DEADLOCK

Definition: A process requests resources. If the resources are not available at that time ,the process enters a wait state. Waiting processes may never change state again because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

A process must request a resource before using it, and must release resource after using it.

1. **Request:** If the request cannot be granted immediately then the requesting process must wait until it can acquire the resource.

2. **Use:** The process can operate on the resource

3. **Release:** The process releases the resource.

1. Deadlock Characterization

Four Necessary conditions for a deadlock

1. Mutual exclusion: At least one resource must be held 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.

4. Circular wait: P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2...Pn-1.

2. Resource-Allocation Graph

It is a Directed Graph with a set of vertices V and set of edges E.

V is partitioned into two types:

- nodes P = {p1, p2,..pn}
- Resource type R ={R1,R2,...Rm}

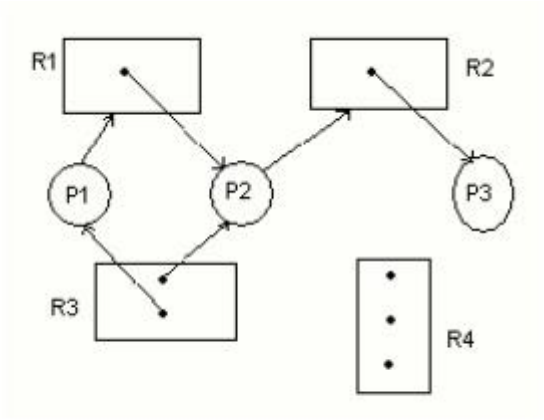
Pi -->Rj - request => request edge

Rj-->Pi - allocated => assignment edge. Pi is denoted as a circle and Rj as a square.

Rj may have more than one instance represented as a dot with in the square.

Sets P,R and E. P = {P1,P2,P3} R = {R1,R2,R3,R4}

E= {P1->R1, P2->R3, R1->P2, R2->P1, R3->P3 }



Resource instances

One instance of resource type R1,Two instance of resource type R2,One instance of resource type R3,Three instances of resource type R4.

Process states

Process P1 is holding an instance of resource type R2, and is waiting for an instance of resource type R1.Resource Allocation Graph with a deadlock

Process P2 is holding an instance of R1 and R2 and is waiting for an instance of resource type R3.Process P3 is holding an instance of R3.

P1->R1->P2->R3->P3->R2->P1

P2->R3->P3->R2->P2

Methods for handling Deadlocks

- ü Deadlock Prevention
- ü Deadlock Avoidance
- ü Deadlock Detection and Recovery

3. Deadlock Prevention:

v This ensures that the system never enters the deadlock state.

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

v By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

Denying Mutual exclusion

§ Mutual exclusion condition must hold for non-sharable resources.

§ Printer cannot be shared simultaneously shared by prevent processes.

§ sharable resource - example Read-only files.

If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file.

A process never needs to wait for a sharable resource.

Denying Hold and wait

o Whenever a process requests a resource, it does not hold any other resource.

o One technique that can be used requires each process to request and be allocated

o all its resources before it begins execution.

o Another technique is before it can request any additional resources, it must release all the resources that it is currently allocated.

Ø These techniques have two main disadvantages :

§ First, resource utilization may be low, since many of the resources may be allocated but unused for a long time.

§ We must request all resources at the beginning for both protocols.

§ starvation is possible.

Denying No preemption

¬ If a Process is holding some resources and requests another resource that cannot be immediately allocated to it. (that is the process must wait), then all resources currently being held are preempted. (ALLOW PREEMPTION)

¬ These resources are implicitly released.

¬ The process will be restarted only when it can regain its old resources.

Denying Circular wait

Ø Impose a total ordering of all resource types and allow each process to request for resources in an increasing order of enumeration.

o Let R = {R1,R2,...Rm} be the set of resource types.

Ø Assign to each resource type a unique integer number.

Ø If the set of resource types R includes tapedrives, disk drives and printers.

F(tapedrive)=1,

F(diskdrive)=5,
F(Printer)=12.

Ø Each process can request resources only in an increasing order of renumeration.

4. Deadlock Avoidance:

ü Deadlock avoidance request that the OS be given in advance additional information concerning which resources a process will request and useduring its life time.

With this information it can be decided for each request whether or not the process should wait.

ü To decide whether the current request can be satisfied or must be delayed, a system must consider the resources currently available, the resources currently allocated to each process and future requests and releases of each process.

