Unit-3- Process Deadlocks

- Ankit Pangeni

DATE

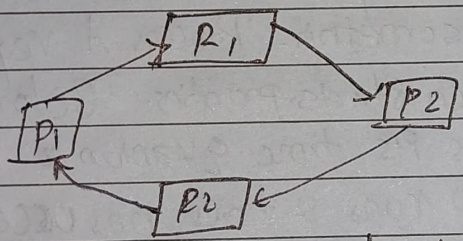
Introduction:

✗ Definition

Deadlock is a condition or a situation in which two programs sharing the same resource are preventing each other from accessing the resource, resulting in none of the program from executing.

In earliest day, the OS ran only one program at a time. All of the resources of the system were available to that program. Nowadays, OS can run multiple program at one. Programs are required to specify in advance what resources they needed so that they could avoid conflicts with other programs running at same time.

Eg:



Here, Process (P_1) is holding resource (R_2) and waiting for resource (R_1) which is acquired by process (P_2) and P_2 is waiting for resource (R_2) which is acquired by P_1.

So, neither program can proceed until the other program releases a resource. At this point, the OS don't know what to do, as a result deadlock occurs & the only alternative solution is to stop all the programs.

DATE

Resources

The resource is the object granted to a process It can be a hardware device (ex: a printer) or a piece of information (ex: a record in a database) A computer can multiple resources that a process can acquire.
A resource is anything that must be acquired, used and released over the course of time.

Resource are of two types:

1. Preemptable: They are the resources that can be taken away from its current owner (and given back later). Ex: Memory.
consider, a system with 32 MB of memory, one printer & two 32MB processes such that each process want's to print somethin. Process A requests & gets the printer, and then starts printing. Before it has finished, it exceeds its time quantum & is swapped out. Process B now runs & tries, unsuccessfully to acquire the printer. Potentially, we have a deadlock situation, as A has printer & B has memory Fortunately, as memory is preemptive, it is possible to take away the memory from B by swapping it out & swapping A in. Now, A can run, do its printing, & then release the printer. No deadlock occurs.

classmate

PAGE

DATE

2. Non-Preemptable.

A non preemptable resource is the one that cannot be taken away from its current owner without causing the computation to fail. Ex: CD-ROM. If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it & giving it to another process will result in corrupted CD-ROM as CD-Recorders are not preemptable. In general, deadlocks involve non-preemptable resources.

* Deadlock Characterization

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting. Various features that characterize deadlock are:-

1. Deadlock prequisites (necessary conditions that lead to deadlock

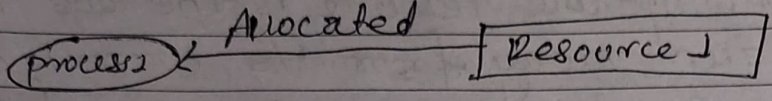
There are four conditions that must hold for deadlock

(a) (Mutual Exclusion)

In a multiprogramming environment, there may be several processes requesting the same resource at a time. The mutual exclusion condition allows only a single process to access the resource at a time. (no sharing). While the other processes requesting

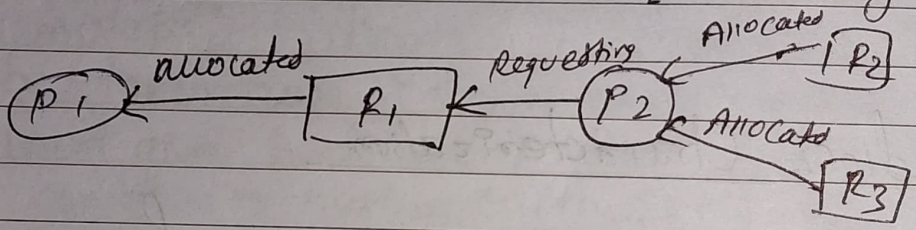
DATE

the same resource must wait and delay their execution until it has been released.



