Deadlock

Unit 4

System Model

- We know that computer systems are full of resources that can only be used by one process at a time.
- Having two processes simultaneously reading/writing to the resources creates a problem. For many applications, a process needs exclusive access to several resources.
- For example: Two processes each want to record scanned document on a CD. Process A request permission to use scanner and is granted. Process B is programmed and requests the CD recorder first and is also granted. Now process A asks for the recorder but the request is denied until process B release it. Unfortunately, instead of releasing the CD recorder process B asks for the scanner. At this point both processes are blocked and will remain so forever. This situation is called deadlock.

System 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:

1. Preemptable
2. Non-Preemptable

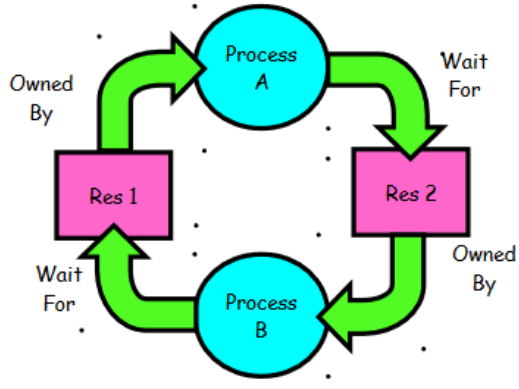
- **Preemptable** – A Preemptable resources is one that can be taken away from the process owing it with no ill effect. Memory is an example of preemptable resources
- **Non-preemptable** – A non-preemptable resources in contrast is one that cannot be taken away from its current owner without causing 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 not 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**: One of two things can happen when a resource is requested; the request can be granted immediately (if the resource is available) or it can be postponed (or blocked) until a later time.
2. **Use the resource**: Once the resource has been acquired, it can be used.
3. **Release the resource**: When the process no longer needs the resource it releases it. Usually it is released as soon as possible but in most systems there is nothing to enforce this policy.

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



Deadlock

Bridge Crossing Example

- Each segment of road can be viewed as a resource
  – Car must own the segment under them
  – Must acquire segment that they are moving into
- For bridge: must acquire both halves
  – Traffic only in one direction at a time
  – Problem occurs when two cars in opposite directions on bridge: each acquires one segment and needs next
  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

Conditions for Deadlock

- Four conditions must hold for there to be a deadlock:

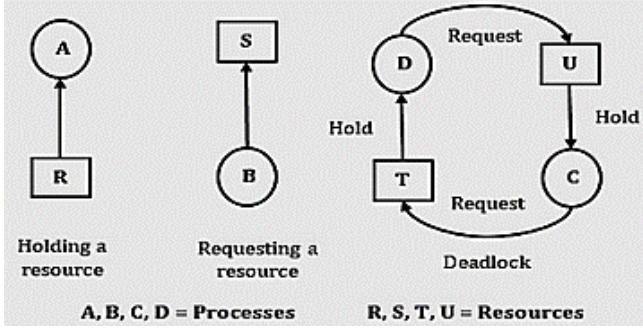
1. **Mutual exclusion**
   Only one process at a time can use a resource.
2. **Hold and wait**
   Process holding at least one resource is waiting to acquire additional resources held by other processes.
3. **No preemption**
   Resources are released only voluntarily by the process holding the resource, 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. If one or more of these conditions is absent, no Deadlock is possible.

Deadlock Modeling

- Holt (1972), showed how the above four conditions can be modeled using directed graphs. The graph has two kinds of nodes:

1. Processes (denoted by circle)
2. Resources (denoted by squares)



- A directed arc from resource node to a process node means that the request has previously been requested by granted to and is currently hold by that process.
- A directed arc from process node to resource node means that the process is requesting for resource.

Methods of Handling Deadlocks:

- Generally speaking there are three ways of handling deadlocks:
  - **Deadlock prevention or avoidance** - Do not allow the system to get into a deadlocked state.
  - **Deadlock detection and recovery** - Abort a process or preempt some resources when deadlocks are detected.
  - **Ignore the problem all together** - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.
- In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)
- Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.
- If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

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

Deadlock Prevention

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:
  • resource-allocation graph algorithm
  • Banker's algorithm

Banker's Algorithm

- 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 exists 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 still request can be 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

Banker's Algorithm

- 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

Banker's Algorithm for a single resource

Customer	Used	Max
A	0	6
B	0	5
C	0	4
D	0	7

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:

Customer	Used	Max
A	1	6
B	1	5
C	2	4
D	4	7

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:

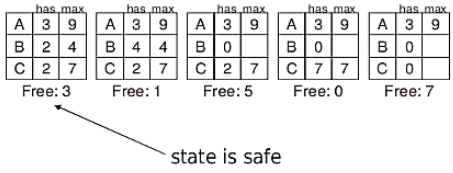
B requests one more unit and is granted.

