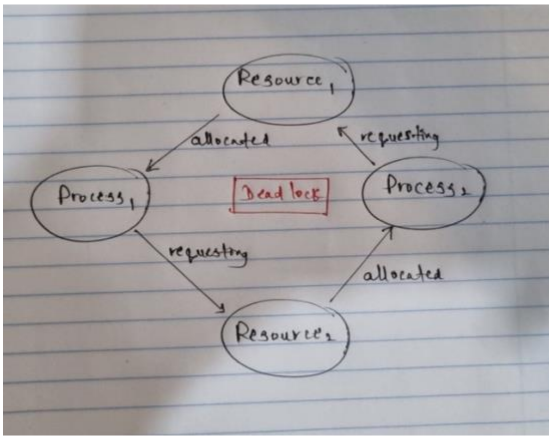
Deadlocks

A process requests resources; and if the resources are not available at that time, the process enters a waiting state.

Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting process…this situation is called a deadlock.

A process uses a resource in the following order,

1. **Request** – is the request cannot be granted immediately, then the requesting process must wait until it can acquire.
2. **Use** – process can operate on the resource.
3. **Release**



Deadlock Characterization

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

Features that characterize deadlocks – necessary conditions

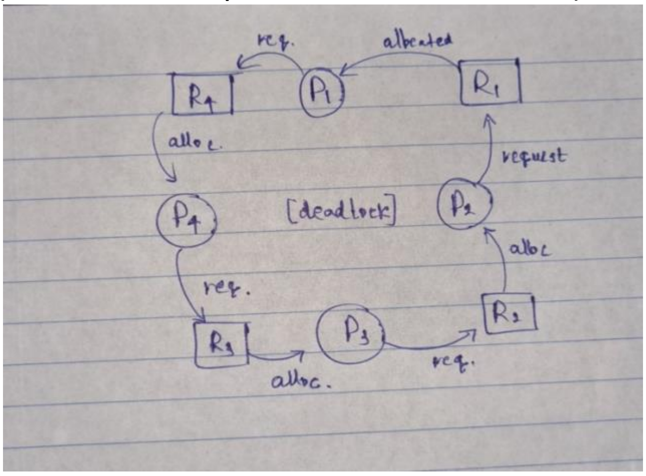
A deadlock situation can arise if the following 4 conditions hold **simultaneously** in a system.

1. Mutual exclusion
   A least one resource must be in a **non-sharable mode;** that is, only one process at a time can use the resource.
   If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait
   A process must be **holding at least one resource** and **waiting to acquire additional resources** that are currently being held by other processes.

3. No preemption
   Resources cannot be preempted; that is, **a resource can be released only voluntarily by the process holding** it, after that process has completed its task.

4. Circular wait
   A set of processes are waiting for each other in circular form.
   (2 and 4 are dependent on each other)



All four conditions must hold true for a deadlock to occur. The **circular-wait implies hold-and-wait**, so the 4 conditions are not completely independent.

