CS353
Lab7
200001053
Nilay Ganvit

```cpp
#include <bits/stdc++.h>
using namespace std;

int num_process, num_resource;
vector<vector<int>> allocated, max_allowed, need;
vector<int> available;

bool is_safe(int taken_p, vector<bool> &completed)
{
   bool all_complete = true;
   for (int p = 1; p <= num_process; p++)
   {
       if (!completed[p] && p != taken_p)
       {
           all_complete = false;
           break;
       }
   }
   if (all_complete)
   {
       return true;
   }

   for (int p = 1; p <= num_process; p++)
   {
       if (completed[p] || p == taken_p)
       {
           continue;
       }

       bool ans = true;

       for (int r = 1; r <= num_resource; r++)
       {
           if (need[p][r] > available[r])
```

```
            {
                ans = false;
                break;
            }
        }

        if (ans)
        {
            return true;
        }
    }

    return false;
}

bool is_allowed(int p, vector<bool> &completed)
{
    for (int r = 1; r <= num_resource; r++)
    {
        if (need[p][r] > available[r])
        {
            return false;
        }
    }

    for (int r = 1; r <= num_resource; r++)
    {
        available[r] += allocated[p][r];
    }

    if (is_safe(p, completed))
    {
        return true;
    }

    for (int r = 1; r <= num_resource; r++)
    {
        available[r] -= allocated[p][r];
    }
```

```
    return false;
}

bool solve(vector<int> &sequence, vector<bool> &completed)
{
    bool all_complete = true;
    for (int p = 1; p <= num_process; p++)
    {
        if (!completed[p])
        {
            all_complete = false;
            break;
        }
    }
    if (all_complete)
    {
        return true;
    }

    for (int p = 1; p <= num_process; p++)
    {
        if (!completed[p] && is_allowed(p, completed))
        {
            completed[p] = true;
            sequence.push_back(p);

            if (solve(sequence, completed))
            {
                return true;
            }

            completed[p] = false;
            sequence.pop_back();
        }
    }

    return false;
}

void printMatrix(vector<vector<int>> &a, int m, int n)
```

```
{
   for (int i = 1; i <= m; i++)
   {
       for (int j = 1; j <= n; j++)
       {
           cout << a[i][j] << " ";
       }
       cout << "\n";
   }

   cout << "\n";
}

bool is_safe_req(int req_proc)
{
   for (int p = 1; p <= num_process; p++)
   {

       bool allowed = true;

       for (int r = 1; r <= num_resource; r++)
       {
           if (need[p][r] > available[r])
           {
               allowed = false;
               break;
           }
       }

       if (allowed)
       {
           return true;
       }
   }

   return false;
}

bool is_allowed_req(int req_proc, vector<int> &request)
{
```

```
    for (int r = 1; r <= num_resource; r++)
    {
        if (request[r] > available[r])
        {
            cout << "Request can not immediately be granted. P" << r << "
has to wait since resources are not available.\n";
            cout << "\n";
            return false;
        }
        if (request[r] > need[req_proc][r])
        {
            cout << "Request can not immediately be granted. P" << r << "
has to wait since resources are not needed.\n";
            cout << "\n";
            return false;
        }
    }

    for (int r = 1; r <= num_resource; r++)
    {
        available[r] -= request[r];
        need[req_proc][r] -= request[r];
        allocated[req_proc][r] += request[r];
    }

    cout << "Update:\n";
    cout << "\n";

    cout << "Allocation Matrix : "
         << "\n";
    printMatrix(allocated, num_process, num_resource);

    cout << "Max Allowed Matrix : "
         << "\n";
    printMatrix(max_allowed, num_process, num_resource);

    cout << "Need Matrix : "
         << "\n";
    printMatrix(need, num_process, num_resource);
```

```
    cout << "Resources available : ";
    for (int r = 1; r <= num_resource; r++)
    {
        cout << available[r] << " ";
    }
    cout << "\n";
    cout << "\n";
    cout << "\n";

    if (!is_safe_req(req_proc))
    {
        for (int r = 1; r <= num_resource; r++)
        {
            available[r] += request[r];
