***Resources:***

Resources are the passive entities needed by processes to do their work. A resource can be a hardware device (eg. a disk space) or a piece of information ( a locked record in the database). Example of resources include CPU time, disk space, memory etc.

There are two types of resources:

**Preemptable** – A Preemptable resources is one that can be taken away from the process owing it with no ill effect. Memory is an example of preemptable resources.

**Non-preemptable** – A non-preemptable resources in contrast is one that cannot be taken away from its current owner without causing the computation to fail. Examples are CD-recorder and Printers. If a process has begun to burn a CD-ROM, suddenly taking the CD recorder away from it and giving it to another process will result in a garbled CD. CD recorders are non preemptable at any arbitrary moment.

Read-only files are typically sharable. Printers are not sharable during time of printing .One of the major tasks of an operating system is to manage resources.

Deadlocks may occur when processes have been granted exclusive access to Resources. A resource may be a hardware device (eg. A tape drive) file or a piece of information ( a locked record in a database). In general Deadlocks involves non preemptable resources. The sequence of events required to use a resource is:

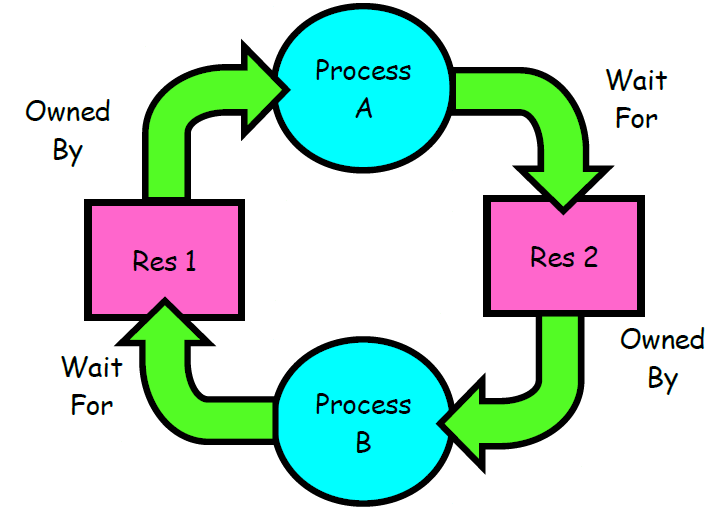
1. Request the resource

2. Use the resource

3. Release the resource

***What is Deadlock?***

In Computer Science a set of process is said to be in deadlock if each process in the set is waiting for an event that only another process in the set can cause. Since all the processes are waiting, none of them will ever cause any of the events that would wake up any of the other members of the set & all the processes continue to wait forever.



**Example 1:**

● Two process A and B each want to record a scanned document on a CD.

● A requests permission to use Scanner and is granted.

● B is programmed differently and requests the CD recorder first and is also granted.

● Now, A ask for the CD recorder, but the request is denied until B releases it. Unfortunately,

instead of releasing the CD recorder B asks for Scanner. At this point both processes are

blocked and will remain so forever. This situation is called Deadlock.

***Conditions for Deadlock:***

Four conditions must hold for there to be a deadlock:

***1. Mutual exclusion:*** Only one process at a time can use a resource.

***2. Hold and wait:*** Process holding at least one resource is waiting to acquire additional resources

held by other processes.

***3. No preemption:*** Resources are released only voluntarily by the process holding the resource,

after the process is finished with it

***4. Circular wait:*** There exists a set {P1 , …, Pn } of waiting processes.

P1 is waiting for a resource that is held by P2

P2 is waiting for a resource that is held by P3

…Pn is waiting for a resource that is held by P1

All of these four conditions must be present for a deadlock to occur.

***Deadlock Modeling:***

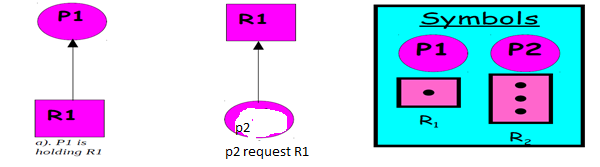
Deadlocks can be described more precisely in terms of Resource allocation graph. Its a set of vertices V and a set of edges E.

V is partitioned into two types:

P = {P1, P 2, ..., P n}, the set consisting of all the processes in the system.