Resource Allocation Graph

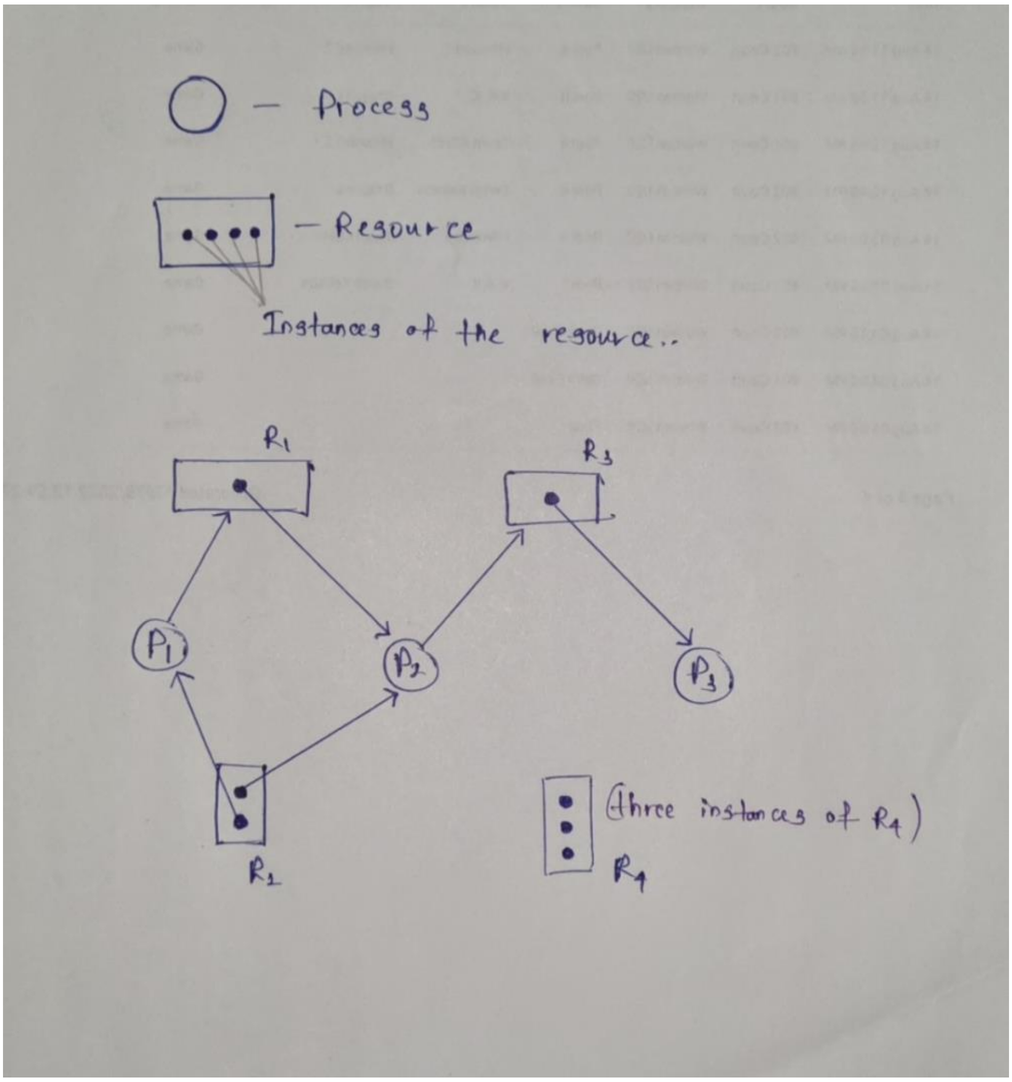
This is a directed graph called a **system resource-allocation graph.** Consists of edges E and vertices V. vertices are partitioned in to two types.

P = (P_1, P_2,…P_n) all the **active processes** in the system
R = (R_1, R_2,…R_n) all the **resource types** in the system

