

Banker's Algorithm

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S . Every time a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than S . It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

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

Characteristics of Banker's Algorithm

The characteristics of Banker's algorithm are as follows:

- If any process requests for a resource, then it has to wait.
- This algorithm consists of advanced features for maximum resource allocation.
- There are limited resources in the system we have.
- In this algorithm, if any process gets all the needed resources, then it is that it should return the resources in a restricted period.
- Various resources are maintained in this algorithm that can fulfill the needs of at least one client.

Let us assume that there are n processes and m resource types.

Data Structures used to implement the Banker's Algorithm

Some data structures that are used to implement the banker's algorithm are:

1. Available

It is an array of length m . It represents the number of available resources of each type. If $Available[j] = k$, then there are k instances available, of resource type R_j .

2. Max

It is an $n \times m$ matrix which represents the maximum number of instances of each resource that a process can request. If $Max[i][j] = k$, then the process P_i can request atmost k instances of resource type R_j .

3. Allocation

It is an $n \times m$ matrix which represents the number of resources of each type currently allocated to each process. If $Allocation[i][j] = k$, then process P_i is currently allocated k instances of resource type R_j .

4. Need

It is a two-dimensional array. It is an $n \times m$ matrix which indicates the remaining resource needs of each process. If $Need[i][j] = k$, then process P_i may need k more instances of resource type R_j to complete its task

$$Need[i][j] = Max[i][j] - Allocation[i][j]$$

Banker's algorithm comprises of two algorithms:

1. Safety algorithm
2. Resource request algorithm

Safety Algorithm

A safety algorithm is an algorithm used to find whether or not a system is in its safe state. The algorithm is as follows:

1. Let **Work** and **Finish** be vectors of length **m** and **n**, respectively. Initially,

```
Work = Available  
Finish[i] = false for i = 0, 1, ..., n - 1.
```

This means, initially, no process has finished and the number of available resources is represented by the **Available** array.

2. Find an index **i** such that both

```
Finish[i] == false  
Needi <= Work
```

If there is no such **i** present, then proceed to step 4.

It means, we need to find an unfinished process whose needs can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:

```
Work = Work + Allocationi  
Finish[i] = true
```

Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If `Finish[i] == true` for all i , then the system is in a safe state.

That means if all processes are finished, then the system is in safe state.

This algorithm may require an order of $m \times n^2$ operations in order to determine whether a state is safe or not.

Resource Request Algorithm

Now the next algorithm is a resource-request algorithm and it is mainly used to determine whether requests can be safely granted or not.

Let $Request_i$ be the request vector for the process P_i . If $Request_i[j] = k$, then process P_i wants k instance of Resource type R_j . When a request for resources is made by the process P_i , the following are the actions that will be taken:

1. If $Request_i \leq Need_i$, then go to step 2; else raise an error condition, since the process has exceeded its maximum claim.

2. If $Request_i \leq Available_i$ then go to step 3; else P_i must have to wait as resources are not available.

3. Now we will assume that resources are assigned to process P_i and thus perform the following steps:

$Available = Available - Request_i$;

$Allocation_i = Allocation_i + Request_i$;

$Need_i = Need_i - Request_i$;

If the resulting resource allocation state comes out to be safe, then the transaction is completed and, process P_i is allocated its resources. But in this case, if the new state is unsafe, then P_i waits for $Request_i$, and the old resource-allocation state is restored.

Disadvantages of Banker's Algorithm

Some disadvantages of this algorithm are as follows:

1. During the time of Processing, this algorithm does not permit a process to change its maximum need.
2. Another disadvantage of this algorithm is that all the processes must know in advance about the maximum resource needs.
3. This algorithm permits the requests to be provided in constrained time, but for one year which is a fixed period.