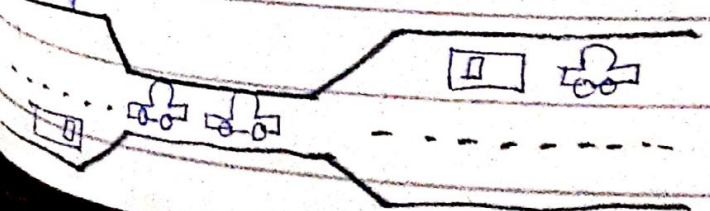
Ch#:-

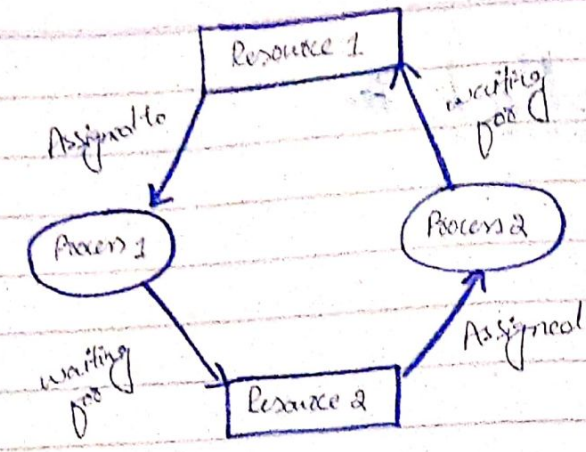
Deadlock

Definition:-

- A process in Operating System uses resources in the following way:
  - ★ Request a resource
  - ★ Use the resource
  - ★ Releases the resource
- A deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- This is because the system has finite number of resources.
- Deadlock is a common problem in multiprocessing where several processes share a specific type of mutually exclusive resource known as Soft lock or Software.

Example:





Examples:

- The system has 2 tape drives.

⇒ P_1 and P_2 each hold one of tape drive and each needs another one.

- Semaphores A and B, initialized to 1, P_0 and P_1 are in deadlock as follows:
  - P_0 executes wait(A) and preempts.
  - P_1 executes wait(B).

Now P_0 and P_1 enter in deadlock.

P_0	P_1
wait(A)	wait(B)
wait(B)	wait(A)

Resources

- A resource is an object that is used by process.
- It can be a piece of hardware:
  - Tape drive
  - Disk drive
  - Printer
- A resource can be a piece of information such as:
  - A record within a file
  - A shared variable
  - A critical section.

Types of Resources:-

i) Preemptible Resources:

- A preemptible resource is one that can be allocated to a given process for a period of time.
- These can be allocated to another process and can be reallocated to the first process without causing the deadlock.

Examples:

- Memory
- Buffers
- CPU
- and
- array of processes.

Non-preemptible Resources:-

- A non-preemptible resource is one that can be allocated to a given process can not be allocated for another process without side effects.

EX:

A printer.

Necessary Conditions for deadlock

- The deadlock situation can only arise if the following four conditions hold simultaneously.

(1) Mutual Exclusion:-

According to this condition,

- Atleast one resource should be non-shareable
- Non-Shareable resources are those that can be used by one process at a time.

(2) Hold and wait condition:

According to this condition:

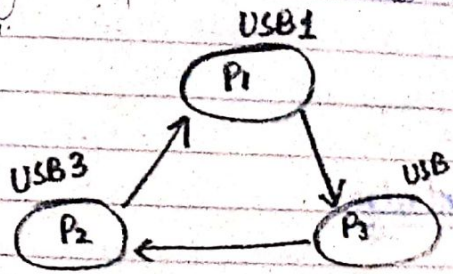
- A process is waiting for additional resources and a process is holding at least one resource.

(3) No Preemption:-

- once a process is holding a resource then that resource cannot be taken away from that process until the process voluntarily releases it.

(4) Circular wait:-

- Once process is waiting for the resources which is held by the second process which is also waiting for the resource held by the third process etc.
- This will continue until the last process is waiting for a resource held by the first process. This creates a circular chain.



Resource-Allocation Graph

- It is a way to represent the situation of a deadlock graphically.
- It is a directed graph.
- It consists of set of vertices and set of edges.

Vertices:

- There are two types of vertices:-
  - Process vertices
  - Resource vertices.

