gradeup

Target NIELIT 2020

A Course for Scientist 'B' Technical Assistant 'A' Aspirant

Computer Science Engineering

Operating Systems

CHAPTER 3

Deadlocks

CHAPTER 3

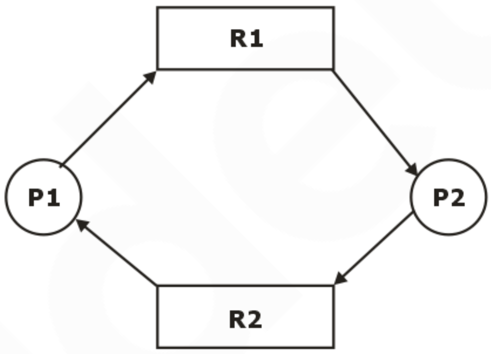
OPERATING SYSTEMS

DEADLOCK

1. INTRODUCTION DEADLOCK

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Example-



Example of a deadlock

Here

- Process P1 holds resource R1 and waits for resource R2 which is held by process P2.
- Process P2 holds resource R2 and waits for resource R1 which is held by process P1.
- None of the two processes can complete and release their resource.
- Thus, both the processes keep waiting infinitely.

2. NECESSARY CONDITIONS FOR DEADLOCKS

Mutual Exclusion

- A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

Hold and Wait

- A process waits for some resources while holding another resource at the same time.

No preemption

- The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

Circular Wait

- All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

NOTE-

- All these 4 conditions must hold simultaneously for the occurrence of deadlock.
- If any of these conditions fail, then the system can be ensured deadlock free.

3. STRATEGIES FOR HANDLING DEADLOCK

3.1. Deadlock Ignorance

- This strategy involves ignoring the concept of deadlock and assuming as if it does not exist.
- This strategy helps to avoid the extra overhead of handling deadlock.
- Windows and Linux use this strategy and it is the most widely used method.
- It is also called as **Ostrich approach**.

3.2. Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

3.2.1. Mutual Exclusion-

- To violate this condition, all the system resources must be such that they can be used in a shareable mode.
- In a system, there are always some resources which are mutually exclusive by nature.
- So, this condition cannot be violated.

3.2.2. Hold and Wait-

This condition can be violated in the following ways-

Approach-01:

In this approach,

- A process has to first request for all the resources it requires for execution.
- Once it has acquired all the resources, only then it can start its execution.
- This approach ensures that the process does not hold some resources and wait for other resources.

Drawbacks-

The drawbacks of this approach are-

- It is less efficient.
- It is not implementable since it is not possible to predict in advance which resources will be required during execution.

Approach-02:

In this approach,

- A process is allowed to acquire the resources it desires at the current moment.
- After acquiring the resources, it starts its execution.
- Now before making any new request, it has to compulsorily release all the resources that it holds currently.
- This approach is efficient and implementable.

3.2.3. No Preemption-

- This condition can by violated by forceful preemption.
- Consider a process is holding some resources and request other resources that cannot be immediately allocated to it.
- Then, by forcefully preempting the currently held resources, the condition can be violated.
- A process is allowed to forcefully preempt the resources possessed by some other process only if-
- It is a high priority process or a system process.
- The victim process is in the waiting state.

3.2.4. Circular Wait-

- This condition can be violated by not allowing the processes to wait for resources in a cyclic manner.
- To violate this condition, the following approach is followed-

Approach-

- A natural number is assigned to every resource.
- Each process is allowed to request for the resources either in only increasing or only decreasing order of the resource number.
- In case increasing order is followed, if a process requires a lesser number resource, then it must release all the resources having larger number and vice versa.
- However, this approach may cause starvation but will never lead to deadlock.

3.3. Deadlock avoidance

In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs. The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step.

- This strategy requires that every process declares its maximum requirement of each resource type in the beginning.
- The main challenge with this approach is predicting the requirement of the processes before execution.
- Banker’s Algorithm is an example of a deadlock avoidance strategy.

3.4. Deadlock detection and recovery

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.

