OPERATING SYSTEM
COMPUTER SOFTWARE
APPLICATION
OS
DATA
TECHNOLOGY
CODE DIGITAL
HARDWARE DESIGN
INTERNET
SCREEN
SERVICE
GRAPHIC
INPUT
APP
WIRELESS
USER
NETWORK
DEVELOPMENT
DEVELOPMENTCONCEPT
INFORMATIONCONCEPT
COMPUTING
CONCEPTUAL
STORAGE
MOBILE
MEMORY
BACKGROUND
DESKTOP
PROGRAMMING
ALLOCATION
PROGRAM
DEVICE
BUSINESS
WEB
COMMUNICATION
SYMBOL
OUTPUT

Jump2Learn
The Online Learning Place

OPERATING SYSTEM - II

CHAPTER 3 : DEADLOCKS

AUTHORS

Mr. Yatin Solanki
M.C.A., Ph.D.(Pursuing)

DOLAT USHA INSTITUTE OF
APPLIED SCIENCES, VALSAD

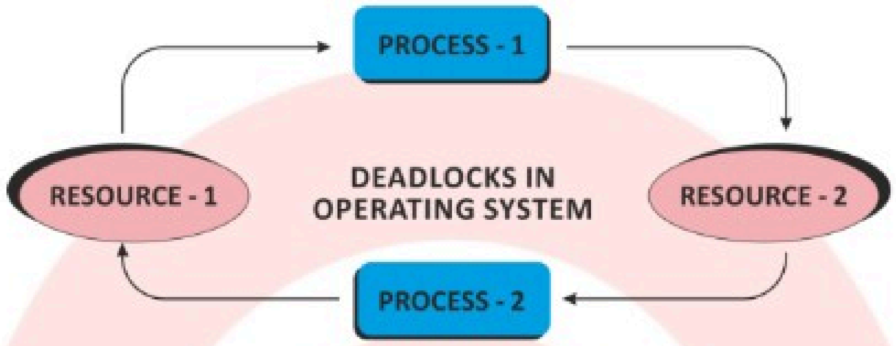
Dr. Jaimin Shukla
M.C.A., M. Phil., Ph.D.(Pursuing)

SUTEX BANK COLLEGE OF
COMPUTER APPLICATIONS & SCIENCE,
AMROLI

Website : www.jump2learn.com| **Email :** info@jump2learn.com|**Instagram :** www.instagram.com/jump2learn
Facebook : www.facebook.com/Jump2Learn | **Whatsapp :** +91-909-999-0960 | **YouTube :** Jump2Learn

3.1 DEADLOCKS

- A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function. The earliest computer operating systems ran only one program at a time.
- A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.



3.1 DEADLOCKS IN OPERATING SYSTEM

- In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is ready to give up their resources.

NECESSARY CONDITIONS (COFFMAN CONDITIONS):

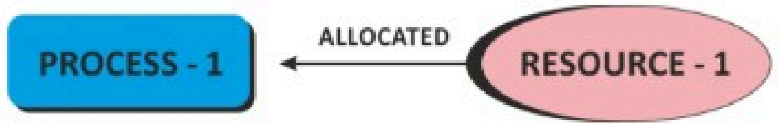
A deadlock situation can arise if the following four condition should occur simultaneously in a system.

A deadlock occurs if the four Coffman conditions hold true.

The Coffman conditions are given as follows:

1. **MUTUAL EXCLUSION**

   There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.

