Silver Oak University

College Of Computer Application

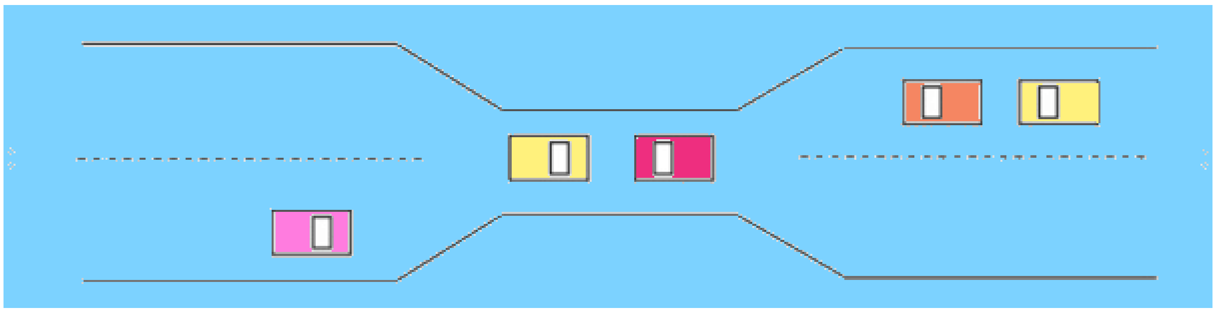
Operating System

Unit 4

1) What is deadlock?

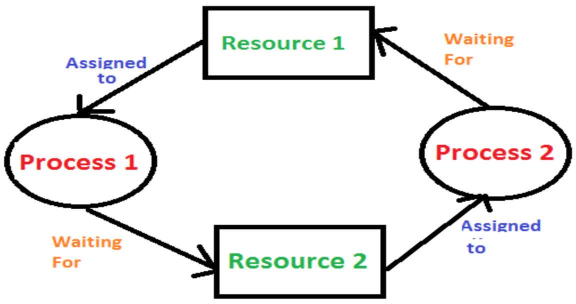
Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each

Other.



A process in operating system uses resources in the following way.
1) Requests a resource
2) Use the resource
3) Releases the resource

A similar 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.



Deadlock can arise if **the** following four conditions hold simultaneously (Necessary Conditions)
Mutual Exclusion: Two or more resources are non-shareable (Only one process can use at a time)
Hold and Wait: A process is holding at least one resource and waiting for resources.
No Preemption: A resource cannot be taken from a process unless the process releases the

resource.
Circular Wait: A set of processes are waiting for each other in circular form.

2) Operating system deadlock problems

Deadlock is **a situation that occurs in OS when any process enters a waiting state because another waiting process is holding the demanded resource**. Deadlock is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a soft lock or software.

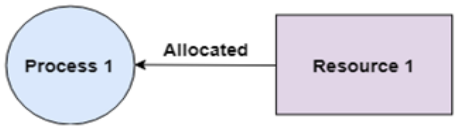
3) Deadlock Characterization

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 −

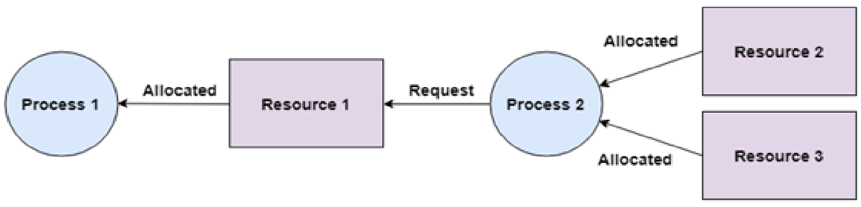
Mutual Exclusion

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.



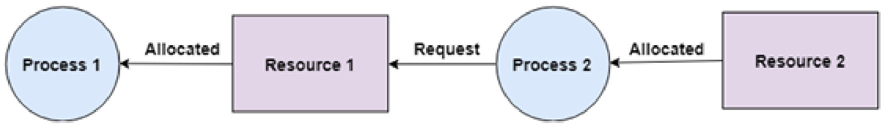
Hold and Wait

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.