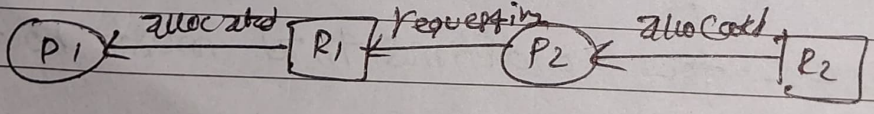
b) (Hold and wait condition):

The hold and wait condition means that a process must be holding access to one resource & must also be waiting to get hold of other resources that have been acquired by other process



c) (Non preemption):

It means previously granted resources cannot be forcibly taken away from process. The process must release the resource when its task is completed



d) (Circular wait condition):

Circular wait condition means there must be a chain of processes such that each member of the chain is waiting for a resource held by the next member of the chain. The processes form a circular chain.

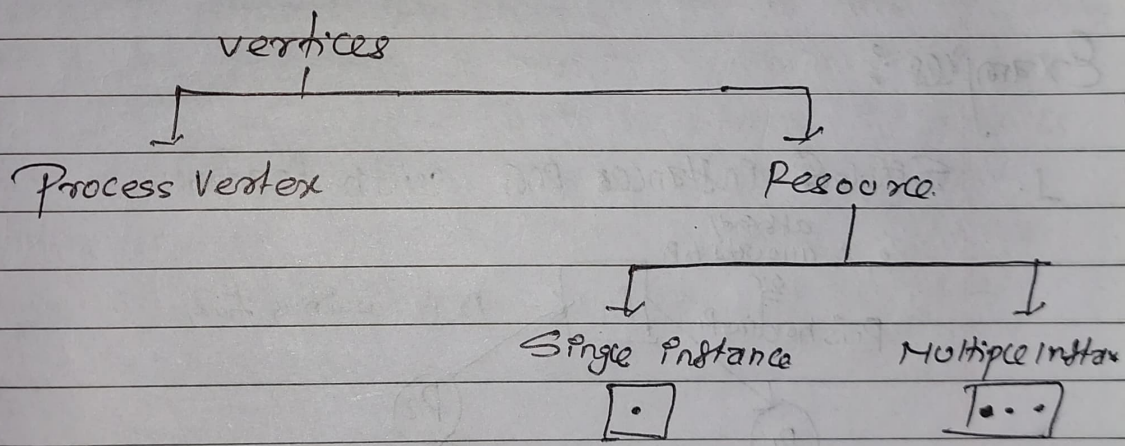
DATE

2. Resource Allocation Graph (RAG).

RAG is a directed graph that briefs about the deadlock more precisely. This is the principal representation showing processes, requested resources and the assigned resource.
One of the advantages of having a graph is that we can see deadlock directly by using RAG. In RAG, Vertices and edges are present.

In RAG Vertices are of two types:-

- Process Vertex: Every process will be represented as a process vertex with a circle.
- Resource vertex: Every resource will be represented as a resource vertex with a rectangle.



↳ Single instance type resource: It represents a box with one dot. The no. of dots indicates the no. of copies of each resource. (Instances)

↳ Multiple Instance type resource: It represents a box with multiple dots.

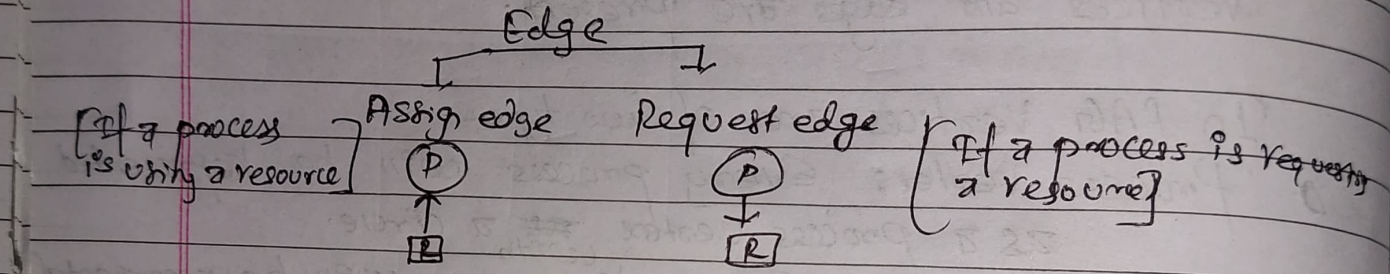
DATE

Similarly, there are two types of edges in RAG

(Allocated)

① Assign edge: If you already allocated (assign) a resource to a process, then it is called assign edge.