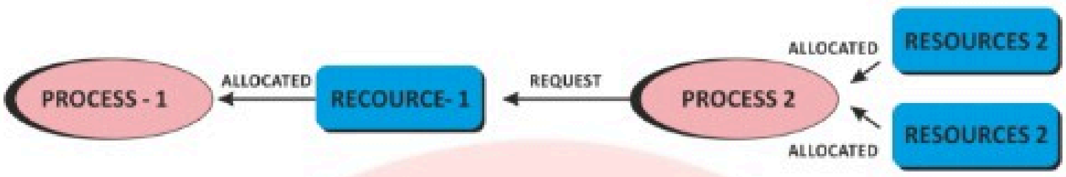


3.2 MUTUAL EXCLUSION

Jump2Learn

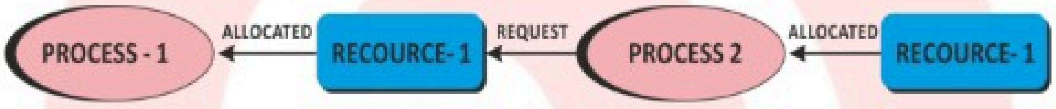
2. HOLD AND WAIT

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



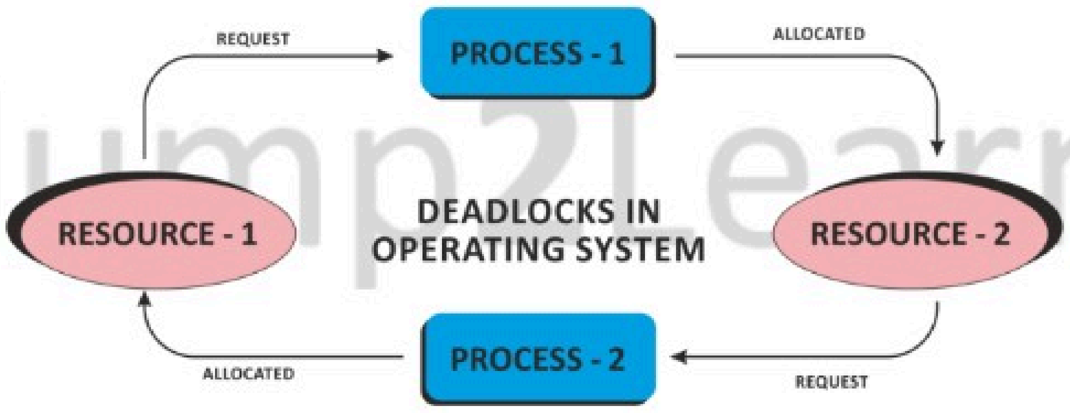
3. NO PREEMPTION

A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 give up it voluntarily after its execution is complete.



4. CIRCULAR WAIT

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Jump2Learn

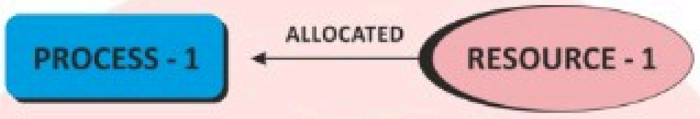
3.2 DEADLOCK CHARACTERISTICS

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows –

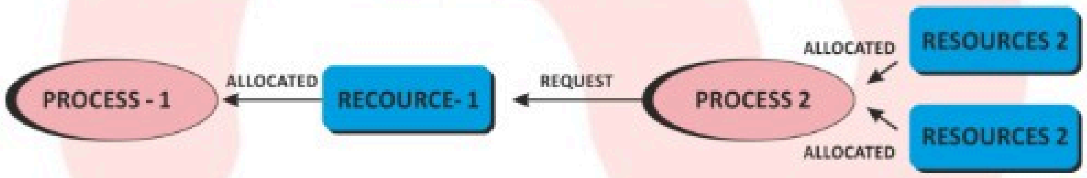
Mutual Exclusion

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



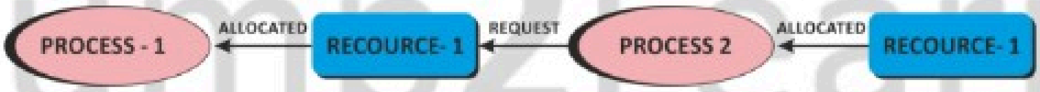
HOLD AND WAIT

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



NO PREEMPTION

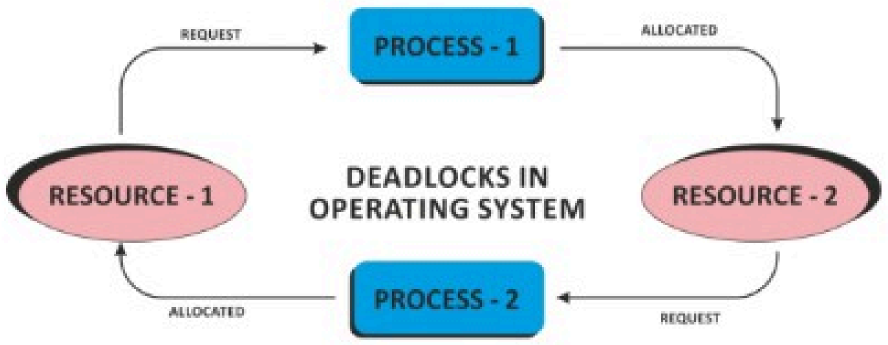
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



Jump2Learn

CIRCULAR WAIT

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



3.3 METHODS FOR HANDLING DEADLOCKS

- Generally speaking there are three ways of handling deadlocks:
  1. Deadlock prevention or avoidance - Do not allow the system to get into a deadlocked state.
  2. Deadlock detection and recovery - Abort a process or preempt some resources when deadlocks are detected.
  3. 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.

Jump2Learn

3.4 DEADLOCK PREVENTION:

- It is very important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight chance that a transaction may lead to deadlock in the future, it is never allowed to execute.
- Preventing deadlocks by constraining how requests for resources can be made in the system and how they are handled (system design). The goal is to ensure that at least one of the necessary conditions for deadlock can never hold.

We can prevent Deadlock by eliminating any of the above four condition:

- **ELIMINATION OF "MUTUAL EXCLUSION" CONDITION:**
  - The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource.
  - This condition is difficult to eliminate because some resources, such as the tap drive and printer, are inherently non-shareable.
  - Note that shareable resources like read-only-file do not require mutually exclusive access and thus cannot be involved in deadlock.

- **ELIMINATION OF "HOLD AND WAIT" CONDITION**
  - There are two possibilities for elimination of the second condition.
  - The first alternative is that a process request be granted all of the resources it needs at once, prior to execution.
  - The second alternative is to disallow a process from requesting resources whenever it has previously allocated resources.
  - This strategy requires that all of the resources a process will need must be requested at once.
  - The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available. While the process waits, however, it may not hold any resources.
  - Thus the "wait for" condition is denied and deadlocks simply cannot occur.
  - This strategy can lead to serious waste of resources. For example, a program requiring ten tap drives must request and receive all ten derives before it begins executing.
  - If the program needs only one tap drive to begin execution and then does not need the remaining tap drives for several hours. Then substantial computer resources (9 tape drives) will sit idle for several hours.
  - This strategy can cause indefinite delay (starvation). Since not all the required resources may become available at once.

Jump2Learn

- **ELIMINATION OF "NO-PREEMPTION" CONDITION**
  - The non-preemption condition can be improved by forcing a process waiting for a resource that cannot immediately be allocated to hand over all of its currently held resources, so that other processes may use them to finish.
  - Suppose a system does allow processes to hold resources while requesting additional resources. Consider what happens when a request cannot be satisfied.
  - A process holds resources a second process may need in order to proceed while second process may hold the resources needed by the first process. This is a deadlock.

- This strategy requires that when a process that is holding some resources is denied a request for additional resources. The process must release its held resources and, if necessary, request them again together with additional resources.
- Implementation of this strategy denies the "no-preemptive" condition effectively.

- **ELIMINATION OF "CIRCULAR WAIT" CONDITION**

  - The last condition, the circular wait, can be denied by imposing a total ordering on all of the resource types and then forcing, all processes to request the resources in order (increasing or decreasing).
  - This strategy impose a total ordering of all resources types, and to require that each process requests resources in a numerical order (increasing or decreasing) of enumeration. With this rule, the resource allocation graph can never have a cycle.

  For example, provide a global numbering of all the resources, as shown

1	≡	Card reader
2	≡	Printer
3	≡	Plotter
4	≡	Tape drive
5	≡	Card punch

Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone.

Jump2Learn

3.5 DEADLOCK AVOIDANCE:

- It is better to avoid a deadlock rather than take measures after the deadlock has occurred. The wait for graph can be used for deadlock avoidance. This is however only useful for smaller databases as it can get quite complex in larger databases.
- This approach to the deadlock problem anticipates deadlock before it actually occurs.
- This approach employs an algorithm to access the possibility that deadlock could occur and acting accordingly.
- This method differs from deadlock prevention, which guarantees that deadlock cannot occur by denying one of the necessary conditions of deadlock.

- If the necessary conditions for a deadlock are in place, it is still possible to avoid deadlock by being careful when resources are allocated.
- The most famous deadlock avoidance algorithm is the Banker's algorithm. So named because the process is analogous to that used by a banker in deciding if a loan can be safely made.

BANKER'S ALGORITHM:

In this analogy

Customers	=	processes
Units	=	Resources , say tap drive
Banker	=	Operating System

CUSTOMER	USED	MAX	
A	0	6	
B	0	5	AVAILABLE UNITS = 10
C	0	4	
D	0	7	

In the above figure, we see four customers each of whom has been granted a number of credit nits. The banker reserved only 10 units rather than 22 units to service them. At certain moment, the situation becomes

CUSTOMER	USED	MAX	
A	1	6	
B	1	5	AVAILABLE UNITS = 2
C	2	4	
D	4	7	

Jump2Learn

SAFE STATE :

- The key to a state being safe is that there is at least one way for all users to finish.
- In other analogy, the state of figure 2 is safe because with 2 units left, the banker can delay any request except *C's*, thus letting *C* finish and release all four resources.

- With four units in hand, the banker can let either *D* or *B* have the necessary units and so on.

UNSAFE STATE:

- Consider what would happen if a request from B for one more unit were granted in above second figure

  We would have following situation

CUSTOMER	USED	MAX	
A	1	6	AVAILABLE UNITS = 1
B	2	5	
C	2	4	
D	4	7	

THIS IS AN UNSAFE STATE.

If all the customers namely *A, B, C,* and *D* asked for their maximum loans, then banker could not satisfy any of them and we would have a deadlock.

IMPORTANT NOTE:

- It is important to note that an unsafe state does not imply the existence or even the eventual existence a deadlock. What an unsafe state does imply is simply that some unfortunate sequence of events might lead to a deadlock.
- The Banker's algorithm is thus to consider each request as it occurs, and see if granting it leads to a safe state.
- If it does, the request is granted, otherwise, it postponed until later.

3.6 DEADLOCK DETECTION

A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be resolved using the following methods:

1. All the processes that are involved in the deadlock are terminated. This is not a good approach as all the progress made by the processes is destroyed.
2. Resources can be preempted from some processes and given to others till the deadlock is resolved.

3.7 DEADLOCK RECOVERY:

- Deadlock recovery performs when a deadlock is detected.
- When deadlock detected, then our system stops working, and after the recovery of the deadlock, our system start working again.
- Therefore, after the detection of deadlock, a method/way must require to recover that deadlock to run the system again.
- The method/way is called as deadlock recovery.

Jump2Learn

Here are various ways of deadlock recovery:

- Deadlock recovery through preemption
- Deadlock recovery through rollback
- Deadlock recovery through killing processes

DEADLOCK RECOVERY THROUGH PREEMPTION:

- 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.
- Deadlock recovery through preemption is too difficult or sometime impossible.

DEADLOCK RECOVERY THROUGH ROLLBACK:

- In this case of deadlock recovery through rollback, whenever a deadlock is detected, it is easy to see which resources are needed.
- To do the recovery of deadlock, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource just by starting one of its earlier checkpoints.

DEADLOCK RECOVERY THROUGH KILLING PROCESSES:

- This method of deadlock recovery through killing processes is the simplest way of deadlock recovery.
- Sometime it is best to kill a process that can be return from the beginning with no ill effects.

Jump2Learn