**Deadlock Handling**

**2- Banker’s algorithm:**

There are multiple instances. Each process must a priori knowledge claim maximum use. When a process requests a resource it may have to wait. When a process gets all its resources it must return them in a finite amount of time.

**Data structures for the “banker’s algorithm”:**

Processes P [n] = [ P1,…,Pi,….Pn ];

Resources R [m] = [ R1,..,Rj, …, Rm];

**Notes:**

* Each of the ***n*** processes has got a maximum demand for instances of each of the **m** existing resources
* You can consider rj as a **number of instances** for each ith process pi.
* All resources are either allocated or available.

**Banker’s algorithm (Safety algorithm):**

* **Available***:* Vector of length *m*. If ***available [j]*** *=* ***k***, there are***k* instances** of resource type *Rj*available.
* ***Max****: n x m* matrix. If *Max [i,j]* = *k*, then process *Pi* may request at most *k* instances of resource type *Rj*.
* **Allocation***: n* x *m* matrix. If *Allocation[i,j] = k* then *Pi* is currently allocated *k* instances of *Rj.*
* **Need***: n* x *m* matrix. If *Need[i,j]* = *k*, then *Pi* may need *k* more instances of *Rj*to complete its task.

***Or*** *:Need* *[i,j]* = *Max[i,j]* – *Allocation* *[i,j].*

**Steps of Banker’s algorithm**

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:

***Work* = *Available***

***Finish [i]* = *false for* i = 1,2, …, n*.***

1. Find an i such that both:

***(a)* *Finish* *[i]* = *false (b)* *Needi*** ≤ ***Work***

**If** no such *i* exists, **go to step 4.**

1. Else: Give & take
2. Work = Work + Allocationi
3. Finish[i] = true

go to step 2.

1. If Finish [i] == true for all i,

🡪 Safe state;

**Otherwise**,

🡪 Unsafe state;

**Remarks**:

In Banker’s Algorithm, allocate resources to processes so that everyone always arrives at a **safe state**

A **safe state**: state in which there exists at least one sequence of allocation of resource instances to processes that does not result in deadlock.

**Example-1: banker’s algorithm**

* 5 processes *P*0 through *P*4; 3 resource types *A*   
  (10 instances), B (5 instances), and *C* (7 instances).
* Snapshot at time *T*0:

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Allocation* | *Need* | *Available* |
|  | *A B C* | *A B C* | *A B C* |
| *P*0 | *0 1 0* | *7 5 3* | *3 3 2* |
| *P*1 | *2 0 0* | *3 2 2* |  |
| *P*2 | *3 0 2* | *9 0 2* |  |
| *P*3 | *2 1 1* | *2 2 2* |  |
| *P*4 | *0 0 2* | *4 3 3* |  |

* The content of the matrix Need is defined to be (Max – Allocation).

|  |  |
| --- | --- |
|  | *Need* |
|  | *A B C* |
| *P*0 | *7 4 3* |
| *P*1 | *1 2 2* |
| *P*2 | *6 0 0* |
| *P*3 | *0 1 1* |
| *P*4 | *4 3 1* |

**Solution:**

* 332 ------Available-a0
* 122 ------Need-p1
* 210 ------Remain
* 322 ------MAX p1
* 532 ------Available-a1
* 011 ------Need-p3
* 521 ------Remain
* 222 ------MAX-p3
* 743 ------Available-a2
* 743 ------Available-a2
* 431 ------Need-P4
* 312 ------Remain
* 433 ------MAX-P4
* 745 ------Available-a3
* 743 ------Need-P0
* 002 ------Remain
* 753 ------MAX-P0
* 755 ------Available-a4
* 600 ------Need-P2
* 155 ------Remain
* 902 ------MAX-P2
* 10 5 7 ------Available-Final= Original A B C

**Conclusion:**

The system is in a **safe state** since the sequence **< *P*1, *P*3, *P*4, *P0*, *P2*>** satisfies safety criteria

**EXAMPLE-2: banker’s algorithm**

* Example : If *P*1 Request (1,0,2)
* Check that *Request ≤ Available* (that is, (1,0,2) ≤ (3,3,2) ⇒ *true.*

|  |  |  |  |
| --- | --- | --- | --- |
|  | *Allocation* | *Need* | *Available* |
|  | *A B C* | *A B C* | *A B C* |
| *P*0 | 0 1 0 | 0 1 0 | 3 3 2 |
| *P*1 | 2 0 0 | 1 2 2 |  |
| *P*2 | 3 0 2 | 6 0 0 |  |
| *P*3 | 2 1 1 | 0 1 1 |  |
| *P*4 | 0 0 2 | 4 3 1 |  |

* Compute Max of requests?
* Executing safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement
* Can request for (3,3,0) by P4 be granted? H.W
* Can request for (0,5,0) by P0 be granted? H.W