هذا الكود هو تنفيذ لخوارزمية (Banker's Algorithm) التي تُستخدم لتجنب حدوث الحالات غير الآمنة في نظم التشغيل، خاصةً عند استخدام مدير الموارد لتخصيص الموارد بين العمليات. الخوارزمية تهدف إلى تجنب حدوث حالات الانتظار التي تؤدي إلى تعليق النظام بسبب انتظار متبادل للموارد.

اساسيات الكود:

المتغيرات:

- available: يُمثل الموارد المتاحة حاليًا.
- max_claim: مصفوفة تحتوي على أقصى مطالب يمكن أن تقدمها كل عملية لكل نوع من الموارد.
- allocation: مصفوفة تحتوي على الموارد التي تم تخصيصها لكل عملية.
- need: مصفوفة تحتوي على الموارد التي تحتاجها كل عملية لإكمال تنفيذها.
-

الدوال:

- is_safe: تُحدد ما إذا كانت العملية آمنة للتنفيذ في اللحظة الحالية أم لا.
- bankers_algorithm: تقوم بتنفيذ خوارزمية المصرف للتحقق من السلامة.
- display_status: تعرض حالة النظام الأولية.
-

التحقق من الحالة الآمنة:

يتم تحميل بيانات النظام (الموارد المتاحة ومطالب العمليات والموارد المخصصة) من قبل المستخدم.

يتم عرض الحالة الأولية للنظام.

يتم استدعاء bankers_algorithm للتحقق من ما إذا كان النظام في حالة آمنة أم لا.

إذا كان النظام في حالة آمنة، يتم عرض التسلسل الآمن لتنفيذ العمليات. إذا لم يكن النظام آمنًا، يتم محاولة البحث عن تسلسل آمن ثم يتم عرضه.

الإخراج:

يتم عرض حالة النظام الأولية.

إما أن يتم عرض رسالة تفيد بأن التسلسل الأولي غير آمن أو يتم عرض التسلسل الآمن إذا كان النظام آمنًا.

الكود يُطلب من المستخدم إدخال المعلومات اللازمة، ومن ثم يُظهر حالة النظام الأولية ويقوم بالتحقق مما إذا كان النظام في حالة آمنة أو لا، ويظهر التسلسل إذا كان النظام آمنًا، أو رسالة بأنه غير آمن إذا لم يكن.

شرح الكود سطر بسطر

```
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;
```

#include: هذه هي تعليمة المعالج القبلي (preprocessor directive) وتُستخدم لتضمين مكتبات الإدخال والإخراج ومكتبة الـ vectors ومكتبة لتنسيق الإخراج.

```
const int MAX_PROCESSES = 5;
const int MAX_RESOURCES = 3;

vector<int> available(MAX_RESOURCES);
vector<vector<int>> max_claim(MAX_PROCESSES, vector<int>(MAX_RESOURCES));
vector<vector<int>> allocation(MAX_PROCESSES, vector<int>(MAX_RESOURCES));
vector<vector<int>> need(MAX_PROCESSES, vector<int>(MAX_RESOURCES));
```

تعريف بعض المتغيرات الثابتة والمتغيرات التي ستحتاج إلى استخدامها في البرنامج، مثل MAX_PROCESSES و MAX_RESOURCES ومصفوفات available ، max_claim، allocation، و need.

```
bool is_safe(int process, vector<int>& work, vector<bool>& finish) {
    for (int i = 0; i < MAX_RESOURCES; ++i) {
        if (need[process][i] > work[i]) {
            return false;
        }
    }
    return true;
}
```

تعريف دالة is_safe التي تقوم بفحص ما إذا كانت العملية آمنة لتنفيذها في اللحظة الحالية أم لا.

```
bool bankers_algorithm(vector<int>& safe_sequence) {
    vector<int> work = available;
    vector<bool> finish(MAX_PROCESSES, false);
    for (int iteration = 0; iteration < MAX_PROCESSES; ++iteration) {
        bool found = false;
        for (int i = 0; i < MAX_PROCESSES; ++i) {
            if (!finish[i] && is_safe(i, work, finish)) {
                for (int j = 0; j < MAX_RESOURCES; ++j) {
                    work[j] += allocation[i][j];
                }
                finish[i] = true;
                safe_sequence.push_back(i);
                found = true;
                break;  // Break after finding a safe process
            }
        }
        if (!found) { // The system is in an unsafe state
            return false;
        }
    }
    // The system is in a safe state
    return true;
}
```

