**Case study on deadlocks bases on process in operating system**

Contributor:

Mounika s-1AT23CS402

[CSE B-3RD SEM]

**INTRODUCTION**

A process in operating system uses resources in the following way.

1. Requests a resource
2. Use the resource
3. Releases the resource

*A****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 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.

**BLOCK DIAGRAM OF DEADLOCK**



**EXAMPLES OF DEADLOCK**

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 pre-empts.
* P1 executes wait(B).
* Now P0 and P1 enter in deadlock.

| **P0** | **P1** |
| --- | --- |
| Waite(a);  Waite(b); | wait(B) ;  Waite(A); |
|  |  |

 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.

**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 Pre-emption:*** A resource cannot be taken from a process unless the process releases the resource.   
***Circular Wait:*** A set of processes waiting for each other in circular form.

**Methods for handling deadlock**   
There are three ways to handle deadlock   
**1) Deadlock prevention or avoidance:**

**Prevention:**

The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free.  
One can zoom into each category individually, Prevention is done by negating one of the above-mentioned necessary conditions for deadlock. Prevention can be done in four different ways:

       1. Eliminate mutual exclusion                                        3. Allow pre-emption

       2. Solve hold and Wait                                                   4. Circular wait Solution

**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 (Which is in turn a gift from Dijkstra) to avoid deadlock.

In prevention and avoidance, we get the correctness of data but performance decreases.  
  
**2) 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.

**Recovery from Deadlock**

1. Manual Intervention:

When a deadlock is detected, one option is to inform the operator and let them handle the situation manually

2. Automatic Recovery: an alternative to enable the system to recover from deadlock automatically

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)

Recovery from Deadlock: **Resource pre-emption**:

1. Selecting a victim:

Resource pre-emption involves choosing which resources and processes should be pre-empted 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.

2. Rollback:

If a resource is pre-empted 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:

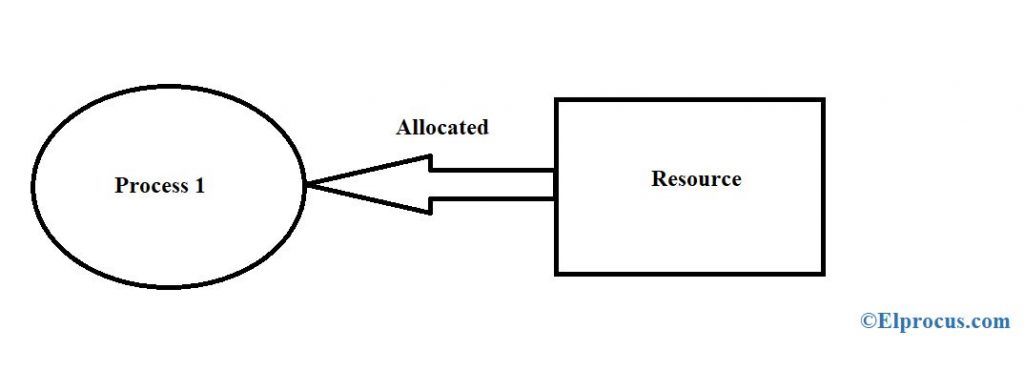
To prevent resource starvation, it is essential to ensure that the same process is not always chosen as a victim. If victim selection is solely based on cost factors, one process might repeatedly lose its resources and never complete its designated task. To address this, it is advisable to limit the number of times a process can be chosen as a victim, including the number of rollbacks in the cost factor.

**3) 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.

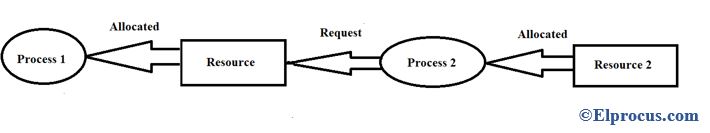
**Mutual Exclusion**

It means whatever resource we are using it must be used in a mutually exclusive way. Where only one processes use one resource at a time only.



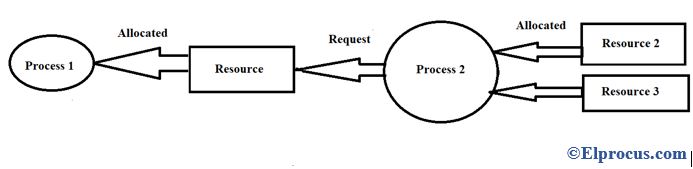
#### No Pre-emption

According to [pre-emptive](https://www.elprocus.com/real-time-operating-system-rtos-and-how-it-works/) based algorithms, if there is a priority task trying to interrupt the current task



#### Hold and Wait

A process is holding some resources and is waiting for additional resources but those resources are acquired by some other process. From the above example, P1 is holding R1 and waiting for R2, where R2



#### Circular Wait

A set of processes are said to be in deadlock if one process is waiting for a resource that is allocated to another process