**Assignment: Bankers Algorithm**

**Title :** Write a program to implement Banker’s Algorithm

**Theory:**

**Deadlock:**

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

**Necessary Conditions for Deadlock:**

* Mutual Exclusion: Only One Process at a time can use a resource.
* Hold and Wait: Processes hold resources already allocated to them while waiting for additional resources.
* No Preemption: Resources cannot be forcibly removed from the process holding them until the resources are used to completion.
* Circular Wait: A circular chain of processes exists such that each process holds one or more resources that are being requested by the next process in the chain.

**Ways of Handling Deadlock:**

* **Deadlock Prevention**

Remove the possibility of deadlock occurring by denying one of the four necessary conditions:

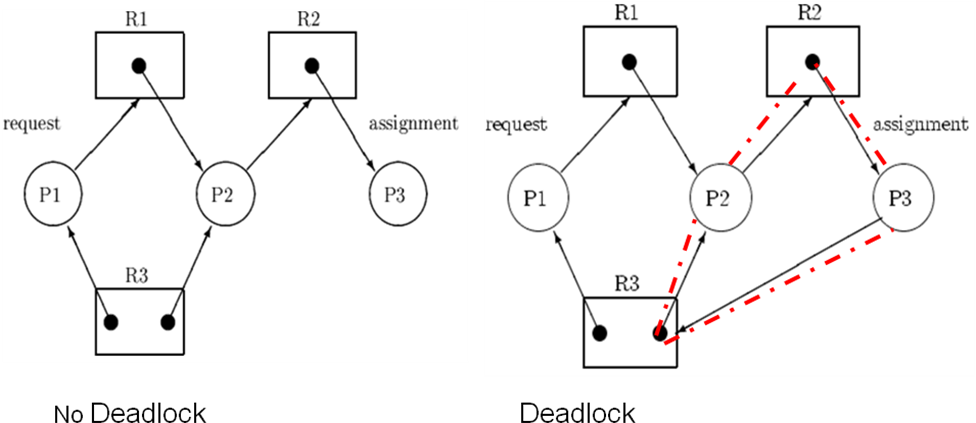
* + Mutual Exclusion
  + Hold & Wait
  + No preemption
  + Circular Wait
* **Deadlock Avoidance**

By giving prior information about the maximum number of resources of each type that may be requested for each process, ensures that system will never enter in deadlock state.

Deadlock Avoidance Algorithm:

* Resource Allocation Graph Algorithm
* Bankers Algorithm
* **Deadlock Detection**

Methods by which the occurrence of deadlock, the processes and resources involved are detected. One method is resource allocation graphs



* **Deadlock Recovery**
* Deadlock can be recovered by one of the two methods:
* Process Termination:
  + Abort all deadlocked processes
  + Abort one process at a time until the deadlock cycle is eliminated
* Resource Preemption:-
  + Preempt some resources from processes and give these resources to other processes and give these resources to other processes until the deadlock cycle is broken.
* **Concepts in Deadlock Avoidance**
* **Safe State:** A State is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.
* **Safe Sequence:** A sequence of processes <P1, P2, ….,Pn> is a safe sequence for the current allocation state if, for each Pi, the resources that Pi can still request can be satisfied by the currently available resources plus the resources held by all the Pj, with j < i.
* If a system is in a safe state, there is no deadlock

**Input :**

1. No of Processes n
2. No. of Resources m
3. No. of Instances of each Resource res[m]
4. Maximum Matrix max[n,m]
5. Allocation Matrix alloc[n,m]

**Output :** Sample Output in following format

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Process** | **Allocation** | | | **MAX** | | | **Available** | | | **Need** | | |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | 7 | 4 | 3 |
| P1 | 2 | 0 | 0 | 3 | 2 | 2 |  |  |  | 1 | 2 | 2 |
| P2 | 3 | 0 | 2 | 9 | 0 | 2 |  |  |  | 6 | 0 | 0 |
| P3 | 2 | 1 | 1 | 2 | 2 | 2 |  |  |  | 0 | 1 | 1 |
| P4 | 0 | 0 | 2 | 4 | 3 | 3 |  |  |  | 4 | 3 | 1 |

Safe SequenceP1, P3, P4, P0, P2

Display multiple safe sequences.

Accept Resource Request for allocation as below:

Request by P1 1 0 2 Display Safe sequence

Request by P1 3 3 0 Request not granted.

Request by P0 0 2 0 No Safe State

**Note:**

If System is in safe state then Display New State of matrices

Else Display Original state of matrices and Error message Request not granted or System is not in safe state.

**Banker Implementation- Main Flow**

1. Accept number of processes n, number of Resources m.
2. Accept instances of each Resources type, res[m].

Accept Maximum need of Each Process, maximum[n, m].

1. Accept Resource Allocation of Each Process, Allocation [n, m].
2. Calculate Need of each Process, Need [n, m].

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

1. Calculate Available instances of each Resource.
   1. Add all row elements of Allocation matrix for each column in temp.
   2. Available= res[i] – temp.
2. Display Process Number, Allocation, Maximum, Need and availability of each Resource type.
3. Check Safety state and generate safe sequence by calling **SafeState** function.
4. IF system is in safe state then,
   1. Accept answer whether any process requesting for resource allocation.
   2. Accept request for resource allocation for particular Process.
   3. Accept requesting Process No. and request for each Resource type.
   4. Check whether system is granting the resources for process and is in safe state by calling **Resource Allocation** function.

Else

Display “Cannot allocate Resources to any Process as system not in safe state.”

**Safe Sate Algorithm**

1. Let Work & Finish be vectors of length m and n. Initialize Work := Available & Finish[i]:= false for i =1,2..,n.
2. If process i exist such that both (Finish [i] = false and Need\_i ≤ Work), Then

1. Work := Work + Allocation

2. Finish [i] := true

3. Add Process i in safeSequence List.

Else go to step 3.

If Finish [i] =true for all i, Then

Display “The system is in a safe state and print safe sequence”.

Else

Display “The system is Not in safe state.”

**Resource Request Allocation Algorithm**

1. If Request\_i ≤ Need\_i go to step2. Otherwise raise an error “Request by Process cannot be granted! Resources not available”.
2. If Request\_i ≤ Available, go to step 3. Otherwise, raise error since the resources are not available.
3. Pretend to allocate requested resources by modifying the state as:

Available: = Available – Request\_i;

Allocation\_i := Allocation\_i + Request\_i;

Need\_i := Need\_i - Request\_i;

1. Check safety of state.
2. If State is safe then,

The resources are allocated.

Display new state of the system.

Else

Cancel the tentative allocation and save original safe state as

Available := Available + Request\_i;

Allocation\_i := Allocation\_i - Request\_i;

Need\_i := Need\_i + Request\_i;

Display Original state of the system.