② Request edge: It means in future the process might want some resource to complete the execution, that is called request edge.



Examples:

1. Single Instance RAG with deadlock.

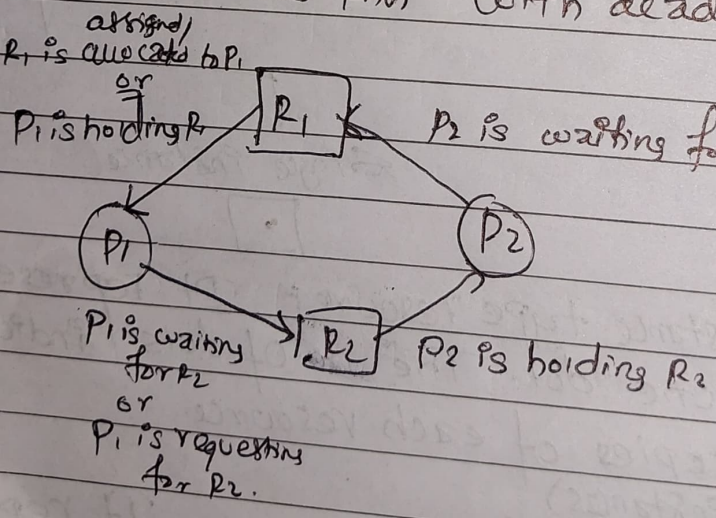


fig: Single Inheritance Resource Type with deadlock.

DATE

If there is a cycle in the RAG & each resource in the cycle provides only one instance, then the process will be in deadlock.
Here, in above example, P_1 holds resource R_1, process P_2 holds resource R_2 & P_1 is waiting for R_2 while P_2 is waiting for R_1. Then, processes P_1 & P_2 will be in deadlock

2. Single instance RAG without deadlock.

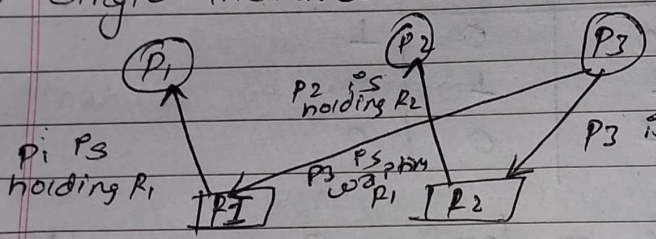
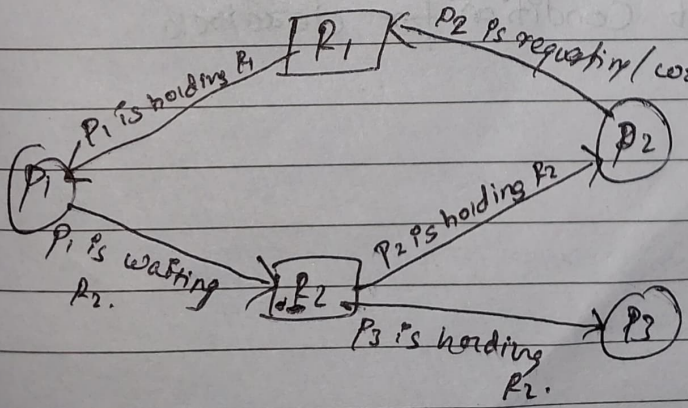


Fig: S.I.R.A.G.W.D

Here, it shows processes P_1 and P_2 are acquiring or holding resources R_1 and R_2 while process P_3 is waiting to acquire both resources. There is no deadlock because there is no circular dependency. So cycle in Single-instance resource type is the sufficient condition for deadlock.