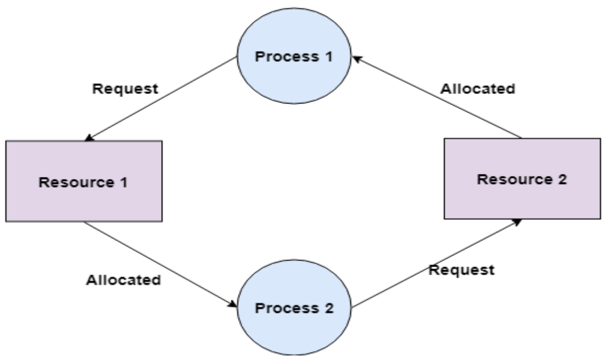
No Preemption

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.



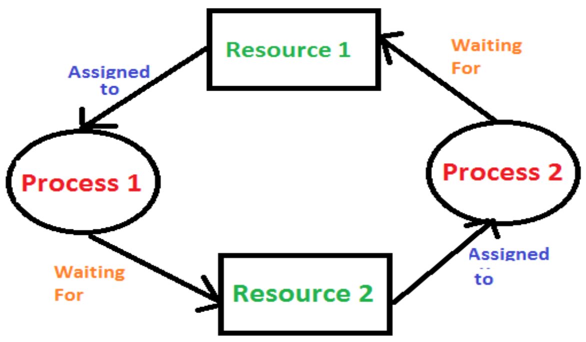
Circular Wait

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.



4) Deadlock Detection :

1. If resources have a single instance –
In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph. The presence of a cycle in the graph is a sufficient condition for deadlock.



2. In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1 → P1 → R2 → P2. So, Deadlock is Confirmed.

3. If there are multiple instances of resources –
Detection of the cycle is necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

5) Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is a time and space-consuming process. Real-time operating systems use Deadlock recovery.

1. **Killing the process –**
   Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till the system

recovers from deadlock. Killing all the processes one by one helps a system to break circular wait condition.

2. **Resource Preemption –**
   Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

6) Deadlock Avoidance :

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

7) Banker's Algorithm

banker algorithm used to **avoid deadlock** and **allocate resources** safely to each process in the computer system. The '**S-State'** examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. In this section, we will learn the **Banker's Algorithm** in detail. Also, we will solve problems based on the **Banker's Algorithm**. To understand the Banker's Algorithm first we will see a real word example of it.

Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

Similarly, it works in an **operating system**. When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as **deadlock avoidance algorithm** or **deadlock detection** in the operating system

Advantages :

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

Disadvantages:

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

When working with a banker's algorithm, it requests to know about three things:

1. How much each process can request for each resource in the system. It is denoted by the [**MAX**] request.
2. How much each process is currently holding each resource in a system. It is denoted by the [**ALLOCATED**] resource.
3. It represents the number of each resource currently available in the system. It is denoted by the [**AVAILABLE**] resource.

Need[i][j]= Max[i][j] - Allocation[i][j], where i corresponds any process P(i) and j corresponds to any resouce type R(j)

8) Data Structures used to implement Banker's Algorithm

1. **Available:** It is a 1-D array that tells the number of each resource type (instance of resource type) currently available. ***Example:*** Available[R1]= A, means that there are A instances of R1 resources are currently available.
2. **Max:** It is a 2-D array that tells the maximum number of each resource type required by a process for successful execution. ***Example:*** Max[P1][R1] = A, specifies that the process P1 needs a maximum of **A** instances of resource R1 for complete execution.

3. **Allocation:** It is a 2-D array that tells the number of types of each resource type that has been allocated to the process. **Example:** Allocation[P1][R1] = A, means that **A** instances of resource type R1 have been allocated to the process P1.

4. **Need:** It is a 2-D array that tells the number of remaining instances of each resource type required for execution. **Example:** Need[P1][R1]= A tells that **A** instances of R1 resource type are required for the execution of process P1.

