

DEADLOCK AVOIDANCE



TEAM MEMBERS

1. Atul Sahu (200905090)
2. Vikyath Shetty B (200905156)
3. Bhavya Kumar Thakur (200905149)

INTRODUCTION

This project is about avoiding the deadlock using Java programming language with the help of well-known concepts in Operating Systems.

We have taken Banker's algorithm and Resource Request algorithm here to detect deadlocks and allocation of resources to avoid deadlocks.

Banker's Algorithm

- Banker's algorithm deals with various concepts like safe sequence. When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources. Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. We need the following data structures, where **n** is the number of processes in the system and **m** is the number of resource types:
 - **Available:** A vector of length **m** indicates the number of available resources of each type. If **Available[j]** equals **k**, then **k** instances of resource type **R_j** are available.

- **Max:** An $n \times m$ matrix defines the maximum demand of each process. If $\text{Max}[i][j]$ equals k , then process P_i may request at most k instances of resource type R_j .
- **Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process. If $\text{Allocation}[i][j]$ equals k , then process P_i is currently allocated k instances of resource type R_j .
- **Need:** An $n \times m$ matrix indicates the remaining resource need of each process. If $\text{Need}[i][j]$ equals k , then process P_i may need k more instances of resource type R_j to complete its task. Note that $\text{Need}[i][j]$ equals to $\text{Max}[i][j] - \text{Allocation}[i][j]$

2. Resource-Request Algorithm

This algorithm for determining whether requests can be safely granted.

Let **Request i** be the request vector for process P_i . If **Request i** $[j] == k$, then process P_i wants k instances of resource type R_j . When a request for resources is made by process P_i , the following actions are taken:

1. If **Request i** \leq **Need i**, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If **Request i** \leq **Available**, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

Available = Available - Request[i];

Allocation[i] = Allocation[i] + Request[i];

Need[i] = Need[i] - Request[i];

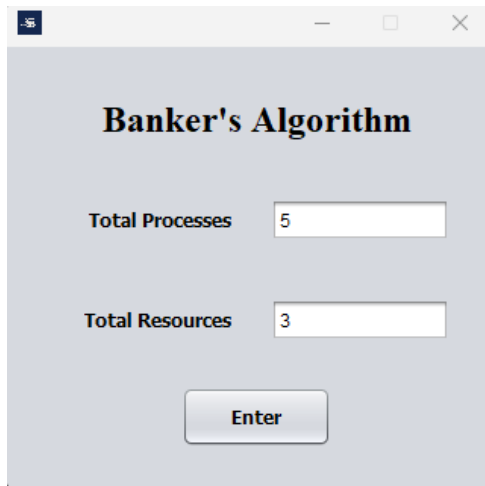
If the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for **Request[i]**, and the old resource-allocation state is restored.

References

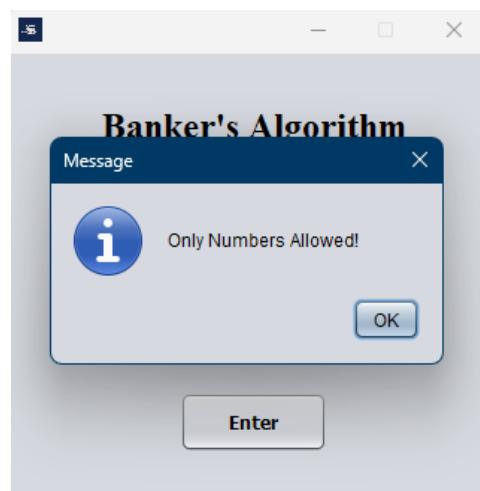
It is taken from example of the book "Abraham-Silberschatz-Operating-System-Concepts 9th edition" on page no #332.

