

Dr. Vishwanath Karad

MIT WORLD PEACE

UNIVERSITY PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

### Banker's Algorithm for Deadlock Avoidance



### References

1. Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Operating System Concepts, WILEY,ISBN 978-1-118-06333-0, 9th Edition

### Banker's Algorithm

#### Aim:

To implement Bankers Algorithm for deadlock avoidance using

- (a) Safety algorithm
- (b) resource request algorithm

#### **Objectives:**

To understand the concept of deadlock.

How to avoid deadlock by implementing safety algorithm.

### Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

**Available**: Vector of length m. If available [j] = k, there are k instances of resource type  $R_i$  available.

**Max**:  $n \times m$  matrix. If Max[i,j] = k, then process  $P_i$  may request at most k instances of resource type  $R_i$ .

**Allocation**:  $n \times m$  matrix. If Allocation[i,j] = k then  $P_i$  is currently allocated k instances of  $R_{j}$ .

**Need**:  $n \times m$  matrix. If Need[i,j] = k, then  $P_i$  may need k more instances of  $R_j$  to complete its task.

Need [i,j] = Max[i,j] - Allocation <math>[i,j].

### **Safety Algorithm**

To determine whether system is in safe state

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

```
Work := Available
Finish [i] = false for i = 1,2,...,n.
```

- 2. Find an *i* such that both:
  - (a) Finish [i] = false
  - (b) Need<sub>i</sub> ≤ WorkIf no such i exists, go to step 4.
- 3. Work := Work + Allocation; Finish[i] := true go to step 2.
- 4. If Finish [i] = true for all i, then the system is in a safe state.

### **Resource-Request Algorithm**

 $Request_i$  = request vector for process  $P_i$ .

If  $Request_{i}[j] = k$  then process  $P_{i}$  wants k instances of resource type  $R_{i}$ .

- 1. If  $Request_i \leq Need_i$  go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
- 2. If  $Request_i \le Available$ , go to step 3. Otherwise  $P_i$  must wait, since resources are not available.
- 3. Pretend to allocate requested resources to  $P_i$  by modifying the state as follows:

Available := Available - Request;

Allocation; := Allocation; + Request;;

 $Need_i := Need_i - Request_{i;i}$ 

- Call safety algorithm
- If safe  $\Rightarrow$  the resources are allocated to P<sub>i</sub>.
- If unsafe  $\Rightarrow$  P, must wait, and the old resource-allocation state is restored

# Example of 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 ·

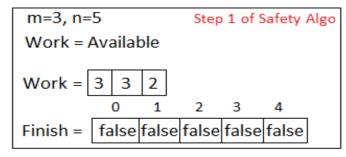
	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	ABC	ABC	ABC
$P_0$	010	753	332
$P_1$	200	322	
$P_2$	302	902	
$P_3$	211	222	
$P_4$	002	433	

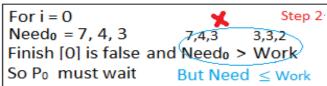
# Example (Cont.)

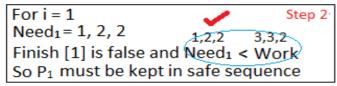
The content of the matrix. Need is defined to be Max - Allocation.

	Need	
	ABC	
$P_{0}$	7 4 3	
$P_1$	122	
$P_2$	600	
$P_3$	011	
$P_4$	431	

The system is in a safe state since the sequence  $\langle P_1, P_3, P_4, P_0, P_2 \rangle$  or  $\langle P_1, P_3, P_4, P_2, P_0 \rangle$  satisfies safety criteria.







$$3, 3, 2 \quad 2, 0, 0 \quad \text{Step 3}$$

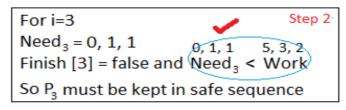
$$Work = Work + Allocation_1$$

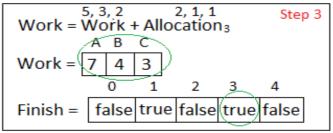
$$Work = 5 \quad 3 \quad 2$$

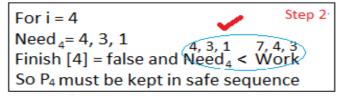
$$0 \quad 1 \quad 2 \quad 3 \quad 4$$

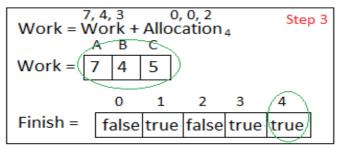
$$Finish = false true false false false$$

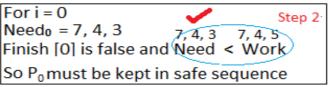
For 
$$i = 2$$
Need<sub>2</sub> = 6, 0, 0
Finish [2] is false and Need<sub>2</sub> > Work
So P<sub>2</sub> must wait

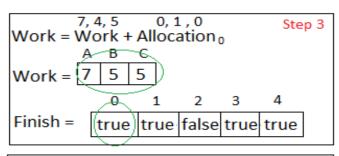


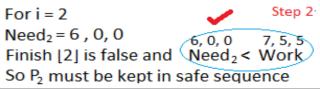


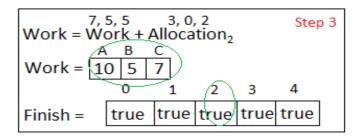












Finish [i] = true for  $0 \le i \le n$  Step 4 Hence the system is in Safe state

The safe sequence is P<sub>1</sub>,P<sub>3</sub>, P<sub>4</sub>,P<sub>0</sub>,P<sub>2</sub>

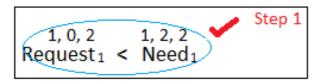
# Example (Cont.).....

