System Programming and Operating Systems Lab

### ASSIGNMENT 12

**Name:Maulick Metkar**

**Roll No: 323045**

**Batch: C2**

# Aim:

Write a Java program to impliment Banker’s Algorithm.

# Objectives:

To implement Banker’s Algorithm by using Java.

The Banker algorithm, sometimes referred to as the detection algorithm, is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety.

# Theory:

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that test for a safety by simulating the allocation for predetermined maximum on ”S-State” check to text for a possible should be allowed to continue

The Banker’s Algorithm derives its name from the fact that this algorithm could be used in a bank- ing system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers[citation needed]. By using the Banker’s algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer’s request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Data structures for the banker’s algorithm, let,

n= Number of processes.

m= Number of resource types.

## Available:

Vector of length m, available [ ]=k, there are k instances of resources type Rj correctly available.

## Max:

nxm matrix if max is [i,j]=k, then process pi will request of most k instances of resources type RJ.

## Allocation:

nxm matrix if allo [i,j]=k then the currently allocates(i.e holding) is k. Instance of Rj.

## Need:

nxm matrix if need [i,j]=k then it may need k more instances of Rj to complete its task. Need[i,j] = Max[i,j] - Allocation[i,j]. n=m-a.

# Safety Algorithm:

1. Let work and finish be vectors of length mond n resp. initialize

work = Available

finish [i] == false for i=1,2,3,. n.

1. Find an i such that both finish [i] == false.

Need = work.

if no such i exists go to step4.

1. work = work Allocation finish [i] = true

go to step2.

1. If finish [i] = true for all i, then the sim is in safe state.

# Resource Request Algorithm for process pi:

request i = request vector for pi

request [i] = k means process pi wants k instance of resource type pj.

1. If request i = Need i goto step2 - otherwise error.
2. If request i = Available go to step3 - otherwise pi burst wait.
3. ”Allocate” requested resources to pi as follows, Available = Available Request;

Alloct = Alloci + Request i; Need = Need i - Request;

if safe = the resources are allocated to pi.

if unsafe = restre the old resource allocation state and block pi.

### C Program:

import java . u t i l . Scanner ; p u b l i c c l a s s Bankers

*{*

p r i v a t e i n t need [ ] [ ] , a l l o c a t e [ ] [ ] , max [ ] [ ] , a v a i l [ ] [ ] , np , nr ;

p r i v a t e void input ( )

*{*

Scanner sc=new Scanner ( System . in ) ;

System . out . p r i n t (” Enter no . o f p r o c e s s e s and r e s o u r c e s : ” ) ; np=sc . next Int ( ) ; // no . o f p r o c e s s

nr=sc . next Int ( ) ; // no . o f r e s o u r c e s

need=new i n t [ np ] [ nr ] ; // i n i t i a l i z i n g a r r a ys max=new i n t [ np ] [ nr ] ;

a l l o c a t e=new i n t [ np ] [ nr ] ; a v a i l=new i n t [ 1 ] [ nr ] ;

System . out . p r i n t l n (” Enter a l l o c a t i o n matrix *>*”); f o r ( i n t i =0; i *<*np ; i++)

*−−*

f o r ( i n t j =0; j *<*nr ; j++)

a l l o c a t e [ i ] [ j ]= sc . next Int ( ) ; // a l l o c a t i o n matrix

System . out . p r i n t l n (” Enter max matrix *>*”); f o r ( i n t i =0; i *<*np ; i++)

*−−*

f o r ( i n t j =0; j *<*nr ; j++)

max [ i ] [ j ]= sc . next Int ( ) ; //max matrix

System . out . p r i n t l n (” Enter a v a i l a b l e matrix *>*”); f o r ( i n t j =0; j *<*nr ; j++)

*−−*

a v a i l [ 0 ] [ j ]= sc . next Int ( ) ; // a v a i l a b l e matrix

sc . c l o s e ( ) ;

*}*

p r i v a t e i n t [ ] [ ] c a l c n e e d ( ) f o r ( i n t i =0; i *<*np ; i++)

*{*

f o r ( i n t j =0; j *<*nr ; j++) // c a l c u l a t i n g need matrix need [ i ] [ j ]=max [ i ] [ j ]*−* a l l o c a t e [ i ] [ j ] ;

return need ;

*}*

p r i v a t e boolean check ( i n t i )

*{*

// checking i f a l l r e s o u r c e s f o r i t h p r o c e s s can be a l l o c a t e d f o r ( i n t j =0; j *<*nr ; j++)

i f ( a v a i l [ 0 ] [ j ]*<* need [ i ] [ j ] ) return f a l s e ;

return true ;

*}*

p u b l i c void i s S a f e ( ) input ( ) ;

*{*

c a l c n e e d ( ) ;

boolean done [ ]= new boolean [ np ] ; i n t j =0;

while ( j *<*np ) // u n t i l a l l p r o c e s s a l l o c a t e d boolean a l l o c a t e d=f a l s e ;

*{*

f o r ( i n t i =0; i *<*np ; i++)

i f ( ! done [ i ] && check ( i ) ) // t r y i n g to a l l o c a t e f o r ( i n t k=0;k*<*nr ; k++)

*{*

a v a i l [ 0 ] [ k]= a v a i l [ 0 ] [ k] need [ i ] [ k]+max [ i ] [ k ] ; System . out . p r i n t l n (” All ocated p r o c e s s : ”+ i ) ; a l l o c a t e d=done [ i ]= true ;

*−*

j ++;

*}*

i f ( ! a l l o c a t e d ) break ; // i f no a l l o c a t i o n

*}*

i f ( j==np ) // i f a l l p r o c e s s e s are a l l o c a t e d System . out . p r i n t l n (” n S a f e l y a l l o c a t e d ” ) ;

*\*

e l s e

System . out . p r i n t l n (” All p r o c e e s s cant be a l l o c a t e d s a f e l y ” ) ;

*}*

p u b l i c s t a t i c void main ( S t r i n g [ ] args ) new Bankers ( ) . i s S a f e ( ) ;

*{*

*}*

*}*

# output:

1. **Conclusion:**

Hence, we implemented the Banker’s Algorithm for deadlock Avoidance.