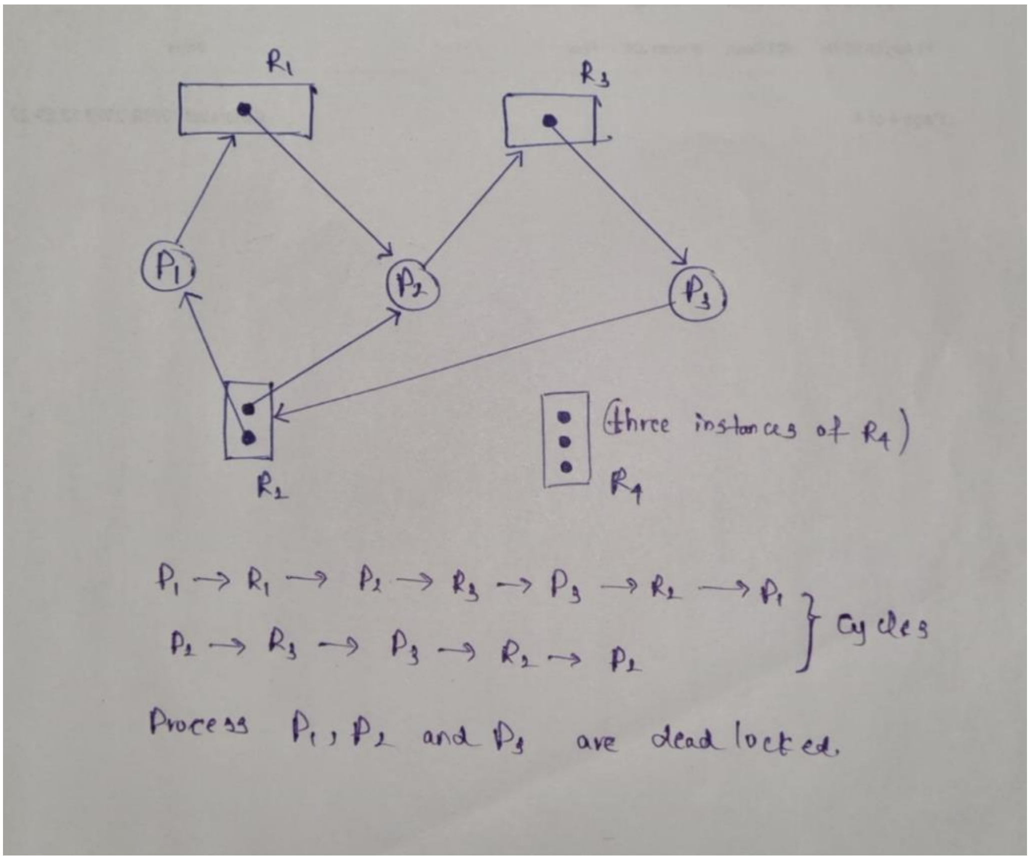
Two types of edges,

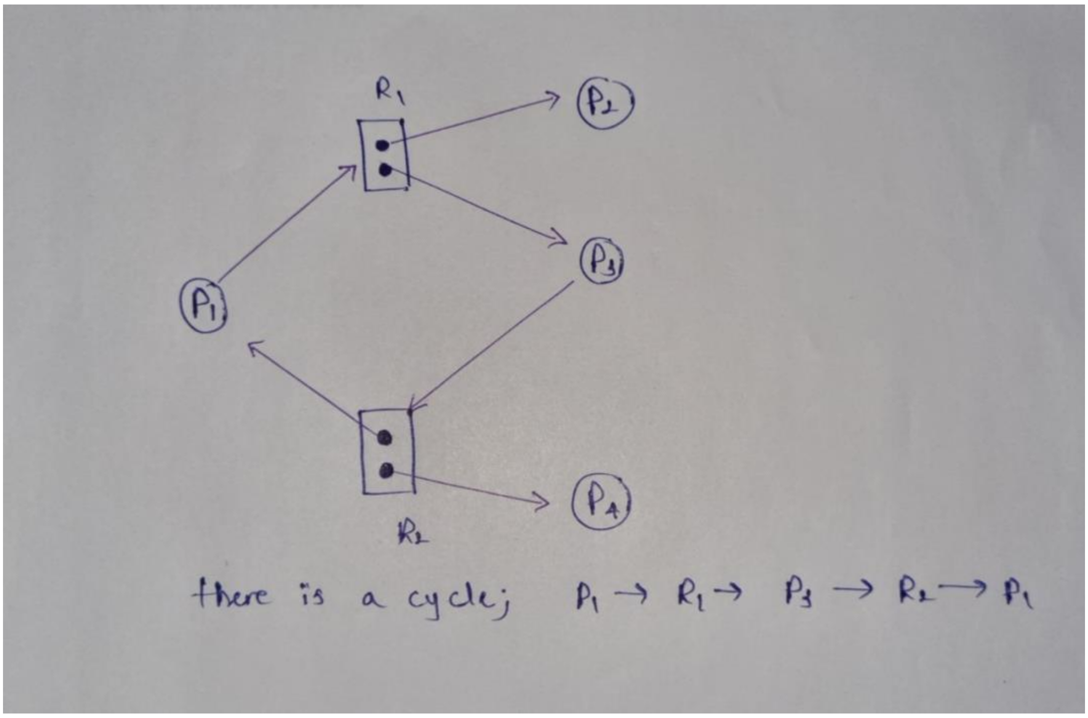
1. [Request edge]
   Directed edge from process P_i to resource R_j, **Pi → Rj** means that **process Pi has requested an instance of resource Rj**, and its currently waiting for the resource.
2. [Assignment edge]
   Directed edge from resource type Rj to process Pi, **Rj → Pi** means that an instance of **resource type Rj has been allocated to Pi**

if the graph has **no cycles**, then no process on the system is deadlocked



If the graph does contain a cycle, then a deadlock **may** exist.





However, there is **no deadlock.** P4 may finish execution and release its instance of R2. Then that can be allocated to P3, breaking the cycle.

Cycle is a necessary condition for a deadlock but it is not a sufficient condition. if there's a deadlock, a cycle must be there, but if there's a cycle doesn't mean that there's a deadlock.

