Practical - 5

Name: Sakshi Deshmukh
PRN: 202301040191
Roll no. 158

Implement Banker algorithm for deadlock handling.

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, m;
vector<int> available;
vector<vector<int>> maximum;
vector<vector<int>> allocation;
vector<vector<int>> need;
void calculateNeed()
{
    need.assign(n, vector<int>(m));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}
bool isSafe(vector<int> &safeSequence)
{
    vector<int> work = available;
    vector<bool> finish(n, false);
    safeSequence.clear();
    int count = 0;
    while (count < n)
```

```
    {
        bool found = false;

        for (int i = 0; i < n; i++)
        {
            if (!finish[i])
            {
                bool canAllocate = true;
                for (int j = 0; j < m; j++)
                {
                    if (need[i][j] > work[j])
                    {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate)
                {
                    for (int j = 0; j < m; j++)
                    {
                        work[j] += allocation[i][j];
                    }
                    safeSequence.push_back(i);
                    finish[i] = true;
                    count++;
                    found = true;
                }
            }
        }
        if (!found)
        {
            return false;
        }
    }
    return true;
}
bool requestResources(int processId, vector<int> &request)
{
    for (int i = 0; i < m; i++)
    {
```

```
        if (request[i] > need[processId][i])
        {

            cout << "error: process has exceeded its maximum claim" <<
endl;
            return false;
        }
    }
    for (int i = 0; i < m; i++)
    {
        if (request[i] > available[i])
        {
            cout << "process must wait - resources not available" << endl;
            return false;
        }
    }
    for (int i = 0; i < m; i++)
    {
        available[i] -= request[i];
        allocation[processId][i] += request[i];
        need[processId][i] -= request[i];
    }
    vector<int> safeSeq;
    if (isSafe(safeSeq))
    {
        cout << "request granted - system remains in safe state" << endl;
        return true;
    }
    else
    {
        for (int i = 0; i < m; i++)
        {
            available[i] += request[i];
            allocation[processId][i] -= request[i];
            need[processId][i] += request[i];
        }
        cout << "request denied - would lead to unsafe state" << endl;
        return false;
    }
}
```

```
bool releaseResources(int processId, vector<int> &release)
{

    for (int i = 0; i < m; i++)
    {
        if (release[i] > allocation[processId][i])
        {
            cout << "error: process trying to release more than allocated"
                 << endl;
            return false;
        }
    }
    for (int i = 0; i < m; i++)
    {
        available[i] += release[i];
        allocation[processId][i] -= release[i];
        need[processId][i] += release[i];
    }
    cout << "resources released successfully" << endl;
    return true;
}
void displayState()
{
    cout << "\ncurrent system state:" << endl;
    cout << "available: ";
    for (int i = 0; i < m; i++)
        cout << available[i] << " ";
    cout << endl;
    cout << "\nallocation matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "p" << i << ": ";
        for (int j = 0; j < m; j++)
            cout << allocation[i][j] << " ";
        cout << endl;
    }
    cout << "\nmaximum matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "p" << i << ": ";
```

```
        for (int j = 0; j < m; j++)
            cout << maximum[i][j] << " ";

        cout << endl;
    }
    cout << "\nneed matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "p" << i << ": ";
        for (int j = 0; j < m; j++)
            cout << need[i][j] << " ";
        cout << endl;
    }
}
int main()
{
    cout << "enter number of processes: ";
    cin >> n;
    cout << "enter number of resource types: ";
    cin >> m;
    available.resize(m);
    maximum.assign(n, vector<int>(m));
    allocation.assign(n, vector<int>(m));
    cout << "\nenter available resources for each type:" << endl;
    for (int i = 0; i < m; i++)
    {
        cout << "resource " << i << ": ";
        cin >> available[i];
    }
    cout << "\nenter maximum matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "process " << i << " (max demand for each resource): ";
        for (int j = 0; j < m; j++)
        {
            cin >> maximum[i][j];
        }
    }
    cout << "\nenter allocation matrix:" << endl;
    for (int i = 0; i < n; i++)
```

