DeadLocks ch6

A Computer System consist of a finite number of resources to be distributed among a number of computing processes. The resources are partitioned into several types, each of which consists of some number of identical instances memory, CPU, cycles, files, and I/O devices are examples of resource types.

Under the normal of operation, a process may utilize a resource in only the following sequence:

a- **Request:** If the request cannot be granted immediately then the requesting process must wait until it can acquire the resource.
b- **Use:** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
c- **Release:** The process releases the resource.

Deadlock definition

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set .

Example

Consider a C/S with two tape drives, suppose that there or two processes each holding one of these tape drives. If each process now requests another tape drive, the two processes will be in a deadlock state, see the fig bellow.

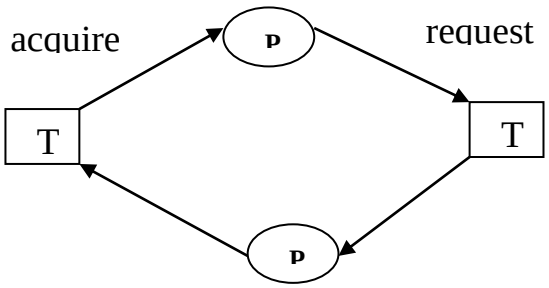


Fig1 : Deadlock

Deadlock necessary conditions

In a deadlock , processes never finish executing and system resources are tied up , preventing other processes from ever starting.
A deadlock situation can arise if and only if the following four conditions hold simultaneously in a system . these condition are :

a- Mutual Exclusion

At least one resource is 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

b- Hold and Wait

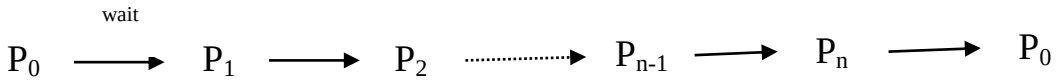
There must exist a process that is holding at least one resource and is waiting to acquire additional resources are currently being held by other processes.

c- No Preemption

Resources can not be preempted , that is a resource can be released only voluntarily by the process holding after that process has completed its task.

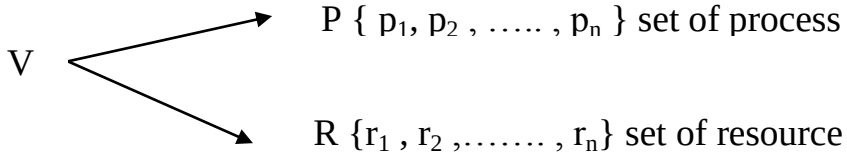
d- Circular Wait

There must exist a set $\{p_0,p_1,\ldots,p_n\}$of waiting processes such that p_0 is waiting for resource 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 .

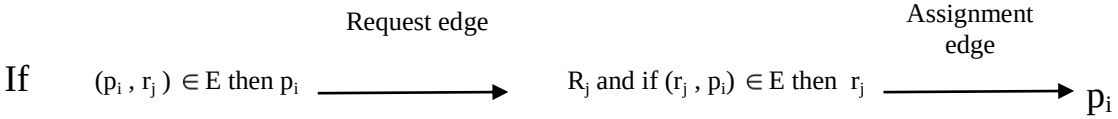


Resource – Allocation Graph(RAG)

Deadlocks can be described more precisely in terms of a directed graph called (RAG).
RAG = (V,E) where V is a set of vertices and E is a set of edges.



Each element in the set E of edges is an ordered pair (p_i,r_j)or (r_j, p_i) where pi is an a process ($p_i \in P$) and r_j is a resource type ($r_j \in R$)



Graphically we represent each process p_i as a circle and each resource type R_j as a square . Since resource type R_j may have more than one instance we represent each such instance as a dot within a square , see fig bellow .