3. Multiple-instances RAG without deadlock.



classmate

PAGE

DATE

Here, it is not possible to say if RAG is in safe state or unsafe. so, to see the state of RAG. let's construct the matrix representing graph.

1 ⇒ True

0 ⇒ False.

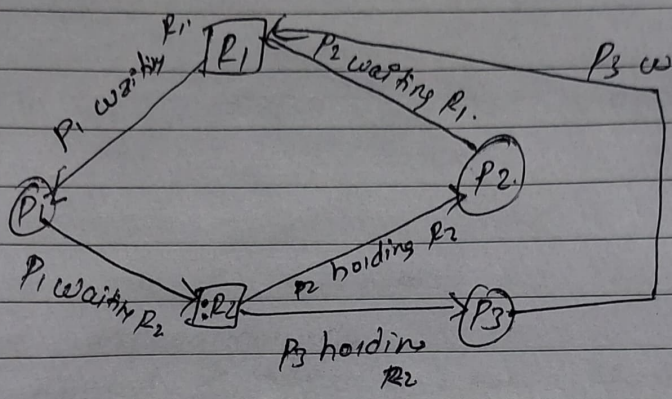
Process	Allocation Resources		Request Resources	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

To see if there is a deadlock in graph, 1st we have to see the available resource. The available resource is [00]. as P_3 doesn't require any resource to complete its execution so it releases R_2 after its completion. Since, there are two instances of R_2, one is holding by P_2 and another is waiting by P_1. Then, P_1 executes & releases R_1. Then finally, P_2 executes and releases R_2. In this way, there is no deadlock in above RAG.

So, we can say, in multiinstance, resource cycle is not sufficient condition for deadlock.

DATE

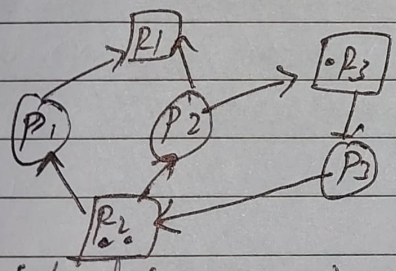
4. Multiinstance resource type with deadlock.



This example is same as previous example except that P3 is requesting for resource R1.

Process	Resource Allocation R1	R2	Resource request R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

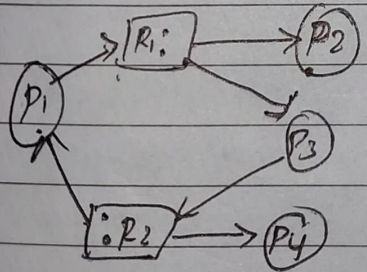
5.



(dead lock occurs)

Process	R.A R1	R2	R3	R.R R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	1	0

6.



(No deadlock)

Process	R.A R1	R2	R.R R1	R2
P1	0	1	1	0
P2	1	0	0	0
P3	1	0	0	1
P4	0	1	0	0

DATE

X Handling deadlocks (Deadlock methods & algorithms)

① Ostrich Algorithm (Deadlock Ignorance)

→ The ostrich algorithm is a strategy of ignoring potential problems on the basis that they may be exceedingly rare.
→ It is named for the ostrich effect which is defined as "to stick one's head in the sand and pretend there is no problem".
→ It is used when it is more cost-effective to allow the problem to occur than to attempt its prevention.
→ This may be used in dealing with deadlocks which occur very rare & the cost of detection or prevention is very high.
→ Ex: If each PC deadlocks occur once per 10 years, the one reboot may be less painful than the restrictions needed to prevent it.
→ The UNIX & windows OS take this approach.

② Deadlock Detection and recovery from deadlock

→ It is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock.
→ The basic idea is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlock state.

