DEPARTMENT OF AIML

23CST202- OPERATING SYSTEMS

II YEAR IV SEM AIML-B

UNIT 2-PROCESS SCHEDULING AND SYNCHRONIZATION

TOPIC –DEADLOCK,SYSTEM MODEL

DEADLOCK

A deadlock is a situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process. In this article, we will discuss deadlock, its necessary conditions, etc. in detail.

- **Deadlock** is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources.
- Key concepts include mutual exclusion, resource holding, circular wait, and no preemption.

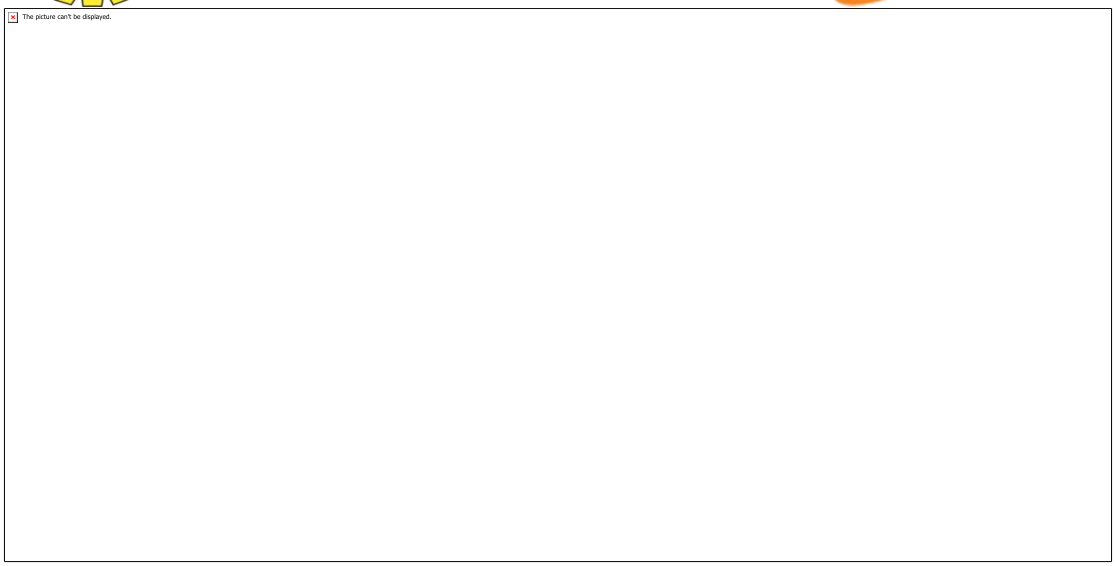
Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. This is a practical example of deadlock.

How Does Deadlock occur in the Operating System?

Before going into detail about how deadlock occurs in the Operating System, let's first discuss how the Operating System uses the resources present. A process in an operating system uses resources in the following way.

- **Requests a resource**
- **Use the resource**
- **Releases the resource**

A situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). **For example**, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Examples of Deadlock

There are several examples of deadlock. Some of them are mentioned below.

1. The system has 2 tape drives. P0 and P1 each hold one tape drive and each needs another one.

2. Semaphores A and B, initialized to 1, P0, and P1 are in deadlock as follows:

- P0 executes wait(A) and preempts.
- P1 executes wait(B).
- Now P0 and P1 enter in deadlock.

P0	P1
wait(A);	wait(B)
wait(B);	wait(A)

3. Assume the space is available for allocation of 200K bytes, and the following sequence of events occurs.

P0	P1
Request 80KB;	Request 70KB;
Request 60KB;	Request 80KB;

Deadlock occurs if both processes progress to their second request.

Necessary Conditions for Deadlock in OS

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

- **Mutual Exclusion:** Only one process can use a resource at any given time i.e. the resources are non-sharable.
- **Hold and Wait:** A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
- **No Preemption:** A resource cannot be taken from a process unless the process releases the resource.
- **Circular Wait:** set of processes are waiting for each other in a circular fashion. For example, lets say there are a set of processes {P0P0,P1P1,P2P2,P3P3} such that P0P0 depends on P1P1, P1P1 depends on P2P2, P2P2 depends on P3P3 and P3P3 depends on P0P0. This creates a circular relation between all these processes and they have to wait forever to be executed.

