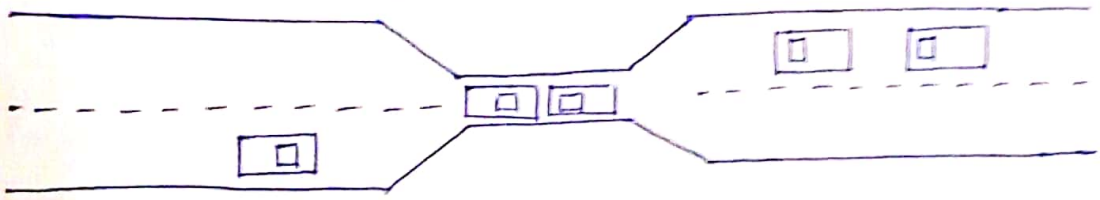
7. Deadlocks

The Deadlock Problem :

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
  * System has 2 disk drivers.
  * P_1 and P_2 each hold one disk drive and each needs another one.
- Example
  * semaphores A and B, initialized to 1 P_0 P_1.

wait(A); wait(B) wait(B); wait(A);

Bridge Crossing Example



- Traffic only in one direction
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.
- Note – Most OSes do not prevent or deal with deadlocks

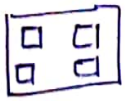
that is held by P_1, P_1 is waiting for a resource that is held by P_2, ... P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Resource - Allocation Graph :

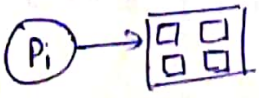
A set of vertices V and set of edges E

- V is partioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all processes in the system.
  - $R = \{R_1, R_2 \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_i \rightarrow R_j$
- assignment edge - directed edge $R_j \rightarrow P_i$
- Process

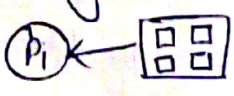
- Resource Type with 4 instances



- P_i request instance of R_j



- P_i is holding an instance of R_j



System Model :

- Resource type $R_1, R_2 \ldots, R_m$
  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilize a resource as follows :
  - request
  - use
  - release

Deadlock Characterization :

Deadlock can arise if four conditions hold simultaneously.

- Mutual Exclusion : only one process at a time can use a resource.
- Hold and wait : a process holding at least one resource is waiting to acquire additional resources held by other processes.
- No preemption : a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- Circular wait : there exists a set $\{P_0, P_1 \ldots P_n\}$ of waiting processes such that P_0 is waiting for a resource

Basic Facts :

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks :

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention :

Restrain the ways request can be made

- Mutual Exclusion – not required for sharable resources; must hold for non-sharable resources.
- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources.

- Require process to request and be allocated all its resources before it begins execution or allow process to request resources only when the process has none.
- Low resource utilization, starvation possible.

- No Preemption :
  - If a process that is holding some resources requests another resource that can not be immediately allocated to it, then all resources currently being held are released.
  - Preempted resources are added to the lists of resources for which the process is waiting.
  - Process will be restarted only when it can regain its old resources, as well as the new ones that is requesting

- Circular wait : impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance :

Requires that the system has some additional a priori information available.

- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State :

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a sequence $< P_1, P_2 \ldots, P_n >$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j;

with $j < i$.

- That is :
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

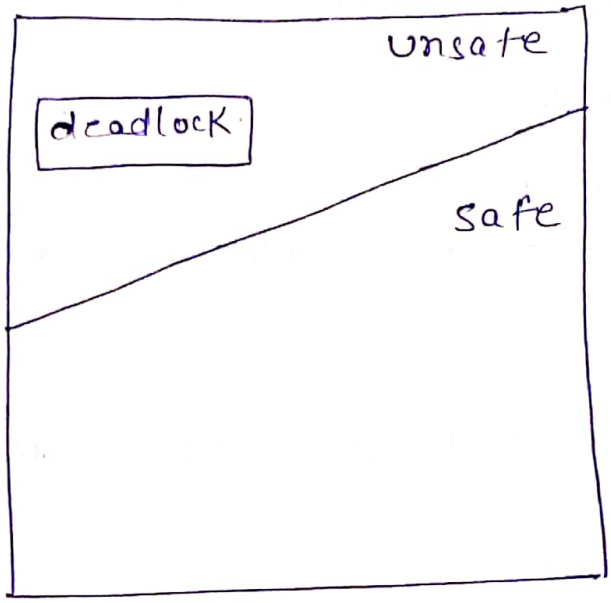
Safe State :

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a sequence $< P_1, P_2, \ldots, P_n >$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j <$

Basic Facts :

- If a system is in safe state ⇒ no deadlocks
- If a system in unsafe state ⇒ possibility of deadlock.

