OPERATING SYSTEMS

CECSC09 - 1

Submitted by :-

NAME: Harshit Gupta

ROLL NO: 2019UCO1580

Program 7

To show the working of deadlock avoidance and deadlock detection through Banker's algorithm

- Banker's algorithm is the algorithm used for the avoidance of deadlocks in system
- It is used for avoiding deadlocks in system which contains multiple instances of single resource types
- It has a worst case run time complexity of $O(mn^2)$ where m is the number of different resource types available in the system and n is the number of processes that the system contains at the present instant.

SYSTEM STATE

- There are 5 processes in the system and 3 resources in the system with 10, 5 and 7 instances.

	Allocated	*Max Required*	*Available*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

CODE

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vector<int> > vvi;

// declare variables
int n,m;
vvi max_p(100,vi(100));
vvi allocated(100,vi(100));
vvi needs(100,vi(100));
vector <int> max_resources(100);
vector <int> available(100);

// returns the safe sequence, if present...
list<int> get_safe_sequence(){

     list<int> result; // first element denotes whether we found or not

    // start with safety algorithm

    // generate finish array
    bool finish[n] = {false};
    int count = n; // all unfinished

    // generate work array
    vector<int> work;
    work.reserve(m);
    for(int i=0;i<m;i++)
        work.push_back(available[i]);

    int less;
    bool found;
    while(count > 0){
        found = false;
        for(int i=0;i<n;i++){
            if(finish[i] == false){
```

```
                // found unfinished process
                less = 0;

                // compare Needi and work
                for(int j=0;j<m;j++){
                    if(needs[i][j] <= work[j]) less++;
                }
                // if all of them are lesser
                if(less == m){
                    // reclaim resources
                    found = true;
                    for(int j=0;j<m;j++)
                        work[j]+= allocated[i][j];

                    //process is finished
                    result.push_back(i);
                    finish[i] = true;
                    count--;
                }

                else {
                    continue;
                }
            }
        }
        if(found == false)
            break;
    }

    // if result is consisting of all processes
    if(result.size() == n){
        result.push_front(1);
        return result;
    }

    // if result does not contain all processes
    else{
        result.push_front(-1);
        return result;
```

```
    }

}

int main(){

    ios::sync_with_stdio(0);
    cout<<"Simulating Banker's algorithm...\n";
    cout<<"Enter the total number of processes in the system :";
    cin>>n;
    cout<<"Enter the total number of resources in the system :";
    cin>>m;


    int rj;

    cout<<"Enter the MAXIMUM availability of the resources :\n";
    for(int i=0;i<m;i++){
        cout<<"R"<<i<<":";
        cin>>rj;
        max_resources[i] = rj;
    }
    cout<<"\nEnter the CURRENT availability of the resources :\n";
    for(int i=0;i<m;i++){
        cout<<"R"<<i<<":";
        cin>>rj;
        available[i] = rj;
    }

    cout<<"PROCESSES' INFORMATION \n\n";

    cout<<"Enter the MAXIMUM possible resources for processes\n";
    for(int i=0;i<n;i++){
        cout<<"For process P"<<i<<":\n";
        for(int j=0;j<m;j++){
            cout<<"R"<<j<<":";
            cin>>rj;
            max_p[i][j] = rj;
        }
```

```
    }

    cout<<"Enter the CURRENT allocated resources";
    for(int i=0;i<n;i++){
        cout<<"For process P"<<i<<":\n";
        for(int j=0;j<m;j++){
            cout<<"R"<<j<<":";
            cin>>rj;
            allocated[i][j] = rj;
        }
    }

    // calculate the need matrix..

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            needs[i][j] = max_p[i][j] - allocated[i][j];
        }
    }

    // show processes in system...
    cout<<"PROCESSES \t ALLOCATED \t MAXIMUM \t AVAILABLE RESOURCES \n";
    for(int i=0;i<n;i++){
        cout<<"     P"<<i<<"\t\t ";
        for(int j=0;j<m;j++){
            cout<<allocated[i][j]<<" ";
        }
        cout<<"\t\t";
        for(int j=0;j<m;j++){
            cout<<max_p[i][j]<<" ";
        }
        cout<<"\t\t";
        if(i==0)
        {for(int j=0;j<m;j++){
            cout<<available[j]<<" ";
        }}

        cout<<endl;
    }
```

```
// now calculate a safe sequence
list<int> safe = get_safe_sequence();

// first element would be 1 or -1

// 1 means -> got a safe sequence
// -1 means -> do not have a safe sequence in this state
auto it = safe.begin();
if(*it == 1){
    cout<<"Safe sequence exists!\n";
    cout<<"SAFE SEQUENCE : ";

    // move to second element
    it++;

    while(it!=safe.end())
        {cout<<"P"<<*it<<" ";
        it++;}
    cout<<endl;
}
else
    cout<<"Sorry, safe sequence does not exist.\n";
return 0;}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Lab4> ./a
Simulating Banker's algorithm...
Enter the total number of processes in the system :5
Enter the total number of resources in the system :3
Enter the MAXIMUM availability of the resources :
R0:10
R1:5
R2:7

Enter the CURRENT availability of the resources :
R0:3
R1:3
R2:2
PROCESSES' INFORMATION

Enter the MAXIMUM possible resources for processes
For process P0:
R0:7
R1:5
R2:3
For process P1:
R0:3
R1:2
R2:2
```

```
For process P2:
R0:9
R1:0
R2:2
For process P3:
R0:2
R1:2
R2:2
For process P4:
R0:4
R1:3
R2:3
Enter the CURRENT allocated resourcesFor process P0:
R0:0
R1:1
R2:0
For process P1:
R0:2
R1:0
R2:0
For process P2:
R0:3
R1:0
R2:2
```

```
For process P3:
R0:2
R1:1
R2:1
For process P4:
R0:0
R1:0
R2:2
PROCESSES        ALLOCATED       MAXIMUM         AVAILABLE RESOURCES
     P0          0 1 0           7 5 3           3 3 2
     P1          2 0 0           3 2 2
     P2          3 0 2           9 0 2
     P3          2 1 1           2 2 2
     P4          0 0 2           4 3 3
Safe sequence exists!
SAFE SEQUENCE : P1 P3 P4 P0 P2
```