            need[req_proc][r] += request[r];
            allocated[req_proc][r] -= request[r];
        }
        cout << "Unsafe.\nP" << req_proc << " must wait. The old
resource-allocation state will restore.\n";
        cout << "\n";
        return false;
    }

    bool all_satisfied = true;

    for (int r = 1; r <= num_resource; r++)
    {
        if (need[req_proc][r] > 0)
        {
            all_satisfied = false;
            break;
        }
    }

    if (all_satisfied)
    {
        for (int r = 1; r <= num_resource; r++)
        {
            available[r] += allocated[req_proc][r];
        }
```

```
    }

    return true;
}

int main()
{
    cout << "Enter No. of processes:";
    cin >> num_process;
    cout << "Enter No. of resources:";
    cin >> num_resource;

    allocated = vector<vector<int>>(num_process + 1,
vector<int>(num_resource + 1));
    max_allowed = vector<vector<int>>(num_process + 1,
vector<int>(num_resource + 1));
    need = vector<vector<int>>(num_process + 1, vector<int>(num_resource +
1));

    vector<int> available_temp;
    cout << "Input allocation matrix:\n";
    for (int p = 1; p <= num_process; p++)
    {
        for (int r = 1; r <= num_resource; r++)
        {
            cin >> allocated[p][r];
        }
    }
    cout << "Input max matrix:\n";
    for (int p = 1; p <= num_process; p++)
    {

        for (int r = 1; r <= num_resource; r++)
        {
            cin >> max_allowed[p][r];
        }
    }

    for (int p = 1; p <= num_process; p++)
    {
```

```
        for (int r = 1; r <= num_resource; r++)
        {
            need[p][r] = max_allowed[p][r] - allocated[p][r];
        }
    }
    cout << "Input available resources:\n";
    available = vector<int>(num_resource + 1);
    for (int r = 1; r <= num_resource; r++)
    {
        cin >> available[r];
    }

    available_temp = available;

    vector<int> sequence;
    vector<bool> completed(num_process + 1, false);

    if (solve(sequence, completed))
    {
        cout << "The system is in a safe state since the sequence < ";
        for (auto x : sequence)
        {
            cout << x << ", ";
        }
        cout << "> satisfies safety criteria\n";
    }
    else
    {
        cout << "The system is in an unsafe state\n";
    }

    available = available_temp;
    cout << "\n";
    cout << "\n";

    while (true)
    {
        cout << "Enter requesting processor: "
             << "\n";
        int req_proc;
```

```
        cin >> req_proc;

        if (req_proc == -1)
        {
            break;
        }

        cout << "Enter resources requested for each type:\n";
        vector<int> request(num_resource + 1);
        for (int r = 1; r <= num_resource; r++)
        {
            cin >> request[r];
        }

        if (is_allowed_req(req_proc, request))
        {
            cout << "Yes resources can be allocated\n";
        }
    }

    return 0;
}
```

Input/Output:

```
nilay@Nilay-PC:~/Downloads$ g++ main.cpp
nilay@Nilay-PC:~/Downloads$ ./a.out
Enter No. of processes:5
Enter No. of resources:3
Input allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Input max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Input available resources:
3 3 2
The system is in a safe state since the sequence < 2, 4, 1, 3, 5, > satisfies safety criteria


Enter requesting processor:
2
Enter resources requested for each type:
1 0 2
Update:

Allocation Matrix :
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2

Max Allowed Matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Need Matrix :
7 4 3
0 2 0
6 0 0
0 1 1
4 3 1

Resources available : 2 3 0
```

```
Yes resources can be allocated
Enter requesting processor:
5
Enter resources requested for each type:
3 3 0
Request can not immediately be granted. P1 has to wait since resources are not available.

Enter requesting processor:
1
Enter resources requested for each type:
0 2 0
Update:

Allocation Matrix :
0 3 0
3 0 2
3 0 2
2 1 1
0 0 2

Max Allowed Matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Need Matrix :
7 2 3
0 2 0
6 0 0
0 1 1
4 3 1

Resources available : 2 1 0


Unsafe.
P1 must wait. The old resource-allocation state will restore.

Enter requesting processor:
```