DATE

Once deadlock is detected, there needs to be a way to recover several alternatives exist:

- Temporarily prevent resources from deadlocked processes
- Successfully kill processes until system is deadlock free.
- Restarting a process

* Deadlock detection for single Resource Instance

If all resources have only a single instance, the deadlock detection algorithm uses a "wait for graph" obtained by removing the nodes of each resource type and collapsing the edges.

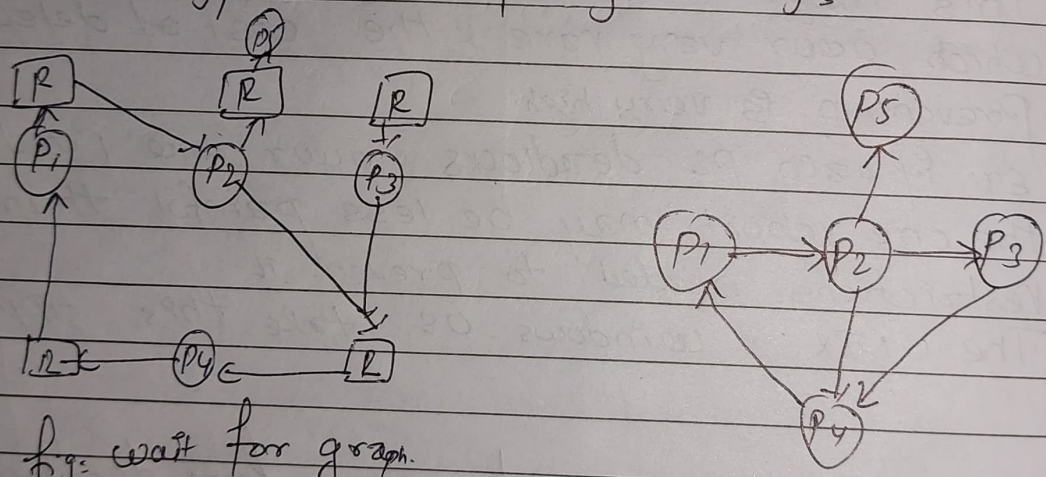


fig: wait for graph.

¥ Deadlock detection for multiple Resource Instance

Wait for graph is not applicable to this system, when multiple instance of each resource type are available. So, a different approach is needed

- Each process is initially said to be unmarked
- As an algorithm, process will be marked indicating that

DATE

that they are able to complete & thus not deadlocked

- When algorithm terminates, any unmarked process is known to be deadlocked.

Note: Recovery from deadlock explained few pages later.

(3) Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four condition.

* Elimination of "Mutual Exclusion" condition

The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource. This condition is difficult to eliminate because some resources, such as the tap drive & printer are inherently (naturally) non-shared bcz. Note that shareable resources do not require mutually exclusive access & thus cannot be involved in deadlock.

* Elimination of "Hold and wait" Condition

There are two possibilities for eliminating this. The first is that, a process request be granted all of the resources it needs at once, prior to execution. The second is, disallow a process from requesting resources whenever it has previously allocated resources. This strategy requires that all of the resources a process will need must be requested at once. The system must grant resources on "all or none" basis. This strategy can cause Starvation, since not all the required resources may ~~become~~ become available at once.

CLASSMATE

PAGE

DATE

* Elimination of "No-preemption" Condition

In this strategy, we allow a process to be preempted. The non-preemption condition can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated to release all of its currently held resources, so that other process may use them to finish.
This strategy requires that, when a process that is holding some resources is denied a request for additional resources. The process must release its held resources & if necessary, request them again together with additional resources. This strategy accepts preemption condition.

* Elimination of "Circular wait" Condition.

This means, we prevent the loop of processes waiting for the resources. The circular wait condition can be denied by ordering all the resources and then forcing all processes to request the resources in either increasing or decreasing order. With this method, the resource allocation graph can never have a cycle or loop.
Let's take an example of 4 resources & 4 processes
1:= Printer, 2:= Plotter, 3:= Tape drive, 4:= card reader
Now, the rule is: processes can request resources whenever they want to, but all requests must be made in increasing or decreasing numerical order.
Here, a process may request 1st printer and then plotter, but it may not request first plotter & then a printer (if we suppose in increasing order). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone.

