Banker's Algorithm in Operating System

It is a 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

Following are the essential characteristics of the Banker's algorithm:

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.

Following are the important data structures terms applied in the banker's algorithm as follows:

Suppose n is the number of processes, and m is the number of each type of resource used in a computer system.

1. **Available**: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.
2. **Max:** It is a [n x m] matrix that indicates each process P[i] can store the maximum number of resources R[j] (each type) in a system.

3. **Allocation:** It is a matrix of m x n orders that indicates the type of resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j] in the system.
4. **Need:** It is an M x N matrix sequence representing the number of remaining resources for each process. When the Need[i] [j] = k, then process P[i] may require K more instances of resources type Rj to complete the assigned work. Nedd[i][j] = Max[i][j] - Allocation[i][j].
5. **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system:

Safety Algorithm

It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm:

1. There are two vectors **Wok** and **Finish** of length m and n in a safety algorithm.

Initialize: Work = Available
Finish[i] = false; for I = 0, 1, 2, 3, 4… n - 1.

2. Check the availability status for each type of resources [i], such as:

Need[i] <= Work
Finish[i] == false
If the i does not exist, go to step 4.

3. Work = Work +Allocation(i) // to get new resource allocation

Finish[i] = true

Go to step 2 to check the status of resource availability for the next process.

4. If Finish[i] == true; it means that the system is safe for all processes.

Resource Request Algorithm

A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.

Let create a resource request array R[i] for each process P[i]. If the Resource $Request_i$ [j] equal to 'K', which means the process P[i] requires 'k' instances of Resources type R[j] in the system.

1. When the number of **requested resources** of each type is less than the **Need** resources, go to step 2 and if the condition fails, which means that the process P[i] exceeds its maximum claim for the resource. As the expression suggests:

If Request(i) <= Need
Go to step 2;

2. And when the number of requested resources of each type is less than the available resource for each process, go to step (3). As the expression suggests:

If Request(i) <= Available
Else Process P[i] must wait for the resource since it is not available for use.

3. When the requested resource is allocated to the process by changing state:

Available = Available - Request
Allocation(i) = Allocation(i) + Request (i)
$Need_i = Need_i - Request_i$

When the resource allocation state is safe, its resources are allocated to the process P(i). And if the new state is unsafe, the Process P (i) has to wait for each type of Request R(i) and restore the old resource-allocation state.

Example: Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	**Allocation**			**Max**			**Available**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			

P3	3	0	2	9	0	2	
P4	2	1	1	2	2	2	
P5	0	0	2	4	3	3	

Answer the following questions using the banker's algorithm:

1. What is the reference of the need matrix?
2. Determine if the system is safe or not.
3. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

Ans. 2: Context of the need matrix is as follows:

Need [i] = Max [i] - Allocation [i]
Need for P1: (7, 5, 3) - (0, 1, 0) = 7, 4, 3
Need for P2: (3, 2, 2) - (2, 0, 0) = 1, 2, 2
Need for P3: (9, 0, 2) - (3, 0, 2) = 6, 0, 0
Need for P4: (2, 2, 2) - (2, 1, 1) = 0, 1, 1
Need for P5: (4, 3, 3) - (0, 0, 2) = 4, 3, 1

Process	**Need**		
	A	**B**	**C**
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Hence, we created the context of need matrix.

Ans. 2: Apply the Banker's Algorithm:

Available Resources of A, B and C are 3, 3, and 2.

Now we check if each type of resource request is available for each process.

Step 1: For Process P1:

Need <= Available

7, 4, 3 <= 3, 3, 2 condition is **false**.

So, we examine another process, P2.

Step 2: For Process P2:

Need <= Available

1, 2, 2 <= 3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) => 5, 3, 2

Similarly, we examine another process P3.

Step 3: For Process P3:

P3 Need <= Available

6, 0, 0 < = 5, 3, 2 condition is **false**.

Similarly, we examine another process, P4.

Step 4: For Process P4:

P4 Need <= Available

0, 1, 1 <= 5, 3, 2 condition is **true**

New Available resource = Available + Allocation

5, 3, 2 + 2, 1, 1 => 7, 4, 3

Similarly, we examine another process P5.

Step 5: For Process P5:

P5 Need <= Available

4, 3, 1 <= 7, 4, 3 condition is **true**

New available resource = Available + Allocation

7, 4, 3 + 0, 0, 2 => 7, 4, 5

Now, we again examine each type of resource request for processes P1 and P3.

Step 6: For Process P1:

P1 Need <= Available

7, 4, 3 <= 7, 4, 5 condition is **true**

New Available Resource = Available + Allocation

7, 4, 5 + 0, 1, 0 => 7, 5, 5

So, we examine another process P2.

Step 7: For Process P3:

P3 Need <= Available

6, 0, 0 <= 7, 5, 5 condition is true

New Available Resource = Available + Allocation

7, 5, 5 + 3, 0, 2 => 10, 5, 7

Hence, we execute the banker's algorithm to find the safe state and the safe sequence like P2, P4, P5, P1 and P3.

Ans. 3: For granting the Request (1, 0, 2), first we have to check that **Request <= Available**, that is (1, 0, 2) <= (3, 3, 2), since the condition is true. So the process P1 gets the request immediately.