Methods for handling deadlocks

We can deal with deadlocks in one of three following ways;

1. Use a protocol to prevent or avoid deadlocks
2. Allow a system to enter a deadlock state, detect it and recover.
3. Ignore the problem altogether and pretend that deadlocks never occur in the system

(1)

To ensure that deadlocks never occur the system can use either a deadlock prevention or a deadlock avoidance scheme.

Deadlock prevention → Provides a set of methods for ensuring that at least one of the necessary conditions cannot hold.

1) Mutual exclusion
2) Hold and Wait
3) No preemption
4) Circular wait

Deadlock avoidance → Requires that the OS be given in advance additional information concerning which resources a process will request and use during its life time..
With this additional knowledge, it can decide for each request whether or not the process should wait.

(2) If a system does not use either deadlock prevention or deadlock avoidance algorithm, then a deadlock situation may arise.

In this environment the system can provid an algorithm that examines the state of the system to determine whether a deadlock has occured, and an algorithm to recover from the deadlock. (If a dl has indeed occured)

(3)

If a system neither ensures that a deadlock will never occur nor provides a mechanism for dl detection and recovery, then a system in a dl state has no way of recognizing what has happened

the undetected dl will deteriorate system's performance.
because resources are being held by processes that cannot run, more and more processes ~~enter~~ requesting resources enter in to dl state.

The system will stop functioning and will need a manual restart.

Deadlock Prevention

1) Mutual exclusion
2) Hold and Wait
3) No preemption
4) Circular wait

} All of these conditions must hold true for a deadlock to occur.

At least one of these does not hold → no deadlock.

1. Mutual exclusion

The mutual exclusion condition must hold for nonsharable resources

ex:- printer; multiple processes can not share a printer simultaneously.

Sharable resources ... do not require mutually exclusive access... cannot be involved in a dl.

ex:- read-only-files

In general we cannot prevent dl's by denying mutual exclusion; becuz some resources are non-sharable.

2. Hold and Wait

To ensure that this condition never occurs; we must garantee that whenever a process requests a resource; it does not hold any resources.

① Each process must request and be allocated all its resources before it begins execution.
(Resource utilization may be low)

② a Process can request resources only when it has none.... it must ~~wa~~ release all the resources currently allocated before requesting additional resour...
(Starvation could occur)

* disadvantages

3. No Preemption

If a process is holding some resour and requests another resource; that is not available (process must wait), then all resources currently being held are preempted.

or

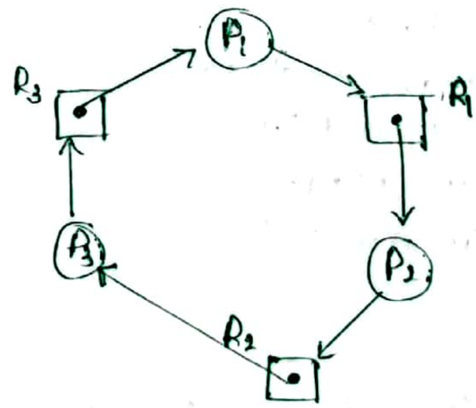
We check whether the requested resources are allocated to some other process that is waiting for additional resources, if so we preempt the desired resources from the waiting process and allocate them to the requesting process

This protocol is often applied to resources whose state can be easily saved and restored

ex:- cpu registers

not applicable for printers, tape drives

4. Circular Wait



Impose a total ordering of all resource types and to require that each process request resources in an increasing order of enumeration

ex:- P_1 is allocated R_3;
if P_1 requests for R_1 or R_2; such ~~we~~ requests will not be granted.

It is upto application developers to write programms that follow the ordering

Deadlock Avoidance

- Safe State -

Require additional information about how resources are to be requested.

The system might ~~m~~ need to know the sequence of requests and releases for each process.

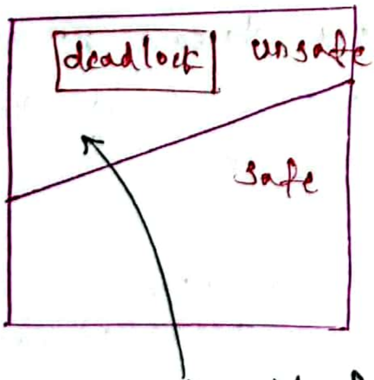
eg:- Process P will request first the ~~Q will~~ tape drive then the printer; both resources; before releasing Q will request first the printer then the tape drive.

- Safe Sate -

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

A system is in a safe state only if there exists a safe sequence.