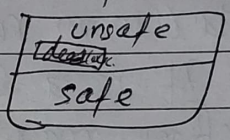
CLASSMATE PAGE

(4) Deadlock Avoidance.

DATE

- (Safe & Unsafe states)

- A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.
- If the system cannot fulfill the request of all processes, then the system's state is unsafe and it might go into deadlock.
- The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case of the resulting state is also a safe state.



The deadlock avoidance approach to the deadlock problem anticipates deadlock before it actually occurs. Here, for every resources allocation, we check the safe and unsafe states. This method differs from dead lock prevention, which guarantees that deadlock cannot occur by denying one of the necessary conditions for deadlock. If the necessary conditions for a deadlock are in place, it is still possible to avoid deadlock by being careful when resources are allocated.

Note: If a system goes into unsafe state, then it may or may not go into deadlock state. So, Unsafe state and deadlock state are different. We should know that, unsafe state implies that some unfortunate sequence of events might lead to deadlock. Thus, we use Banker's algorithm to see if granting each resource request leads to a safe state. If it leads, request is granted, o/w postponed for (a)

PAGE

DATE

Banker's algorithm (deadlock avoidance)

Let 'n' be the number of processes in system and 'm' be the no. of resources types.

a. Available resources
It is a 1-dimensional array of size 'm' indicating the no. of available resources. Here, Available [j] = k. This means there are k instances of resource type Rj.

b. Maximum resources
It is a 2-dimensional array of size n x m that defines maximum demand of each process in a system
Max [i, j] = k.
This means process Pi may request at most 'k' instances of resource type Rj.

c. Allocation of resources
It is also 2-dimensional array of size n x m that define no. of resource of each type currently allocated to each process
Allocation [i, j] = k.
This means, process Pi is currently allocated 'k' instances of resource type Rj.

d. Needed resources.
It is also a 2 dimensional array of size n x m that indicates remaining resource needed for each process. Need [i, j] = k.
It means, process Pi currently needs 'k' instance of resource type Rj for its execution

Note: Banker's algorithm can also be used for deadlock detection.

DATE

Need [i,j] = Max[i,j] - Allocation [i,j]

Example: Here, customers = processes
Units = resources (like printer, tape recorder)
Banker = OS.

customers	used	Max	
A	0	6	
B	0	5	Available units = 10
C	0	4	
D	0	7	

Table 1

There are 4 customers. The banker has 10 units available. At certain time, the situation become

customers	used	Max	
A	1	6	
B	1	5	Available units = 2
C	2	4	
D	4	7	

Table 2.

(Safe state) Being in safe state means there is at least one way for all users to finish so ther will be no deadlock. The table 2 is in safe state, because with 2 units left, the banker can delay any request except C's, thus letting C finish and release all four resources. With 4 units in hand, banker can let either D or B have necessary units and so on.

(Unsafe state):-

Consider: What would happen if a request from B for one more unit were granted in above table 2. We would have following situation:

Customers	Used	Max	
A	1	6	Available units = 1
B	2	5	
C	2	4	
D	4	7	

Table 3

This is an unsafe state. If all customers A, B, C, D asked for their maximum loans, the banker could not satisfy any of them & we would have a deadlock.

limitations:
- requires fixed no. of resources & in advance.
- predicts all process returns within finite time but system doesn't guarantee it.

(5) Deadlock Recovery.

Traditional OS such as windows doesn't deal with deadlock recovery as it is time and space consuming process. Real time systems use deadlock recovery. Some of the common recovery methods are:

For resource:

- (Preempt the resource): We can snatch (steal) one of the resources from ~~the~~ process & give it to other process with the expectation that it will complete the execution and will release the resource sooner. It is a bit difficult process for choosing a resource.