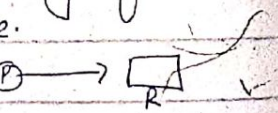
Edges:-

- There are two types of edges:
  - Request Edge
  - Assignment edge

Request edge:-

- A set of directed arcs from P_i to R_j, indicating that process P_i has requested R_j, and is currently waiting for that resource to become available.

$P_i \rightarrow R_j$



Assignment edge:

- A set of directed arcs from R_j to P_i indicating that resource R_j has been allocated to process P_i and that P_i is currently holding resource R_j.

Note:

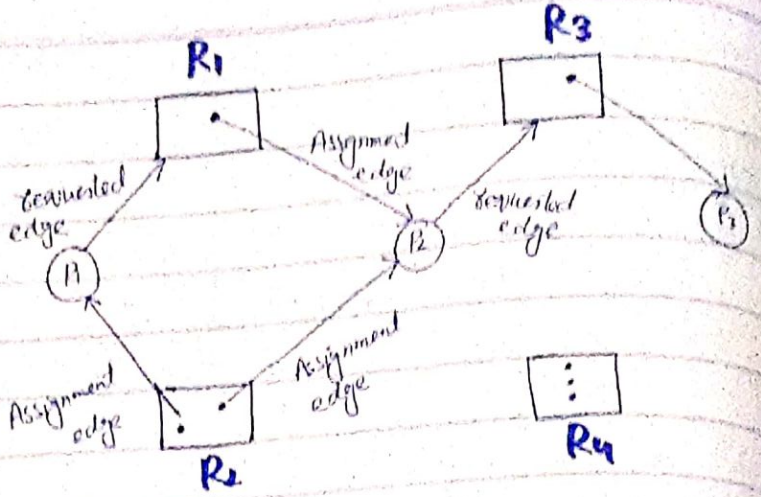
⇒ Requested edge can be converted into an assignment edge by reversing the direction of the arc when the request is granted.

⇒ If a resource-allocation graph contains no cycles, then the system is not deadlocked.

⇒ If a resource-allocation graph does contain cycles And each resource category contains only a single instance, then a deadlock exists.

⇒ If a resource category contains more than one instance, then the presence of a cycle in the resource-allocation graph indicates the possibility of a deadlock but does not guarantee.

Instance of resource:-

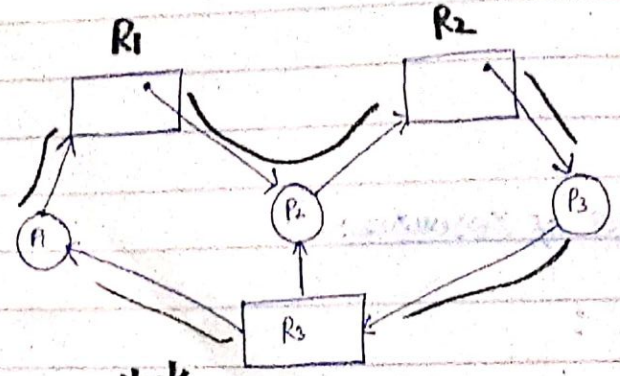


$P = \{ P_1, P_2, P_3 \}$

$R = \{ R_1, R_2, R_3, R_4 \}$

$E = \{ P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3 \}$

Situation where cycle & confirm Deadlock:-

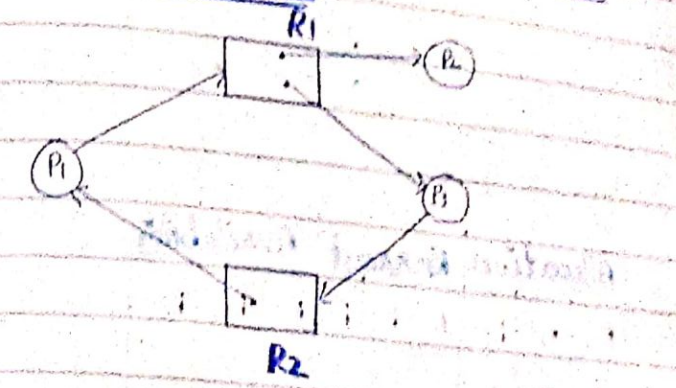


Deadlocks

	Allocation			Requested			Available		
	P_1	P_2	P_3	P_1	P_2	P_3	P_1	P_2	P_3
P_1	0	0	1	1	0	0	0	0	0
P_2	1	0	1	0	1	0	0	0	0
P_3	0	1	0	0	0	1	0	0	0