$(P_1, P_2, \ldots P_n)$; P_i can be ~~satist~~ satisfied by the currently available resources + resources held by all P_i



possible for a deadlock

A safe state is not a dl state.
A deadlock state is an unsafe state.
Not all unsafe states are deadlocks

Consider a system with 12 magnetic tap drives and three processes; P_0, P_1 and P_2.

Safe sequence (P_1, P_0, P_2)

	Maximum Needs	Current Needs
P_0	10	5
P_1	4	2
P_2	9	2
		9

at time to free tape drives;
$12 - 9 = 3$

P_1

P_1 needs 2 more
free tape drives $= 3 - 2 = 1$

P_1 finishes execution and releases all 4 tape drives
free tap drives $= 1 + 4 = 5$

P_0

P_0 needs 5 more
free tape drives $= 5 - 5 = 0$

P_0 finishes and releases all ~~5~~ 10;
free tape drives $= 10$

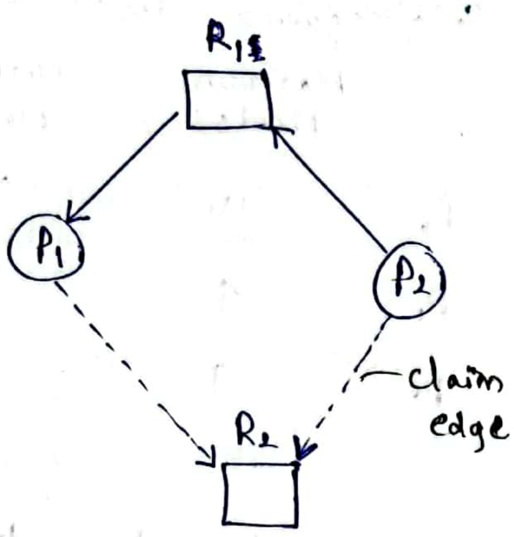
P_2

P_2 needs 7 more
free tap drives $= 10 - 7 = 3$

this sequence satisfies the safe state condition

→ Suppose P_0 requests one more tap drive.
the system is no longer in a safe state.

Resource allocation graph algorithm



A claim edge $P_i \rightarrow R_j$ indicates that P_i may request resource R_j, at some point in the future.
Similar to request edge in direction but a dashed line.

When P_i requests R_j, claim edge is converted to a request edge.

When R_j is realeased by P_i assignment edge $R_j \rightarrow P_i$ is reconverted to a claim edge $P_i \rightarrow R_j$

Which ever resources needed by a process that should appear as a claim edge before the execution begins

A request can be granted only if converting the request edge to an assignment edge does not result a formation of a cycle.

for safety chekking; a cycle-detection algorithm is used.
cycle occurs → Puts the system in an unsafe state.
no cycle → safe state.

Banker's Algorithm

Resources should not be allocated in a such way that the needs of the Processes can not be met / satisfied.

Data structures used to implement banker's algorithm

~~Available~~
n = no. of processes
m = no. of resource types.

Available

1-D array of size m
number of available resources of each type.
Available [j] = 3 ⇒ 3 instances of resource type R_j

Max

2-D array of size n×m
maximum demand of each process in a system
Max[i,j] = 3 ⇒ P_i may request max of 3 instances of R_j

Allocation

2-D array of size n×m
number of resources of each type currently allocated to each process.
Allocation [i,j] = 3 ⇒ P_i is allocated 3 instances of R_j

Need

2-D array of size n×m
remaining resource needs of each process
Need [i,j] = 3 ⇒ P_i needs 3 instances of R_j

N[i,j] = Max [i,j] − Allocation [i,j]

Allocation$_i$ → resources currently allocated to process P_i

Need$_i$ → resources that P_i may request

Safety Algorithm

1) Work → size m } vectors
finish → size n

initialize

work = available

Finish [i] = false ; for i = 0, 1, 2 ... n-1

2) Find an i such that both
a) Finish [i] = false
b) Need$_i$ <= Work

If no such i exists goto step (4)

3) work = work + Allocation [i]
Finish[i] = true
goto step (2)

4) If Finish [i] = true for all i then the system is in a safe state.

Resource - Request Algorithm

1) If Request$_i$ <= Need$_i$
goto step (2);
else ; raise an error

2) If Request$_i$ <= Available
goto step (3)
else; P_i must wait

3) Have the system pretend to have allocated the requested resources to P_i by modifying the state as follows.