DATE

- (Rollback to safe state): The OS can roll back the system back to the previous safe state. The system passes through various states to get into the deadlock state, so by implementing check point at every state the OS can rollback the system to the safe state whenever we get into deadlock.

For Process:

- (Kill a process): Killing a process whenever a system goes into deadlock can solve our problem but the bigger concern is to decide which process to kill. Generally, OS kills a process which has done least amount of work till now.

- (Kill all processes): This is not a good approach but still can be implemented if the problem becomes very serious. It makes system inefficient because all processes will need to execute again from the beginning.

PAGE

VIMP:

Some numerical Examples of Bankers Algorithm

Q. Consider a system with 5 processes. P_0, P_1, P_2, P_3, P_4 and 3 resources types A, B, C. The following scenario is given, A=10, B=5, C=7.

Process	Allocation A	B	C	MAX Need A	B	C
P_1	0	1	0	7	5	3
P_2	2	0	0	3	2	2
P_3	3	0	2	9	0	2
P_4	2	1	1	2	2	2
P_5	0	0	2	4	3	3

Available: A B C = 3 3 2

c. Determine the total sum of each type of resource?

a. Find the need matrix

b. Is the system is in safe state? Why/why not? If yes find the safe sequence.

a) ⇒ we know, Need[i,j] = Max[i,j] - Allocation[i]

Sol^n a)

Process	Need A	B	C
P_1	7	4	3
P_2	1	2	2
P_3	6	0	0
P_4	2	1	1
P_5	5	3	1

b) To check safe sequence, we need safety algorithm.

→ Check for each process, the need of that process with the availability.

If available ≥ need (or need ≤ available) then execute process.

Same

DATE

New available = available + Allocation

Else

donot execute

Go forward

i) For process P_1

Process	Need			Available.		
	A	B	C	A	B	C
P_1	7	4	3	3	3	2.

Since, available $\leq$ need. So, P_1 is not executed.
Let's Go to another process

ii) For process P_2

	Need			Available.		
P_2	1	2	2	3	3	2

Since, available $\geq$ need. so, P_2 is executed.

New available = Available + Allocated

$= 3\ 3\ 2 + 2\ 0\ 0$

$= 5\ 3\ 2.$

iii) For process P_3

	Need			Available		
P_3	6	0	0	5	3	2

Since, available $\leq$ need. So, P_3 is not executed.

~~New available = Available + Allocated~~

~~$= 5\ 3\ 2 + 6\ 0\ 0$~~

$=$

iv) For process P4

	Need			Available		
P4	2	1	1	5	3	2

Since, available $\geq$ Need. So, P4 is executed.
New available: Available + allocated
= 5 3 2 + 2 1 1
= 7 4 3

v) For process P5

	Need			Available		
P5	5	3	1	7	4	3

Since, Available $\geq$ need. So P5 is executed
New available: Available + allocated
= 7 4 3 + 0 0 2
= 7 4 5

Since, P1 and P3 are not executed still as sufficient resources were not available. Now, we test for them again

vi) For process P1

	Need			Available.		
P1	7	4	3	7	4	5

Sine, Available $\geq$ Need. So, we execute P1
New available: available + allocated
= 7 4 5 + 0 1 0
= 7 5 5

DATE

vii) For process P_3

	Need	Available
P_3	6 0 0	7 5 5

Since, Available ≥ Need. So, P_3 is executed.

New Available = Available + allocated

= 7 5 5 + 3 0 2

= 10 5 7

Here, we can see, the system is in safe state because system allocated all the resources types requested by all processes without entering into deadlock.

