Smt. Chandibai Himathmal Mansukhani College

USCS3P01:USCS303-Operating System (OS) Practical-06

Banker's Algorithm

Contents

Practical Date: 20th August 2021

Practical Aim:

Banker's Algorithm

Banker's Algorithm

1. Banker's algorithm is a deadlock avoidance algorithm.
2. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.
3. The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

Data Structures

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 Rj are available.

Max: An n x m matrix defines the maximum demand of each thread. If Max[i][j] equals k, then thread Ti may request at most k instances of resource type Rj.

Allocation: An n x m matrix defines the number of resources of each type currently allocated to each thread. If Allocation[i][j] equals k, then thread Ti is currently allocated k instances of resource type Rj.

Need: An n x m matrix indicates the remaining resource need of each thread. If Need[i][j] equals k, then thread Ti may need k more instances of resource type Rj to complete its task.

Need[i][j] = Max[i][j] - Allocation[i][j]

Safety Algorithm

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

Initialize Work = Available and Finish[i] = false for i= 0, 1,..., n-1. Step 2: Find an index i such that both

Step 2.1: Finish[i] == false Step 2.2: Need; ≤ Work

If no such i exists, go to Step 4.

Step 3: Work = Work + Allocation,

Finish[i] = true

Go to Step 2. Step 4: If Finish[i] == true for all i, then the system is in a safe state.

Resource-Request Algorithm

Let Request, be the request vector for thread Ti.

• If Request; [j] ==k, then thread T_1 wants k instances of resource type Rj.

When a request for resources is made by thread T_1,actions are taken the following

Step 1: If Request, ≤ Need,, go to Step 2. Otherwise, raise an error condition, since the thread has exceeded its maximum claim.

Step 2: If Request, ≤ Available, go to Step 3. Otherwise, T, must wait, since the resources are not available.

Step 3: Have the system pretend to have allocated the requested resources to thread T_1 by modifying the state as follows:

Available = Available - Request;
Allocation; = Allocation; + Request;
Need; = Need; - Request;

If the resulting resource-allocation state is safe, the transaction is completed, and thread T_1 is allocated its resources. However, if the new state is unsafe, then T_1 must wait for Request;, and the old resource allocation state is restored.

Q.1Write a Java program that implements the banker's algorithm.

Source Code:

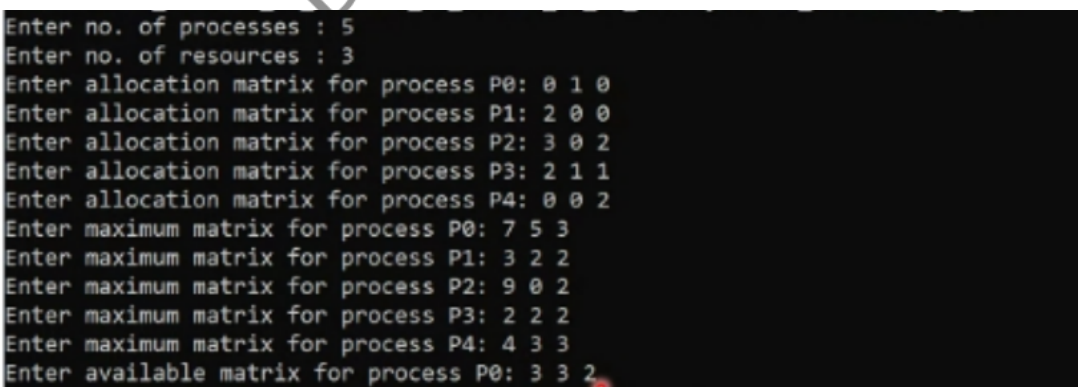
```
//Name: Subrat Sahu
// Batch: B2
// PRN: 2020016400833692
// Date: 20th August 2021
// Prac-06: Banker's Algorithm

import java.util.Scanner;

public class P6_BankersAlgo_SS {

private int need[][], allocate[][], max[][], avail[][], np, SS;

private void input() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter no. of processes : ");

np = sc.nextInt(); // no. of process

System.out.print("Enter no. of resources: ");

SS = sc.nextInt(); // no. of resources
need = new int[np][SS]; // initializing arrays

max = new int[np][SS];

allocate = new int[np][SS];

vail=newint[1][SS];
```

```
for (int i = 0; i < np; i++) {

System.out.print("Enter allocation matrix for process P" + i + ": "); allocate[i][j] =
sc.nextInt(); // allocation matrix

for (int j = 0; j < SS; j++)

}

for (int i = 0; i < np; i++) {

System.out.print("Enter maximum matrix for process P" + i + ": ");

for (int j = 0; j<SS; j++)

max[i][j] = sc.nextInt(); // max matrix }
private boolean check(int i) {

// checking if all resources for ith process can be allocated for (int j = 0; j<n; j++)

f(avail * [0] * [j] < need * [i] * [j])

return false;

return true;

} // check() ends public void isSafe() {

input();
```

```
calc_need():

boolean done[] = new boolean[np];

int i=0;
// printing Need Matrix

System.out.println("========Need

for (int a = 0; a < np; a++) {

for (int b= 0; b < SS; b++) {

System.out.print(need[a][b] + "\t");

}

System.out.println();

Matrix========");

}         for (int i = 0; i < np; i++)

if (!done[i] && check(i)) { // trying to allocate for (int k = 0; k < SS; k++)

avail[0][k]=avail[0][k]* need[i][k]+max[i][k]; System.out.print("P" + i + ">");

allocated = done[i] = true;
if (j == np) // if all processes are allocated System.out.println("\nSafely allocated");
```

```
else System.out.println("All/Remaining process can\'t be allocated safely");

}//isSafe() ends

public static void main(String[] args) {

new P6_BankersAlgo_SS().isSafe();

}

}// class ends
```


Input:



```
Enter no. of processes : 5
Enter no. of resources : 3
Enter allocation matrix for process P0: 0 1 0
Enter allocation matrix for process P1: 2 0 0
Enter allocation matrix for process P2: 3 0 2
Enter allocation matrix for process P3: 2 1 1
Enter allocation matrix for process P4: 0 0 2
Enter maximum matrix for process P0: 7 5 3
Enter maximum matrix for process P1: 3 2 2
Enter maximum matrix for process P2: 9 0 2
Enter maximum matrix for process P3: 2 2 2
Enter maximum matrix for process P4: 4 3 3
Enter available matrix for process P0: 3 3 2
```

Output:

```
========Need Matrix========
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1
Allocated process:
P1 > P3 > P4 > P0 > P2 >
Safely allocated
```