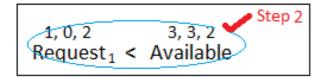
At time  $T_1$ :

 $P_1$  request (1,0,2)

# Resource-Request algorithm

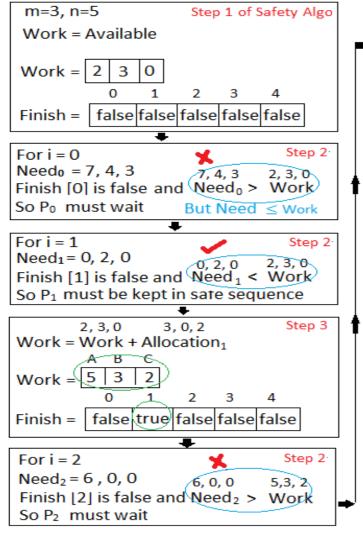
To decide whether the request is granted we use Resource Request algorithm

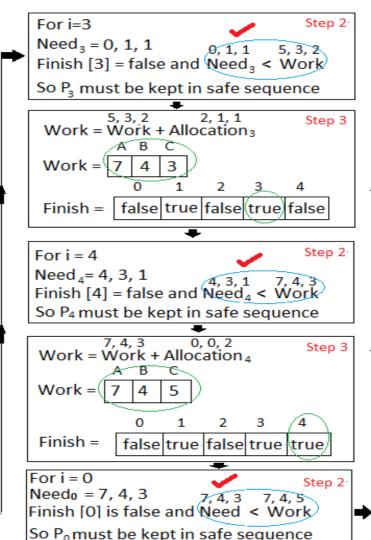


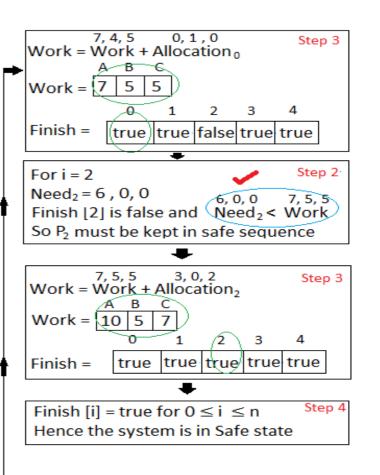


Available = Available - Request <sub>1</sub> Allocation <sub>1</sub> = Allocation <sub>1</sub> + Request <sub>1</sub> Need <sub>1</sub> = Need <sub>1</sub> - Request <sub>1</sub>					
Process	Allocation	Need	Available		
	АВС	A В С	A B C		
P <sub>0</sub>	0 1 0	7 4 3	2 3 0		
P <sub>1</sub>	(3 0 2 )	0 2 0			
P <sub>2</sub>	3 0 2	6 0 0			
P <sub>3</sub>	2 1 1	0 1 1			
P <sub>4</sub>	0 0 2	4 3 1			

Sten 3







The safe sequence is P<sub>1</sub>,P<sub>3</sub>, P<sub>4</sub>,P<sub>0</sub>,P<sub>2</sub>

## Example (Cont.): $P_1$ request (1,0,2)

Check that Request  $\leq$  Available (that is,  $(1,0,2) \leq (3,3,2) \Rightarrow true$ .

	Allocation	Need	Available
	ABC	ABC	ABC
$P_0$	010	743	230
$P_1$	302	020	
$P_2$	302	600	
$P_3$	211	011	
$P_4$	002	431	

Executing safety algorithm shows that sequence  $\langle P_1, P_3, P_4, P_0, P_2 \rangle$  satisfies safety requirement.

### **Example (Cont.)**

At time  $T_2$ : Can request for (3,3,0) by  $P_4$  be granted?

At time  $T_3$ : Can request for (0,2,0) by  $P_0$  be granted?

# Steps for implementation

- 1. Accept no of processes
- 2. Accept no of resource types
- 3. Accept available, max, allocation matrices
- 4. Determine need matrix
- 5. Invoke safety algorithm and display the safety sequence
- 6. If system is in safe state, accept resource request from user
- 7. Invoke resource-request algorithm
- 8. Display whether request can be granted and display safe sequence if any

# Steps for implementation

- 1. Accept no of processes
- 2. Accept no of resource types
- 3. Accept available, max, allocation matrices
- 4. Determine need matrix
- 5. Invoke safety algorithm
- 6. If safe, accept resource request from user
- 7. Invoke resource-request algorithm
- 8. Display whether request can be granted and display safe sequence if any

# output

Following is the SAFE Sequence

P1 -> P3 -> P4 -> P0 -> P2