$\{P_1 P_2 P_3\} = \{0\ 0\ 0\}$

Situation where cycles but No deadlock:-



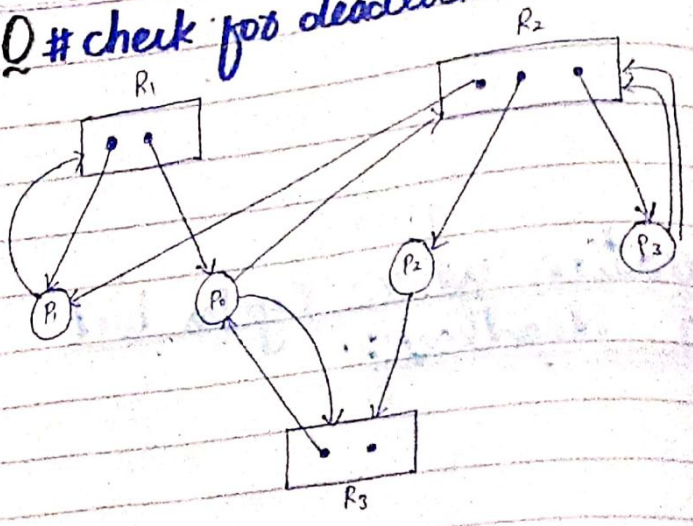
Pros

- It helps us to understand:
  - which resource is required by the process
  - How many resources are required by the Process?
  - How many resources are available?

Cons

- It is better to store the resource allocation detail in a table if the system contains many processes and resources. RAGs is better only when the system contains fewer process & resource.

Q # check for deadlock.



	Allocation			Request			Availability		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1	0	0	1
P_1	1	1	0	1	0	0	0	1	1
P_2	0	1	0	0	0	1	0	1	2
P_3	0	1	0	0	2	0	2	2	2

Safe sequence? P_2, P_0, P_1, P_3

Deadlock? No

Methods for Deadlock:-

There are four methods for handling deadlock:

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection and recovery
- Deadlock ignorance

Deadlock:-

⇒ This strategy involves designing a system that violates one of the four necessary conditions required for the occurrence of deadlock.

⇒ This ensures that the system remains free from the deadlock.

⇒ The various conditions of deadlock occurrence maybe violated as:

Mutual Exclusion:-

⇒ To violate this condition, all the system resources must be such that they can be used in shareable mode.

⇒ In a system there are always some resources which are mutually exclusive by nature.

⇒ So, this condition cannot be violated. Shareable resources do not require mutually exclusive access and thus cannot be involved in a deadlock.

⇒ Read only files are good example of Shareable resources.

⇒ In general, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-shareable.

for example: a mutex lock cannot be simultaneously shared by several threads.

(2) Hold and wait:

This condition can be violated in the following ways:

(1) Only holding no waiting:-

⇒ A process has to first request for all the resources it requires for execution.

⇒ Once it has acquired all the resources, only then it can start its execution.

⇒ This approach ensures that the process does not hold some resources and wait for other resources.

Cons:

⇒ It is less efficient.

⇒ It is not implementable since it is not possible to predict in advance, which resources will be required during execution.

Approach-02:-

⇒ A process is allowed to acquire the resources it desires at the current moment.

⇒ After acquiring the resources, it start its execution.

⇒ Now before making any new request, it has to compulsorily release all the resources that it holds currently.

⇒ This approach is efficient and implementable

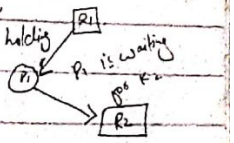
Approach-3:

⇒ A times is set after the process acquires any resource.

⇒ After the times expires, a process has to compulsorily release the resource.

Cons:

- Low Resource utilization.
- Starvation



No preemption:-

Preemption of resources means take away resources from processes when they are waiting for other resources.

⇒ This condition can be violated forceful preemption.
⇒ Consider a process is holding some resources and request other resources that cannot be immediately allocated to it.
⇒ All of its held resources must be released.
o Now the process is waiting for previously held resources and new resources.
⇒ Then, by forcefully preempting the currently held resources, the condition can be violated.
⇒ A process is allowed to forcefully preempt the resources possessed by some other process only if:
⇒ it is a high priority process or system process.
⇒ The victim process is in the waiting state.

Use:-

o This protocol is often used or applied to resources whose state can be easily saved and restored later, such as CPU registers and database transactions.
o It cannot apply to resources such as mutex locks and semaphores.