Program Demonstration & Outcome

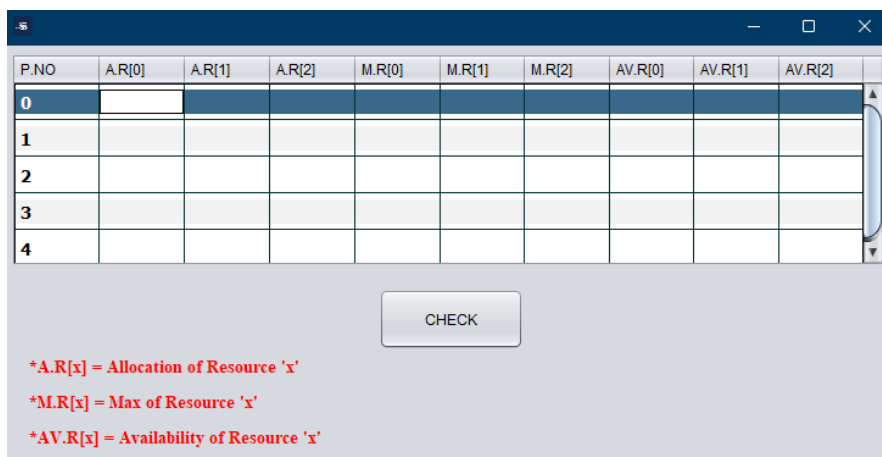
Step – 01: Execute the program and enter the data. In this program we must provide the numbers/digits. This program restricts the unwanted type of data, example: String, Characters.



A window titled "Banker's Algorithm" with two input fields. The first field is labeled "Total Processes" and contains the value "5". The second field is labeled "Total Resources" and contains the value "3". Below the fields is a button labeled "Enter".



A window titled "Banker's Algorithm" with a message box overlay. The message box has a blue information icon and the text "Only Numbers Allowed!". There is an "OK" button. Below the message box is a button labeled "Enter".

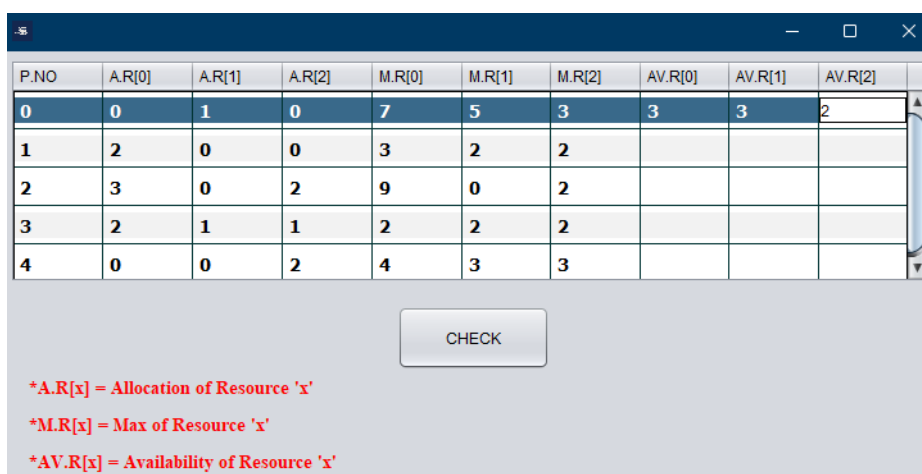


A window titled "Banker's Algorithm" showing a table for entering data. The table has 10 columns: P.NO, A.R[0], A.R[1], A.R[2], M.R[0], M.R[1], M.R[2], AV.R[0], AV.R[1], AV.R[2]. The rows are numbered 0 to 4. Below the table is a button labeled "CHECK".

P.NO	A.R[0]	A.R[1]	A.R[2]	M.R[0]	M.R[1]	M.R[2]	AV.R[0]	AV.R[1]	AV.R[2]
0									
1									
2									
3									
4									

*A.R[x] = Allocation of Resource 'x'
*M.R[x] = Max of Resource 'x'
*AV.R[x] = Availability of Resource 'x'

Now we will enter the data into this table.

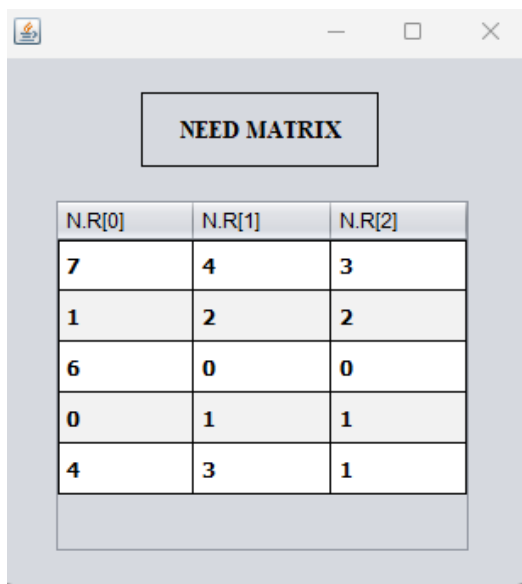


A window titled "Banker's Algorithm" showing the same table as before, but now with data entered. The rows are numbered 0 to 4. Below the table is a button labeled "CHECK".