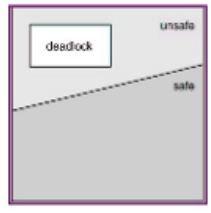
Banker's Algorithm for a single resource

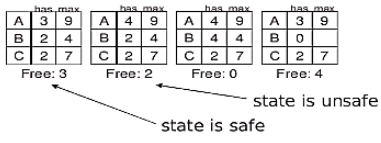
Customer	Used	Max
A	1	6
B	2	5
C	2	4
D	4	7

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.







Banker's Algorithm for a 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.

	Assigned resources			
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

	Resources still needed			
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

a). Current Allocation Matrix b). Request Matrix

E (Existing Resources): (6 3 4 2)
P (Processed Resources): (5 3 2 2)
A (Available Resources): (1 0 2 0)

Banker's Algorithm for a multiple resources

Solution:
Process A, B & C can't run to completion since for Process for each process, Request is greater than Available Resources. Now process D can complete since its requests row is less than that of Available resources.

Step 1:
When D run to completion the total available resources is:
A = (1, 0, 2, 0) + (1, 1, 0 , 1)= (2, 1, 2, 1)
Now Process E can run to completion
Step 2:
Now process E can also run to completion & return back all of its resources.
==> A = (0 , 0, 0, 0) + (2, 1, 2, 1) = (2, 1, 2, 1)
Step 3:
Now process A can also run to completion leading A to
(3, 0, 1, 1) + (2, 1, 2, 1) = (5, 1, 3, 2)
Step 4:
Now process C can also run to completion leading A to
(5, 1, 3, 2) + (1, 1, 1, 0) = (6, 2, 4, 2)
Step 5:
Now process B can run to completion leading A to
(6, 2, 4, 2) + (0, 1, 0 , 0) = (6, 3, 4, 2)
This implies the state is safe and Dead lock free

	Assigned resources			
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

	Resources still needed			
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

a). Current Allocation Matrix b). Request Matrix

E (Existing Resources): (6 3 4 2)
P (Processed Resources): (5 3 2 2)
A (Available Resources): (1 0 2 0)

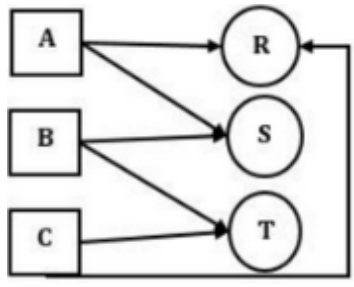
Deadlock Detection

- Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock.
- Once a deadlock is detected, there needs to be a way to recover and several alternatives exits:
  1. Temporarily prevent resources from deadlocked processes.
  2. Back off a process to some check point allowing preemption of a needed resource and restarting the process at the checkpoint later.
  3. Successively kill processes until the system is deadlock free.
- These methods are expensive in the sense that each iteration calls the detection algorithm until the system proves to be deadlock free.
- The complexity of algorithm is O(N2) where N is the number of proceeds.
- Another potential problem is starvation; same process killed repeatedly.
- In order to detect deadlock, we use resource allocation graph. Imagine, we have three processes A, B and C and three resources R, S and T

Deadlock Detection

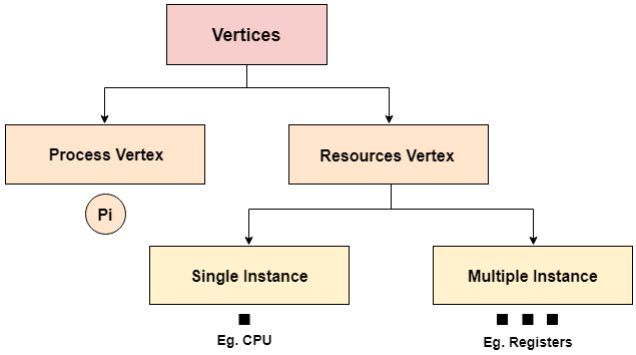
Process A	**Process B**	**Process C**
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
NO DEADLOCK		

Process A	**Process B**	**Process C**
Request R	Request S	Request T
Request S	Request T	Request R
DEADLOCK		