(4) Circular Wait:

⇒ This condition can be violated by not allowing the processes to wait for resources in a cyclic manner.
⇒ We can define order by which processes get resources to prevent circular wait.
⇒ Each resource is assigned a unique number.
⇒ Each process is allowed to request for the resources either in only increasing or only decreasing order of the resource numbers.
⇒ In case increasing order is followed, if a process requires a lower number resource, then it must release all the resources.
⇒ This approach is the most practical approach and implementable.

Example:

o Suppose a tape drive has number 2, disk drive has number 5 and printer has number 12.
o A process wants to read the disk drive and print out the results.

- It will first need to deallocate the disk drive then the printer.
- it will be prevented from doing it in reverse order.

(2) Deadlock detection & Recovery:

=> if a system does not use a deadlock prevention technique or the deadlock avoidance technique, then is possibility that a deadlock will occurs.

=> In order to avoid deadlocks, the OS examines the system for any deadlocks a regular basis.

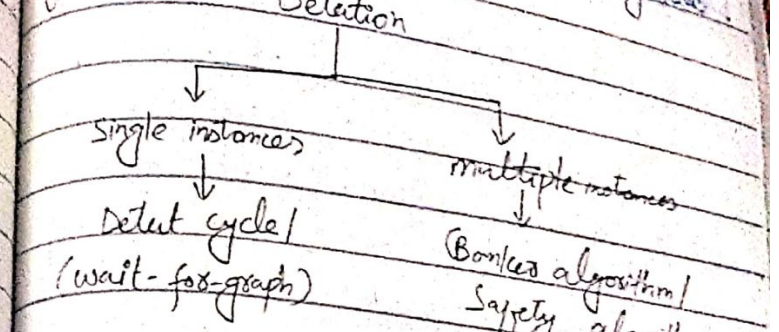
=> The OS will recover from the deadlock using recovery mechanisms after it has discovered.

How to detect deadlock in OS?

=> The OS periodically checks the system for any deadlock.

=> If it finds any of the deadlock then OS will recover the system using some recovery techniques.

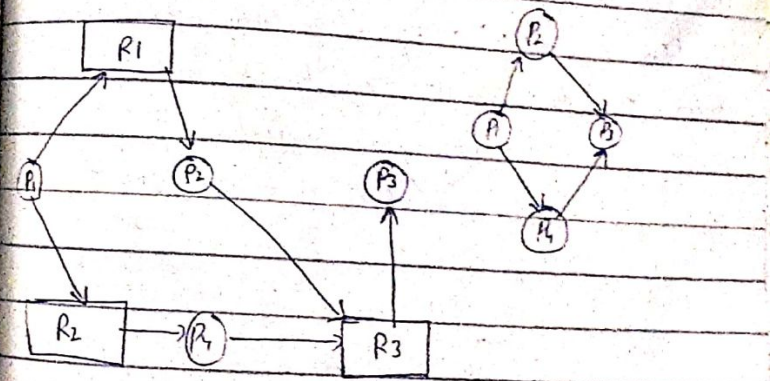
The OS can detect deadlocks by the following:



Wait-for-graph:

- it is a variant of resource allocation graph.
- The only difference is only process are written not resources.

Example:



RAG

Wait-for-graph

Detection Algorithm Usage:

=> Detection algorithms need to be executed in order to detect deadlock.

=> The frequency and time when we run such algorithm is dependent on how often we assume deadlocks occurs and how many processes they may affect.

(1) Banker's algorithm

(2) Safety algorithm

(3) Deadlock Recovery

=> OS examines either resources or processes to recover the system from deadlocks.

When a detection algorithm determines that a dead lock exists.

Recovery from Deadlock

- Processes
  - Abort one process at a time until the elimination of deadlock cycle.
  - Abort all deadlocked processes
- Resources
  - Preemption
  - Rollback

1: Process Termination:

- In order to eliminate deadlock by aborting a process, the following methods are used.

Aborting all deadlocked processes:

=> it is helpful in breaking the cycle

=> Expensive approach & not suggestable but can be used if problem become very serious.

=> If all the processes are killed then there may occur insufficiency in the system and all processes will execute again from starting.

Abort one process at a time until the elimination of the deadlock cycle.

=> Can be used but we have to decide which process to kill and this method incurs considerable overhead.

