Chapter - 5

Deadlocks

Deadlock

- Deadlock can be defined as the permanent blocking of a set of processes that either compete for system resources or communicate with each other.

Deadlocks

- A process request for some resources. If the resources are not available at that time , the process enters a waiting state . The resources was held by other processes.The waiting process may never able to get the resource.

  This situation is called deadlock.

Deadlock state

- A set of processes are said to be in deadlock state when every process in the set is waiting for an event that can be caused by some another process in the set.

System Model

- A system consists of a finite number of resources which must be distributed among several processes.
- **Resources**

Resources are of two types

- **pre-emptable resource**

It is the one that can be taken away from its current owner. (e-x) memory

- **Non pre-emptable resource**

It is the one that cannot be taken away from its current owner(e-x) printer

The utilization of the resource by a process must be done in the following sequence.

Request-→ use----→ Release

- Deadlock Characterization
- Deadlock prevention
- Deadlock avoidance
- Deadlock detection
- Deadlock Recovery

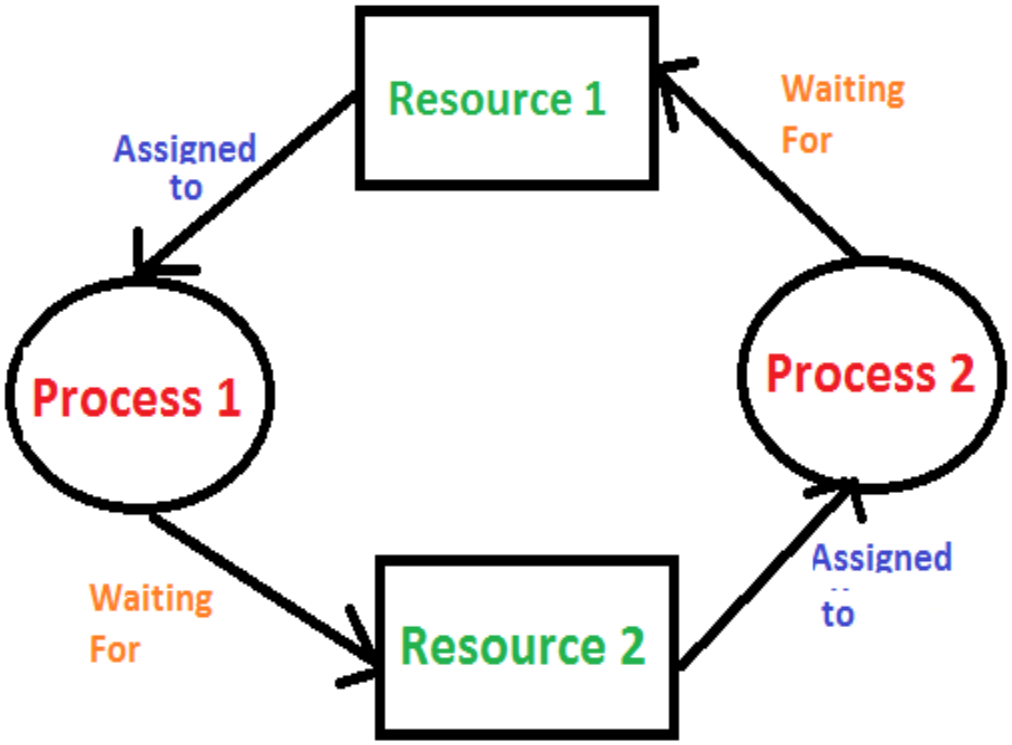
Deadlock characterization

- Necessary conditions
- Resource allocation graph

- Deadlock can arise if four conditions hold simultaneously
- **Mutual exclusion:** only one process at a time can use a resource. If another process requests the same resource, the requesting process must wait until the resource is released.
- **Hold and wait:** Processes currently holding resources granted earlier , can request for new resources , that are currently held by other.

- **No preemption:** a resource can be released by the process holding it only after that process has competed its task.
- **Circular wait:** The circular chain of two or more processes must exist such that each of them is waiting for a resource held by next member.

