Banker's Algorithm

Perumalla Dharan
AP21110010201

Q) Write a C++ program to simulate the Bankers algorithm for the purpose of deadlock avoidance.

CODE :

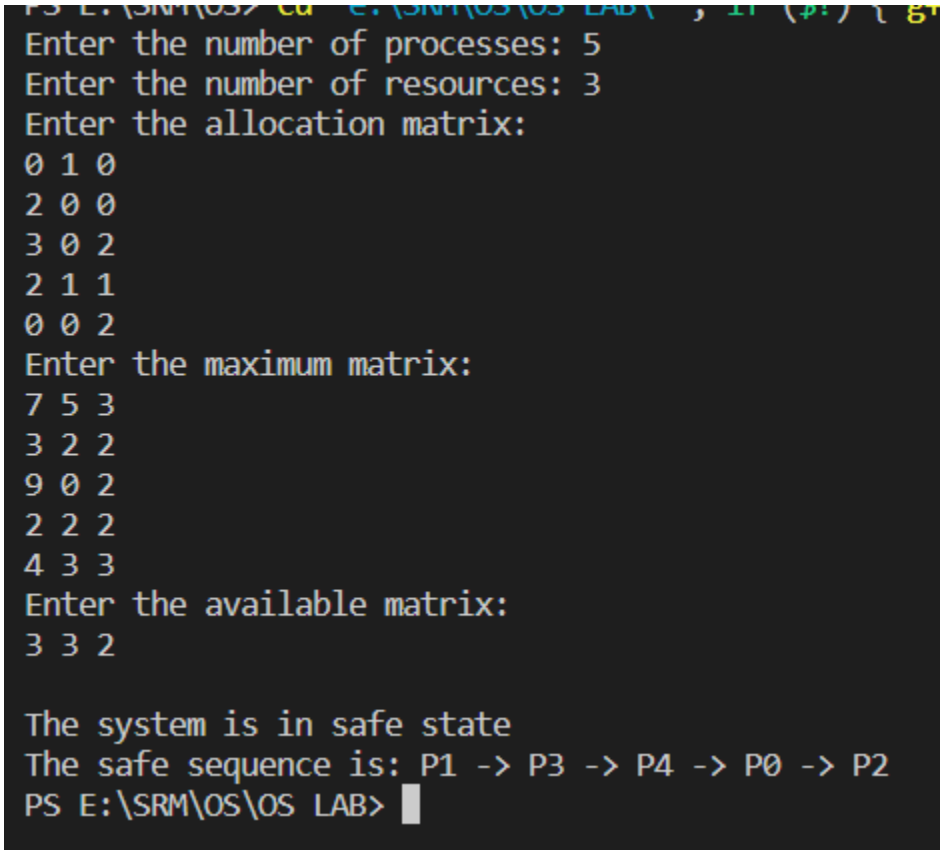
```
// Initialising the header files
#include <iostream>
using namespace std;

// Main function
int main()
{
    // Taking the input
    int n, m;
    cout << "Enter the number of processes: ";
    cin >> n;
    cout << "Enter the number of resources: ";
    cin >> m;

    // Declaring the allocation, maximum, available, need
matrix
    int alloc[n][m], max[n][m], avail[m], need[n][m];
    cout << "Enter the allocation matrix: " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cin >> alloc[i][j];
        }
```

```
    }
    cout << "Enter the maximum matrix: " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cin >> max[i][j];
        }
    }
    cout << "Enter the available matrix: " << endl;
    for (int i = 0; i < m; i++)
    {
        cin >> avail[i];
    }

    // Calculating the need matrix
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    // Declaring the work and finish array
    int work[m];
    for (int i = 0; i < m; i++)
    {
        work[i] = avail[i];
    }
    bool finish[n];
    for (int i = 0; i < n; i++)
    {
```

```
        finish[i] = false;
    }

    // Declaring the safe sequence array
    int safeSeq[n];
    int count = 0;
    for (int k = 0; k < n; k++)
    {
        for (int i = 0; i < n; i++)
        {
            if (finish[i] == false)
            {
                bool flag = true;
                for (int j = 0; j < m; j++)
                {
                    if (need[i][j] > work[j])
                    {
                        flag = false;
                        break;
                    }
                }
                if (flag)
                {
                    for (int j = 0; j < m; j++)
                    {
                        work[j] += alloc[i][j];
                    }
                    finish[i] = true;
                    safeSeq[count++] = i;
                }
            }
        }
    }
```

```
    // Checking if the system is in a safe state or not
    bool safe = true;
    for (int i = 0; i < n; i++)
    {
        if (finish[i] == false)
        {
            safe = false;
            break;
        }
    }

    // Printing the safe sequence
    if (safe)
    {
        cout << endl;
        cout << "The system is in the safe state" << endl;
        cout << "The safe sequence is: ";
        for (int i = 0; i < n - 1; i++)
        {
            cout << "P" << safeSeq[i] << " -> ";
        }
        cout << "P" << safeSeq[n - 1];
        cout << endl;
    }
    else
    {
        cout << "The system is not in a safe state" << endl;
    }

    return 0;
}
```

OUTPUT :


Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the available matrix:
3 3 2
The system is in safe state
The safe sequence is: P1 -> P3 -> P4 -> P0 -> P2
PS E:\SRM\OS\OS LAB>