**Experiment No. 11: LAB PROJECT**

**PROGRAM :**

#include <bits/stdc++.h>

**using** **namespace** std;

**class** **Farmer**

{

public:

**friend** **class** **manager**;

vector<int> allocation ;

vector<int>request;

Farmer()

= **default**;

Farmer(vector<int> &allocations , vector<int>& requests)

{

allocation.clear();

**for**(**auto** it : allocations)

allocation.push\_back(it);

request.clear();

**for**(**auto** it : requests)

request.push\_back(it);

}

};

**class** **manager**{

public:

int no\_of\_farmers;

int number\_of\_resources;

vector<vector<int>> allocated\_resources;

vector<vector<int>> request\_matrix;

vector<vector<int>> needed\_resources;

vector<int> available\_resources;

vector<int> finished\_allocation;

manager()

{

no\_of\_farmers = 0;

number\_of\_resources = 0;

}

manager(int resource\_no , vector<int> &values)

{

no\_of\_farmers = 0;

number\_of\_resources = resource\_no;

**for**(**auto** it : values)

{

available\_resources.push\_back(it);

}

}

void print()

{

cout << "**\n**Farmer**\t** Allocation**\t** Need**\t\t**Request**\t\t** Available**\t**";

**for** (int i = 0; i < no\_of\_farmers; i++){

cout << "**\n**F" << i + 1 << "**\t** ";

**for** (int j = 0; j < number\_of\_resources; j++)

{

cout << allocated\_resources[i][j] << " ";

}

cout << "**\t\t**";

**for** (int j = 0; j < number\_of\_resources; j++)

{

cout << needed\_resources[i][j] << " ";

}

cout << "**\t\t**";

**for** (int j = 0; j < number\_of\_resources; j++)

{

cout << request\_matrix[i][j] << " ";

}

cout << "**\t\t** ";

**if** (i == 0)

{

**for** (int j = 0; j < number\_of\_resources; j++)

cout << available\_resources[j] << " ";

}

}

cout<<"**\n**";

}

bool deadlock\_check()

{

finished\_allocation.clear();

finished\_allocation.resize(no\_of\_farmers);

finished\_allocation.assign(no\_of\_farmers , 0);

int i , j , flag = 1;

int n = no\_of\_farmers , r = number\_of\_resources;

vector<vector<int>> need\_resource\_temp = needed\_resources;

vector<int> avail\_resource = available\_resources;

vector<vector<int>> allocated\_resource\_temp = allocated\_resources;

**while** (flag)

{

flag = 0;

**for** (i = 0; i < n; i++)

{

int c = 0;

**for** (j = 0; j < r; j++)

{

**if** ((finished\_allocation[i] == 0) && (need\_resource\_temp[i][j] <= avail\_resource[j]))

{

c++;

**if** (c == r)

{

**for** (int k = 0; k < r; k++)

{

avail\_resource[k] += allocated\_resource\_temp[i][j];

finished\_allocation[i] = 1;

flag = 1;

}

**if** (finished\_allocation[i] == 1)

{

i = n;

}

}

}

}

}

}

j = 0;

flag = 0;

**for** (i = 0; i < n; i++)

{

**if** (finished\_allocation[i] == 0)

{

j++;

flag = 1;

}

}

**return** flag;

}

void add\_farmer(Farmer &f1)

{

no\_of\_farmers++;

allocated\_resources.push\_back(f1.allocation);

request\_matrix.push\_back(f1.request);

vector<int> need;

need.reserve(number\_of\_resources);

need.reserve(number\_of\_resources);

**for**(int i = 0 ; i < number\_of\_resources ; i++)

{

need.push\_back(f1.request[i] - f1.allocation[i]);

}

needed\_resources.push\_back(need);

need.clear();

}

void safe\_sequence() **const**

{

int n = no\_of\_farmers;

int r = number\_of\_resources;

vector<bool> finish(n);

int safeSeq[n];

int work[r];

vector<int> avail = available\_resources;

**for** (int i = 0; i < r ; i++)

work[i] = avail[i];

int count = 0;

vector<vector<int>> need = needed\_resources;

vector<vector<int>> allot = allocated\_resources;

**while** (count < n)

{

bool found = false;

**for** (int p = 0; p < n; p++)

{

**if** (finish[p] == 0)

{

int j;

**for** (j = 0; j < r; j++)

**if** (need[p][j] > work[j])

**break**;

**if** (j == r)

{

**for** (int k = 0 ; k < r ; k++)

work[k] += allot[p][k];

safeSeq[count++] = p;

finish[p] = true;

found = true;

}

}

}

**if** (!found)

{

cout << "System is not in safe state";

}

}

**for** (int i = 0; i < n-1 ; i++)

cout << "Farmer"<<safeSeq[i]+1 <<" -> ";

cout<<"Farmer"<<safeSeq[n-1] + 1<<"**\n**";

}

void remove\_deadlock()

{

vector<int> max\_possible = available\_resources;

**for**(int j = 0 ; j < no\_of\_farmers ; j++)

{

**for**(int k = 0 ; k < number\_of\_resources ; k++)

{

max\_possible[k] += allocated\_resources[j][k];

