**EXPERIMENT 5**

**OBJECTIVE**:-

Write a program to implement Banker’s Algorithm.

**CODE**:-

//------------------------------------------------------------------------------

//

// Desc: Program to emulate banker's algorithm

// input is 2 matrix of size mxn namely: Allocated and Max Request

// calculate the Need matrix and then run the algorithm

// to find out the safe sequence of processes

// there are m processes and n resource types

//

// Date: 21/2/2018

//

// Author: VU DUY DU - 2k16/CO/364

//

//------------------------------------------------------------------------------

#include<stdio.h>

#include<iostream>

#include<vector>

struct Matrix{

Matrix(std::vector< std::vector<int> >container):container(container){

printf("Created Matrix (%ix%i)\n",GetRow(),GetColumn());

}

Matrix(){}

std::vector< std::vector<int> >container;

size\_t GetRow()const{return container.size();}

size\_t GetColumn()const{return container[0].size();}

int GetElement(int i, int j)const{

return container[i][j];

}

void EnterInput(int row, int col){

int itemp;

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

std::vector<int>vtemp;

for(int j =0; j<col;j++){

printf("[%i,%i]=",i,j);

std::cin>>itemp;

vtemp.push\_back(itemp);

}

container.push\_back(vtemp);

}

}

void Display(){

for(size\_t i = 0; i<container.size();i++){

for(size\_t j = 0; j<container[i].size();j++)

printf("%i ",container [i][j]);

printf("\n");

}

}

std::vector<int>GetRowElement(int i){

return container[i];

}

friend Matrix operator-(const Matrix&a, const Matrix&b);//subtract two same size matrix

};

Matrix operator-(const Matrix&a,const Matrix&b){

std::vector< std::vector<int> >container;

int itemp;

for(size\_t i = 0; i<a.GetRow();i++){

std::vector<int>vtemp;

for(size\_t j = 0; j<a.GetColumn();j++){

itemp = a.GetElement(i,j) - b.GetElement(i,j);

vtemp.push\_back(itemp);

}

container.push\_back(vtemp);

}

return Matrix(container);

}

bool compare(std::vector<int>a, std::vector<int>b){

for(int i = 0; i<a.size(); i++)

if(a[i]>b[i])return false;

return true;

}

void add(std::vector<int>&a,std::vector<int>b){//add to a

for(int i = 0; i<a.size(); i++)

a[i]+=b[i];

}

void RunBankerAlgorithm(Matrix&Allocated,Matrix&MaxRequest, std::vector<int>Available){

int process\_number = Allocated.GetRow();

std::vector<bool> Finish(Allocated.GetRow());

for(int i = 0; i<process\_number;i++)

Finish.push\_back(false);

Matrix Need = MaxRequest - Allocated;

Need.Display();

bool running = true;

int count = 0;

while(running){

running = false;

for(int i = 0; i<process\_number;i++){

if(!Finish[i]){

if(compare(Need.GetRowElement(i),Available)){//compare 2 vectors

add(Available,Allocated.GetRowElement(i));//add 2 vectors

Finish[i]=true;

std::cout<<"P"<<i<<" ";

//if no Need's vector is select in one complete loop

running = true;

count++;

}

}

}

if(count == Finish.size())

break;

}

printf("\n");

for(int i = 0; i<process\_number;i++)

if(!Finish[i]){

std::cout<<"Not A Safe State\n";

return;

}

std::cout<<"Safe State\n";

}

int main(){

printf("Banker's Algorithm\n");

std::cout<<"Enter Allocated matrix:\n";

std::cout<<"Enter size mxn: ";

int row, col;

std::cin>>row>>col;

Matrix Allocated;

Allocated.EnterInput(row, col);

Allocated.Display();

std::cout<<"Enter Max Request matrix:\n";

Matrix MaxRequest;

MaxRequest.EnterInput(row,col);

MaxRequest.Display();

printf("Enter Available Matrix\n");

std::vector<int> Available;

int t;

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

std::cin>>t;

Available.push\_back(t);

}

RunBankerAlgorithm(Allocated, MaxRequest, Available);

return 0;

}

**OUTPUT:-**

Banker's Algorithm

Enter Allocated matrix:

Enter size mxn: 5 3

[0,0]=0

[0,1]=1

[0,2]=0

[1,0]=2

[1,1]=0

[1,2]=0

[2,0]=3

[2,1]=0

[2,2]=2

[3,0]=2

[3,1]=1

[3,2]=1

[4,0]=0

[4,1]=0

[4,2]=2

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter Max Request matrix:

[0,0]=7

[0,1]=5

[0,2]=3

[1,0]=3

[1,1]=2

[1,2]=2

[2,0]=9

[2,1]=0

[2,2]=2

[3,0]=2

[3,1]=2

[3,2]=2

[4,0]=4

[4,1]=3

[4,2]=3

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter Available Matrix

3

3

2

Created Matrix (5x3)

7 4 3

1 2 2

6 0 0

0 1 1

4 3 1

P1 P3 P4 P0 P2

Safe State

**DISCUSSION:-**

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

*Source code of this experiment can be found here:*

*https://github.com/Sieunguoimay/OSLab-4thsem-2018/tree/master/Exp6*