تعريف دالة bankers_algorithm التي تقوم بتنفيذ خوارزمية المصرف للتحقق من السلامة.

```
void display_status() {
    cout << "Initial Status of the System:" << endl;
    cout << "Available Resources: ";
    for (int i = 0; i < MAX_RESOURCES; ++i) {
        cout << available[i] << " ";
    }
    cout << endl << setw(20) << "Max Claim Matrix" << setw(20) << "Allocation
Matrix" << setw(20) << "Need Matrix" << endl;
    for (int i = 0; i < MAX_PROCESSES; ++i) {
        for (int j = 0; j < MAX_RESOURCES; ++j) {
            cout << setw(5) << max_claim[i][j];
        }
        cout << setw(10) << " ";
        for (int j = 0; j < MAX_RESOURCES; ++j) {
            cout << setw(5) << allocation[i][j];
        }
        cout << setw(10) << " ";
        for (int j = 0; j < MAX_RESOURCES; ++j) {
            cout << setw(5) << need[i][j];
        }
        cout << endl;
    }
}
```

تعريف دالة display_status التي تقوم بعرض حالة النظام الأولية.

```
int main() {
    // Initialize available resources
    cout << "Enter the available resources:" << endl;
    for (int i = 0; i < MAX_RESOURCES; ++i) {
        cin >> available[i];
    }
```

دالة main هي نقطة بداية التنفيذ. يتم في هذا الجزء استخراج مدى الموارد المتاحة من المستخدم.

```
    // Initialize maximum claim matrix
    cout << "Enter the maximum claim matrix:" << endl;
    for (int i = 0; i < MAX_PROCESSES; ++i) {
        for (int j = 0; j < MAX_RESOURCES; ++j) {
            cin >> max_claim[i][j];
        }
    }
```

يُطلب من المستخدم إدخال المطالب القصوى التي يمكن أن تقدمها كل عملية.

```
    // Initialize allocation matrix
    cout << "Enter the allocation matrix:" << endl;
    for (int i = 0; i < MAX_PROCESSES; ++i) {
        for (int j = 0; j < MAX_RESOURCES; ++j) {
            cin >> allocation[i][j];
            need[i][j] = max_claim[i][j] - allocation[i][j];
        }
    }
```

يُطلب من المستخدم إدخال الموارد التي تم تخصيصها لكل عملية، ويُحسب المتبقي باستخدام need.

```
    // Display the initial status
    display_status();
```

يتم عرض حالة النظام الأولية.

```
    // Check if the system is in a safe state
    vector<int> safe_sequence;
    if (!bankers_algorithm(safe_sequence)) {
        cout << "The given sequence is not safe." << endl;

        // Attempt to find a safe sequence
        safe_sequence.clear();
        if (bankers_algorithm(safe_sequence)) {
            cout << "\nSafe Sequence: ";
            for (int i = 0; i < MAX_PROCESSES; ++i) {
                cout << "P" << safe_sequence[i];
                if (i != MAX_PROCESSES - 1) {
                    cout << " -> ";
                }
            }
            cout << endl;
            cout << "The system is in a safe state." << endl;
        } else {
            cout << "Unable to find a safe sequence." << endl;
        }
    } else {
        cout << "\nSafe Sequence: ";
        for (int i = 0; i < MAX_PROCESSES; ++i) {
            cout << "P" << safe_sequence[i];
            if (i != MAX_PROCESSES - 1) {
                cout << " -> ";
            }
        }
        cout << endl;
        cout << "The system is in a safe state." << endl;
    }

    return 0;
}
```

في هذا الجزء يتم استخدام دالة bankers_algorithm للتحقق مما إذا كان النظام في حالة آمنة، وفي حالة كان غير آمن يتم المحاولة في العثور على تسلسل آمن آخر، ثم يتم عرض النتائج.