=> The process that has done the least amount of work is killed by the OS firstly.

=> Some priority must be considered while killing a process.

=> The priority might be:

- How many resources are being held by that process.
- How long has it executed.
- How long it has to go before it completes.
- How many processes will have to be terminated.
- How many resources it needs to complete its job etc.

a: Resource preemption:-

=> This approach takes resources from waiting processes and gives them to other processes. The victim process cannot continue regularly.

Preempt the resource:

- Snatching the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and release this resource sooner.
- In this case, the system goes into starvation.

b) Rollback:

- When a resource is preempted from a process, it cannot continue its normal execution.
- That process should be rollback to a safe state so that it can be restarted later.

Advantages:

=> Improved system stability
=> Better resource utilization
=> Better system design

Disadvantages:

- Performance Overhead
- Complexity
- False positive & negatives
- Risk of data loss

Deadlock Avoidance

=> It is a technique used in OS to prevent the situation where two or more processes are unable to proceed because each is waiting for one of the other to release a resource.

=> The deadlock avoidance is used by the OS in order to check whether the system is in a safe state or unsafe state, in order to avoid deadlock.

-> The process must need to tell the OS about the maximum number of resources a process can request in order to avoid deadlock & complete its execution.

Safe state:

=> A state is safe if the system can allocate resources to each process (up to its maximum requirements) in some order and still avoid a deadlock.

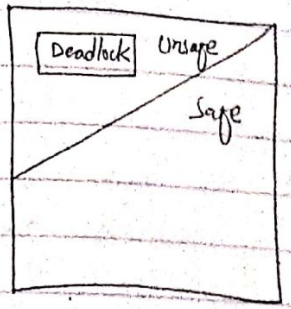
=> Formally, a system is in safe state only if there exists a safe sequence.

=> So, safe state is not a deadlock state. Can be more than one

Unsafe State:

=> In an unsafe state the OS cannot prevent processes from requesting resources in such a way that any deadlock occurs.

=> It is not necessary that all unsafe states are deadlocks; an unsafe state may lead to a deadlock.



+ example:-

=> Note:

if the system is unable to fulfill the request of all processes then the state of the system is called safe state.

=> Deadlock avoidance can be done with the help of:
- Banker's Algorithm
- Resource allocation graph

Banker's Algorithm:-

=> It is deadlock avoidance algorithm. It is named because this algorithm is used in banking systems to determine whether a loan can be granted or not.

=> Consider there are n account holders in a bank and the sum of the money in all of their account is S.

=> Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S.

=> It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

=> Banker's algorithm works in a similar way in computers.

=> Whenever a new process is created, it must specify the maximum instances of each resource type that it needs exactly.

Characteristics of Banker's Algorithm

The characteristics of banker's algorithm are as follows:

- If any process requests for a resource then it has to wait.
- This algorithm consists of advanced features for maximum resource allocation.
- There are limited resources in the system we have.
- In this algorithm, if any process gets all the needed resources, then it should return the resources in a restricted period.
- Various resources are maintained that can fulfill the need of at least one client.

Consider:

- There are n processes
- There are m resource types.

Data Structure used to implement the algorithm are:-

⇒ Data Structures that are used to implement the bankers algorithm are as follows:

(1) Available:

- it is an array of length m.
- it represents the number of available resources of each type.

(2) Max:

- it is an nxm matrix that represents each the maximum requirements of Type.

(3) Allocated:

- it is an nxm matrix that represent the number of resources that each resource type currently allocated to each proces

(4) Need:

- it is an nxm matrix that represent the remaining resource need of each pr

Bankers algorithm comprised:

⇒ Comprises of two algorithms

- Safety Algorithm
- Resource request algorithm

Safety Algorithm

⇒ it is used to find whether or not system is in its Safe state.

Resource request Algorithm

⇒ used to determine whether requests can be granted safely or not granted.

① How to find need matrix?

Need = Max need - currently allocated

② How to find max matrix?

Max need = Need + currently allocated

Disadvantages:

- During the time of processing, this algorithm does not permit a process to change its maximum need.
- All the processes must know in advance about the maximum resource needs.

Criteria

Request ≤ Available

Example:

Proceries	Allocation A	B	C	Man A	B	C	Available A	B	C
P_0	1	1	2	4	3	3	2	1	0
P_1	2	1	2	3	2	2			
P_2	4	0	1	9	0	2			
P_3	0	2	0	7	5	3			
P_4	1	1	2	1	1	2			

(1) find need matrix?
(2) check whether the system is in safe state?
(3) Determine the total sum of each resource type?