Note : *Need[i][j]= Max[i][j] - Allocation[i][j], where i corresponds any process P(i) and j corresponds to any resouce type R(j)*

9) Deadlock Prevention in Operating System

There are four necessary conditions for deadlock. Deadlock happens only when all four conditions occur *simultaneously for unshareable single instance resources*.

The conditions for deadlock are:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait.

There are three ways to handle deadlock:

1. **Deadlock prevention:** The possibility of deadlock is excluded before making requests, by eliminating one of the necessary conditions for deadlock. Example: Only allowing traffic from one direction, will exclude the possibility of blocking the road.
2. **Deadlock avoidance:** Operating system runs an algorithm on requests to check for a safe state. Any request that may result in a deadlock is not granted. **Example:** Checking each car and not allowing any car that can block the road. If there is already traffic on road, then a car coming from the opposite direction can cause blockage.
3. **Deadlock detection & recovery:** OS detects deadlock by regularly checking system state, and recovers to safe state using recovery techniques. **Example:** Unblocking the road by backing cars from one side.

Deadlock prevention and deadlock avoidance are carried out before deadlock occurs.

10) <u>Deadlock Prevention Techniques</u>

Deadlock prevention techniques refer to violating any one of the four necessary conditions. We will see one by one how we can violate each of them to make safe requests and which is the best approach to prevent deadlock.

Mutual Exclusion

Some resources are inherently unshareable, for example: Printer. For unshareable resources, processes require exclusive control of the resources.

Mutual exclusion means that unshareable resources cannot be accessed simultaneously by processes.

Shared resources do not cause deadlock but some resources can't be shared among processes, leading to a deadlock.

For example: *read* operation on a file can be done simultaneously by multiple processes, but *write* operation cannot. *Write* operation requires sequential access, so, some processes have to wait while another process is doing a *write* operation.

It is not possible to eliminate mutual exclusion, as some resources are inherently non shareable,

For example: Tape drive, as only one process can access data from a Tape drive at a time.

For other resources like printer, we can use a technique called **Spooling**.

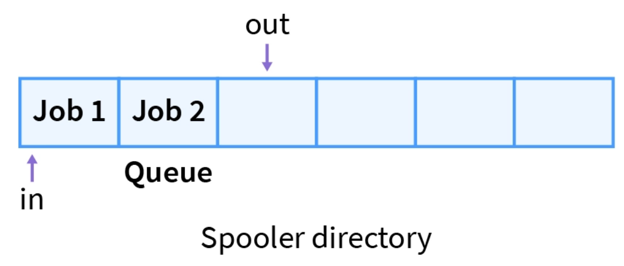
Spooling: It stands for Simultaneous Peripheral Operations On-line.

A Printer has associated memory which can be used as a spooler directory (memory which is used to store files that are to be printed next).

In spooling, when multiple processes request for the printer, their jobs (instructions of the processes that require printer access) are added to the queue in the spooler directory.

The printer is allocated to *jobs* on a first come first serve (FCFS) basis. In this way, the process does not have to wait for the printer and it continues its work after adding its *job* in the queue.

We can understand the working of Spooler directory better with the diagram given below:



in : pointer to next file to printed.

out : pointer to next empty slot.

Challenges of Spooling:

- Spooling can only be used for the resources with associated memory, like a Printer.
- It may also causes **race condition**. Race condition is the situation where two or more processes are accessing a resource and the final results cannot be definitively determined.

  For example: In printer spooling, if process A overwrites the job of process B in the queue, then process B will never receive the output.