Methods of Handling Deadlocks in Operating System

There are three ways to handle deadlock:

1. Deadlock Prevention or Avoidance
2. Deadlock Detection and Recovery
3. Deadlock Ignorance

Deadlock Prevention or Avoidance

Deadlock Prevention and Avoidance is the one of the methods for handling deadlock. First, we will discuss Deadlock Prevention, then Deadlock Avoidance.

Deadlock Prevention

In deadlock prevention the aim is to not let full-fill one of the required condition of the deadlock. This can be done by this method:

(i) Mutual Exclusion

We only use the Lock for the non-share-able resources and if the resource is share- able (like read only file) then we not use the locks here. That ensure that in case of share -able resource , multiple process can access it at same time. Problem- Here the problem is that we can only do it in case of share-able resources but in case of no-share-able resources like printer , we have to use Mutual exclusion.

(ii) Hold and Wait

To ensure that Hold and wait never occurs in the system, we must guarantee that whenever process request for resource , it does not hold any other resources.

- we can provide the all resources to the process that is required for it's execution before starting it's execution . **problem –** for example if there are three resource that is required by a process and we have given all that resource before starting execution of process then there might be a situation that initially we required only two resource and after one hour we want third resources and this will cause starvation for the another process that wants this resources and in that waiting time that resource can allocated to other process and complete their execution.

- We can ensure that when a process request for any resources that time the process does not hold any other resources. Ex- Let there are three resources DVD, File and Printer . First the process request for DVD and File for the copying data into the file and let suppose it is going to take 1 hour and after it the process free all resources then again request for File and Printer to print that file.

(iii) No Preemption

If a process is holding some resource and requestion other resources that are acquired and these resource are not available immediately then the resources that current process is holding are preempted. After some time process again request for the old resources and other required resources to re-start.

For example – Process p1 have resource r1 and requesting for r2 that is hold by process p2. then process p1 preempt r1 and after some time it try to restart by requesting both r1 and r2 resources.

Problem – This can cause the Live Lock Problem .

Live Lock : Live lock is the situation where two or more processes continuously changing their state in response to each other without making any real progress.

Example:

- suppose there are two processes p1 and p2 and two resources r1 and r2.
- Now, p1 acquired r1 and need r2 & p2 acquired r2 and need r1.
- so according to above method- Both p1 and p2 detect that they can't acquire second resource, so they release resource that they are holding and then try again.
- continuous cycle- p1 again acquired r1 and requesting to r2 p2 again acquired r2 and requesting to r1 so there is no overall progress still process are changing there state as they preempt resources and then again holding them. This the situation of Live Lock.

(iv) Circular Wait:

To remove the circular wait in system we can give the ordering of resources in which a process needs to acquire.

Ex: If there are process p1 and p2 and resources r1 and r2 then we can fix the resource acquiring order like the process first need to acquire resource r1 and then resource r2. so the process that acquired r1 will be allowed to acquire r2 , other process needs to wait until r1 is free.

This is the Deadlock prevention methods but practically only fourth method is used as all other three condition removal method have some disadvantages with them.

Deadlock Avoidance

Avoidance is kind of futuristic. By using the strategy of "Avoidance", we have to make an assumption. We need to ensure that all information about resources that the process will need is known to us before the execution of the process. We use Banker's algorithm to avoid deadlock.

In prevention and avoidance, we get the correctness of data but performance decreases.

Deadlock Detection and Recovery

If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

1. In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
2. If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

In Deadlock detection and recovery, we get the correctness of data but performance decreases.

Deadlock Detection

Deadlock detection is a process in computing where the system checks if there are any sets of processes that are stuck waiting for each other indefinitely, preventing them from moving forward. In simple words, deadlock detection is the process of finding out whether any process are stuck in loop or not. There are several algorithms like;

- Resource Allocation Graph
- Banker's Algorithm

These algorithms helps in detection of deadlock in Operating System.

Deadlock Recovery

There are several Deadlock Recovery Techniques:

- Manual Intervention
- Automatic Recovery
- Process Termination
- Resource Preemption

1. Manual Intervention

When a deadlock is detected, one option is to inform the operator and let them handle the situation manually. While this approach allows for human judgment and decision-making, it can be time-consuming and may not be feasible in large-scale systems.

