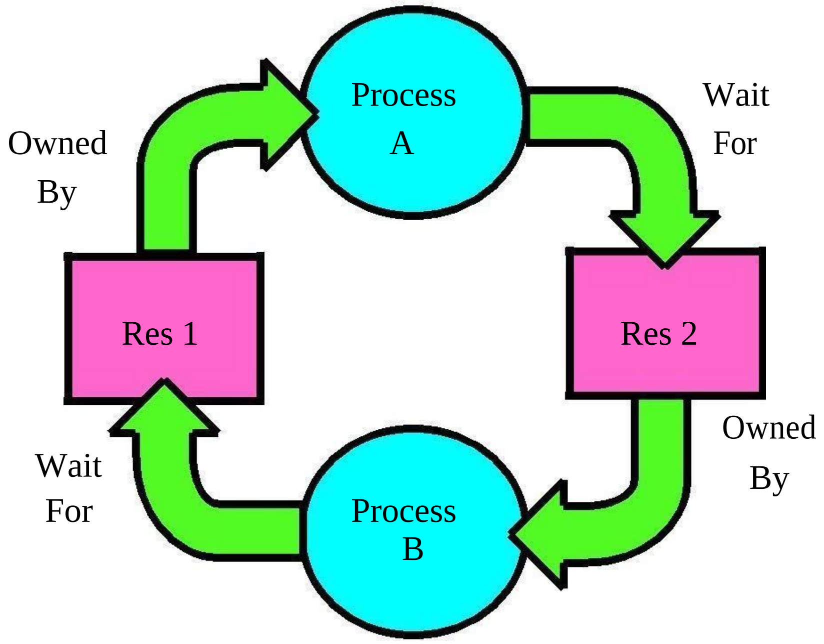
Deadlock and Indefinite Postponement

1. <u>Introduction:</u>

Indefinite Postponement is a process to delay indefinitely the scheduling of a process while other processes receive the system's attention.

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:

- Two processes 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 asks 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.

2. 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 includes 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 current owner (and give back later). 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 failure. 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.

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:

i. Request the resource
ii. Use the resource
iii. Release the resource

3. Conditions for Deadlock:

The four conditions must hold for there to be deadlocks are:

a) Mutual exclusion:

Only one process at a time can use a resource.

b) Hold and wait:

Process holding at least one resource is waiting to acquire additional resources held by other processes.

c) No preemption:

Resources are released only voluntarily by the process holding the resource, after the process is finished with it.

d) 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.

4. Deadlock Modeling:

Deadlocks can be described more precisely in terms of Resource allocation graph. It‟s 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

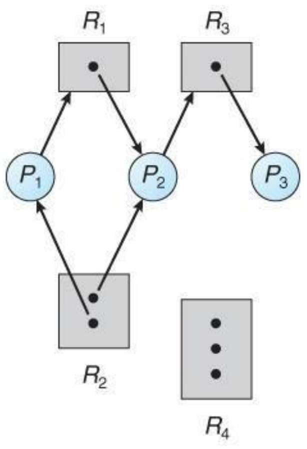


Figure a

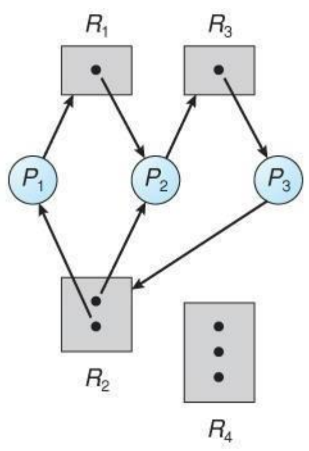


Figure b

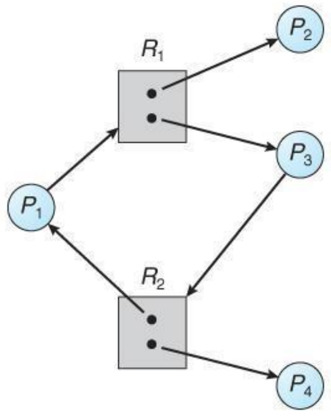


Figure c

- ❖ Fig a: Resource Allocation Graph
- ❖ Fig b: Resource Allocation graph with deadlock
- ❖ Fig c: Resource Allocation graph with a cycle but no deadlock