Available = Available - Request$_i$
Allocation$_i$ = Allocation$_i$ + Request$_i$
Need$_i$ = Need$_i$ - Request$_i$

↓

now we have a safe - state ; again we have to check using the safety algo. whether the new state is safe or not.

If it is actually safe; then we can grant the request made by P_i
else; P_i has to wait

example - Safety algorithm.

Resource type A has 10 instances
B has 5 "
C has 7 "

Process	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
4 P_0	0	1	0	7	5	3	7	4	3	3	3	2
1 P_1	2	0	0	3	2	2	1	2	2			
5 P_2	3	0	2	9	0	2	6	0	0			
2 P_3	2	1	1	2	2	2	0	1	1			
3 P_4	0	0	2	4	3	3	4	3	1			
total	7	2	5									

Step 1 n = 3, m = 3

work = available

work = 3 3 2

Process = 0 1 2 3 4

Finish = f f f f f

Step 2 — P_0 — for i = 0

$need_0$ = 7 4 3

7, 4, 3 > 3, 3, 2

Finish [0] = false and $Need_0$ > ~~Work~~ Work

P_0 must wait

Step 2 — P_1 — for i = 1

$Need_i$ = 1 2 2

1, 2, 2 < 3, 3, 2

Finish [1] = false and $Need_i$ < Work

P_1 can be kept in a safe state 1

Step 3 — P_1

Work = work + Allocation

3, 3, 2 + 2, 0, 0

5 3 2

Process = 0 1 2 3 4

Finish = f t f f f

Step 2 P_2 for $i = 2$

$Need_2 = 6 \quad 0 \quad 0$

$6,0,0 > 5,3,2$

Finish [2] = false and $Need_2 >$ ~~allocation~~ work

P_2 must wait

Step 2 – P_3 – for $i = 3$

$Need_3 = 0 \quad 1 \quad 1$

$0,1,1 < 5,3,2$

Finish [3] = false and $Need_3 <$ Work

P_3 can be kept in a safe sequence (2)

Step 3 – P_3

Work = work + Allocation

$5,3,2 + 2,1,1$

$7 \quad 4 \quad 3$

Process =	0	1	2	3	4
	f	t	f	t	f

Step 2 – P_4 – for $i = 4$

$Need_4 = 4 \quad 3 \quad 1$

$4,3,1 < 7,4,3$

Finish [4] = false and $Need_4 <$ Work

P_4 can be kept in a safe state (3)

Step 3 – P_4

Work = work + allocation

$7,4,3 + 0,0,2$

$7 \quad 4 \quad 5$

Process =	0	1	2	3	4
Finish =	f	t	f	t	t

Step 2 — P_0 — for $i = 0$

$Need_0$ 7 4 3

$7,4,3 \le 7,4,5$

Finish [0] = false and $Need_0 \le Work$

P_0 can be kept in a safe state. 4

Step 3 — P_0

Work = Work + Allocation

$7,4,5 + 0,1,0$

7 5 5

Process =	0	1	2	3	4
finish =	t	t	f	t	t

Step 2 — P_2 — for $i = 2$

$Need_2$ = 6 0 0

$6,0,0 < 7,5,5$

Finish [0] = false and $Need_2 < Work$

P_2 can be kept in a safe sate 5

Step 3 — P_2

Work = Work + allocation

$7,5,5 + 3,0,2$

10 5 7

Process =	0	1	2	3	4
Finish =	t	t	t	t	t

* Safe Sequence → P_1, P_3, P_4, P_0, P_2

Example of Resource request algorithm

Suppose that P_1 requests 1 additional instance of resource type A and 2 instances of C

the request = $(1, 0, 2)$

Step 1

$1,0,2 <= 1,2,2$

$Request_1 <= Need_1$ ✓

↓ go to step 2

Step 2

$1,0,2 <= 3,3,2$

$Request_1 <= Available$ ✓

Step 3

$Available_1 = Available - Req_1$

$(2,3,0) \Leftarrow (3,3,2) - (1,0,2)$

$Allocation_1 = Allocation_1 + Req_1$

$(3,0,2) \Leftarrow (2,0,0)\ (1,0,2)$

$Need_1 = Need_1 - Request_1$

$(0,2,0) \Leftarrow (1,2,2) - (1,0,2)$

We pretend that the request is granted. Now we have to check whether this state is safe or not using the safety algorithm

Deadlock Detection

When each resource have only one single instance; we use the Wait-for Graph.