- Avoidance ⇒ ensure that a system will never enter an unsafe state.



Safe, Unsafe, Deadlock State

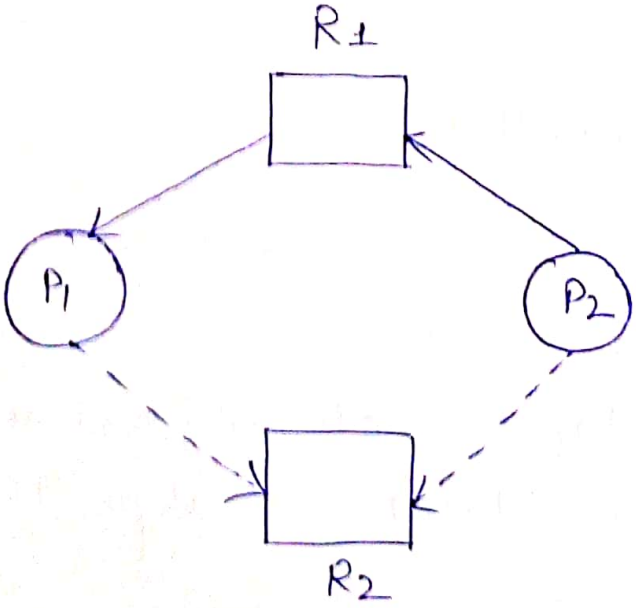
Avoidance Algorithm :

- Single instance of a resource type.
  - Use a resource-allocation graph.
- Multiple instances of a resource type
  - Use the banker's algorithm.

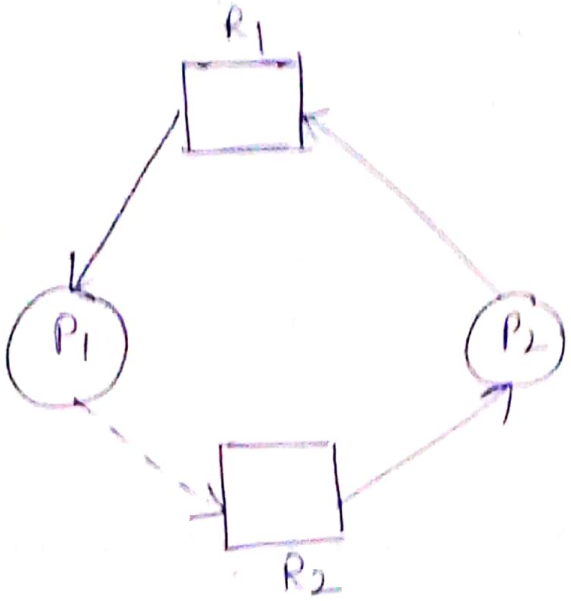
- Resource - Allocation Graph Scheme :

- Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j, represented by a dashed Line.
- Claim edge converts to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed a priori in the system.

Resource - Allocation Graph



Unsafe State in Resource-Allocation Graph



Resource-Allocation Graph Algorithm

- Suppose that process Pi requests a resource Rj.
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.

Banker's Algorithm:

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Bankers Algorithm. (Deadlock avoidance)

Total A = 10, B = 5, C = 7

Process	Allocation A	B	C	Max Need A	B	C	Current Available A	B	C	Remaining. (Max. Need − Allocation) A	B	C
							3	3	2			
P_1	0	1	0	7	5	3	+2	0	0	7	4	3
							5	3	2			
P_2	2	0	0	3	2	2	+2	1	1	1	2	2
							7	4	3			
P_3	3	0	2	9	0	2	+0	0	2	6	0	0
							7	4	5			
P_4	2	1	1	4	2	2	+0	1	0	2	1	1
							7	5	5			
P_5	0	0	2	5	3	3	+3	0	2	5	3	1
	7	2	5				10	5	7			

Safe Sequence :

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$.

Data Structure for the Banker's Algorith

2)

Let n = number of processes, and m = no. o resources types.

- Available : Vector of length m. If available [j] = K, there are K instances of resource type R_j available.
- Max : $n \times m$ matrix. If Max[i, j] = K, then process P_i may request at most K instances of resource type R_j.
- Allocation : $n \times m$ matrix. If Allocation [i, j] = K then P_i is currently allocated K instances of R_j.
- Need : $n \times m$ matrix. If Need [i, j] = K, then P_i may need K more instances of R_j to complete its task.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$$

1. Safety Algorithm :

(1) Let Work and finish be vectors of length m and n, respectively.

Initialize :

$$\text{Work} = \text{Available}$$

$$\text{Finish}[i] = \text{false for } i = 0, 1, \ldots, n-1$$

(2) Find an i such that both :

(a) Finish [i] = false

(b) Needi ≤ Work

If no such i exists, goto step 4.