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

5. Methods for Handling Deadlock:

i. **Detection and recovery:** Allow system to enter deadlock and then recover
   - ❖ It requires deadlock detection algorithm
   - ❖ It must ensure some technique for forcibly preempting resources and/or terminating tasks which ensure that system will never enter a deadlock
   - ❖ It need to monitor all lock acquisitions
   - ❖ Selectively deny those that might lead to deadlock

ii. **Deadlock Prevention:** Ignore the problem and pretend that deadlocks never occur in the system
   - ❖ Used by most operating systems, including UNIX

6. 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:

I. 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.

II. 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 “request resources only when it does not occupy 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.

III. 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.

IV. 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).

7. 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. One of the deadlock avoidance algorithms is Banker's algorithm.

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 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.

A state is said to be safe if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately. 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.

Bankers Algorithms for a single resource:

Customer = Processes
Units = Resource say tape drive
Bankers = OS

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 customers 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:

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

Available units: 1

Suppose B requests one more unit and is granted. This is an unsafe condition. If all of the customers 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 state does imply is that some unfortunate sequence of events might lead a deadlock.

8. Detection and Recovery

It is a technique for handling deadlock. When this technique is used, the system does not attempt to prevent deadlocks from occurring. Instead, it lets them occur, tries to detect when this happens and then takes some action to recover after the fact.

Once the deadlock has been detected, it can be recovered through various ways:

a. **Recovery through preemption:** In this method, a resource can be temporarily taken away from its current owner and given to another process. The ability to take a resource away from a process, have another process use it and then give it back without the process noticing it, is highly dependent on the nature of the resource.

b. **Recovery through rollback:** When a deadlock is detected, it is easy to see which resources are needed. To do the recovery, a process that owns a needed resource is rolled back to an earlier point when it did not have the resource. The resource can now be assigned to one of the deadlocked processes.

c. **Recovery through killing processes:** The simplest way to break a deadlock is to kill one or more processes. 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.

9. Issues related to deadlocks:

9.1. Starvation:

In a dynamic system, request for resources happen all the time. Some policy is need to make a decision about who gets which resource when. This policy may lead to some processes never getting service even though they are not deadlocked. This situation is known as starvation. Starvation may be caused by errors in a scheduling or mutual exclusion algorithm, but can also be caused by resource leaks, and can be intentionally caused via a denial-of-service attack such as a fork bomb.

For example; Consider allocation of the printer. Imagine that the system uses some algorithm to ensure that allocating the printer does not lead to deadlock. Now, suppose that several processes wants it at once. Now deciding who will get the printer is a difficult task. Suppose one possible allocation algorithm is to give it to the process with the smallest file to print. Now consider what happens in a busy system when one process has a huge file to print. Every time the printer is free, the system will look around and choose the process with the shortest file. If there is a constant stream of processes with short files, the process with the huge file will never be allocated the printer. It will simply starve to death even though it is not blocked.

It can be avoided by using FCFS algorithm. With this approach, the process waiting the longest gets served next.

9.2. Two phase locking

It is a way to ensure serializability in accessing database item in a mutually exclusive manner. i.e; while one transaction is assessing a data item, no other transaction can modify that that data item. The most common method used to implement this requirement is known as lock. There are generally two modes in which a data item may be locked. They are

a) Shared lock: If a transaction T_i has obtained a shared mode lock denoted by „s‟ on data item Q then T_i can read but cannot write Q.
b) Exclusive lock: if a transaction T_i has obtained an exclusive mode lock denoted by X on data item Q then T_i can both read and write Q.