2. Automatic Recovery

An alternative approach is to enable the system to recover from deadlock automatically. This method involves breaking the deadlock cycle by either aborting processes or preempting resources. Let's delve into these strategies in more detail.

3. Process Termination

- **Abort all Deadlocked Processes**
  This approach breaks the deadlock cycle, but it comes at a significant cost. The processes that were aborted may have executed for a considerable amount of time, resulting in the loss of partial computations. These computations may need to be recomputed later.
- **Abort one process at a time**
  Instead of aborting all deadlocked processes simultaneously, this strategy involves selectively aborting one process at a time until the deadlock cycle is eliminated. However, this incurs overhead as a deadlock-detection algorithm must be invoked after each process termination to determine if any processes are still deadlocked.
- **Factors for choosing the termination order:**
  The process's priority
  Completion time and the progress made so far
  Resources consumed by the process
  Resources required to complete the process
  Number of processes to be terminated
  Process type (interactive or batch)

4. Resource Preemption

- **Selecting a Victim**
  Resource preemption involves choosing which resources and processes should be preempted to break the deadlock. The selection order aims to minimize the overall cost of recovery. Factors considered for victim selection may include the number of resources held by a deadlocked process and the amount of time the process has consumed.
- **Rollback**
  If a resource is preempted from a process, the process cannot continue its normal execution as it lacks the required resource. Rolling back the process to a safe state and restarting it is a common approach. Determining a safe state can be challenging, leading to the use of total rollback, where the process is aborted and restarted from scratch.
- **Starvation Prevention**
  To prevent resource starvation, it is essential to ensure that the same process is not always chosen as a victim. If victim selection is solely based on cost factors, one process might repeatedly lose its resources and never complete its designated task. To address this, it is advisable to limit the number of times a process can be chosen as a victim, including the number of rollbacks in the cost factor.

Deadlock Ignorance

If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take. we use the ostrich algorithm for deadlock ignorance.

"In Deadlock, ignorance performance is better than the above two methods but the correctness of data is not there."

Safe State

A safe state can be defined as a state in which there is no deadlock. It is achievable if:

- If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
- All the requested resources are allocated to the process.

Difference between Starvation and Deadlocks

Aspect	Deadlock	Starvation
Definition	A condition where two or more processes are blocked forever, each waiting for a resource held by another.	A condition where a process is perpetually denied necessary resources, despite resources being available.
Resource Availability	Resources are held by processes involved in the deadlock.	Resources are available but are continuously allocated to other processes.
Cause	Circular dependency between processes, where each process is waiting for a resource from another.	Continuous preference or priority given to other processes, causing a process to wait indefinitely.
Resolution	Requires intervention, such as aborting processes or preempting resources to break the cycle.	Can be mitigated by adjusting scheduling policies to ensure fair resource allocation.

SYSTEM MODEL

Overview :

A deadlock occurs when a set of processes is stalled because each process is holding a resource and waiting for another process to acquire another resource. In the diagram below, for example, Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.

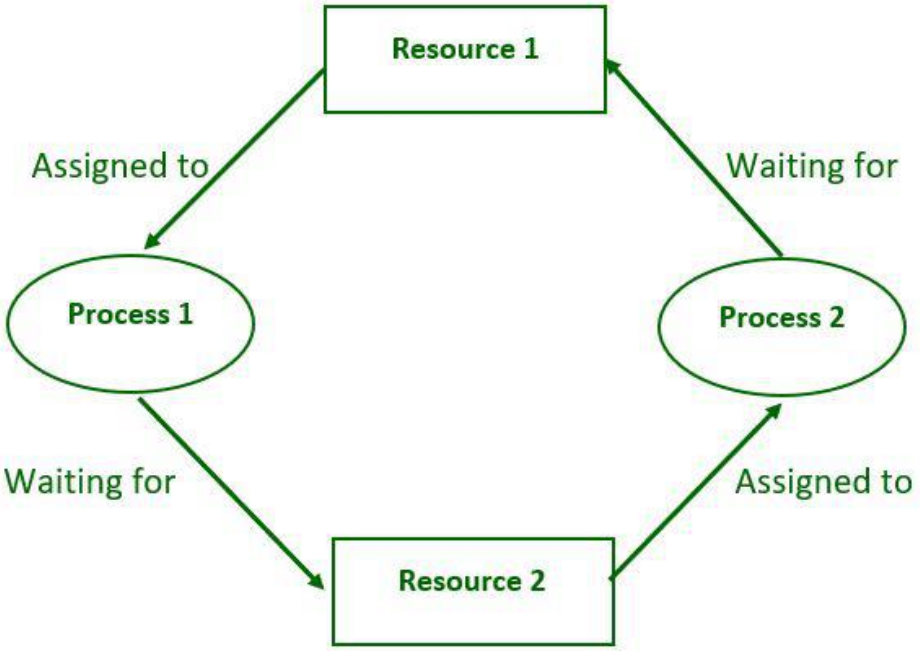


Figure: Deadlock in Operating system

System Model :

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources that can be divided into different categories and allocated to a variety of processes, each with different requirements.
- Memory, printers, CPUs, open files, tape drives, CD-ROMs, and other resources are examples of resource categories.
- By definition, all resources within a category are equivalent, and any of the resources within that category can equally satisfy a request from that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category must be subdivided further. For example, the term "printers" may need to be subdivided into "laser printers" and "color inkjet printers."
- Some categories may only have one resource.
- The kernel keeps track of which resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available for all kernel-managed resources. Mutexes or wait() and signal() calls can be used to control application-managed resources (i.e. binary or counting semaphores.)
- When every process in a set is waiting for a resource that is currently assigned to another process in the set, the set is said to be deadlocked.

Operations :

In normal operation, a process must request a resource before using it and release it when finished, as shown below.

1. **Request –**
   If the request cannot be granted immediately, the process must wait until the resource(s) required to become available. The system, for example, uses the functions open(), malloc(), new(), and request ().
2. **Use –**
   The process makes use of the resource, such as printing to a printer or reading from a file.
3. **Release –**
   The process relinquishes the resource, allowing it to be used by other processes.

Necessary Conditions :

There are four conditions that must be met in order to achieve deadlock as follows.

1. **Mutual Exclusion** –
   At least one resource must be kept in a non-shareable state; if another process requests it, it must wait for it to be released.

2. **Hold and Wait** –
   A process must hold at least one resource while also waiting for at least one resource that another process is currently holding.

3. **No preemption** –
   Once a process holds a resource (i.e. after its request is granted), that resource cannot be taken away from that process until the process voluntarily releases it.

4. **Circular Wait** –
   There must be a set of processes P0, P1, P2,…, PN such that every P[I] is waiting for P[(I + 1) percent (N + 1)]. (It is important to note that this condition implies the hold-and-wait condition, but dealing with the four conditions is easier if they are considered separately).

Methods for Handling Deadlocks :

In general, there are three approaches to dealing with deadlocks as follows.

1. Preventing or avoiding deadlock by Avoid allowing the system to become stuck in a loop.
2. Detection and recovery of deadlocks, When deadlocks are detected, abort the process or preempt some resources.
3. Ignore the problem entirely.
4. To avoid deadlocks, the system requires more information about all processes. The system, in particular, must understand what resources a process will or may request in the future. (Depending on the algorithm, this can range from a simple worst-case maximum to a complete resource request and release plan for each process.)
5. Deadlock detection is relatively simple, but deadlock recovery necessitates either aborting processes or preempting resources, neither of which is an appealing option.
6. If deadlocks are not avoided or detected, the system will gradually slow down as more processes become stuck waiting for resources that the deadlock has blocked and other waiting processes. Unfortunately, when the computing

requirements of a real-time process are high, this slowdown can be confused with a general system slowdown.

Deadlock Prevention :

Deadlocks can be avoided by avoiding at least one of the four necessary conditions: as follows.

Condition-1 :

Mutual Exclusion :

- Read-only files, for example, do not cause deadlocks.
- Unfortunately, some resources, such as printers and tape drives, require a single process to have exclusive access to them.