(3) Work = Work + Allocationj

Finish [i] = true

goto step 2

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

2. Resource-Request Algorithm for Process Pi.

Request = request vector for process Pi. If Requesti[j] = k then process Pi wants k instances of resource type Rj.

(1) If Requesti ≤ Needj goto step 2. Otherwise raise error condition, since process has exceeded its maximum claim.

(2) If Requesti ≤ Available, go to step 3. Otherwise Pi must wait, since resources are not available.

(3) Pretend to allocate requested resources to Pi by modifying the state as follows :

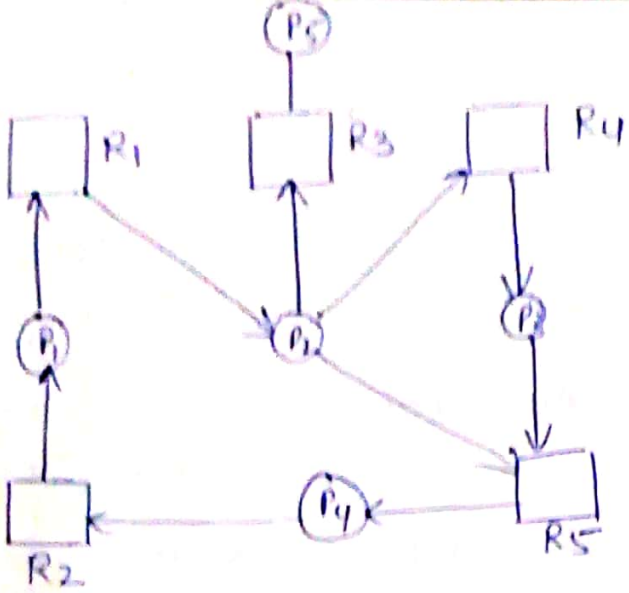
Available = Available − Request;
Allocationi = Allocationi + Requesti;
Needi = Needi − Requesti; (7.12)

- If safe ⇒ the resources are allocated to Pi
- If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

Deadlock Detection

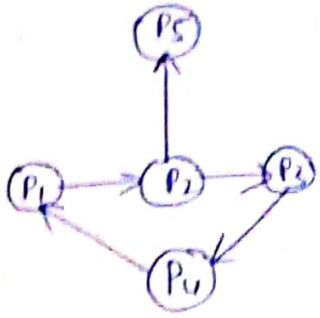
- Allow system to enter deadlock state.
- Detection algorithm.
- Recovery scheme.

1. Single Instance of Each Resource Type :
   - Maintain wait for graph
     - Nodes are processes
     - $P_i \rightarrow P_j$ if P_i is waiting for P_j.
   - Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.
   - An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.



(a)

Resource - Allocation Graph



(b)

corresponding wait for graph.

2. Several Instances of a Resource Type

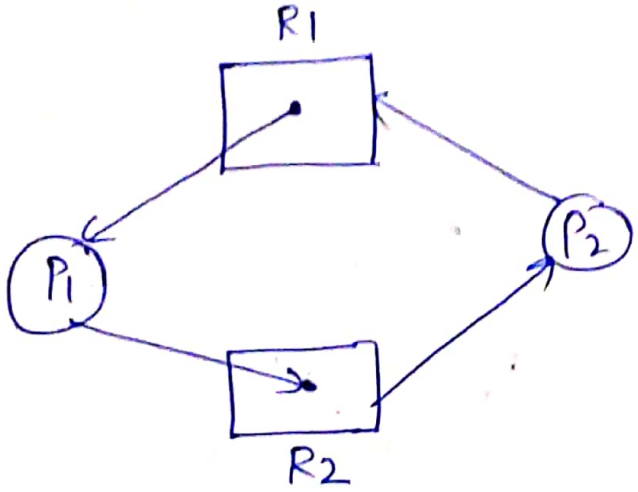
- Available : A vector of length m indicates the number of available resources of each type.
- Allocation : An n x m matrix defines the number of resources of each type currently allocated to each process.
- Request : An n x m matrix indicates the current request of each process. If Request [i][j] = K, then process Pi is requesting K more instances of resource type Rj

Resource Allocation Graph (RAG)

Single Instance

(i)

