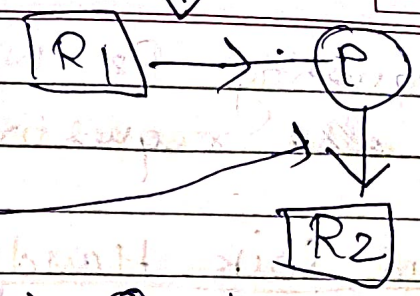
* Deadlock | Part:-1

- A Situation where each participant is waiting for a resource held by another participant in the group forming a circular dependency is called as deadlock.
- It is an bug present in the process / thread Synchronization method.
- RAG (Resource Allocation Graph)

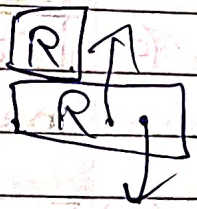
A directed graph used to visualize the state of processes and their resource.

① Vertex → ① Process Vertex = P
② Resource Vertex = R

(2) Edges → (1) Assign
(2) Request

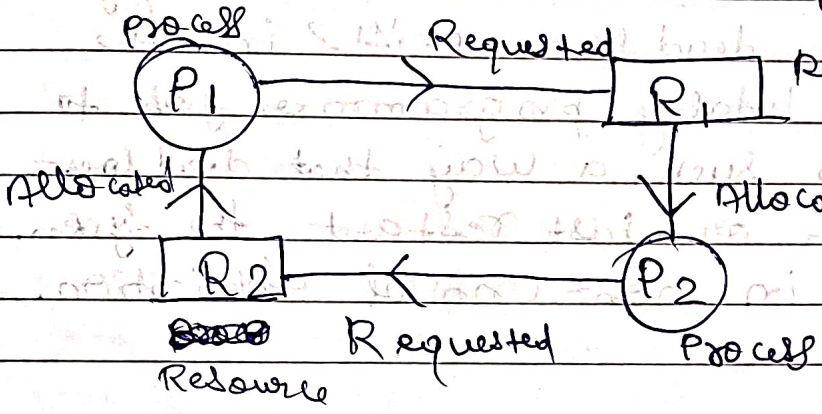


(3) Resource instance → (1) single
(2) Multiple



- How a process / thread utilize a resource?

(a) Request :- Process "P_1" request for "R_1" resource
(b) Use the resource
(c) Release :- Now available for other processes.



- Deadlock necessary condition

4 necessary condition :-

(a) **Mutual Exclusion** :- only 1 process at a time can use the resource, other processes wait until the resource has been released.

(b) **Hold & Wait** :- A process must be holding atleast one resource and requesting to acquire additional resources.

(c) **NO-preemption** :- Resource must be released by the process after completing the execution. voluntarily

(d) **Circular Wait** :- P_1 is held R_1 (resource) and

Date:

requesting R2 which is held by P2 which is requesting R1.

- **Dead lock Handling**

(a) **Prevent or avoid** :- ensuring the system will never enter in a deadlock situation.

(b) **Detect & recover** :- If system goes in deadlock then detect it and recover it.

(c) **Ostrich algorithm** :- Here OS simply ignore that dead lock exists in the system, it's totally programmer job to implement in such a way that dead lock will not occur or just restart the system. Mostly used in non-critical application.

- **Deadlock Prevention**:- Ensuring that at least one of the necessary condition of the deadlock cannot hold.

(a) **Mutual Exclusion** :-

i) use mutual locks only for non-sharable resource.

ii) Sharable resources like-read only files can be accessed by multiple processes / threads

iii) However, we can't prevent deadlock by using this only because some resources non-sharable.

Page No.:
Date:

(b) Hold & Wait :-

To ensure this condition never happens we use below two ways :-

i) Protocol (A), each has to request all its required resources and allocate it before its execution.

ii) Protocol (B), allow a process to request resources only when it has none, it can request additional resources after it must released all the resources that it is currently allocated.

(c) No preemption :-

i) If a process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the process is currently holdings are preempted. The process will restart only when it can regain its old resources, as well as the new one that is requesting. Now, if apply the lock then it take 1 sec gap to every resource to apply to avoid the issue of live lock (a situation where two or more processes continuously change their state in response to each other, without making any actual progress).

ii) If a process requests some resources, we first check if not available, then we check whether they are allocated to some other process that is waiting for additional resources, if so then we preempt the desired resource from waiting process and allocate them to the requesting process.

(d) Circular Wait :- P1 & P2 both require R1 and R2 resources, first both try to lock R1 process and whoever acquire the R1 then only that process acquire R2. R2 is still waiting for R1 resource.