}

}

**for**(int j = 0 ; j < no\_of\_farmers ; j++)

{

**for**(int k = 0 ; k < number\_of\_resources ; k++)

{

**if**(request\_matrix[j][k] > max\_possible[k])

{

cout<<"Deadlock cannot be Removed as one or more farmers are requesting resources greater than the overall available resources.**\n**";

**return**;

}

}

}

int ans = (1<<no\_of\_farmers);

ans--;

int cnt = no\_of\_farmers;

vector<vector<int>> temp = allocated\_resources;

vector<int> avail\_rs = available\_resources;

vector<vector<int>> need = needed\_resources;

**for**(int i = 0 ; i <(1<<no\_of\_farmers) ; i++)

{

allocated\_resources = temp;

available\_resources = avail\_rs;

needed\_resources = need;

**for**(int j = 0 ; j < no\_of\_farmers ; j++)

{

**if**(i & (1<<j))

{

**for**(int k = 0 ; k < number\_of\_resources ; k++)

{

available\_resources[k] += allocated\_resources[j][k];

allocated\_resources[j][k] = 0;

needed\_resources[j][k] = request\_matrix[j][k];

}

}

}

**if**(!deadlock\_check())

{

int mini = 0;

**for**(int j = 0 ; j < no\_of\_farmers ; j++)

{

**if**(i & (1<<j))

{

mini++;

}

}

**if**(mini < cnt){

cnt = mini;

ans = i;

}

}

}

cout<<"Deallocate all Resources from following farmers :**\n**";

**for**(int j = 0 ; j < no\_of\_farmers ; j++)

{

**if**(ans & (1<<j))

{

cout<<"Farmer"<<j+1<<" ";

}

}

cout<<endl;

available\_resources = avail\_rs;

allocated\_resources = temp;

needed\_resources = need;

**for**(int j = 0 ; j < no\_of\_farmers ; j++)

{

**if**(ans & (1<<j))

{

**for**(int k = 0 ; k < number\_of\_resources ; k++)

{

available\_resources[k] += allocated\_resources[j][k];

allocated\_resources[j][k] = 0;

needed\_resources[j][k] = request\_matrix[j][k];

}

}

}

cout<<"After De-allocation , data is :**\n**";

print();

safe\_sequence();

available\_resources = avail\_rs;

allocated\_resources = temp;

needed\_resources = need;

}

};

int main()

{

cout<<"**\n**Enter the number of resources: ";

int number\_of\_resources;

cin>>number\_of\_resources;

cout<<"**\n**Enter the available resources:**\n**";

vector<int> avail\_resources(number\_of\_resources);

**for**(int i=0; i < number\_of\_resources; i+=1){

cin>>avail\_resources[i];

}

manager man(number\_of\_resources , avail\_resources);

int x = 1;

cout<<"Press 1 for adding farmer**\n**Press 2 for checking safe state and getting safe sequence**\n**Press 3 to print available date**\n**Press 4 to exit.**\n** ";

cin>>x;

vector<Farmer> farmers;

vector<int> allocated(number\_of\_resources) , request(number\_of\_resources);

**while**( x< 4)

{

**if**(x == 1)

{

cout<<"Allocation Matrix of Farmer **\n**";

**for**(int i = 0 ; i < number\_of\_resources ; i++)

{

cin>>allocated[i];

}

cout<<"Request Matrix of Farmer **\n**";

**for**(int i = 0 ; i < number\_of\_resources ; i++)

{

cin>>request[i];

}

Farmer f(allocated , request);

farmers.push\_back(f);

man.add\_farmer(f);

cout<<"Press 1 for adding farmer**\n**Press 2 for checking safe state and getting safe sequence**\n**Press 3 to print available date**\n**Press 4 to exit. **\n**";

cin>>x;

}

**else** **if**(x == 2)

{

**if**(!man.deadlock\_check()){

cout<<"No deadlock**\n**Safe Sequence is :**\n**";

man.safe\_sequence();

}

**else**

{

cout<<"Deadlock Occurs :**\n**";

man.remove\_deadlock();

}

cout<<"Press 1 for adding farmer**\n**Press 2 for checking safe state and getting safe sequence**\n**Press 3 to print available date**\n**Press 4 to exit.**\n** ";

cin>>x;

}

**else** **if**(x == 3)

{

man.print();

cout<<"Press 1 for adding farmer**\n**Press 2 for checking safe state and getting safe sequence**\n**Press 3 to print available date**\n**Press 4 to exit.**\n** ";

cin>>x;

}

**else**

{

**break**;

}

}

**return** 0;

}

**Sample Input:**

**3**

**3 3 2**

**1**

**0 1 0**

**7 5 3**

**1**

**2 0 0**

**3 2 2**

**1**

**3 0 2**

**9 0 2**

**1**

**2 1 1**

**2 2 2**

**1**

**0 0 2**

**4 3 3**

**3**

**2**

**1**

**0 0 0**

**10 5 7**

**3**

**2**

**4**

**ScreenShot:**