Need matrix

Need = man - Allocation

Process	Need A	B	C
P_0	3	2	1
P_1	1	1	0
P_2	7	3	3
P_3	0	0	0
P_4			

Safe Sequence:

(1)

⇒ For Process P_0, Need = (3, 2, 1) and Available = (2, 1, 0)

⇒ Need <= Available = False

So, the system move to the next process

(2)

⇒ For process P_1, Need = (1, 1, 0) and Available = (2, 1, 0)

∘ Need <= Available = True

Request P_1 is granted.

∘ Available = Available + Allocation

= (2, 1, 0) + (2, 1, 2)

= (4, 2, 2) (New available)

.
.
.

Safe sequence:

$\langle P_1, P_4, P_2, P_3, P_0 \rangle$

Past Papers

1) Differentiate between progress and bounded waiting time.
2) What is the difference between deadlock avoidance, prevention and detection?
3) What are deadlock characterizations?
4) When cycle is both necessary and sufficient to detect deadlock?
5) When and for what purpose bankers algorithm is used?
6) What are necessary conditions for deadlock?
7) Define Spinclock?
8) What is deadlock?
9) What is circular wait condition in deadlock?

Long

1) Write about deadlock conditions & bankers algorithm in details.
2) What is deadlock? Explain in detail how deadlock can be prevented?
3) Write down bankers algorithm & Safety algorithm?

Q What could be the possible conditions, which can ensure that a deadlock will not occur? explain?

Q What are deadlock avoidance? write details of bankers algorithm.

Q) For the given data apply the bankers algorithm? (repeat)

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			

(a) Can request for (3,3,0) by P_4 be granted? (NO)

(b) Can request for (0,2,0) by P_0 be granted? Yes.

Q) Use banker algorithm and determine the system is in safe state or find safe sequence and check for deadlock?

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	1	1	2	0	1	1	2	1	5	2	0
P_1	0	0	5	0	1	7	5	0	1	6	3	2
P_2	1	0	5	4	2	2	5	6	1	8	7	4
P_3	0	1	2	0	0	6	5	3	1	8	12	4
P_4	0	2	4	2	0	6	5	6	2	8	17	8
	1	4	17	8					2	9	19	8

Total no of resources = Allocated + Available

= 1 4 17 8

1 5 2 0

2 9 19 8

A	B	C	D
2	9	19	8

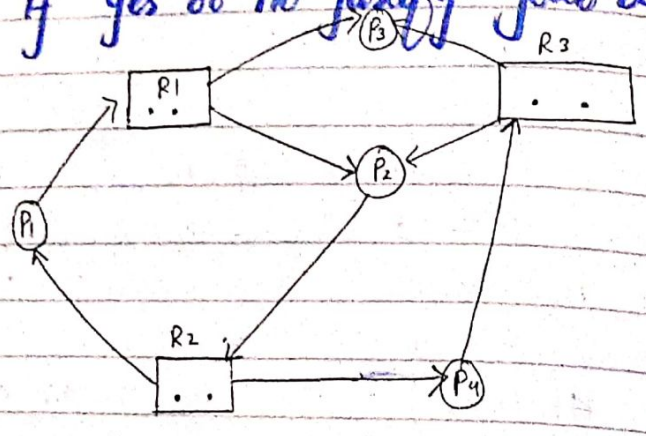
Need matrix			
A	B	C	D
0	0	0	0
1	7	0	0
1	2	0	2
0	5	3	3
0	4	1	4

Safe sequence:

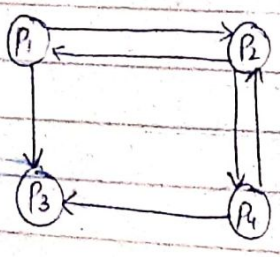
$\langle P_0, P_4, P_1, P_2, P_3 \rangle$

=> There is no deadlock System is in safe state.

Q) Given the following resource allocation graph:
(a) Draw wait for graph?
(b) Determine whether or not there is deadlock?
If yes or no justify your answer.