```
    {
        cout << "process " << i << " (currently allocated): ";

        for (int j = 0; j < m; j++)
        {
            cin >> allocation[i][j];
        }
    }
    calculateNeed();
    displayState();
    vector<int> safeSeq;
    cout << "\n--- checking initial system safety ---" << endl;
    if (isSafe(safeSeq))
    {
        cout << "system is in safe state" << endl;
        cout << "safe sequence: ";
        for (int i = 0; i < safeSeq.size(); i++)
        {
            cout << "p" << safeSeq[i];
            if (i < safeSeq.size() - 1)
                cout << " -> ";
        }
        cout << endl;
    }
    else
    {
        cout << "system is in unsafe state" << endl;
    }
    int choice;
    while (true)
    {
        cout << "\n--- menu ---" << endl;
        cout << "1. request resources" << endl;
        cout << "2. release resources" << endl;
        cout << "3. check safe state" << endl;
        cout << "4. display current state" << endl;
        cout << "5. exit" << endl;
        cout << "enter choice: ";
        cin >> choice;
        if (choice == 1)
```

```
        {
            int pid;
            cout << "enter process id (0 to " << n - 1 << "): ";

            cin >> pid;
            if (pid < 0 || pid >= n)
            {
                cout << "invalid process id" << endl;
                continue;
            }
            vector<int> request(m);
            cout << "enter request for each resource type: ";
            for (int i = 0; i < m; i++)
                cin >> request[i];
            requestResources(pid, request);
        }
        else if (choice == 2)
        {
            int pid;
            cout << "enter process id (0 to " << n - 1 << "): ";
            cin >> pid;
            if (pid < 0 || pid >= n)
            {
                cout << "invalid process id" << endl;
                continue;
            }
            vector<int> release(m);
            cout << "enter resources to release for each type: ";
            for (int i = 0; i < m; i++)
                cin >> release[i];
            releaseResources(pid, release);
        }
        else if (choice == 3)
        {
            vector<int> seq;
            if (isSafe(seq))
            {
                cout << "system is in safe state" << endl;
                cout << "safe sequence: ";
                for (int i = 0; i < seq.size(); i++)
```

```
                {
                    cout << "p" << seq[i];
                    if (i < seq.size() - 1)
                        cout << " -> ";
                }
                cout << endl;
            }
            else
            {
                cout << "system is in unsafe state" << endl;
            }
        }
        else if (choice == 4)
        {
            displayState();
        }#include <bits/stdc++.h>
using namespace std;
int n, m;
vector<int> available;
vector<vector<int>> maximum;
vector<vector<int>> allocation;
vector<vector<int>> need;
void calculateNeed()
{
    need.assign(n, vector<int>(m));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            need[i][j] = maximum[i][j] - allocation[i][j];
        }
    }
}
bool isSafe(vector<int> &safeSequence)
{
    vector<int> work = available;
    vector<bool> finish(n, false);
    safeSequence.clear();
    int count = 0;
    while (count < n)
```

```
    {
        bool found = false;

        for (int i = 0; i < n; i++)
        {
            if (!finish[i])
            {
                bool canAllocate = true;
                for (int j = 0; j < m; j++)
                {
                    if (need[i][j] > work[j])
                    {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate)
                {
                    for (int j = 0; j < m; j++)
                    {
                        work[j] += allocation[i][j];
                    }
                    safeSequence.push_back(i);
                    finish[i] = true;
                    count++;
                    found = true;
                }
            }
        }
        if (!found)
        {
            return false;
        }
    }
    return true;
}
bool requestResources(int processId, vector<int> &request)
{
    for (int i = 0; i < m; i++)
    {
```

```
        if (request[i] > need[processId][i])
        {

            cout << "error: process has exceeded its maximum claim" <<
endl;
            return false;
        }
    }
    for (int i = 0; i < m; i++)
    {
        if (request[i] > available[i])
        {
            cout << "process must wait - resources not available" << endl;
            return false;
        }
    }
    for (int i = 0; i < m; i++)
    {
        available[i] -= request[i];
        allocation[processId][i] += request[i];
        need[processId][i] -= request[i];
    }
    vector<int> safeSeq;
    if (isSafe(safeSeq))
    {
        cout << "request granted - system remains in safe state" << endl;
        return true;
    }
    else
    {
        for (int i = 0; i < m; i++)
        {
            available[i] += request[i];
            allocation[processId][i] -= request[i];
            need[processId][i] += request[i];
        }
        cout << "request denied - would lead to unsafe state" << endl;
        return false;
    }
}
```

