Fokane Sakshi Anil
TE - A - 42

Assignment No: 9

Banker Algorithm

- Aim: Banker algorithm for deadlock detection and avoidance.

- Problem statement: write a java program to implement Bankers algorithm.

- Operating System: Bankers Algorithm: The bankers algorithm is resource allocation & deadlock avoidance algorithm that test for safety by simulating the allocation for predetermined maximum possible amount of all resource then makes an = s-states check for Possible activities. following datastructure are used to implement :
let 'n' be the Number of process in the System, & 'm' be the No. of resource type.

Available :~
It is a 1-array of size 'm' indicating the n. of available resource of each type.
- **Available [j]** = K means there are 'k) instance of resource type Rj
- **MAX**: It is 2-d array of size n * m that defines the maximum demand of each process in a System.
- **Allocation [i.j] = K** means process Pi

is currently allocated 'K' instance of resource type R_j.

- Need :

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

- Need [i·j] = K mean process P, currently needs K instance of resource type R_j for its execution.
- Need [ij] = Max[i, j) - Allocation [i·j)
- Allocation specifies the resource currently allocated to process, P_i and need, specifies the additional resource that.
- Banker's algorithm consist of safety algorithm and resource request algorithm.
- Safety Algorithm :-

- The algorithm for finding out whenever or, not a system is in a safe state can be described as follows.

1) Let work & finish be vectors of length 'm' & 'n' resp.

Initialize : work = Available

finish [i] = false; for i = 1, 2 ... n

2) find an i such that both

a) finish [i] = false

b] need <= work

if no such i exist goto step (4)

3) work = work + Allocation [i]

finish [i] = true.
goto step (2)

4) if finish [i] = true for all i
then the System is in a safe state.

java code for Banker's Algorithm:

```
import java.util.Scanner;
public class Bankers{
 private int need[][], allocate[][],
max[][] avail, n,p,r;
 private void input(){
  Scanner sc = new Scanner(System.in)
System.out.println("Enter no. of processes
and reesources");
 np = sc.nextINT();
 nr = sc.nextINT();
 need = new int [np][nr];
 max = new int [np][nr];
 allocate = new int [np][nr]
 avail = new int [i][nr]
System.out.println("Enter allocat matrix");
for(int i=0; i<p ; i++)
   for (int j=0 ; j<nr ; j++).
      allocate[i][j] = sc.nextINT();
System.out.println("Enter max matrix");
for(int i=0; i<np ; i++)
   for(int j=0; j<nr ; j++)
     max[i][j] = sc.nextINT();
System.out.println("Enter available matrix");
```

```
   for(int j=0; j<nr ; j++)
     avail[0][j] = sc.nextInt();
   sc.close();
}
 private int[][] calc-need(){
  for(int i=0; i<np ; i++)
     for( int j=0; j<nr ; j++)
      need[i][j]=max[i][j]-allocate[i][j];
     returnneed;
}
 private boolean check(int i){
   for(int j=0; j<nr ; j++)
     if( avail[0][i]< need [i][j])
      return false;
  return true;
}

 public void isSafe(){
    input();
    calc_need();
    boolean done[] =new boolean[np];
    int j=0;
    while(j<np){
    boolean allocated=false;
   for(int i=0; i<np ; i++)
    if( !done[i] && check(i))
       for(int k=0; k<nr; k++)
         avail[0][i]= avail[0][k]-need[i]
                          [k]+max
    System.out.println("Allocate process"+i);
    allocated=done[i]=true;
     j++;
  }
 if(!allocated) break;
}
```

```
if(j==np)
 System.out.println("an safety
          allocated");
else
  system.out.println("All process
can't be allocated:");
}
public static void main(string[]
   args) {
     new Banker() is safe();
}
```

```
//Name:Fokane Sakshi Anil
// TE A 42
//  ASSINGNMENT:GROUP_C_1
// Java program to illustrate Banker's Algorithm

import java.util.*;
class banker_algo
{
static int P = 5;
static int R = 3;

// Function to find the need of each process
static void calculateNeed(int need[][], int maxm[][],
int allot[][])
{
// Calculating Need of each P
for (int i = 0 ; i < P ; i++)
for (int j = 0 ; j < R ; j++)

// Need of instance = maxm instance -
//              allocated instance
need[i][j] = maxm[i][j] - allot[i][j];
}

// Function to find the system is in safe state or not
static boolean isSafe(int processes[], int avail[], int maxm[][],
int allot[][])
{
int [][]need = new int[P][R];

// Function to calculate need matrix
calculateNeed(need, maxm, allot);

// Mark all processes as infinish
boolean []finish = new boolean[P];

// To store safe sequence
int []safeSeq = new int[P];

// Make a copy of available resources
int []work = new int[R];
for (int i = 0; i < R ; i++)
work[i] = avail[i];

// While all processes are not finished
// or system is not in safe state.
int count = 0;
while (count < P)
{
// Find a process which is not finish and
// whose needs can be satisfied with current
// work[] resources.
boolean found = false;
for (int p = 0; p < P; p++)
{
// First check if a process is finished,
// if no, go for next condition
if (finish[p] == false)
{
// Check if for all resources of
// current P need is less
// than work
int j;
for (j = 0; j < R; j++)
if (need[p][j] > work[j])
break;

// If all needs of p were satisfied.
if (j == R)
{
// Add the allocated resources of
// current P to the available/work
// resources i.e.free the resources
for (int k = 0 ; k < R ; k++)
work[k] += allot[p][k];

// Add this process to safe sequence.
safeSeq[count++] = p;

// Mark this p as finished
finish[p] = true;

found = true;
}
}
}

// If we could not find a next process in safe
// sequence.
if (found == false)
{
System.out.print("System is not in safe state");
return false;
}
}

// If system is in safe state then
// safe sequence will be as below
System.out.print("System is in safe state.\nSafe"
+" sequence is: ");
for (int i = 0; i < P ; i++)
System.out.print(safeSeq[i] + " ");

return true;
}
```

```
// Driver code
public static void main(String[] args)
{
int processes[] = {0, 1, 2, 3, 4};

// Available instances of resources
int avail[] = {3, 3, 2};

// Maximum R that can be allocated
// to processes
int maxm[][] = {{7, 5, 3},
{3, 2, 2},
{9, 0, 2},
{2, 2, 2},
{4, 3, 3}};

// Resources allocated to processes
int allot[][] = {{0, 1, 0},
{2, 0, 0},
{3, 0, 2},
{2, 1, 1},
{0, 0, 2}};

// Check system is in safe state or not
isSafe(processes, avail, maxm, allot);
}
}
```

________________OUTPUT____________

```
System is in safe state.

Safe sequence is: 1 3 4 0 2
```