UNIT - 3
Deadlocks

Imp
★ **Deadlock :-** A deadlock is a situation in which some process wait for each other actions indefinitely. It can arise in a synchronisation situation.
Example- When process wait for the other process to send them message or to release resources that they need.

• **Definition of deadlock** – Deadlock means a lock having no keys.
A set of process is in deadlock if each of them wait for an event that can be cause only by process in the set. Thus each process wait for an event that cannot occur. This situation arises if the following conditions are satisfied-

1- Process Pi in a set of processes D is blocked on some event Ej.
2- Event Ej can be cause only by actions of other process in D.
Each process Pi in D must satisfy above two conditions for D to be in deadlock

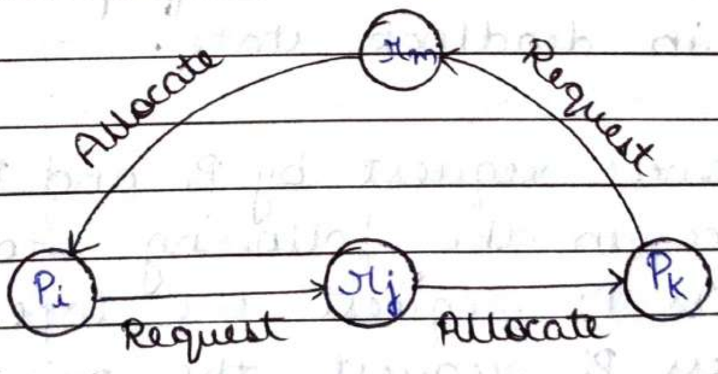
state.

* **Deadlocks in Resource allocation :-** Three events concerned the resource allocation can occur in a system.

1. Request for a resource.
2. Allocation for a resource.
3. Release of a resource.

A request event occurs when process Pi makes a request for a resource r_1. If r_1 is currently allocated to some process P_K, process Pi gets locked on an allocation event for r_1. In effect Pi is waiting for P_K to release r_1.

A release event by P_K free resource r_1. It can cause the allocation event for which Pi is waiting. Process Pi will face an indefinite wait if P_K is release of r_1 is indefinitely delayed.



Deadlock

There may be another resource r_m which is allocated to the process P_i and process P_K request for it to accomplish its task then same will occur that is until P_i release the resource r_m. P_K can't complete its task so both process P_i and P_K are locked. No (process) task is perform in the system.

xample- A system contains one tape and one printer and two process P_i and P_j that use these resource as follow-

Process i	Process j
Request tape	Request Printer
Request printer	Request tape
Use tape and printer	Use tape and printer
Release tape	Release Printer
Release Printer	Release tape

Show that the set of process P_i, P_j is in deadlock state.

tion:- Resource request by P_i and P_j take place in the following order-

1- Process P_i request the tape.
2- Process P_j request the printer.
3- Process P_i request the printer

PAGE NO :
DATE : / /

4- Process Pj request the tape.
The first two requests are granted immediately because a tape and printer exist in the printer. Now process Pi hold the tape and Pj hold the printer. When Pi ask for the printer it is blocked until Pj release the printer.
Similarly Pj is locked until Pi releas the tape. So the set of process Pi, Pj is in deadlock state.

* Necessary conditions for a resource deadlock -

1. Mutual exclusion (Non-sharable resources) - Resources cannot be shared a process need exclusive access to a resource. If another process request that resource, it is block until the resource is release by the first process.

2. Hold and wait - A process continue to hold the resources allocated to it while waiting for other resources.

3. No preemption - OS preempt a resource from one process in order to allocate it to another process. A resource can be

release after that process completed its task.

4. Circular wait – A circular chain of hold and wait conditions exist in the system. A set of waiting process $P_0, P_1, \ldots P_n$ must exist in the system such that P_0 is waiting for the resource held by P_1, P_1 is waiting for the resource held by P_2 and so on. P_{n-1} is waiting for the resource held by P_n and P_n is waiting for the resource held by P_0.

* Handling deadlocks :-

There are 3 fundamental approach use in deadlock handling. Each approach has different implications for user process and for the operating system.

1- Deadlock Prevention – The kernal uses a resource allocation policy that ensure that the four conditions for resource deadlock do not arise simultaneouslly. This approach makes deadlock impossible.

2- Deadlock Avoidance – The kernal analyze allocation state to determine whether granting a resource request might lead to be a deadlock later. Only request that cannot lead to a deadlock are granted, others are kept pending until they can be granted.

3- Deadlock detection and resolution - The kernal analyze the resource state to check whether a deadlock exist, if so it abort some process and allocates the resources held by them to other process. So that the deadlock classes to exist.