(a)



(b) Check for deadlock:

Process	Allocate			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	0	1	0	0	0	0	0
P_2	1	0	1	0	1	0	1	0	0
P_3	1	0	1	0	0	0	1	1	1
P_4	0	1	0	0	0	1	2	1	2
							2	2	2

Safe sequence:

$P_3 \rightarrow P_1 \rightarrow P_2 \rightarrow P_4$

So, there is no deadlock because P_3 does not need any resource it can be granted permission to executed
=> Even though there are cycles but it is multiinstances RAG so, in this case there may or may not be deadlock
=> In our case there is no deadlock.

#2020 X

	A	B	C	1	1	3
P0	0	1	0	0	2	0
P1	3	0	2	6	0	0
P2	3	0	2	0	1	1
P3	2	1	1	4	3	1
P4	0	0	2			

#2020

Process	Allocated			Need			Available			Max-Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0	7	5	3
P1	3	0	2	0	2	0	5	3	2	3	2	2
P2	3	0	2	6	0	0	7	4	3	9	0	2
P3	2	1	1	0	1	1	7	4	5	2	2	2
P4	0	0	2	4	3	1	7	5	5	4	3	3
	8	2	7				10	5	7			

Safe sequence:

<P1, P3, P4, P0, P2>

Total:

8 2 7
+ 2 3 0
10 5 7

Deadlock Prevention VS Deadlock detection

Deadlock prevention

Def: It blocks at least one of the conditions necessary for deadlock to occur.

Resource request:

2 All the resources are requested together.

Information required

o It does not requires information about existing resources, available resources and resource requests.

Procedure:

o it prevents deadlock by constraining resource request process and handling of resources.

Deadlock Avoidance

o it ensure that system does not go in unsafe state

o Resource requests are done according to the available Safe path.

o it requires such type of information.

o it automatically considers requests & check whether it is safe for system or not

Preemption	There is no preemption
Occurs more frequently	
Resource allocation Strategy:	
Conservative	Not Conservative
Future resource request:	
Does not require any knowledge of future resource request	it requires the knowledge of future resource request type.
Pros:	
Does not have any cost	Costly
Examples:	
Spooling & non-blocking synchronization algorithms are used.	Bankers & safety algorithms are used.

Q When cycle is both necessary and sufficient to detect deadlock?

- if there is a cycle in the graph and each resource has only one instance, then there is deadlock.
- In this case cycle is necessary & sufficient condition for deadlock.
- if there is a cycle in the graph and each resource has more than one instance, then there may or may not be deadlock.

Q. Define livelock? Difference b/w livelock & deadlock?

- Live lock is a variant of deadlock.
- This is a situation in which two or more processes continuously change their state in response to change in the other process without doing any useful work.
- This is similar to deadlock in that no progress is made

but differ in that neither process is blocked or waiting for any thing.

Deadlock

- it is situation where a process or set of processes is blocked and waiting for an event that will never occur.
- It is common in CS. ~~Computer Science~~ where resource sharing is frequent.
- it can be handled by various deadlock handling strategies like prevention, detection & avoidance.

Livelock

- Live lock is a situation in which single failure of a process can cause an infinite no of rollbacks preventing the system from making progress.
- it is common in distributed system where synchronization is maintained by message passing.
- Can be prevented by coordinating the processes either at the time of checkpointing or at the beginning of recovery.

Q11 Define Spin clock?

- A spin lock is a type of lock used in operating systems and concurrent programming to protect shared resources.
- It employs busy waiting, or spinning, where a thread repeatedly checks the lock's availability instead of blocking or suspending.
- Spin locks are efficient when the anticipated wait time is short, but they can waste CPU cycles if the wait is too long.
- They are commonly used in low-level kernel code, while higher-level programming languages often provide more user-friendly synchronization abstractions like mutexes.

Q. Describe a monitor Solution to the dining - philosphers problem?

```
monitor dining philosphers
{
enum { THINKING , HUNGRY, EATING } state[5];
 condition self[5];
void pickup(int i)
{
   state[i] = HUNGRY;
    test(i);
 if ( state[i] != EATING)
        self[i].wait();
}
void putdown(int i)
{
   state[i] = THINKING;
    test ( (i+4)%5);
    test ( (i+1)%5);
}
void test (int i)
{
 if ( (state[(i+4)%5] != EATING) &&
      (state[i] == HUNGRY) &&
      (state[(i+1)%5] != EATING))
{
 state[i] = EATING;
 self[i].signal();
}
}
initialization_code() {
for (int i = 0; i<5; i++)
      state[i] = THINKING;
}
```

Q. Describe a monitor Solution to Reader-writer problem?