Safe State

v A state is safe if the system can allocate resources to each process in some order and still avoid a dead lock.

v A deadlock is an unsafe state.

v Not all unsafe states are dead locks

v An unsafe state may lead to a dead lock

Two algorithms are used for deadlock avoidance namely;

1. **Resource Allocation Graph Algorithm -** single instance of a resource type.

2. **Banker's Algorithm** – several instances of a resource type.

Resource allocation graph algorithm

Claim edge - Claim edge Pi---> Rj indicates that process Pi may request resource Rj at some time, represented by a dashed directed edge.

- When process Pi request resource Rj, the claim edge Pi -> Rj is converted to a request edge.

- Similarly, when a resource Rj is released by Pi the assignment edge Rj -> Pi is reconverted to a claim edge Pi -> Rj

Banker's algorithm

Available: indicates the number of available resources of each type.

Max: Max[i, j]=k then process Pi may request at most k instances of resource type Rj

Allocation : Allocation[i. j]=k, then process Pi is currently allocated K instances of resource type Rj

Need : if Need[i, j]=k then process Pi may need K more instances of resource type Rj Need [i, j]=Max[i, j]-Allocation[i, j]

Safety algorithm

1 Initialize work := available and Finish [i]:=false for i=1,2,3 .. n

2 Find an i such that both

1 Finish[i]=false b. Needi<= 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

Let Requesti be the request from process Pi for resources.

1. If Requesti<= Needi goto step2, otherwise raise an error condition, since the process has exceeded its maximum claim.

2. If Requesti <= Available, goto step3, otherwise Pi must wait, since the resources are not available.

3. Available := Availabe-Requesti;

Allocationi := Allocationi + Requesti

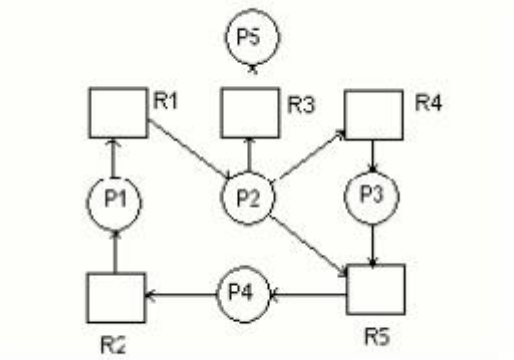
Needi := Needi - Requesti;

Now apply the safety algorithm to check whether this new state is safe or not.
If it is safe then the request from process Pi can be granted.

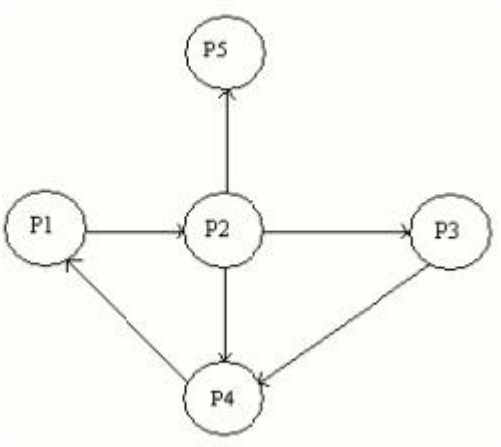
5. Deadlock Detection

(i) Single instance of each resource type

ResourceAllocation Graph



Wait for S



ii) Several Instance of a resource type
Available : Number of available resources of each type
Allocation : number of resources of each type currently allocated toeach process
Request : Current request of each process
If Request [i,j]=k, then process Pi is requesting K more instances of resource type Rj.

1. Initialize work := available

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 go to step4.

3. Work:=work+allocationi

Finish[i]:=true goto step2

4. If finish[i]=false

then process Pi is deadlocked

6. Deadlock Recovery

1. Process Termination

1. Abort all deadlocked processes.
2. Abort one deadlocked process at a time until the deadlock cycle is eliminated.

After each process is aborted , a deadlock detection algorithm must be invoked to determine where any process is still dead locked.

2. Resource Preemption

Preemptive some resources from process and give these resources to other processes until the deadlock cycle is broken.

i. **Selecting a victim:** which resources and which process are to be preempted.

ii. **Rollback:** if we preempt a resource from a process it cannot continue with its normal execution. It is missing some needed resource. we must rollback the process to some safe state, and restart it from that state.

iii. **Starvation :** How can we guarantee that resources will not always be preempted from the same process