The safe sequence is:

< P_2 P_4 P_5 P_1 P_3 >

Q.c ⇒ Total amount of resources = Sum of columns of Allocation + Available = 7 5 5 + 3 3 3

= 10 5 7

Q. Let us consider 3 resources types A, B, C and 4 processes P_0, P_1, P_2, P_3. AT To we have following Snapshot of the system

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	1	0	1	2	1	1	2	1	1
P_1	2	1	2	5	4	4			
P_2	3	0	0	3	1	1			
P_3	1	0	1	1	1	1			

classmate

PAGE

DATE

a. Create the need matrix (max-allocation)
b. Is the system in a safe state? Why or why not?
c. If a request from process P0 arrives for (3 4 3), can the request be granted immediately?

a) Sol^n Let's make the need matrix

We know, need = Max - Allocation

Need matrix

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

b) Sol^n In order to check whether the system is in safe state, we run the safety algorithm.

(write algorithm)

i) For process P0.

	Need			Available		
	A	B	C	A	B	C
P0	1	1	0	2	1	1

Since, available >= Need. So, P0 is executed

New available = available + allocated

= 2 1 1 + 1 0 1

= 3 1 2

ii) For process P1

	Need			Available		
P1	3	3	2	3	1	2

DATE

Since, available ≤ Need so, P_1 is not executed.

iii) For process P_2.

	Need	Available.
P_2	0 1 1	3 1 2

Since, Available ≥ Need, so P_2 is executed.

New available: available + allocated

= 3 1 2 + 3 0 0

= 6 1 2

iv) For process P_3

	Need	Available
P_3	0 1 0	6 1 2

Since, available ≥ Need, so P_3 is executed.

New available = available + allocated

= 6 1 2 + 1 0 1

= 7 1 3

Since, P_1 isn't executed yet. So, let's check again

v) For process P_1

	Need	Available.
P_1	3 3 2	7 1 3

Since, available ≥ Need, so P_1 is executed.

New available = 7 1 3 + 2 1 2

= 9 2 5

So, the system is in safe state. as ∴

The safe sequence is < P_0 P_2 P_3 P_1 >

classmate PAGE

c) Here, for process P0

	Need (request)			Available		
P0	3	4	3	2	1	1

Since, Available ≤ Need. So, the request from P_1 can't be granted immediately.

DATE

Questions asked from this chapter in board exam

Q. How unsafe state differs from deadlocked state? Consider following initial state and identify whether requested resource is and granted or denied for the given cases:

Process	Has (Allocated)	Max
A	2	6
B	1	5
C	2	3
D	3	8

Free = 2

- What will happen if process D request 1 resource?
- What will happen if process A request 1 resource?

(2078 - 10 marks)

Q. Why do deadlock occur? How can you detect and recover from deadlock? (2075 - 5 marks)

Q. A system has 2 processes & 3 identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain (2066 - 5 marks)

Q.

PAGE

classmate

DATE

Q. Considering a system with 5 processes P_0 through P_4 and 3 resource type A, B, C. Resource A has 10 instances, B has 5, & C has 7

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			

Q.a What will be the need matrix?

Q.b Is system in safe state? If yes, then what is the safe sequence?

Q. What will happen if process P_1 requests one additional instance of resource type A and two instances of resource type C?

~~If request from P_1 arrives for~~ (1, 0, 2) ~~can it be~~ (immediately granted)

or.

=> Q.a & b already done in 1st question

Q.c Request$_1$ for P_1 = A B C
1 0 2

We know,

New available = available - request$_1$

Allocation$_1$ = Allocation$_1$ + Request$_1$

Need$_1$ = Need$_1$ - Request$_1$

DATE

we know,

∴ Request 1 = 1 0 2

So, New available = 3 3 2 - 1 0 2

= 2 3 0

New allocation for P_1 = 2 0 0 + 1 0 2

= 3 0 2

New need for P_1 = 1 2 2 - 1 0 2

= 0 2 0

Process	Allocation			NEED			Available		
	A	B	C	A	B	C	2	3	0
P_0	0	1	0	7	4	3			
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	2	1	1			
P_4	0	0	2	5	3	1			

Now, again repeat all steps for this. and find safe sequence.

If it is in safe state, resource will be immediately granted

PAGE

classmate