**Practical no: 9**

**Problem Statement: : Write a Java program to implement Banker's Algorithm.**

**Name: Aditi Dinesh Mulay**

**Class: T.E. Computer**

**Subject: SPOS**

**Div: A**

**Roll no: 02**

**PRN No. 71918146B**

SPOS

Assignment C-2

Aditi Dinesh Muday
T.E. Comp Div: A
Roll no: 02.

Aim: Bankers algorithm for deadlock detection & avoidance.

Theory:

Banker's algorithm is a resource allocation & deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an 'S-state' check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following Data Structures used to implement Banker's Algorithm.

Let 'n' be the number of processes in the system & 'm' be the number of resources type.

Available:

1) It is a 1-d array of size 'm' indicating the no of available resources of each type.
2) Available [j] = k means there are 'k' instances of resource type Rj.

Max:

1) It is a 2-d array of size n*m that defines the maximum demand of each process in a system.
2) max[i,j] = k means process Pi may request atmost 'k' instances of resource type Rj.

Allocation:

1) It is 2d array of size 'n*m' that defined the no. of resources of each type currently allocated to each process.

2) Allocation [i,j] = k means process $P_i$ is currently allocated 'k' instance of resource type $R_j$.

Need:

1) It is 2-d array of size 'n*m' that indicates the remaining resource need of each process.

2) Need [i,j] = k means process $P_i$ currently need 'k' instances of resource type $R_j$ for its execution.

3) Need [i,j] = Max [i,j] − Allocation [i,j]

Banker's algorithm consist of Safety algorithm & Resource request algorithm:

Safety Algorithm:

1) Let work and Finish be vectors of length 'm' & 'n' resp
Initialize: Work = Available
Finish[i] = false; for i = 1,2,3,4...n.

2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work
if no such i exists goto step 4.

3) Work = Work + Allocation [i]
Finish [i] = true
goto step(2)

4) if Finish[i] = true for all i
then system is in safe state.

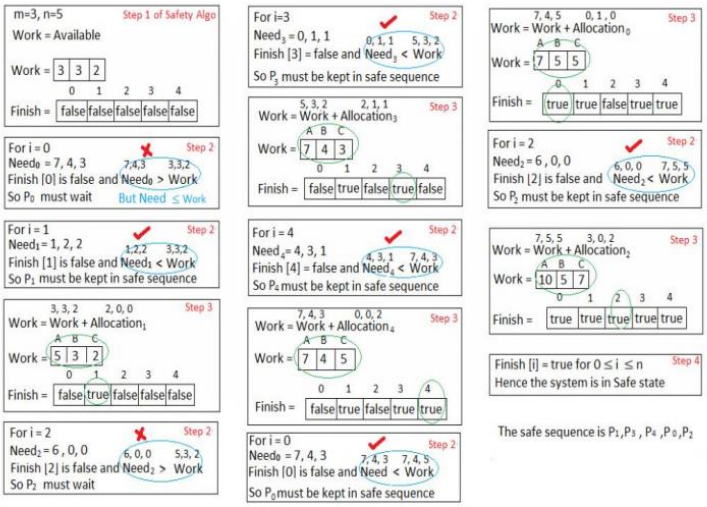
Ex: Considering a system with 5 processes $P_0$ through $P_4$ & 3 resources type A, B, C. Resource type A has 10 instances, B has 5 & C has 7 instances.
At time $t_0$.

| Process | Allocation | | | Max | | | 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 | | | |

1) What will be the content of Need matrix?
Need[i, j] = Max[i, j] − Allocation[i, j]
∴ Need of Matrix is:

| Process | 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 |

2) Is the system in safe state? If yes, then what is safe sequence?
→ Applying a safety algorithm on system,



- **Question3. What will happen if process $P_1$ requests one additional instance of resource type A and two instances of resource type C?**

A B C

$Request_1$ = 1, 0, 2

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

Step 1

1, 0, 2 1, 2, 2

$Request_1 < Need_1$

Step 2

1, 0, 2 3, 3, 2

$Request_1 < Available$

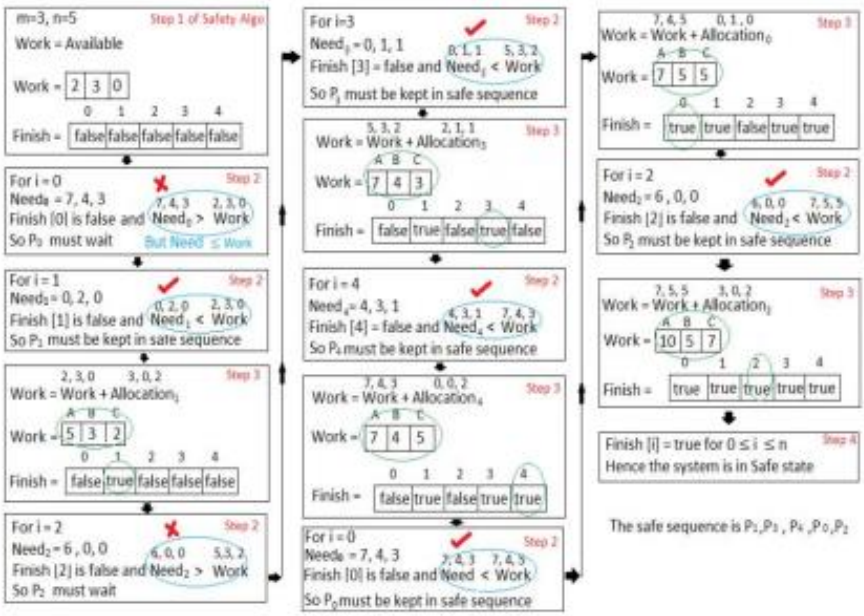
Step 3

$Available = Available - Request_1$

$Allocation_1 = Allocation_1 + Request_1$

$Need_1 = Need_1 - Request_1$

| Process | Allocation | Need | Available |
|---|---|---|---|
| | A B C | A B C | A B C |
| $P_0$ | 0 1 0 | 7 4 3 | 2 3 0 |
| $P_1$ | 3 0 2 | 0 2 0 | |
| $P_2$ | 3 0 2 | 6 0 0 | |
| $P_3$ | 2 1 1 | 0 1 1 | |
| $P_4$ | 0 0 2 | 4 3 1 | |




Conclusion:

Thus, we have studied the Banker's Algorithm.

# Program:

```
/*Name: Aditi Mulay */
/*Roll.No: 02 */
/*Div: A */

%{

        #include<stdio.h>
        int compund = 0;
%}


%%
[ \t\n][aA][nN][dD][ \t\n] {compund=1;}
[ \t\n][oO][rR][ \t\n] {compund=1;}
[ \t\n][bB][uU][tT][ \t\n] {compund=1;}
 ;
%%

int yywrap(){

}

int main(){
    printf("\nEnter the Sentence\n");
    yylex();

    if(compund==1){
        printf("Sentence is Compound\n");
    }
    else{
     printf("Sentence is Simple\n");
    }
    return 0;
}
```