↳ Varient of the Resource Allocation Graph.

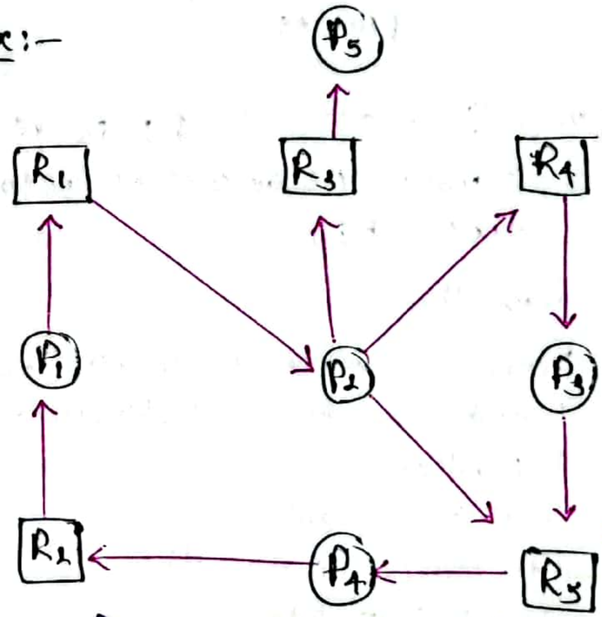
Remove the resource nodes and Collapsing the appropriate edges

In a Wait-For Graph;

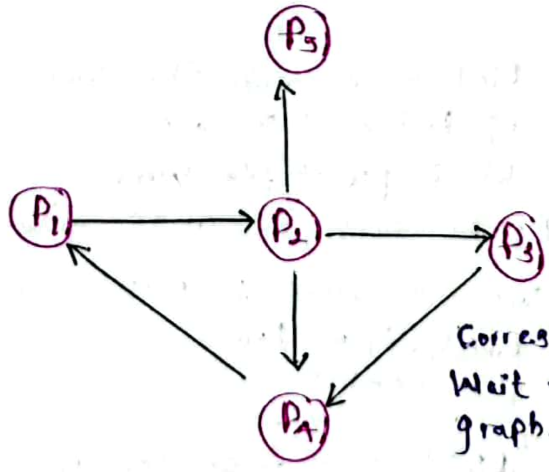
$P_i \rightarrow P_j$ means that P_i is waiting for P_j to release a resource needed by P_i.

Some times; $P_i \rightarrow R_k$ and $R_k \rightarrow P_j$

ex:-



Resource-Allocation-Graph



Corresponding Wait-For graph.

There is a deadlock since there exist a cycle in the system.

* If the resources have more than one instance we can not use the wait for graph.

Multiple instances of each resource type; this algorithm employs several time-varying data structures similar to those used in Banker's Algo.

Available – vector of length of m
# of available resources of each type.

Allocation – n x m matrix.
# of resources of each type currently allocated to each process.

Request – n x m matrix
Current request of each process.

Request [i][j] = k → P_i is requesting k more instances of resource type R_j

the algorithm

1) Work → vector of length m
Finish → vector of length n.
Initialize
Work = available
If allocation ≠ 0 then
Finish[i] = false
else
Finish[i] = true for i = 0, 1, 2, ..., n-1

2) Find an i such that both
a) Finish[i] = false
b) $Request_i$ <= Work
If no such i exists goto step 4

3) Work = Work + Allocation [i]
Finish [i] = true.
goto step 2

4) if Finish [i] = false for some i, 0 < i < n
then the system is in a deadlock state.; P_i is deadlocked.

Example

Processes P_0 ... P_4
Resource A – 7 instances
B – 2
C – 6

T = t_0

Process	Allocation			Request			Available		
	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			
	7	2	6						

Safe sequence – (P_0, P_2, P_3, P_1, P_4)

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

Suppose that now P_2 makes one additional request for an instance of type C.

Process	Allocation			Request			Available
P_2	3	0	3	0	0	1	

	Currently available		
	A	B	C
	0	0	1
P_0	0	1	0

$Request_2$ < available

Now the system is deadlocked..

Recovery from Deadlock

Process termination

When a deadlock detection algo. determines that a deadlock exists;

Possibility 01

Inform the operator and let the operator deal with the deadlock manually.

Possibility 02

Let the system recover from the deadlock automatically.