Example on Deadlock-

A system is having 3 processes each requiring 2 units of resource R. The minimum number of units of R such that no deadlock will occur-

a) 3

b) 4

c) 5

d) 6

Answer- b

<u>Solution-</u>

In worst case,

The number of units that each process holds = One less than its maximum demand

- So, Process P1 holds 1 unit of resource R
- Process P2 holds 1 unit of resource R
- Process P3 holds 1 unit of resource R

Thus,

Maximum number of units of resource R that ensures deadlock = 1 + 1 + 1 = 3

Minimum number of units of resource R that ensures no deadlock = 3 + 1 = 4

Alternate-

Using the: Formula-: $\boldsymbol{n(x-1)+1 \leq r}$

Where, n= number of process, x=need for each process, r= total instances of resource.

"The above formula states that, the number of resources **r** must be greater than the overall need of all the processes"

Here, n= 3, number of processes given in problem

x=2, each process requires 2 unit of resource

r=? we need to find the minimum resources to ensure no deadlock occurs.

Putting the values in above formula,

$3(2-1)+1 \leq \mathrm{r}$

$3+1 \leq \mathrm{r}$

$4 \leq \mathrm{r}$

it means when r =4, there will be no deadlock.(minimum resources required)

For r =3, the system will cause deadlock.

4. BANKER'S ALGORITHM

- Banker's Algorithm is a deadlock avoidance strategy.
- It is called so because it is used in banking systems to decide whether a loan can be granted or not.

Banker's Algorithm requires-

Whenever a new process is created, it specifies the maximum number of instances of each resource type that it exactly needs.

- Banker's Algorithm is executed whenever any process puts forward the request for allocating the resources.

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.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfil the request of all processes then the state of the system is called unsafe.

5. RESOURCE ALLOCATION GRAPH

Resource Allocation Graph (RAG) is a graph that represents the state of a system pictorially.

Components Of RAG-

There are two major components of a Resource Allocation Graph-

1. Vertices
2. Edges

Vertices-

There are following types of vertices in a Resource Allocation Graph-

1. Process Vertices
2. Resource Vertices

Process Vertices-

- Process vertices represent the processes.
- They are drawn as a circle by mentioning the name of process inside the circle.

Resource Vertices-

- Resource vertices represent the resources.
- Depending on the number of instances that exists in the system, resource vertices may be single instance or multiple instance.
- They are drawn as a rectangle by mentioning the dots inside the rectangle.
- The number of dots inside the rectangle indicates the number of instances of that resource existing in the system.

Edges-

There are two types of edges in a Resource Allocation Graph-

1. Assign Edges
2. Request Edges

Assign Edges-

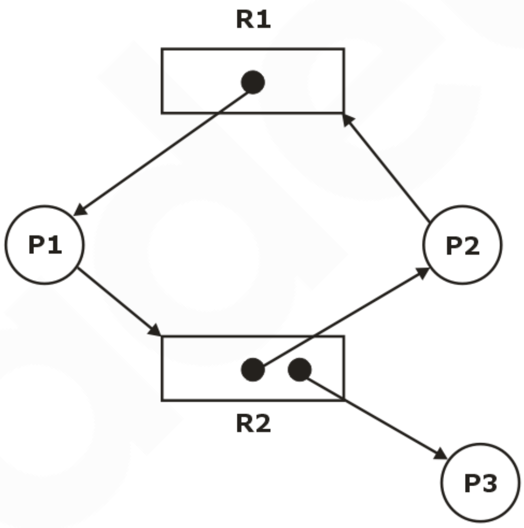
- Assign edges represent the assignment of resources to the processes.
- They are drawn as an arrow where the head of the arrow points to the process and tail of the process points to the instance of the resource.

Request Edges-

- Request edges represent the waiting state of processes for the resources.
- They are drawn as an arrow where the head of the arrow points to the instance of the resource and tail of the process points to the process.
- If a process requires 'n' instances of a resource type, then 'n' assign edges will be drawn.

Example Of RAG-

The following diagram represents a Resource Allocation Graph-



It gives the following information-

- There exist three processes in the system namely P1, P2 and P3.
- There exist two resources in the system namely R1 and R2.
- There exists a single instance of resource R1 and two instances of resource R2.
- Process P1 holds one instance of resource R1 and is waiting for an instance of resource R2.
- Process P2 holds one instance of resource R2 and is waiting for an instance of resource R1.
- Process P3 holds one instance of resource R2 and is not waiting for anything.

6. DEADLOCK DETECTION USING RAG

Using Resource Allocation Graph, it can be easily detected whether system is in a **Deadlock** state or not.

The rules are-

Rule-01:

In a Resource Allocation Graph where all the resources are single instance,

- If a cycle is being formed, then system is in a deadlock state.
- If no cycle is being formed, then system is not in a deadlock state.