★ **Deadlock Avoidance :-**
Given a prior information about the maximum number of resources of each type that may be requested for each process. It is possible to construct an algorithm that ensure that the system will never enter a deadlock state. This approach is called as deadlock avoidance.

• Safe state - A state is safe if the system can allocate resources to each process in some order and avoid deadlock.
Formally a system is said to be in safe state only if there exist a safe sequence.

- Unsafe state – If no such sequence exist then the system state is said to be unsafe.
  A safe state is not a deadlock state conversely, a deadlock state is an unsafe state.

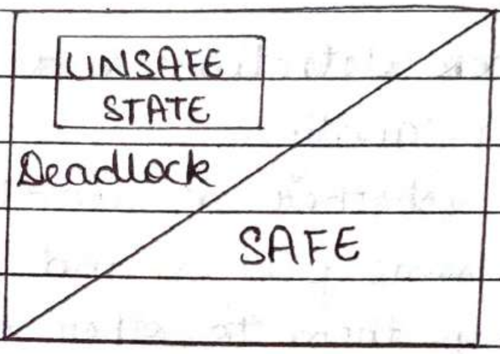


Fig :- Safe, Unsafe, Deadlock State

Example – Consider a system with 12 tape driver and 3 process as–

Process	Maximum needs	Current need	Available
P_0	10	5	12
P_1	4	2	
P_2	9	~~2~~ 3	

Out of 12 available tape drives suppose at time to process P_0 is holding 5 tapes, P_1 hold 2 tapes, P_2 hold ~~3~~ 2 tapes and 3 tape drives are free. Whether this system is in deadlock state or not.

★ **Banker's algorithm** :- The resource allocation graph algorithm is not applicable to a resource allocation system with multiple instance of each resource type, the deadlock avoidance scheme that we describe next is applicable to such a system. But is less efficient than resource allocation graph scheme. This algorithm is commonly known as Banker's algorithm.

Several data structures are required to implement the Banker algorithm. Let n be the number of process in the system and m be the number of resource type. We need the following data structures -

1. Available - A vector of length M indicates the number of available resources of each type. If available[j] = k instance of resource type rj are available. It is a one-dimensional array of length m.

2. Max - It is a 2-dimensional array (matrix) of length n×m that defines the maximum demand of each process in a system. If max [i,j] = k then

PAGE NO :
DATE : / /

process P_i may request at most k instance of resource type r_j.

3- Allocation — It is also a 2-D array of length n X m that defines the number of resources of each type currently allocated to each process.
If allocation [i,j] = k then process P_i currently allocate k instance of resource type r_j.

4. Need — If need [i,j] = k then process P_i may need k more instance of resource type r_j to complete its task.
Need [i,j] = Max[i,j] — Allocation [i,j]

* Safety Algorithm :-
The algorithm for finding out whether or not a system is in a safe state can be describe as follow—
1. Let work n finish by vector m and n.
Initialize work = available
finish [i] = false;

2. Finish an i such that both —
(i) finish [i] = false
(ii) Need ≤ work
if no such i exist, go to step 4.

PAGE NO :
DATE : / /

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.

★ Resource request algorithm –
Let $request_i$ be the request vector for process P_i. If $request_i[j] = k$ then process P_i wants k instance of resource type R_j. When a request for resources is made by process P_i, the following actions are taken :

(1) If $Request_i \leq Need_i$
   goto step (2) ; otherwise,
raise an error condition, since the process has exceeded its maximum claim.

(2) If $Request_i \leq Available$
   goto step (3) ; otherwise,
P_i must wait, since the resources are not available.

(3) Have the system pretend to have

allocated the requested resources to process P_i by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i$$
$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i$$
$$\text{Need}_i = \text{Need}_i - \text{Request}_i$$

If the resulting resource - allocation state is safe, the transaction is completed and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for Request_i and the old resource - allocation state is restored.

Example:- Consider a system with 5 process - P_0, P_1, P_2, P_3, P_4 and 3 resources types A, B, C. Resource type A has 10 instance, B has 5 instance and type C has 7 instance. Suppose at time t_0 following snapshot of the system has been taken —

Process	Allocation	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	

(i) What will be the content of need matrix?

(ii) Is the system in safe state? If yes, what is the safe sequence.

~~(iii) If request (3,3,0)~~

(iii) What will happen if process P_1 request 1 additional instance of resource type A and 2 instance of resource type C.

(iv) If request (3,3,0) by process P_4 arrive in the state define by (iii), can it be granted immediately?

(v) If a request (0,2,0) by process P_0 arrive then check whether it is granted or not?

Solution:- (i) Content of need matrix -

Need = Max − Allocation

$$= \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}$$

(ii) Applying safety algorithm on the given system for P_i if Need ≤ Available then P_i is in safe sequence.