```
bool releaseResources(int processId, vector<int> &release)
{

    for (int i = 0; i < m; i++)
    {
        if (release[i] > allocation[processId][i])
        {
            cout << "error: process trying to release more than allocated"
                 << endl;
            return false;
        }
    }
    for (int i = 0; i < m; i++)
    {
        available[i] += release[i];
        allocation[processId][i] -= release[i];
        need[processId][i] += release[i];
    }
    cout << "resources released successfully" << endl;
    return true;
}
void displayState()
{
    cout << "\ncurrent system state:" << endl;
    cout << "available: ";
    for (int i = 0; i < m; i++)
        cout << available[i] << " ";
    cout << endl;
    cout << "\nallocation matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "p" << i << ": ";
        for (int j = 0; j < m; j++)
            cout << allocation[i][j] << " ";
        cout << endl;
    }
    cout << "\nmaximum matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "p" << i << ": ";
```

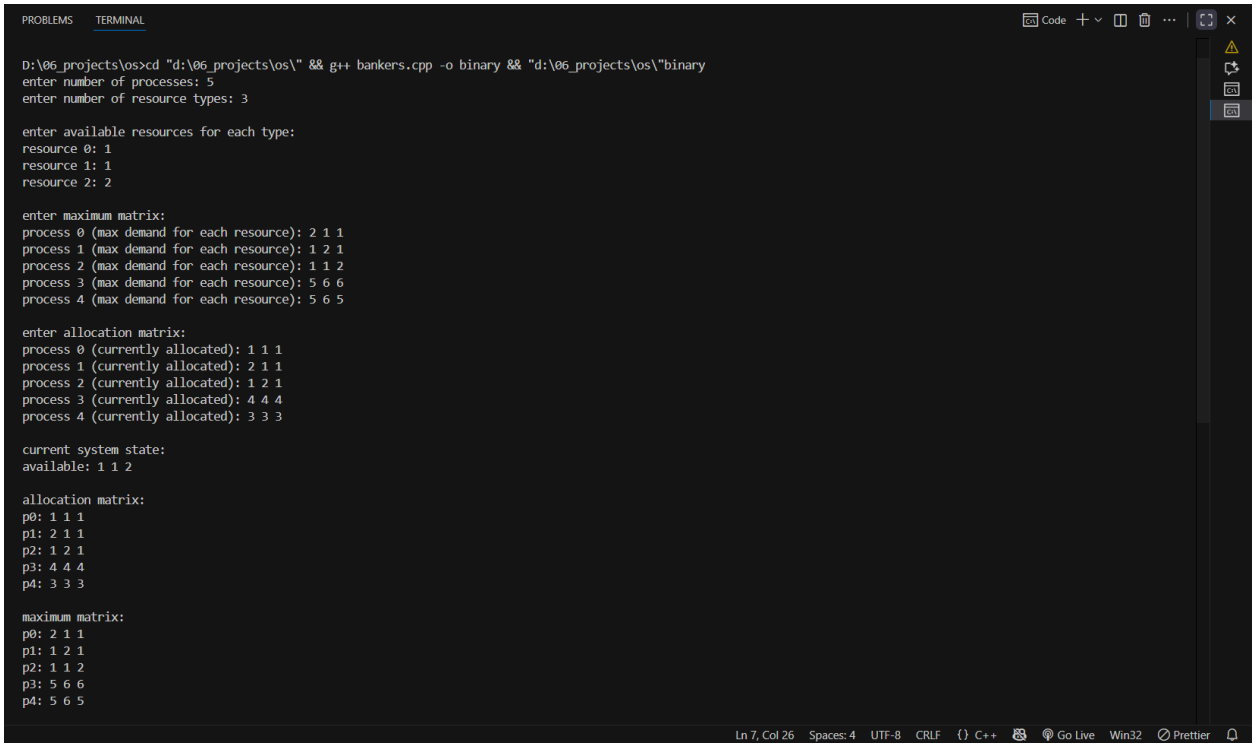
```
        for (int j = 0; j < m; j++)
            cout << maximum[i][j] << " ";

        cout << endl;
    }
    cout << "\nneed matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "p" << i << ": ";
        for (int j = 0; j < m; j++)
            cout << need[i][j] << " ";
        cout << endl;
    }
}
int main()
{
    cout << "enter number of processes: ";
    cin >> n;
    cout << "enter number of resource types: ";
    cin >> m;
    available.resize(m);
    maximum.assign(n, vector<int>(m));
    allocation.assign(n, vector<int>(m));
    cout << "\nenter available resources for each type:" << endl;
    for (int i = 0; i < m; i++)
    {
        cout << "resource " << i << ": ";
        cin >> available[i];
    }
    cout << "\nenter maximum matrix:" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << "process " << i << " (max demand for each resource): ";
        for (int j = 0; j < m; j++)
        {
            cin >> maximum[i][j];
        }
    }
    cout << "\nenter allocation matrix:" << endl;
    for (int i = 0; i < n; i++)
```