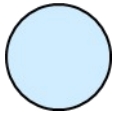
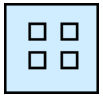
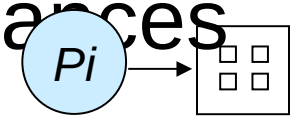
Circular wait



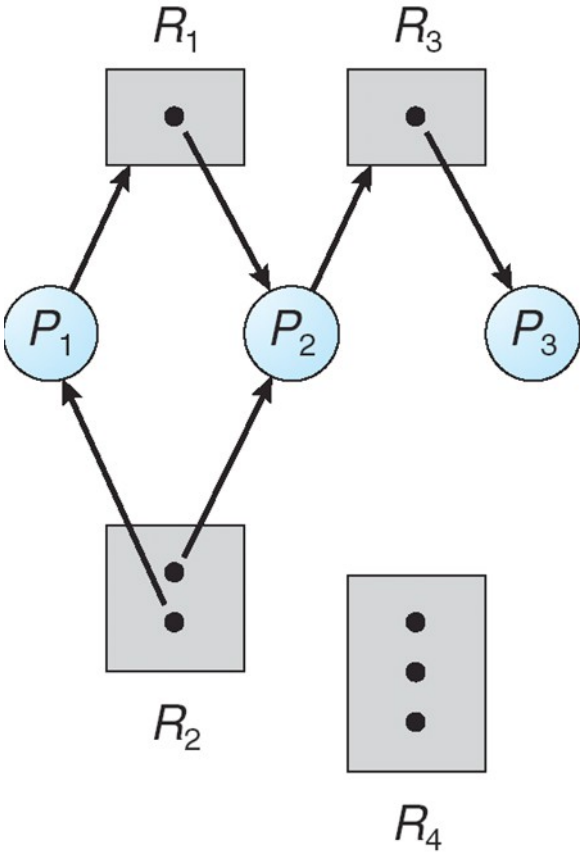
Resource allocation graph

- A set of vertices *V* and a set of edges *E*.
- V is partitioned into two types:
  - *P* = {*P*1, *P*2, …, *Pn*}, the set consisting of all the processes in the system
  - *R* = {*R*1, *R*2, …, *Rm*}, the set consisting of all resource types in the system
- **request edge** – directed edge *Pi* → *Rj*

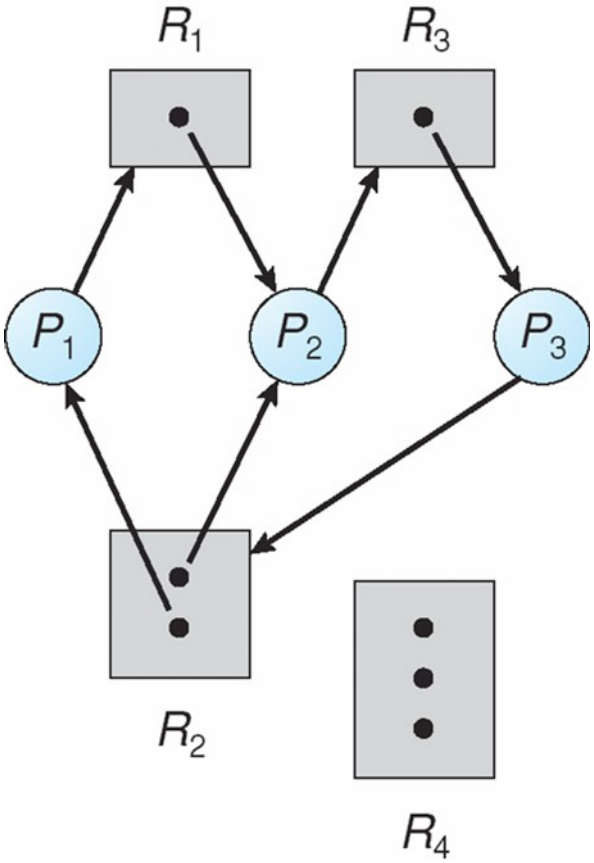
Resource allocation graph

- Process 
- Resource Type with 4 instances 
- *Pi* requests instance of *Rj* 

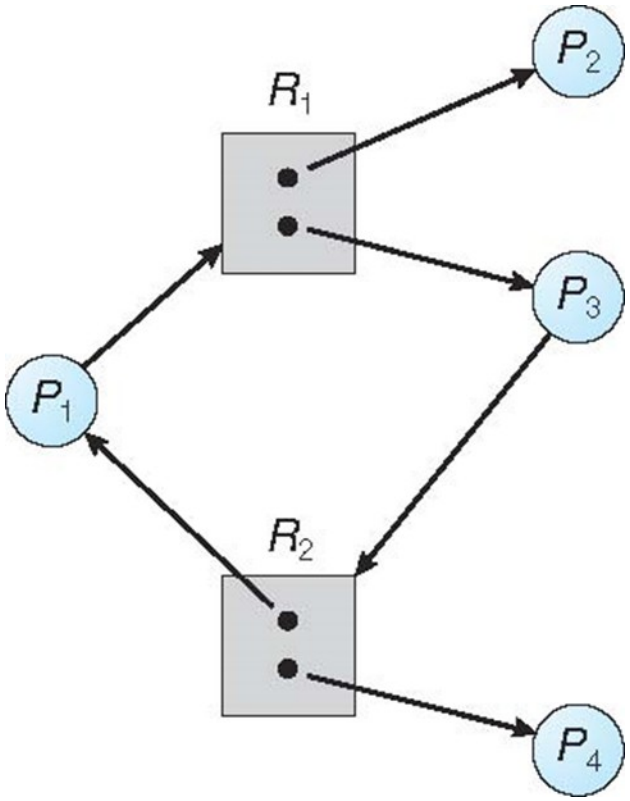
Example for Resource allocation graph



Resource allocation graph with deadlock



Graph With A Cycle



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:
  - Deadlock prevention
  - Deadlock avoidance
  - Deadlock detection
  - Deadlock Recovery

Deadlock prevention

- Mutual exclusion
- Hold and wait
- No pre-emption
- Circular wait

Mutual Exclusion

Not always possible to prevent deadlock by preventing mutual exclusion (making all resources shareable) as certain resources are cannot be shared safely.

Hold and Wait

A process can get all required

- **No preemption**

We will see two approaches here. If a process request for a resource which is held by another process, then the resource may be preempted from the other process. In the second approach, if a process request for a resource which are not readily available, all other resources that it holds are preempted.

