources
   4) Consumable resources

3) Necessary condition for deadlock –
   1) Mutual exclusion
   2) Hold and wait
   3) No preemption
   4) Circular wait

4) Resource allocation graph – Graphical representation of a relationship between process and resources.

5) Handling of deadlock –
   1) Deadlock prevention
   2) Deadlock avoidance
   3) Deadlock detection and recovery
   4) Ostrich approach