Assignment 5

Name: Kumar Deshmukh

Roll: 33143

Title: Banker's Algorithm

Problem Statement:

Implement C program for Deadlock Avoidance: Banker's Algorithm

Theory:

Banker's algorithm

Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not. Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank; it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Characteristics of Banker's Algorithm:

1. If any process requests for a resource, then it has to wait.
2. This algorithm consists of advanced features for maximum resource allocation.
3. There are limited resources in the system we have.
4. In this algorithm, if any process gets all the needed resources, then it is that it should return the resources in a restricted period.
5. Various resources are maintained in this algorithm that can fulfill the needs of at least one client.

Some data structures that are used to implement the banker's algorithm are:

1. Available

It is an array of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type Rj.

2. Max

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process Pi can request at most k instances of resource type Rj.

3. Allocation

It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process Pi is currently allocated k instances of resource type Rj.

4. Need

It is a two-dimensional array. It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process Pi may need k more instances of resource type Rj to complete its task.

Need[i][j] = Max[i][j] - Allocation [i][j]

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state 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) Needi <= Work

   if no such i exists goto step (4)

3) Work = Work + Allocation[i]

   Finish[i] = true

   goto step (2)

4) if Finish [i] = true for all i

Resource-Request Algorithm

Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

1. If Requesti <= Needi

   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If Requesti <= Available

   Goto step (3); otherwise, Pi must wait, since the resources are not available.

3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

   Available = Available – Requesti

   Allocationi = Allocationi + Requesti

   Needi = Needi– Requesti

Example:

Input:

Total Resources	R1	R2	R3
	10	5	7

Process	Allocation			Max		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3
P2	2	0	0	3	2	2
P3	3	0	2	9	0	2
P4	2	1	1	2	2	2

Output:

Safe sequences are:

P2--> P4--> P1--> P3

P2--> P4--> P3--> P1

P4--> P2--> P1--> P3

P4--> P2--> P3--> P1

There are total 4 safe-sequences

Explanation:

Total resources are R1 = 10, R2 = 5, R3 = 7 and allocated resources are R1 = (0+2+3+2 =) 7, R2 = (1+0+0+1 =) 2, R3 = (0+0+2+1 =) 3. Therefore, remaining resources are R1 = (10 – 7 =) 3, R2 = (5 – 2 =) 3, R3 = (7 – 3 =) 4.

Remaining available = Total resources – allocated resources

and

Remaining need = max – allocated

Process	Max			Allocation			Available			Needed		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	7	5	3	0	1	0	3	3	4	7	4	3
P2	3	2	2	2	0	0				1	2	2
P3	9	0	2	3	0	2				6	0	0
P4	2	2	2	2	1	1				0	1	1
				7	2	3						

So, we can start from either P2 or P4. We can not satisfy remaining need from available resources of either P1 or P3 in first or second attempt step of Banker's algorithm. There are only four possible safe sequences. These are :

P2–> P4–> P1–> P3

P2–> P4–> P3–> P1

P4–> P2–> P1–> P3

P4–> P2–> P3–> P1

CODE:

```
#include <iostream>
using namespace std;

int main()
{
    int P, R;
    cout << "Enter number of Processes (P): ";
    cin >> P;
    cout << "Enter number of Resources (R): ";
    cin >> R;

    int i,j,k;

    int Allocated[P][R] = {};
    cout<<"\nReading 'Allocated' matrix ::"<<endl;
    for(i=0; i<P; i++){
        cout<<"For P"<<i<<":"<<endl;
        for(j=0; j<R; j++){
            cout<<"Enter allocated of resource R"<<j<<": ";
            cin>>Allocated[i][j];
        }
    }

    int Max[P][R];
    cout<<"\nReading 'Max' matrix ::"<<endl;
    for(i=0;i<P; i++){
        cout<<"For P"<<i<<"::"<<endl;
        for(j=0; j<R; j++){
            cout<<"Enter max need of resource R"<<j<<": ";
            cin>>Max[i][j];
        }
    }
    cout<<endl;
    int avail[R];
    for(i=0; i<R; i++){
        cout<<"Resource R"<<i<<" available: ";
        cin>>avail[i];
    }


    int need[P][R];
    int finished[P]={0}, SafeSeq[P];

    int ind = 0;

    for (i = 0; i < P; i++){
        for (j = 0; j < R; j++)
            need[i][j] = Max[i][j] - Allocated[i][j];
    }

    int y = 0;
    for (k = 0; k < 5; k++){
        for (i = 0; i < P; i++){
            if (finished[i] == 0){
                int flag = 0;
```

```
                for (j = 0; j < R; j++)
                {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                        break;
                    }
                }

                if (flag == 0){
                    SafeSeq[ind++] = i;
                    for (y = 0; y < R; y++){
                        avail[y] += Allocated[i][y];
                    }
                    finished[i] = 1;
                }
            }
        }
    }

    bool isSafe = true;
    for(i=0; i<P; i++){
        if(SafeSeq[i]<0 || SafeSeq[i]>P-1){
            isSafe = false;
            break;
        }
    }

    if(isSafe){
        cout << "\nSAFE Sequence is::" << endl;
        for (i = 0; i < P - 1; i++)
            cout << " P" << SafeSeq[i] << " ->";
        cout << " P" << SafeSeq[P - 1] << endl;
    }
    else
        cout<<"\nDEADLOCK occured"<<endl;

    return 0;
}
```

OUTPUT:

```
(base) kumar@pop-os:~/Desktop/OS/Assignments$ g++ Assig5.cpp
(base) kumar@pop-os:~/Desktop/OS/Assignments$ ./a.out
Enter number of Processes (P): 5
Enter number of Resources (R): 3

Reading 'Allocated' matrix ::
For P0:
Enter allocated of resource R0: 0
Enter allocated of resource R1: 1
Enter allocated of resource R2: 0
For P1:
Enter allocated of resource R0: 2
Enter allocated of resource R1: 0
Enter allocated of resource R2: 0
For P2:
Enter allocated of resource R0: 3
Enter allocated of resource R1: 0
Enter allocated of resource R2: 2
For P3:
Enter allocated of resource R0: 2
Enter allocated of resource R1: 1
Enter allocated of resource R2: 1
For P4:
Enter allocated of resource R0: 0
Enter allocated of resource R1: 0
Enter allocated of resource R2: 2

Reading 'Max' matrix ::
For P0::
Enter max need of resource R0: 7
Enter max need of resource R1: 5
Enter max need of resource R2: 3
For P1::
Enter max need of resource R0: 3
Enter max need of resource R1: 2
Enter max need of resource R2: 2
For P2::
Enter max need of resource R0: 9
Enter max need of resource R1: 0
Enter max need of resource R2: 2
For P3::
Enter max need of resource R0: 2
Enter max need of resource R1: 2
Enter max need of resource R2: 2
For P4::
Enter max need of resource R0: 4
Enter max need of resource R1: 3
Enter max need of resource R2: 3

Resource R0 available: 3
Resource R1 available: 3
Resource R2 available: 2

SAFE Sequence is::
 P1 -> P3 -> P4 -> P0 -> P2
```

```
(base) kumar@pop-os:~/Desktop/OS/Assignments$ ./a.out
Enter number of Processes (P): 5
Enter number of Resources (R): 3

Reading 'Allocated' matrix ::
For P0:
Enter allocated of resource R0: 0
Enter allocated of resource R1: 1
Enter allocated of resource R2: 0
For P1:
Enter allocated of resource R0: 2
Enter allocated of resource R1: 0
Enter allocated of resource R2: 0
For P2:
Enter allocated of resource R0: 3
Enter allocated of resource R1: 0
Enter allocated of resource R2: 2
For P3:
Enter allocated of resource R0: 2
Enter allocated of resource R1: 1
Enter allocated of resource R2: 1
For P4:
Enter allocated of resource R0: 0
Enter allocated of resource R1: 0
Enter allocated of resource R2: 2

Reading 'Max' matrix ::
For P0::
Enter max need of resource R0: 7
Enter max need of resource R1: 5
Enter max need of resource R2: 3
For P1::
Enter max need of resource R0: 3
Enter max need of resource R1: 2
Enter max need of resource R2: 2
For P2::
Enter max need of resource R0: 9
Enter max need of resource R1: 0
Enter max need of resource R2: 2
For P3::
Enter max need of resource R0: 2
Enter max need of resource R1: 2
Enter max need of resource R2: 2
For P4::
Enter max need of resource R0: 4
Enter max need of resource R1: 3
Enter max need of resource R2: 3

Resource R0 available: 3
Resource R1 available: 3
Resource R2 available: 0

DEADLOCK occured
(base) kumar@pop-os:~/Desktop/OS/Assignments$
```