Condition-2 :

Hold and Wait :

To avoid this condition, processes must be prevented from holding one or more resources while also waiting for one or more others. There are a few possibilities here:

- Make it a requirement that all processes request all resources at the same time. This can be a waste of system resources if a process requires one resource early in its execution but does not require another until much later.
- Processes that hold resources must release them prior to requesting new ones, and then re-acquire the released resources alongside the new ones in a single new request. This can be a problem if a process uses a resource to partially complete an operation and then fails to re-allocate it after it is released.
- If a process necessitates the use of one or more popular resources, either of the methods described above can result in starvation.

Condition-3 :

No Preemption :

When possible, preemption of process resource allocations can help to avoid deadlocks.

- One approach is that if a process is forced to wait when requesting a new resource, all other resources previously held by this process are implicitly released (preempted), forcing this process to re-acquire the old resources alongside the new resources in a single request, as discussed previously.
- Another approach is that when a resource is requested, and it is not available, the system looks to see what other processes are currently using those resources and are themselves blocked while waiting for another resource. If such a process is discovered, some of their resources may be preempted and added to the list of resources that the process is looking for.
- Either of these approaches may be appropriate for resources whose states can be easily saved and restored, such as registers and memory, but they are generally inapplicable to other devices, such as printers and tape drives.

Condition-4 :

Circular Wait :

- To avoid circular waits, number all resources and insist that processes request resources is strictly increasing (or decreasing) order.
- To put it another way, before requesting resource Rj, a process must first release all Ri such that I >= j.

- The relative ordering of the various resources is a significant challenge in this scheme.

Deadlock Avoidance :

- The general idea behind deadlock avoidance is to avoid deadlocks by avoiding at least one of the aforementioned conditions.
- This necessitates more information about each process AND results in low device utilization. (This is a conservative approach.)
- The scheduler only needs to know the maximum number of each resource that a process could potentially use in some algorithms. In more complex algorithms, the scheduler can also use the schedule to determine which resources are required and in what order.
- When a scheduler determines that starting a process or granting resource requests will result in future deadlocks, the process is simply not started or the request is denied.
- The number of available and allocated resources, as well as the maximum requirements of all processes in the system, define a resource allocation state.

Deadlock Detection :

- If deadlocks cannot be avoided, another approach is to detect them and recover in some way.
- Aside from the performance hit of constantly checking for deadlocks, a policy/algorithm for recovering from deadlocks must be in place, and when processes must be aborted or have their resources preempted, there is the possibility of lost work.

Recovery From Deadlock :

There are three basic approaches to getting out of a bind:

1. Inform the system operator and give him/her permission to intervene manually.
2. Stop one or more of the processes involved in the deadlock.
3. Prevent the use of resources.

Approach of Recovery From Deadlock :

Here, we will discuss the approach of Recovery From Deadlock as follows.

Approach-1 :

Process Termination :

There are two basic approaches for recovering resources allocated to terminated processes as follows.

1. Stop all processes that are involved in the deadlock. This does break the deadlock, but at the expense of terminating more processes than are absolutely necessary.
2. Processes should be terminated one at a time until the deadlock is broken. This method is more conservative, but it necessitates performing deadlock detection after each step.

In the latter case, many factors can influence which processes are terminated next as follows.

1. Priorities in the process
2. How long has the process been running and how close it is to completion.

3. How many and what kind of resources does the process have? (Are they simple to anticipate and restore?)
4. How many more resources are required for the process to be completed?
5. How many processes will have to be killed?
6. Whether the process is batch or interactive.

Approach-2 :

Resource Preemption :

When allocating resources to break the deadlock, three critical issues must be addressed:

1. **Selecting a victim –**
   Many of the decision criteria outlined above apply to determine which resources to preempt from which processes.
2. **Rollback –**
   A preempted process should ideally be rolled back to a safe state before the point at which that resource was originally assigned to the process. Unfortunately, determining such a safe state can be difficult or impossible, so the only safe rollback is to start from the beginning. (In other words, halt and restart the process.)
3. **Starvation –**
   How do you ensure that a process does not go hungry because its resources are constantly being preempted? One option is to use a priority system and raise the priority of a process whenever its resources are preempted. It should eventually gain a high enough priority that it will no longer be preempted.