**NAME: SHASHWAT SHAH**

**SAP ID: 60004220126**

**DIV: B3**

**EXPERIMENT NAME: Banker’s Algorithm**

**OPERATING SYSTEMS – EXPERIMENT Banker’s Algorithm**

**AIM**: To implement Banker’s Algorithm.

**THEORY:**

The Banker’s algorithm is a resource allocation and deadlock avoidance algorithm. Banker’s

algorithm is applicable to resource allocation system with multiple instances of each

resource type. Newly entered process should declare maximum number of instances of each

resource type which it may require. The request should not be more than total number of

resources in the system. It tests for safety by simulating the allocation for predetermined

maximum possible amounts of all resources, then makes an ‘safe state’ check to test for

possible activities, before deciding whether allocation should be allowed to continue.

Let ‘n’ be the number of processes in the system and ‘m’ be the number of resources types.

Available:

• It is a 1-d array of size ‘m’ indicating the number of available resources of each type.

• Available [j] = k means there are ‘k’ instances of resource type Rj

Max:

• It is a 2-d array of size ‘n\*m’ that defines the maximum demand of each process in a

system.

• Max [ i, j] = k means process Pi may request at most ‘k’ instances of resource type Rj.

Allocation:

• It is a 2-d array of size ‘n\*m’ that defines the number of resources of each type

currently allocated to each process.

• Allocation [ i, j] = k means process Pi is currently allocated ‘k’ instances of resource

type Rj

Need:

• It is a 2-d array of size ‘n\*m’ that indicates the remaining resource need of each

process.

• Need [ i, j] = k means process Pi currently need ‘k’ instances of resource type Rj

• Need [ i, j] = Max [ i, j] – Allocation [ i, j]

Allocationi specifies the resources currently allocated to process Pi and Needi specifies the

additional resources that process Pi may still request to complete its task.

Algorithm:

The algorithm for finding out whether a system is in a safe state can be described as follows:

1. Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.

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. Needi <= Work

if no such i exists go to step (4)

3. Work = Work + Allocation[i]

Finish[i] = true

go to step (2)

4. if Finish [i] = true for all i

then the system is in a safe state

**CODE AND OUTPUT:**

**// Shashwat Shah - 60004220126**

import java.util.Scanner;

public class Bankers {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int m, n;

System.out.print("Enter number of recourses: ");

m = sc.nextInt();

System.out.print("Enter number of processes: ");

n = sc.nextInt();

int resource[] = new int[m];

int max[][] = new int[n][m];

int allocation[][] = new int[n][m];

int need[][] = new int[n][m];

int work[] = new int[m];

int finish[] = new int[n];

int tot[] = new int[m];

int i, j;

for (i = 0; i < m; i++) {

System.out.print("Enter number of instances of resource " + (i + 1) + ": ");

resource[i] = sc.nextInt();

tot[i] = 0;

}

for (i = 0; i < n; i++) {

System.out.println("Maximum and Allocated resources for Process " + i);

for (j = 0; j < m; j++) {

System.out.print("Max for resource " + (j + 1) + ": ");

max[i][j] = sc.nextInt();

System.out.print("Allocation for resource " + (j + 1) + ": ");

allocation[i][j] = sc.nextInt();

tot[j] += allocation[i][j];

need[i][j] = max[i][j] - allocation[i][j];

}

finish[i] = 0;

}

System.out.print("\nInstances of recourses available:\n");

for (i = 0; i < m; i++) {

work[i] = resource[i] - tot[i];

System.out.print("R" + (i + 1) + ": " + work[i] + "\t");

}

System.out.print("\nInstances of resources needed by Processes:\n");

for (i = 0; i < m; i++)

System.out.print("\tR" + (i + 1));

for (i = 0; i < n; i++) {

System.out.print("\nP" + i);

for (j = 0; j < m; j++)

System.out.print("\t" + need[i][j]);

}

System.out.println();

int total = 0, check = 0;

int safe[] = new int[n];

while (total < n) {

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++) {

if (need[i][j] > work[j])

check++;

}

if (check == 0 && finish[i] == 0) {

for (j = 0; j < m; j++)

work[j] += allocation[i][j];

finish[i] = 1;

safe[total] = i;

total++;

}

check = 0;

}

}

System.out.print("\n\nThe safe sequence is");

for (i = 0; i < n; i++)

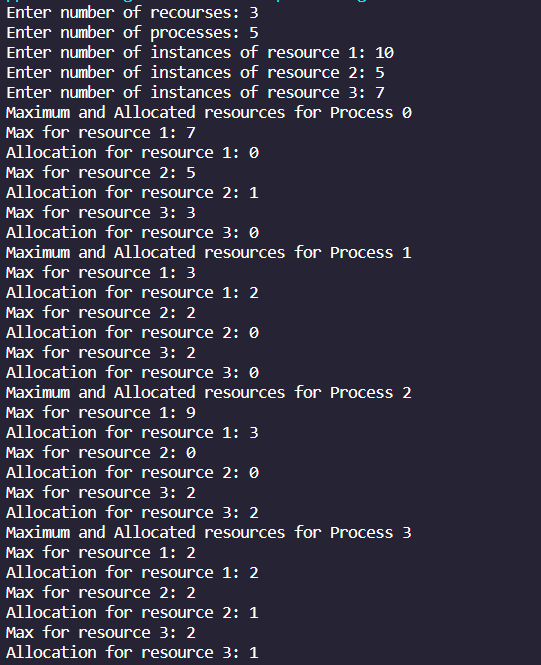
System.out.print("\tP" + safe[i]);

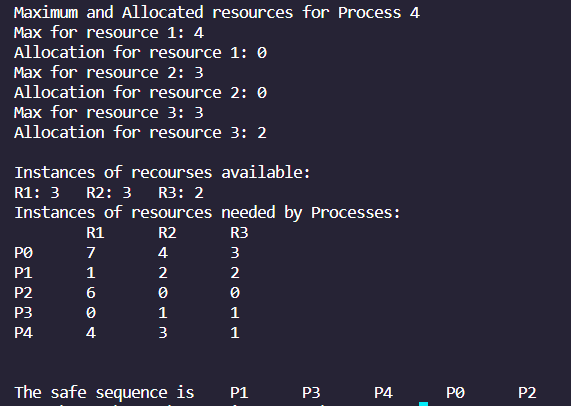
System.out.println();

sc.close();

}

}





**Conclusion:** Thus, we have learnt about Banker’s Algorithm and implemented it’s program in Java.