Rule-02:

In a Resource Allocation Graph where all the resources are **NOT** single instance,

- If a cycle is being formed, then system may be in a deadlock state.
- **Banker's Algorithm** is applied to confirm whether system is in a deadlock state or not.
- If no cycle is being formed, then system is not in a deadlock state.
- Presence of a cycle is a necessary but not a sufficient condition for the occurrence of deadlock.

7. DEADLOCK DETECTION AND RECOVERY

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore, the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.

In order to recover the system from deadlocks, either OS considers resources or processes.

For Resource

- **Preempt the resource**

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

- **Rollback to a safe state**

System passes through various states to get into the deadlock state. The operating system can roll back the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

For Process

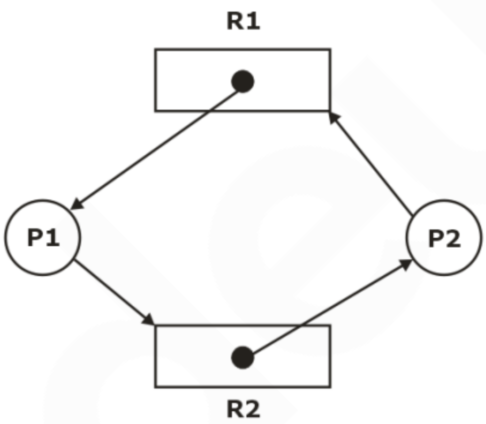
- **Kill a process**

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

- **Kill all process**

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

Example- Consider the resource allocation graph in the figure-



Find if the system is in a deadlock state otherwise find a safe sequence.

Solution-

The given resource allocation graph is single instance with a cycle contained in it.

Thus, the system is definitely in a deadlock state.

8. THREADS IN OS

A thread is a flow of execution through the process code.

- A thread is also called a **lightweight process**.
- Threads provide a way to improve application performance through parallelism.
- Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- Each thread belongs to exactly one process and no thread can exist outside a process.

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

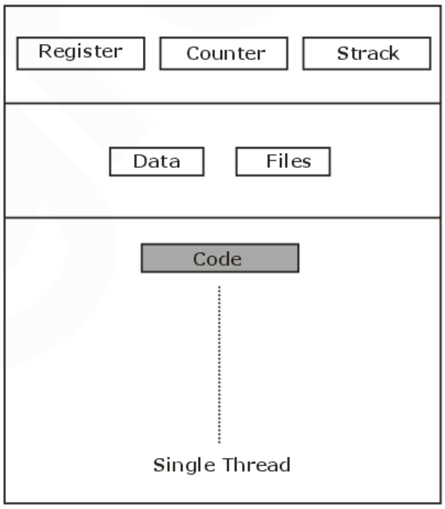
8.1. Difference between Process and Thread

8.2. Advantages of Threads

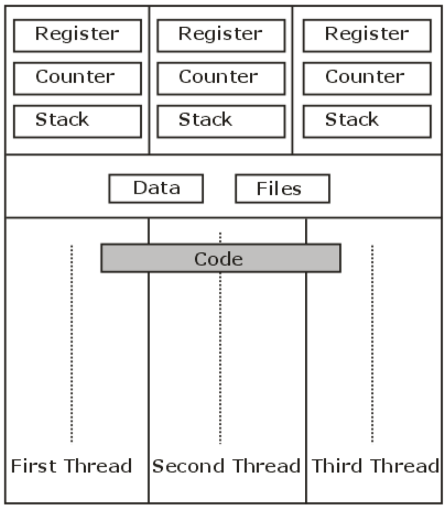
- **Responsiveness**: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
- **Faster context switch**: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
- **Effective utilization of multiprocessor system**: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.
- **Resource sharing:** Resources like code, data, and files can be shared among all threads within a process.

Note-1: stack, register, and Counter can't be shared among the threads. Each thread has its own stack and registers.

Note-2: A thread shares with its peer threads few information like code segment, data segment and open files.



Single Process P with single thread



Single Process P with three threads

- **Communication:** Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.
- **Enhanced throughput of the system**: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

9. TYPES OF THREAD

Threads are implemented in following two ways –

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

9.1. User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.

9.2. Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

9.3. Difference between User Level thread and Kernel Level thread

USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread performs blocking operation then another thread can continue execution.
User level threads are designed as dependent threads.	Kernel level threads are designed as independent threads.

****