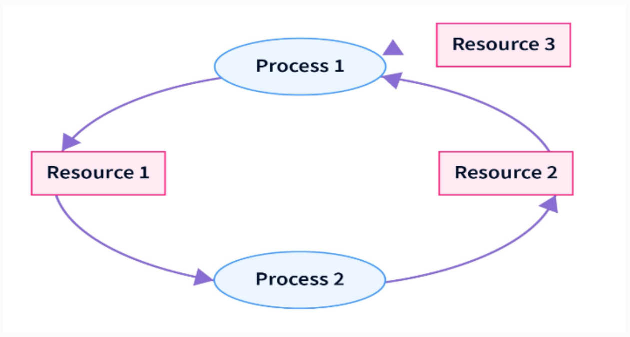
- It is not a full proof method as after the queue becomes full, incoming processes go in a waiting state.

By: Payal Dhanesha, Assistant Professor, SOCCA

For example: If the size of the queue is 10 blocks then whenever there are more than 10 processes, they will go in waiting state.

Hold and Wait

Hold and wait is a condition in which a process is holding one resource while simultaneously waiting for another resource that is being held by another process. The process cannot continue till it gets all the required resources.



In the diagram given below:

- Resource 1 is allocated to Process 2
- Resource 2 is allocated to Process 1
- Resource 3 is allocated to Process 1
- Process 1 is waiting for Resource 1 and holding Resource 2 and Resource 3
- Process 2 is waiting for Resource 2 and holding Resource 1

There are two ways to eliminate hold and wait:-

1. **By eliminating wait**:

The process specifies the resources it requires in advance so that it does not have to wait for allocation after execution starts.

For Example: Process1 declares in advance that it requires both Resource1 and Resource2

2. **By eliminating hold**:

The process has to release all resources it is currently holding before making a new request.

For Example: Process1 has to release Resource2 and Resource3 before making request for Resource1

No preemption

Preemption is temporarily interrupting an executing task and later resuming it.

For example, if a process P1 is using a resource and a high priority process P2 requests for the resource, process P1 is stopped and the resources are allocated to P2.

There are two ways to eliminate this condition by preemption:

1. If a process is holding some resources and waiting for other resources, then it should release all previously held resources and put a new request for the required resources again. The process can resume once it has all the required resources.

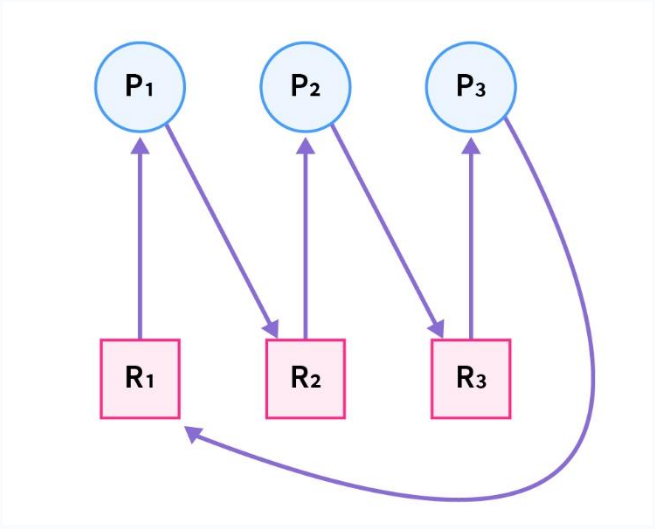
For example: If a process has resources R1, R2, and R3 and it is waiting for resource R4, then it has to release R1, R2, and R3 and put a new request of all resources again.

2. If a process P1 is waiting for some resource, and there is another process P2 that is holding that resource and is blocked waiting for some other resource. Then the resource is taken from P2 and allocated to P1. This way process P2 is preempted and it requests again for its required resources to resume the task.

The above approaches are possible for resources whose states are easily restored and saved, such as memory and registers.

Circular Wait

In circular wait, two or more processes wait for resources in a circular order. We can understand this better by the diagram given below:



To eliminate circular wait, we assign a priority to each resource. A process can only request resources in an increasing order of priority.

In the example above, process **P3** is requesting resource **R1**, which has a number lower than resource **R3** which is already allocated to process **P3**. So this request is invalid and cannot be made, as **R1** is already allocated to process **P1**.