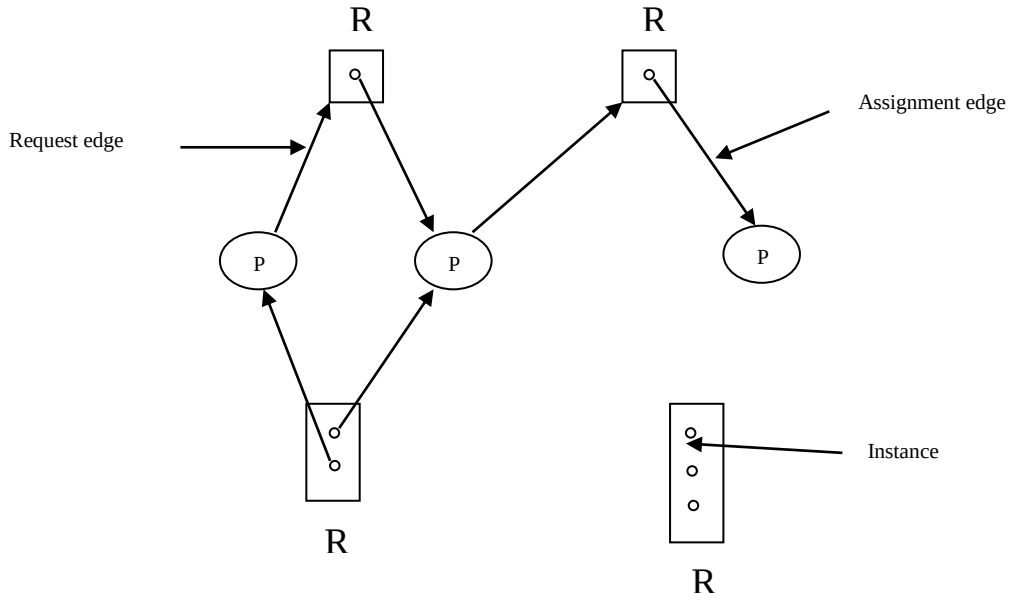


Fig2: Resource – allocation graph

The RAG in this fig depicts the following situation
The sets P, R and E :

- P = {p1 , p2 p3}
- R = {r1, r2 , r3 , r4 }
- E= { (p1,r1) , (p2,r3) , (r1,p2) , (r2, p2) ,(r2, p1) , (r3,p3)}

Resource states:
R1= 1, R2= 2, R3= 3 , and R4 = 4 instances.

Process states:
P1 →r1, P2→r3 , r1→P2 , r2→P1 , r2→P2 , r3→p3

By using a RAG it can be easily shown that if the graph contain no cycles, than no process in the system is deadlocked. If on the other hand the graph contains a cycle than a deadlock may exist .

If each resource type has exactly one instance then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types each of which has only a single instance then a deadlock has occurred.
Each process involved in the cycle is deadlock.
In this case a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock .

If each resource type has several instances then a cycle does not necessarily imply that a deadlock occurred . In this case a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.
To illustrate this concept let us return to fig bellow suppose that p3 requests r2 . Since no resource instance is currently available a request edge p3→r2 is added to the graph fig. At this point two minimal cycles exist in the system :
P2 →r1→ p2→ r3→ p3 →r2→ p1
P2 →r3→ p3→ r2→ p2
Process p1,p2 and p3 are deadlocked.

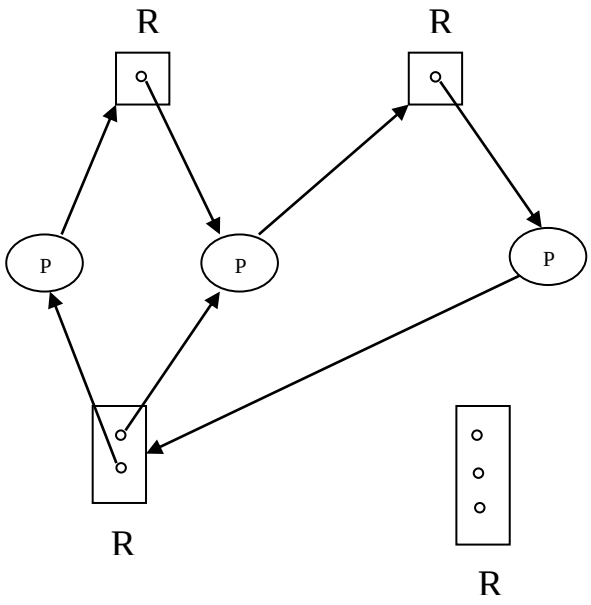


Fig3 : RAG with deadlock

In the fig bellow we have a cycle but there is no deadlock .
P1⟶r1⟶p3⟶r2⟶p1
Where p4 way release its instance of R2 that source can then be allocated to p3 breaking the cycle .

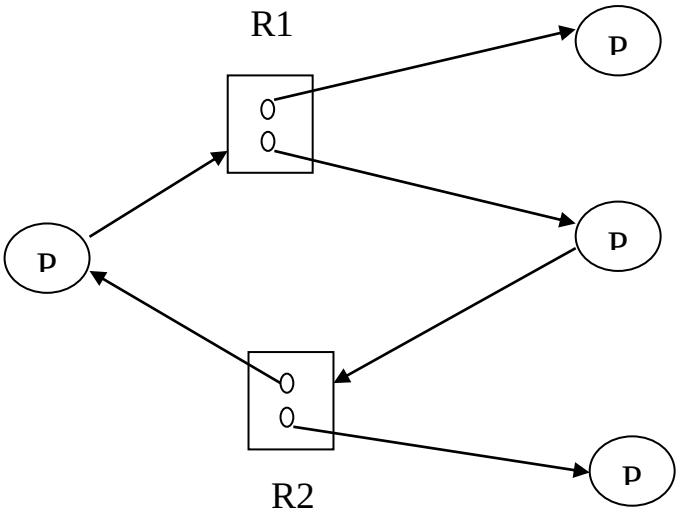


Fig4 : RAG with a cycle but no deadlock

Methods for Handling Deadlock

There are three methods for dealing with the deadlock problem:

a- We can use a protocol to ensure the system will never enter a deadlock state.

b- Allow the system to enter a deadlock state and then recover.

c- We can ignore the problem all together and pretend that deadlocks never occur in the system.

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