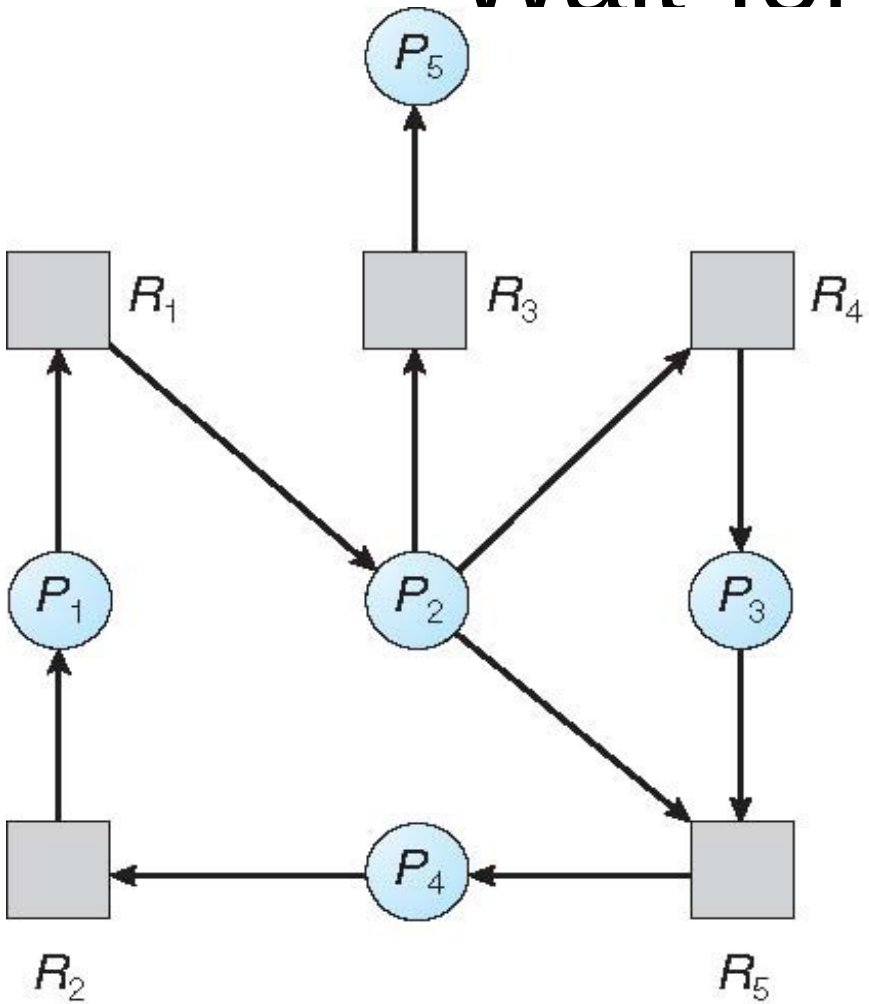
Circular wait

Deadlock detection Single Instance of Each Resource Type

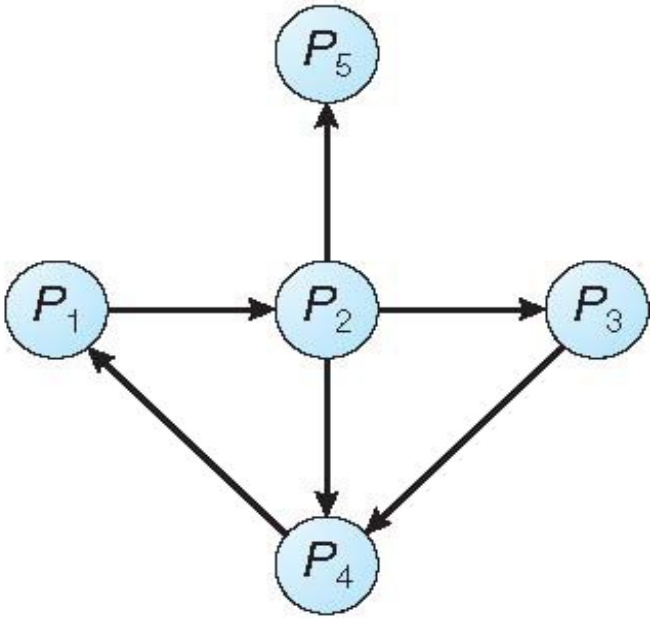
- Maintain **wait-for** graph
  - Nodes are processes
  - ***Pi*** → ***Pj*** if ***Pi*** is waiting for ***Pj***

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

Resource-Allocation Graph and Wait-for Graph



(a)



(b)

Several Instances of a Resource Type

The algorithm uses the following data structures

- **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

Deadlock detection algorithm

1.Let n = no of process and m= no of resources

- Allocation[n][m] and request[n][m] are matrices of nxm.
- Available[m],temp[m],done[n] are the vectors of length m,m and n respectively.

2.Initialize temp[j]=available[j] for all j. if allocation = 0 then done[i]=true otherwise done[i]=false

3) Find an i such that both

a) done[i] = false

b) request[i][j] < = temp[j]

if no such i exists , go to step 5.

4) temp[i] = temp[i] + allocation[i][j]for all j

done[i] = true

go to step 3

5) If done[i] = false, for some i,the system is deadlocked state .

DEADLOCK AVOIDANCE

- **SAFE STATE**

A state is safe if the system can allocate resources to each process and avoid a deadlock.

- **UNSAFE STATE**

A system is in unsafe state, it may lead to deadlock.

BANKER’S ALGORITHM

- It is a resource allocation and deadlock avoidance algorithm developed by Dijkstra that is applicable to resource allocation system with multiple instances of each resource type.

The algorithm uses the following data structures

- **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
- **Need***:* An ***n*** **x** ***m*** matrix indicates the remaining resources required by each process.
- **Max** : It is an ***n*** **x** ***m*** matrix defines the

1.Let n = no of process and m= no of resources

- Allocation[n][m] ,need[n][m] and max[n][m] are matrices of nxm.
- Available[m],temp[m],done[n] are the vectors of length m,m and n respectively.

2.Initialize temp[j]=available[j] for all j. Done[i]=false for all i.

3) Find an i such that both

a) done[i] = false

b) need[i][j] < = temp[j]

if no such i exists , go to step 5.

4) temp[i] = temp[i] + allocation[i][j]for all j

done[i] = true

go to step 3

5) If done[i] = false, for some i,the system is deadlocked state .

Deadlock recovery

- Process termination
- Resource pre-emption
- Check point / roll back mechanism

Process termination

- Abort all deadlocked process
- Successively abort each deadlocked process until the deadlock no longer exists.

Resource pre-emption

Roll back

A process that has a resource pre-empted from it must be roll back to the point to its acquiring of that resource.

Total roll back – Abort the process and restart it.

Check point

- Keep checkpointing periodically
- When a deadlock is detected , see which resource is needed
- Take away the resource from the process currently having it
- Restart the process from the checkpointed state.