How to break from the deadlock,

option 01

Abort one or more processors to ~~ber~~ break the circular wait

option 02

Preempt some resources from one or more of the deadlocked processes.

eliminating a deadlock by aborting a process;

Abort all deadlock processes

will break the deadlock;

expense :- partial computations of the deadlocked processes must be discarded and probably will have to be recomputed later

Abort one process at a time until the deadlock cycle is eliminated.

overhead - after each process is aborted; a dl detection algorithm must be invoked.

Aborting a process maynot be easy.

ex:- a process was in the midst of updating a file, terminating it will leave that file in ~~an~~ an incorrect state

:- midst of printing data on a printer the system must reset the printer to a correct state before printing the next job.

If the partial termination method is used;

⇒ We must determine which deadlocked process/processes should be terminated

⇒ this determination is a policy decision, similar to cpu-scheduling.

We should abort the processes whose termination will ~~incur~~ incur the minimum cost

Factors that affect the choosing of processes to be aborted.

1) Priority of the process
2) How long the process has computed ~~and~~ and how much computations remaining before completion.
3) How many and what type of resources the process has used? (simple to preempt ?)
4) How many resources the process needs for completion
5) How many process will need to be terminated. (We should terminate minimum as possible.)

6) Whether the process is interactive or batch.

Resource Preemption

three issues need to be addressed.

1) Selecting a victim
2) Rollback
3) Starvation

Selecting a Victim

which resources and which processes are to be preempted?

Determine the order of preemption to minimize cost.

- number of res. a deadlocked process is holding.
- amount of time the process has consumed during its execution.

Rollback

what should be done with a process if we preempt its resources?

It ~~can~~ can not continue its normal execution as it is missing some or all of its needed resources.

We must ~~~~ roll back the process to some safe state and restart it ~~it~~ from that state.

Safe state? (more algorithms ?)

the simplest solution is

total rollback

Abort the process and then restart it.

Starvation

How can we guarantee that resources will not always be preempted from the same process?

If victim selection is based primarily on cost factors, it may happen that the same process is always picked as a victim. ~~~~ As a result, this process never completes its designated task.

A process should be picked as a victim only a (small) finite number of times.
(limit, count)

Solution

Include the number of rollbacks in the cost factor.

Question

	Allocation			Max		
	x	y	z	x	y	z
P_0	0	0	1	8	4	3
P_1	3	2	0	6	2	0
P_2	2	1	1	3	3	3

req 1 — P_0 requests 0 units of x; 0 units of y; 2 units of z

req 2 — P_1 requests 2 units x; 0 units of y; 0 units of z.

3 units of x; 2 units of y and 2 units of z still available.

	Allocation x y z	Max			Need			Available		
		x	y	z	x	y	z	x	y	z
P_0	0 0 1	8	4	3	8	4	2	3	2	2
P_1	3 2 0	6	2	0	3	0	0			
P_2	2 1 1	3	3	3	1	2	2			

req 1 ; $P_0 \longrightarrow (0, 0, 2)$; $Need_0 >= Request_0$; ✓

Assume that the req. was granted. $Req_0 <= Available$ ✓

	Allocation			Max			Need			Available		
	x	y	z	x	y	z	x	y	z	x	y	z
P_0	0	0	3	8	4	3	8	4	0	3	2	0
P_1	3	2	0	6	2	0	3	0	0			
P_2	2	1	1	3	3	3	1	2	2			

P_1 can be allowed to be executed.

after P_1 executes

Available		
x	y	z
6	4	0

with this available count neither P_0 nor P_2 can be executed.
Hence req 1 can not be permitted.

req 2 ; $P_1 \rightarrow (2, 0, 0)$

Request$_1$ <= Need$_1$ ✓

Request$_1$ <= Available ✓

Assume that the request was granted.

	Allocation X Y Z	Max X Y Z	Need X Y Z	Available X Y Z
P_0	0 0 1	8 4 3	8 4 2	1 2 2
P_1	5 2 0	6 2 0	1 0 0	
P_2	2 1 1	3 3 3	1 2 2	

P_1 or P_2 can be allowed to execute

If P_1 executes

Available		
X	Y	Z
6	4	2

with this available count P_2 can be executed

After P_2 executes

Available		
X	Y	Z
8	5	3

with this available count P_0 can be executed.

All processes executed; safe sequence (P_1, P_2, P_0)

Hence, req 2 can be permitted.