Peitam Mogal TE-B-17

DATE

Assignment No. 9

Bankers Algorithm

Assignment No:- C2

Aim:- Bankers algorithm for deadlock detection and avoidance.

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

Following Data structures are used to implement the Banker's algorithm:

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' intances of resoura 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' intances of resource type Rj.

Allocation:

- It is a 2-d array of size 'n*m' that defines the number of resoures of each type currently allocated to each process.
- Allocation [i,j] = K means process Pi is currently allocated 'K' intances of resource type Rj.

Need:

- it is 2-d array of size 'n*m' that indicates

the remaining resource need for each process.
- Need [i,j] = K means process Pi currently need 'K' intances of resource type Rj.
- Banker's Algorithm consist of Safety algorithm and Resource request algorithm.

- Safety algorithm
  - The algorithm for finding out wheather or not a system is in a safe 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) Need$_i$ <= Work
   if no such i exists goto setup (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.

Name: Pritam Gotiram Mogal **DivRoll**: TE-B-17

Assignment No. 09

Problem Satement: Write a Java program to implement Banker's Algorithm

1. Banker's Algorithm Program:

```
import java.util.Scanner; public class Bankers{
private int need[][],allocate[][],max[][],avail[][],np,nr;


 private void input(){
 Scanner sc=new Scanner(System.in);
 System.out.print("Enter no. of processes and resources : ");
np=sc.nextInt(); //no. of process  nr=sc.nextInt(); //no. of
resources  need=new int[np][nr]; //initializing arrays
max=new int[np][nr];  allocate=new int[np][nr];  avail=new
int[1][nr];


 System.out.println("Enter allocation matrix -->");
for(int i=0;i<np;i++)  for(int j=0;j<nr;j++)
 allocate[i][j]=sc.nextInt(); //allocation matrix


 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(); //max matrix


 System.out.println("Enter available matrix -->");
for(int j=0;j<nr;j++)  avail[0][j]=sc.nextInt();
//available matrix


 sc.close();
 }
```

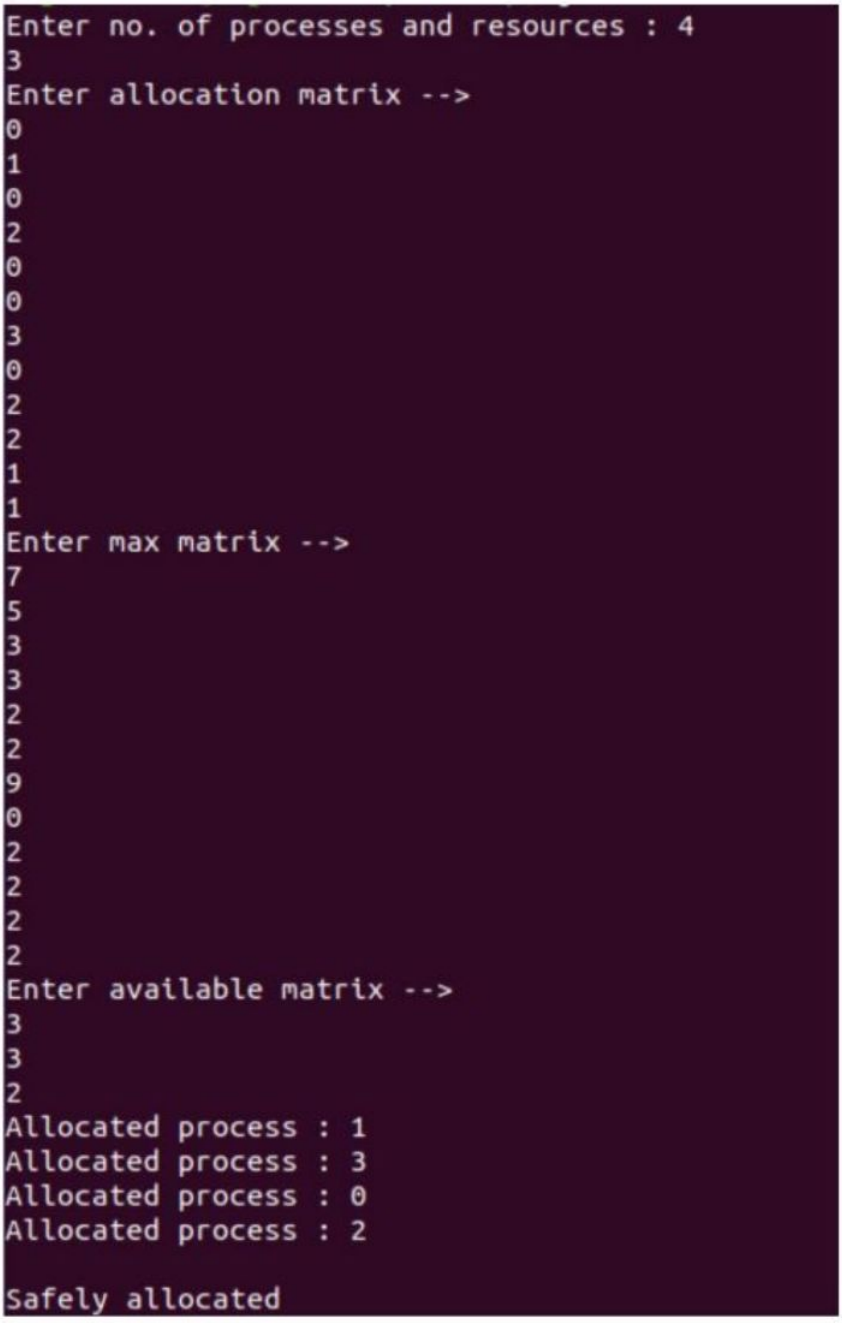
```
 private int[][] calc_need(){  for(int
i=0;i<np;i++)  for(int j=0;j<nr;j++)
//calculating need matrix
need[i][j]=max[i][j]-allocate[i][j];


 return need;  }  private
boolean check(int i){
 //checking if all resources for ith process can be allocated
for(int j=0;j<nr;j++)  if(avail[0][j]<need[i][j])  return
false;


 return true;  }  public void isSafe(){  input();
calc_need();  boolean done[]=new
boolean[np];  int j=0;  while(j<np){ //until all
process allocated  boolean allocated=false;
for(int i=0;i<np;i++)  if(!done[i] &&
check(i)){ //trying to allocate  for(int
k=0;k<nr;k++)
 avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
System.out.println("Allocated process : "+i);
allocated=done[i]=true;  j++;  }  if(!allocated)
break; //if no allocation
 }  if(j==np) //if all processes are
allocated  System.out.println("\nSafely
allocated");  else
 System.out.println("All proceess cant be allocated safely");
 }


 public static void main(String[] args) {
new Bankers().isSafe();
 }
}
```

OUTPUT:


Enter no. of processes and resources : 4
3
Enter allocation matrix -->
0
1
0
2
0
0
3
0
2
2
1
1
Enter max matrix -->
7
5
3
3
2
2
9
0
2
2
2
2
Enter available matrix -->
3
3
2
Allocated process : 1
Allocated process : 3
Allocated process : 0
Allocated process : 2
Safely allocated