P.NO	A.R[0]	A.R[1]	A.R[2]	M.R[0]	M.R[1]	M.R[2]	AV.R[0]	AV.R[1]	AV.R[2]
0	0	1	0	7	5	3	3	3	2
1	2	0	0	3	2	2			
2	3	0	2	9	0	2			
3	2	1	1	2	2	2			
4	0	0	2	4	3	3			

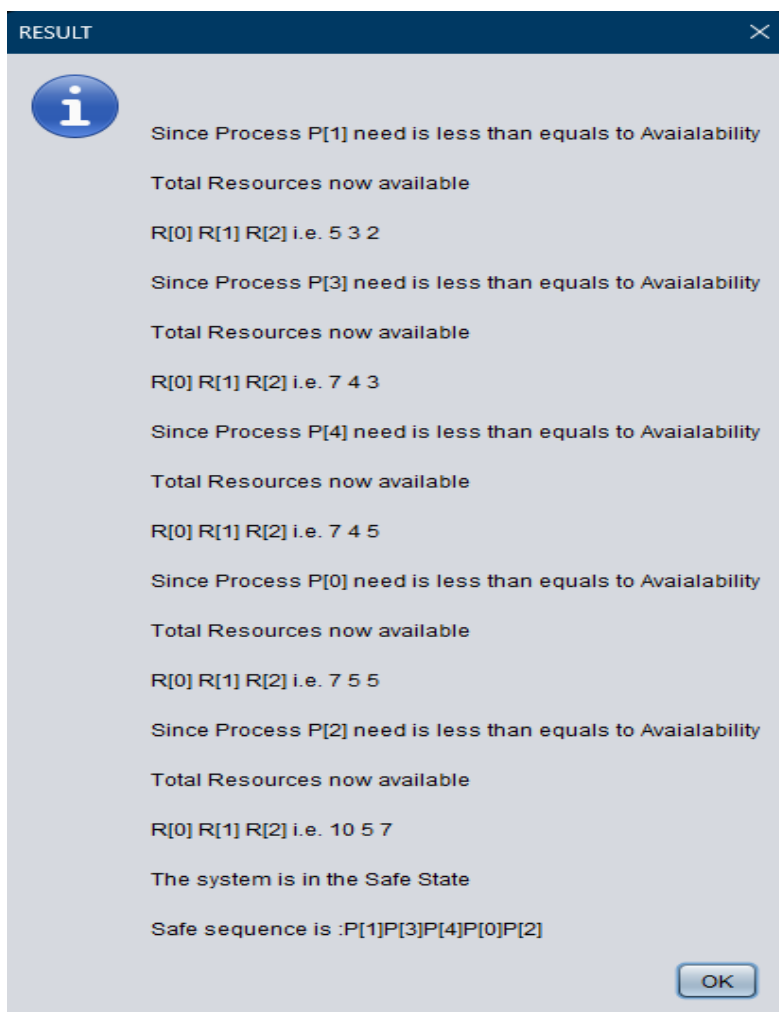
*A.R[x] = Allocation of Resource 'x'
*M.R[x] = Max of Resource 'x'
*AV.R[x] = Availability of Resource 'x'

Step – 03: After clicking the check button, the program will generate a need matrix.



N.R[0]	N.R[1]	N.R[2]
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Step – 04: After generating the need matrix, the program will apply the Banker's algorithm and check the system state. As we can see in Fig, the system is in the safe state with the safe sequence.



RESULT

i

Since Process P[1] need is less than equals to Avaialability
Total Resources now available
R[0] R[1] R[2] i.e. 5 3 2

Since Process P[3] need is less than equals to Avaialability
Total Resources now available
R[0] R[1] R[2] i.e. 7 4 3

Since Process P[4] need is less than equals to Avaialability
Total Resources now available
R[0] R[1] R[2] i.e. 7 4 5

Since Process P[0] need is less than equals to Avaialability
Total Resources now available
R[0] R[1] R[2] i.e. 7 5 5