Available = Available + Allocation

So for P_0
(i=0) $Need_0 \rightarrow 7,4,3$
Available → 3,3,2
Condition is false.
So P_0 must wait.
for P_1
(i=1) $Need_1 \rightarrow 1,2,2$
Available → 3,3,2
(Need < Available)
P_1 will be kept in safe sequence.
Now available will be updated as-
Available = Available + Allocation
= 3,3,2 + 2,0,0
= 5,3,2
for P_2
(i=2) $Need_2 \rightarrow 6,0,0$
Available → 5,3,2
Condition is false and P_2 must wait.
for P_3
(i=3) $Need_3 \rightarrow 0,1,1$
Available → 5,3,2
Now, Available = 5,3,2 + 2,1,1
= 7,4,3
for P_4
(i=4) $Need_4 \rightarrow 4,3,1$
Available → 7,4,3
Need < Available

Now Available = 7,4,3 + 0,0,2
= 7,4,5

Now we have two process P_0 and P_2 in waiting state. As current available either P_0 or P_2 is kept in safe sequence

Firstly we take P_2 whose need = 6,0,0
Available → 7,4,5
$Need_2$ < Available
So P_2 now comes in safe state.
Available = 7,4,5 + 3,0,2
= 10,4,7
Next, P_0 whose Need → 7,4,3 and
Available → 10,4,7
P_0 now comes in safe state.
and Available = 10,4,7 + 0,1,0
= 10,5,7
So the safe sequence is –
$\langle P_1, P_3, P_4, P_2, P_0 \rangle$

(iii) Since P_1 request some additional instance of resources such that
$Request_1$ (1,0,2)
To decide whether this request is immediately granted as we first check that Request ≤ Available which hold true. So the request may be granted

To confirm that this request is granted we check the new state by applying safety algorithm that our system is in safe state or not.

If the new state is in safe state then only this request is granted otherwise not.

To define the new state of the system because of the arrival of request P_1. We follow the resource request algorithm which result as-

Process	Allocation	Need	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	

We must determine whether this new system state is safe. We again execute a safety algorithm and find the safe sequence as $<P_1, P_3, P_4, P_2, P_0>$ which satisfy our safety requirements hence we can immediately grant the request for process P_1.

(iv) The request for (3 3 0) by P_4 cannot be granted because $\text{Request}_4 = 3\,3\,0$
$\text{Available} = 2\,3\,0$

In this situation the condition is false. So it is not granted since resource are not available.

(v) The request for (0 2 0) by process P_0.

Request$_0$ = 0 2 0

Available = 2 3 0

So the condition is true and the request may be granted.

If it is granted then the new state of the system is define as—

Available = Available − Request
= 2,3,0 − 0,2,0
= 2,1,0

Allocation = Allocation + Request
= 0,1,0 + 0,2,0
= 0,3,0

Need = Need − Request
= 7,4,3 − 0,2,0
= 7,2,3

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 3 0	7 2 3	2 1 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	

Ques Consider a system with 5 process P_0 ---- P_4. and 3 resources types A, B and C. Resource type A has 7 instance, B has 2 and C has 6 instance. Suppose at time t_0 we have following state –

	Allocation	Request	Available
Process	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

(i) Is the given system is in deadlock state?

(ii) Suppose P_2 makes an additional request (001). What will be the effect of this request to the system?

Sol:- (i) For P_0 ⇒ $Request_0$ = 000

Available = 000

Condition is true

Now available = Available + Allocation

= 000 + 010

= 010

for P_1 ⇒ $Request_1$ = 202

Available = 010

(Request > Available)

Condition is false, so P_1 must wait.

for P_2 ⇒ $Request_2 = 000$
Available = 010
Condition is true.
Now available = 010 + 303
= 313
for P_3 ⇒ $Request_3 = 100$
Available = 313
Condition is true.
Now available = 313 + 211
= 524
for P_4 ⇒ $Request_4 = 002$
Available = 524
Condition is true
Now available = 524 + 002
= 526
Now for the waiting process P_1 ⇒
$Request_1 = 202$
Available = 526
Condition is true.
So P_1 comes in safe state.
Now available = 526 + 200
= 726.

(ii) Since P_2 request some additional instance of resources such that
$Request_2 = 001$
Available = 726
Request < Available

PAGE NO :

DATE : / /

So the request may be granted.

The new state of the system will be—

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

★ Deadlock resolution or recovery from deadlock :-

When a detection algorithm determines that a deadlock exist, several alternatives approach can be implemented. One possibility is to inform the operator that a deadlock has occur and to let the operator deal with the deadlock manually. The other possibility is to let the system to recover from the deadlock automatically. There are two options for breaking a deadlock -

1- Terminate some process P_j which is a subset of D to free the resource required by P_i.

2- Add new unit of resources requested by P_i or preempt some resources from one or more of the deadlock process.