```
    {
        cout << "process " << i << " (currently allocated): ";

        for (int j = 0; j < m; j++)
        {
            cin >> allocation[i][j];
        }
    }
    calculateNeed();
    displayState();
    vector<int> safeSeq;
    cout << "\n--- checking initial system safety ---" << endl;
    if (isSafe(safeSeq))
    {
        cout << "system is in safe state" << endl;
        cout << "safe sequence: ";
        for (int i = 0; i < safeSeq.size(); i++)
        {
            cout << "p" << safeSeq[i];
            if (i < safeSeq.size() - 1)
                cout << " -> ";
        }
        cout << endl;
    }
    else
    {
        cout << "system is in unsafe state" << endl;
    }
    int choice;
    while (true)
    {
        cout << "\n--- menu ---" << endl;
        cout << "1. request resources" << endl;
        cout << "2. release resources" << endl;
        cout << "3. check safe state" << endl;
        cout << "4. display current state" << endl;
        cout << "5. exit" << endl;
        cout << "enter choice: ";
        cin >> choice;
        if (choice == 1)
```

```
        {
            int pid;
            cout << "enter process id (0 to " << n - 1 << "): ";

            cin >> pid;
            if (pid < 0 || pid >= n)
            {
                cout << "invalid process id" << endl;
                continue;
            }
            vector<int> request(m);
            cout << "enter request for each resource type: ";
            for (int i = 0; i < m; i++)
                cin >> request[i];
            requestResources(pid, request);
        }
        else if (choice == 2)
        {
            int pid;
            cout << "enter process id (0 to " << n - 1 << "): ";
            cin >> pid;
            if (pid < 0 || pid >= n)
            {
                cout << "invalid process id" << endl;
                continue;
            }
            vector<int> release(m);
            cout << "enter resources to release for each type: ";
            for (int i = 0; i < m; i++)
                cin >> release[i];
            releaseResources(pid, release);
        }
        else if (choice == 3)
        {
            vector<int> seq;
            if (isSafe(seq))
            {
                cout << "system is in safe state" << endl;
                cout << "safe sequence: ";
                for (int i = 0; i < seq.size(); i++)
```

```
                {
                    cout << "p" << seq[i];
                    if (i < seq.size() - 1)
                        cout << " -> ";
                }
                cout << endl;
            }
            else
            {
                cout << "system is in unsafe state" << endl;
            }
        }
        else if (choice == 4)
        {
            displayState();
        }
        else if (choice == 5)
        {
            break;
        }
        else
        {
            cout << "invalid choice" << endl;
        }
    }
    return 0;
}
        else if (choice == 5)
        {
            break;
        }
        else
        {
            cout << "invalid choice" << endl;
        }
    }
    return 0;
}
```

Output:


PROBLEMS
TERMINAL
Code
D:\06_projects\os>cd "d:\06_projects\os\" && g++ bankers.cpp -o binary && "d:\06_projects\os\"binary
enter number of processes: 5
enter number of resource types: 3
enter available resources for each type:
resource 0: 1
resource 1: 1
resource 2: 2
enter maximum matrix:
process 0 (max demand for each resource): 2 1 1
process 1 (max demand for each resource): 1 2 1
process 2 (max demand for each resource): 1 1 2
process 3 (max demand for each resource): 5 6 6
process 4 (max demand for each resource): 5 6 5
enter allocation matrix:
process 0 (currently allocated): 1 1 1
process 1 (currently allocated): 2 1 1
process 2 (currently allocated): 1 2 1
process 3 (currently allocated): 4 4 4
process 4 (currently allocated): 3 3 3
current system state:
available: 1 1 2
allocation matrix:
p0: 1 1 1
p1: 2 1 1
p2: 1 2 1
p3: 4 4 4
p4: 3 3 3
maximum matrix:
p0: 2 1 1
p1: 1 2 1
p2: 1 1 2
p3: 5 6 6
p4: 5 6 5
Ln 7, Col 26
Spaces: 4
UTF-8
CRLF
C++
Go Live
Win32
Prettier


PROBLEMS
TERMINAL
Code
maximum matrix:
p0: 2 1 1
p1: 1 2 1
p2: 1 1 2
p3: 5 6 6
p4: 5 6 5
need matrix:
p0: 1 0 0
p1: -1 1 0
p2: 0 -1 1
p3: 1 2 2
p4: 2 3 2
--- checking initial system safety ---
system is in safe state
safe sequence: p0 -> p1 -> p2 -> p3 -> p4
--- menu ---
1. request resources
2. release resources
3. check safe state
4. display current state
5. exit
enter choice: