Student name: Rudra Kaniya
Student ID: 11803187
Email Address: rudra.kaniya.rk@gmail.com
GitHub Link: https://github.com/Rudrakaniya/

Question: *(Problem Statement 11)*
Reena’s operating system uses an algorithm for deadlock avoidance to manage the allocation of resources say three namely A, B, and C to three processes P0, P1, and P2. Consider the following scenario as reference .user must enter the current state of system as given in this example:

Suppose P0 has 0,0,1 instance, P1 is having 3,2,0 instances and P2 occupies 2,1,1 instances of A, B, C resource respectively.
Also, the maximum number of instances required for P0 is 8,4,3 and for p1 is 6,2,0 and finally for P2 there are 3,3,3 instances of resources A, B, C respectively. There are 3 instances of resource A, 2 instances of resource B and 2 instances of resource C available. Write a program to check whether Reena’s operating system is in a safe state or not in the following independent requests for additional resources in the current state:

1. Request1: P0 requests 0 instances of A and 0 instances of B and 2 instances of C.
2. Request2: P1 requests for 2 instances of A, 0 instances of B and 0 instances of C.

All the request must be given by user as input.

Code: *Mention solution code assigned to you*
1. Explain the problem in terms of operating system concept? (Max 200 word)

Description:

- **Banker’s Algorithm**
  Banker’s Algorithm is a deadlock avoidance strategy. It is called so because it is used in banking systems to decide whether a loan can be granted or not.

  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.

  Banker's algorithm works in a similar way in computers.

  Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

Data structures:

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

- **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 R(j).

- **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 P(i) can request at most k instances of resource type R(j).

- **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 P(i) is currently allocated k instances of resource type R(j).

- **Need**
  It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process P(i) may need k more instances of resource type R(j) to complete its task.

Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively. Initially,

   Work = Available
   Finish[i] = false for i = 0, 1, ... , n - 1.

   This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index i such that both

   Finish[i] ==false
   Need i <= Work

   If there is no such i present, then proceed to step 4.

   It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:

   Work = Work + Allocation;
   Finish[i] = true;
   Go to step 2.

   When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If *Finish[i] == true*
   for all i, then the system is in a safe state.
   That means if all processes are finished, then the system is in safe state.

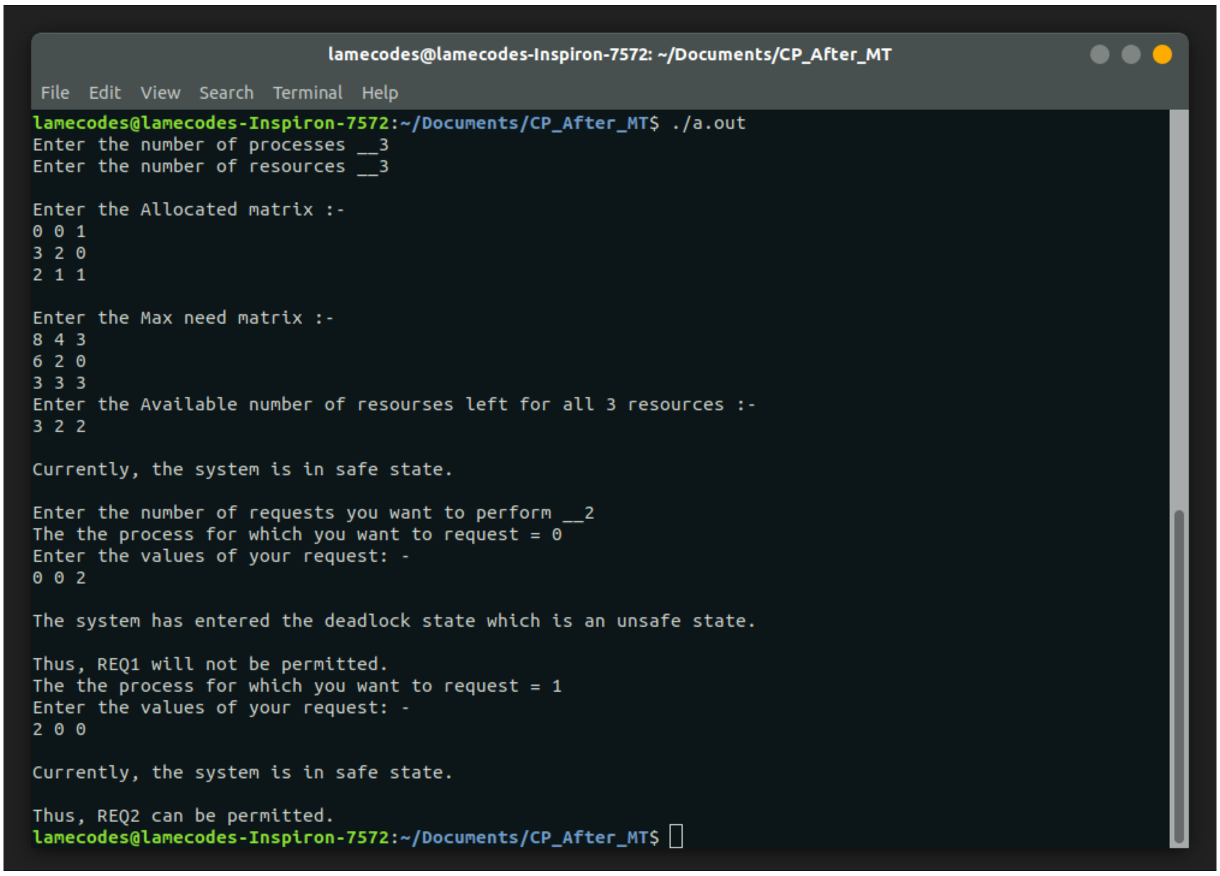
Code snippet:

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef vector< vector<int> > Matrix;
bool fredo = false;

bool bankerHere(int processes, int resources, Matrix& need, vector<int> avail, Matrix& alloc, int totalResources){
    vector<bool> executed(processes, false);
    for (int k = 0; k < processes; k++)
    {
        for (int i = 0; i < processes; i++) {
            if (executed[i] == false) {

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

                if (flag == 0) {
                    for (int y = 0; y < resources; y++){
                        avail[y] += alloc[i][y];
                    }
                    executed[i] = true;
                    //cout << "a" << endl;
                }
            }
        }
    }

    int t{0};
    for (int i = 0; i < resources; ++i)
    {
        t += avail[i];
    }

    if(t == totalResources){
        cout << endl << "Currently, the system is in safe state." << endl;
        return true;
    }
    else
    {
        cout << endl << "The system has entered the deadlock state which is an unsafe state." << endl;
        //cout << "t = " << t << " total = " << totalResources << endl;
        return false;
    }
}

int32_t main(){
    int processes, resources, totalResources{0};;
    cout << "Enter the number of processes __";
    cin >> processes;

    cout << "Enter the number of resources __";
    cin >> resources;

    Matrix alloc(processes, vector<int>(resources));
    Matrix max(processes, vector<int>(resources));
    Matrix req(processes, vector<int>(resources));

    cout << endl << "Enter the Allocated matrix :- "<<endl;

    for (int i = 0; i < processes; ++i){
        for (int j = 0; j < resources; ++j){
            cin >> alloc[i][j];
            totalResources += alloc[i][j];
        }
    }
    cout << endl << "Enter the Max need matrix :-" << endl;

    for (int i = 0; i < processes; ++i){
        for (int j = 0; j < resources;++j){
            cin >> max[i][j];
            req[i][j] = max[i][j] - alloc[i][j];
        }
    }

    vector<int> avail(resources);
    cout << "Enter the Available number of resourses left for all " << resources << " resources :- " << endl;
    for (int i = 0; i < resources;++i){
        cin >> avail[i];
        totalResources += avail[i];
    }

    bool boo = bankerHere(processes, resources, req , avail, alloc, totalResources);

    int noReq;
    cout << endl
         << "Enter the number of requests you want to perform __";
    cin >> noReq;

    for (int i = 0; i < noReq; ++i){
        Matrix vv = alloc;
        Matrix xvv = req;
        vector<int> anvi = avail;
        int pp, tr = totalResources;
        cout << "The the process for which you want to request = ";
        cin >> pp;
                cout << "Enter the values of your request: - " << endl;
                int sg[3];
                for (int l = 0; l < 3; ++l){
                    cin >> sg[l];
                    vv[pp][l] += sg[l];
                    xvv[pp][l] -= sg[l];
                    anvi[l] -= sg[l];
                }
                boo = bankerHere(processes, resources, xvv , anvi, vv, tr);

                if(boo){
                    cout << endl
                         << "Thus, REQ" << i + 1 << " can be permitted." << endl;

                }else{
                    cout << endl
                         << "Thus, REQ" << i + 1 << " will not be permitted." << endl;
                }
            }

                return 0;
        }
```

Input:

- Enter your allocated matrix
- Enter the Max need matrix
- Enter the value of available instances of all the resources
- Enter the number of requests
- Enter the request.

Output:


lamecodes@lamecodes-Inspiron-7572: ~/Documents/CP_After_MT
File Edit View Search Terminal Help

```
lamecodes@lamecodes-Inspiron-7572:~/Documents/CP_After_MT$ ./a.out
Enter the number of processes __3
Enter the number of resources __3

Enter the Allocated matrix :-
0 0 1
3 2 0
2 1 1

Enter the Max need matrix :-
8 4 3
6 2 0
3 3 3
Enter the Available number of resourses left for all 3 resources :-
3 2 2

Currently, the system is in safe state.

Enter the number of requests you want to perform __2
The the process for which you want to request = 0
Enter the values of your request: -
0 0 2

The system has entered the deadlock state which is an unsafe state.

Thus, REQ1 will not be permitted.
The the process for which you want to request = 1
Enter the values of your request: -
2 0 0

Currently, the system is in safe state.

Thus, REQ2 can be permitted.
lamecodes@lamecodes-Inspiron-7572:~/Documents/CP_After_MT$ 
```

As clearly seen by the output that,

- **Request 1** will not be permitted to execute. And,
- **Request 2** will be allowed to permit.

GitHub Link:

https://github.com/Rudrakaniya/OS_Assignment2020