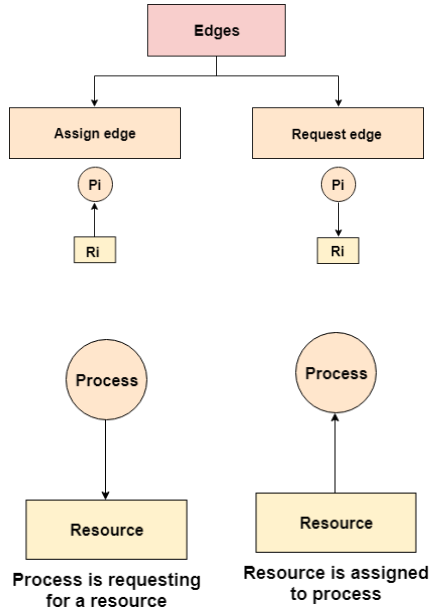


Resource Allocation Graph

- The resource allocation graph is the pictorial representation of the state of a system.
- As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.
- It also contains the information about all the instances of all the resources whether they are available or being used by the processes.
- In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.

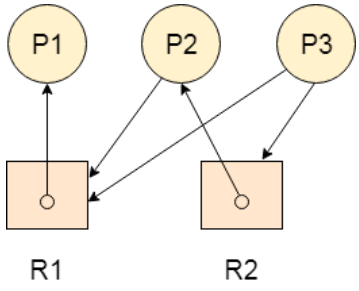


- Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.
- A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



- Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.
- A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.
- A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

- **Example**
- Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.
- According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.
- The graph is deadlock free since no cycle is being formed in the graph.



Recovery from Deadlock

Suppose that our deadlock detection algorithm has succeeded and detected a deadlock. Some way is needed to recover and get the system going again. So, here we have some ways to recover from deadlock and they are:

1. Recovery Through Preemption:

In some cases, it may be possible to temporarily take a resource away from its current owner and give it to other process. For example, To take a laser printer away from its owner, the operator can collect all the sheets already printed and put them in a pile. Then the process can be suspended (marked as not runnable).

At this point the printer can be assigned to another process. When the process finishes, the pile of sheets can be put back in the printer output tray and the process restarted. It depends upon the nature of resources.

2. Recovery Through Rollback:

In this approach processes are check pointed periodically (i.e. its state is written in file so that it can be restarted later). The checkpoint contains not only memory image but also the resource state i.e. which resources are currently assigned to which process.

When deadlock occurs, it is easy to see which resources are needed. To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired the resource.

3. Recovery Through Killing Processes:

It is the simplest way to break a deadlock. In this approach we kill one or more processes. By killing a process in a cycle if other process get chance to continue then the deadlock is ended but, if the processes did not continue, we again start to kill processes until the cycle get broken.

Ostrich Algorithm

- "stick your head in the sand and pretend there is no problem at all".
- Different people react to this strategy in very different ways. Mathematicians find it completely unacceptable and say that deadlocks must be prevented at all costs.
- Most operating systems, including UNIX, MINIX 3, and Windows, just ignore the problem on the assumption that most users would prefer an occasional deadlock to a rule restricting all users to one process, one open file, and one of everything. If deadlocks could be eliminated for free, there would not be much discussion.
- The Ostrich algorithm pretends there is no problem and is reasonable to use if deadlocks occur very rarely and the cost of their prevention would be high.

Starvation

- Starvation is closely related to deadlock. At certain condition, some processes never get resources even though they are not deadlocked, these processes goes on starvation due to which they get lost or dead.
- Foe example: Allocation of a printer through a file (i.e. shortest file to print). If new files are continually being added, the huge file will starve to death (postponed indefinitely even though it is not deadlocked).
- Some of the common causes of starvation are as follows:
  - If a process is never provided the resources it requires for execution because of faulty resource allocation decisions, then starvation can occur.
  - A lower priority process may wait forever if higher priority processes constantly monopolize the processor
  - Starvation may occur if there are not enough resources to provide to every process as required.
  - If random selection of processes is used then a process may wait for a long time because of non-selection.
- Some solutions that can be implemented in a system to handle starvation are as follows:
  - An independent manager can be used for allocation of resources. This resource manager distributes resources fairly and tries to avoid starvation.
  - Random selection of processes for resource allocation or processor allocation should be avoided as they encourage starvation.
  - The priority scheme of resource allocation should include concepts such as aging, where the priority of a process is increased the longer it waits. This avoids starvation.

Difference between Deadlock and Starvation

- Starvation: thread waits indefinitely
  Example, low-priority thread waiting for resources constantly in use by high-priority threads
- Deadlock: circular waiting for resources
  Thread A owns Res 1 and is waiting for Res 2 Thread B owns Res 2 and is waiting for Res 1
- Deadlock ==> Starvation but not vice versa
- Starvation can end (but doesn't have to)
- Deadlock can't end without external intervention