R = {R1, R2 , ..., Rm}, the set consisting of all resource types in the system.

request edge – directed edge Pi🡪Rj

Assignment edge – directed edge Rj🡪 Pi

***Deadlock Prevention:***

To prevent the system from deadlocks, one of the four discussed conditions that may create a deadlock should be discarded. The methods for those conditions are as follows:

**1. Mutual Exclusion:** In general, we do not have systems with all resources being sharable. Some resources like printers, processing units are non-sharable. So it is not possible to prevent deadlocks by denying mutual exclusion.

**2. Hold and Wait:** One protocol to ensure that hold-and-wait condition never occurs says each process must request and get all of its resources before it begins execution. Another protocol is “Each process can request resources only when it does not occupies any resources.” The second protocol is better. However, both protocols cause low resource utilization and starvation. Many resources are allocated but most of them are unused for a long period of time. A process that

requests several commonly used resources causes many others to wait indefinitely.

**3. No Preemption:** One protocol is “If a process that is holding some resources requests another resource and that resource cannot be allocated to it, then it must release all resources that are currently allocated to it.” Another protocol is “When a process requests some resources, if they are available, allocate them. If a resource it requested is not available, then we check whether it is being used or it is allocated to some other process waiting for other resources. If that resource is not being used, then the OS preempts it from the waiting process and allocate it to the requesting process. If that resource is used, the requesting process must wait.” This protocol can be applied to resources whose states can easily be saved and restored (registers, memory space). It cannot be applied to resources like printers.

**4. Circular Wait:** One protocol to ensure that the circular wait condition never holds is “Impose a linear ordering of all resource types.” Then, each process can only request resources in an increasing order of priority.

***For example***, set priorities for r1 = 1, r2 = 2, r3 = 3, and r4 = 4. With these priorities, if process P wants to use r1 and r3, it should first request r1, then r3.

Another protocol is “Whenever a process requests a resource rj, it must have released all resources rk with priority(rk) ≥ priority (rj).

***Deadlock Avoidance:***

Given some additional information on how each process will request resources, it is possible to

construct an algorithm that will avoid deadlock states. The algorithm will dynamically examine the resource allocation operations to ensure that there won't be a circular wait on resources.

Two deadlock avoidance algorithms:

1. Resource-allocation graph algorithm
2. Banker's algorithm

***Resource-allocation graph algorithm***

• only applicable when we only have 1 instance of each resource type

• claim edge (dotted edge), like a future request edge

• when a process requests a resource, the claim edge is converted to a request edge

• when a process releases a resource, the assignment edge is converted to a claim edge

***Bankers Algorithms:***

The Banker’s algorithm is a resource allocation and deadlock avoidance algorithm developed by

Edsger Dijkstra . Resource allocation state is defined by the number of available and allocated

resources and the maximum demand of the processes.

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

The system is in a safe state if there exist a safe sequence of all processes:

Sequence < P1, P2, ... Pn > is safe for the current allocation state if, for each Pi, the resources which Pi can request can be still satisfied by

• the currently available resources plus

• the resources held by all of the Pj's, where j < i.

If the system is in a safe state, there can be no deadlock. If the system is in an unsafe state, there is the possibility of deadlock.

A state is safe if the system can allocate resources to each process in some order avoiding a deadlock. A deadlock state is an unsafe state.

Customer =Processes

Units = Resource say tape drive

Bankers=OS

The Banker algorithm does the simple task

– If granting the request leads to an unsafe state the request is denied.

– If granting the request leads to safe state the request is carried out.

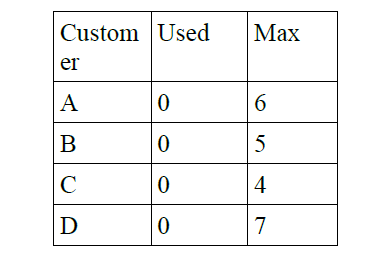
***Basic Facts:***

● If a system is in safe state ⇒ no deadlocks.

● If a system is in unsafe state ⇒ possibility of deadlock.

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

***Bankers Algorithms for a single resource:***

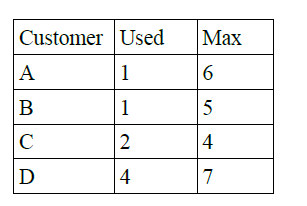


Available units: 10

In the above fig, we see four customers each of whom has been granted a certain no. of credit units ( eg. 1 unit=1K dollar). The Banker reserved only 10 units rather than 22 units to service them since not all

customer need their maximum credit immediately.