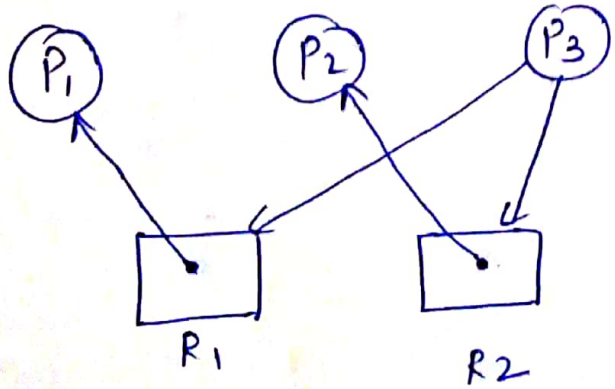
	Allocate		Request		Availability	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	1	0	0	1	0	0
P_2	0	1	1	0		



(Deadlock)

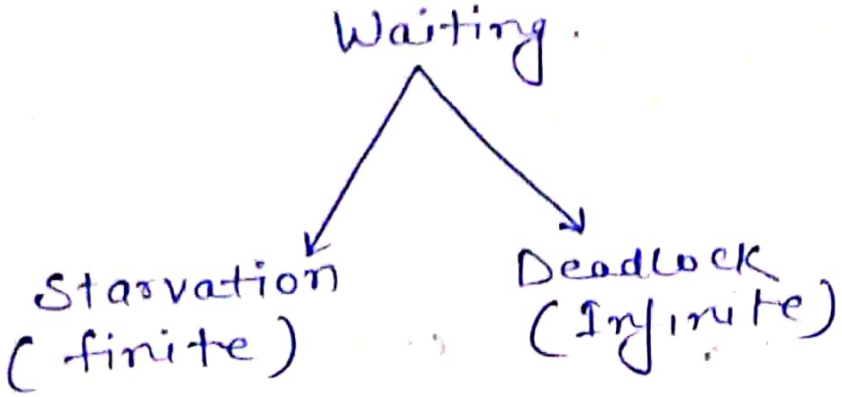
(ii)

	Allocate		Request		Availability	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	1	0	0	0	0	0
P_2	0	1	0	0	1	0
P_3	0	0	1	1	1	1



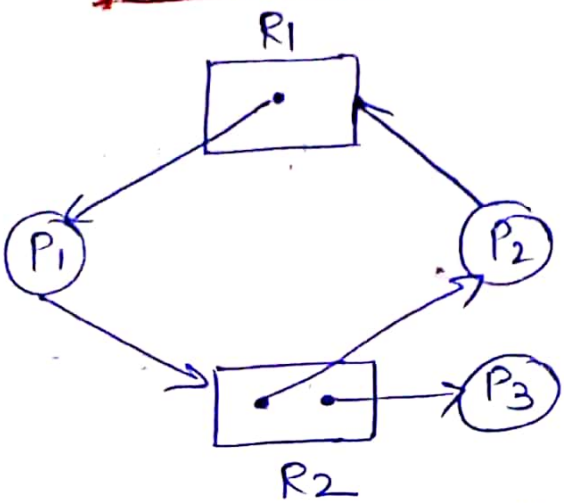
(No Deadlock)

$P_1 \rightarrow P_2 \rightarrow P_3$ (All executed)



Multi Instance.

(i)



NO Deadlock

	Allocate R1	Allocate R2	Request R1	Request R2	Availability R1	Availability R2	
P_1	1	0	0	1	0	0	
P_2	0	1	1	0	0, 1		(P_3 executed) P_1 can execute.
P_3	0	1	0	0	1	1	(P_1 executed) P_2 can execute.

$P_3 \rightarrow P_1 \rightarrow P_2$ (All executed)

Detection Algorithm

1. Let Work and Finish be vectors of length m and n, respectively.

   Initialize :

   (a) Work = Available

   (b) For i = 1, 2, ... n if Allocation ≠ 0 then Finish[i] = false; otherwise, Finish[i] = true

2. Find an index i such that both :

   (a) Finish[i] == false

   (b) Requesti ≤ Work.

   If no such i exists, goto step 4.

3. Work = Work + Allocationi

   Finish[i] = true

   goto step 2

4. If Finish[i] == false, for some i, $1 \le i \le n$, then the system is in deadlock state. Moreover, if Finish[i] == false, then Pi is deadlocked.

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Detection Algorithm Usage :

When, and how often, to invoke depends on:

- How often a deadlock is likely to occur ?
- How many processes will need to be rolled back ?
  > one for each disjoint cycle.

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Recovery from Deadlock :

1. Process Termination

- Abort all deadlocked process.
- Abort one process at a time untill the deadlock cycle is eliminated.
- In which order should we choose to abort
  - Priority of the process
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.

- Resources process needs to complete.
- How many processes will need to be terminated.
- Is process interactive or batch?

Recovery from Deadlock:

Resource Preemption

- Selecting a victim - minimize cost.
- Rollback - return to some safe state, restart process for that state.
- Starvation - some process may always be picked as victim, include number of rollback in cost factor.