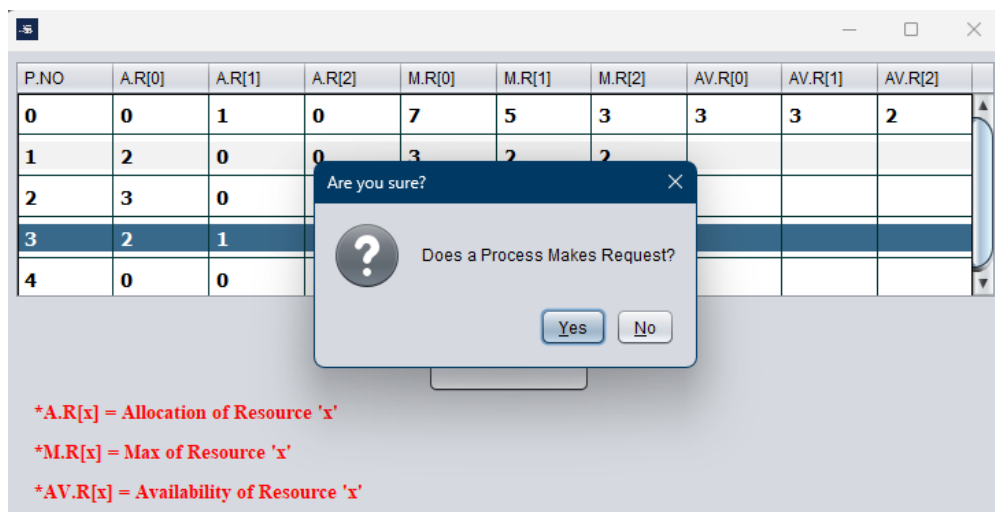
Since Process P[2] need is less than equals to Avaialability
Total Resources now available
R[0] R[1] R[2] i.e. 10 5 7

The system is in the Safe State

Safe sequence is :P[1]P[3]P[4]P[0]P[2]

OK

Step – 05: After clicking the "OK" button, the program will ask whether there is a Process Request or not, as shown in Fig.



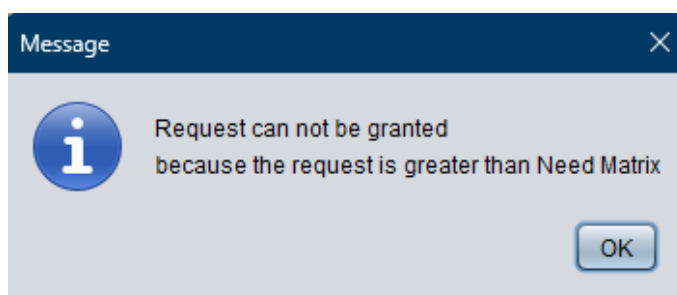
Step – 06: Suppose now that the process P1 requests one additional instance of resource type A and two instances of resource type C, so $Request_1 = (1,0,3)$, As shown in Fig.

The screenshot shows a window titled "Process Request". It contains a text input field for "P.NO" with the value "1". Below it is a table for resource requests.

R.R[0]	R.R[1]	R.R[2]
1	0	3

Check

Since the request of resource C (3) is greater than need(2) it raises an error message.



Step – 07: Suppose now that the process P1 requests one additional instance of resource type A and two instances of resource type C, so $Request_1 = (1,0,0)$, As shown in Fig.

Process Request

P.NO

R.R[0]	R.R[1]	R.R[2]
1	0	<input type="text" value="0"/>

NEED MATRIX

N.R[0]	N.R[1]	N.R[2]
7	4	3
0	2	2
6	0	0
0	1	1
4	3	1

Step – 08: After clicking on the check button, the program first executes the process request algorithm and then generates the need matrix and result, as shown in fig.

RESULT
✕

i

Since Process P[1] need is less than equals to Avaialability

Total Resources now available

R[0] R[1] R[2] i.e. 5 3 2

Since Process P[3] need is less than equals to Avaialability

Total Resources now available

R[0] R[1] R[2] i.e. 7 4 3

Since Process P[4] need is less than equals to Avaialability

Total Resources now available

R[0] R[1] R[2] i.e. 7 4 5

Since Process P[0] need is less than equals to Avaialability

Total Resources now available

R[0] R[1] R[2] i.e. 7 5 5

Since Process P[2] need is less than equals to Avaialability

Total Resources now available

R[0] R[1] R[2] i.e. 10 5 7

The system is in the Safe State

Safe sequence is :P[1]P[3]P[4]P[0]P[2]