At a certain moment the situation becomes:



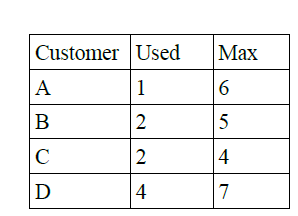
Available units: 2

Safe State:

With 2 units left, the banker can delay any requests except C's, thus letting C finish and release all four of his resources. With four in hand, the banker can let either D or B have the necessary units & so on.

Unsafe State:

B requests one more unit and is granted.



Available units: 1

this is an unsafe condition. If all of the customer namely A, B,C & D asked for their maximum loans, then Banker couldn't satisfy any of them and we would have deadlock. It is important to note that an unsafe state does not imply the existence or even eventual existence of a deadlock. What an unsafe does imply is that some unfortunate sequence of events might lead a deadlock.

***Bankers Algorithms for Multiple Resources:***

The algorithm for checking to see if a state is safe can now be stated.

1. Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such

row exists, the system will eventually deadlock since no process can run to completion.

2. Assume the process of the row chosen requests all the resources it needs (which is guaranteed

to be possible) and finishes. Mark that process as terminated and add all its resources to the A

vector.

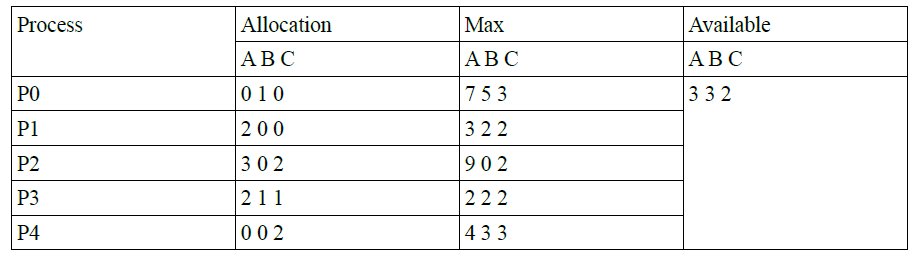
3. Repeat steps 1 and 2 until either all processes are marked terminated, in which case the initial

state was safe, or until a deadlock occurs, in which case it was not.

Q). Consider a system with five processes P0 through P4 and three resources types A, B, C.

Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at

time t0 following snapshot of the system has been taken

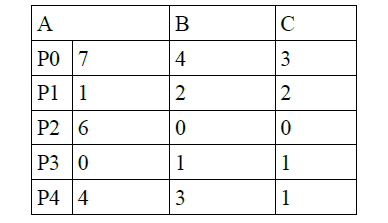


1) What will be the content of the need Matrix?

2) Is the system in safe state? If yes, then what is the safe sequence?

1. Need [i,j]= Max [i,j] – Allocation[i,j]

content of Need Matrix is



1. applying Safety algorithms

For Pi if Needi <= Available, then pi is in Safe sequence,

Available = Available + Allocation i

**For P0,** need0=7,4,3

Available = 3,3,2

==> Condition is false, So P0 must wait.

**For P1 ,** need1= 1,2,2

Available=3,3,2

need1< Available

So P1 will be kept in safe sequence. & Available will be updated as:

Available= 3,3,2 + 2,0,0 = 5,3,2

**For P2,** need2= 6,0,0

Available = 5,3,2

==> condition is again false, so P2 also have to wait.

**For P3,** need3= 0,1,1

Available= 5,3,2

==> condition is true , P3 will be in safe sequence.

Available = 5,3,2 + 2,1,1 = 7,4,3

**For P4,** need4= 4,3,1

Available = 7,4,3

==> condition Needi <= Available is true, so P4 will be in safe sequence

Available = 7,4,3 + 0,0,2 = 7,4,5

Now we have two processes P0 and P2 in waiting state. Either P0 or P2 can be choosen.

**Let us take P2** whose need = 6,0,0

Available = 7,4,5

Since condition is true, P2 now comes in safe state leaving the

Available = 7,4,5 + 3,0,2 = 10, 4,7

**Next P0** whose need = 7, 4, 3

Available = 10,4,7 since condition is true P0 also can be kept in safe state. So system is in safe state & the safe sequence is <P1, P3, P4, P2, P0>

***Detection and Recovery***

A second technique is detection and recovery. When this technique is used, the system does not do anything except monitor the requests and releases of resources. Every time a resource is requested or released, the resource graph is updated, and a check is made to see if any cycles exist. If a cycle exists, one of the processes in the cycle is killed. If this does not break the deadlock, another process is killed, and so on until the cycle is broken.