01 实验概述

1.1 实验目的

银行家算法是操作系统中避免死锁的典型算法，通过本实验加深对银行家算法的理解。

1.2 实验内容

用C语言或C++编写一个简单的银行家算法模拟程序，实现多个进程争用系统临界资源时的分配过程。要求实现如下功能：

1. 初始化当前系统状态。
2. 显示当前资源分配情况。
3. 检查当前系统是否安全。
4. 分配资源。某进程请求分配一组资源时，先进行安全性检查，分配后系统是否处于安全状态？如果安全，显示安全序列，将资源分配给该进程，否则该进程等待。
5. 新建一个进程。
6. 撤销一个进程。

02 实验设计与实现

2.1 银行家算法所需数据结构

假设实验共有m个资源类别，n个进程。

银行家算法的数据结构主要有以下几个：

1. `Available[n]`：
    - 表示系统中可用的资源数目;
    - `Available[j]` 表示第j类资源的可用数目;
2. `Max[n][m]`：
    - 表示各进程对各类资源的最大需求量;
    - `Max[i][j]` 表示进程i对第j类资源的最大需求量;
3. `Allocation[n][m]`：
    - 表示系统已分配给各进程的各类资源数目;
    - `Allocation[i][j]` 表示进程i已分配给第j类资源的数目;
4. `Need[n][m]`：
    - 表示各进程对各类资源的尚需求量;
    - `Need[i][j]` 表示进程i对第j类资源的尚需求量;
    - `Need[i][j] = Max[i][j] - Allocation[i][j]`;
5. `Request[m]`：
    - 表示进程一次请求对各类资源的请求量;
    - `Request[j]` 表示进程请求第j类资源的数目;
6. `Work[m]`：
    - 表示系统可提供给进程继续运行所需的各类资源数目;
    - 初始值为 `Available[m]`;

对于上述数据结构，在实现上使用C++的vector容器进行封装，以便于使用。

定义如下全局变量：

```
vector<string> resource_name;   // 资源名称
vector<int> Available;          // 可用资源个数
vector<vector<int>> Max;        // 最大需求矩阵
vector<vector<int>> Allocation; // 已分配矩阵
vector<vector<int>> Need;       // 需求矩阵
int resource_num;   // 资源种类数
int task_num;       // 进程数

```

2.2 数据结构封装

为方便实验，下面对vector的部分运算符进行重载。

2.2.1 重载运算符 <<

重载运算符 `<<` ，使得可以直接使用 `cout` 输出vector容器的内容。

```
// 重载vector输出
template <typename T>
std::ostream &operator<<(std::ostream &o, const std::vector<T> &vec) {
    int num = vec.size();
    for (int i = 0; i < num; i++)
        cout << (i ? " " : "") << vec[i];
    return o;
}
```

2.2.2 重载运算符 +

重载运算符 `+` ，使得可以直接使用 `vector` 容器进行向量加法运算。

```
// 重载vector加法
vector<int> operator+(vector<int> &a, vector<int> &b) {
    if (a.size() != b.size()) {
        cout << "vector运算的两个操作数大小不同！\n";
        exit(-1);
    }
    int n = a.size();
    vector<int> res(n);
    for (int i = 0; i < n; i ++) {
        res[i] = a[i] + b[i];
    }
    return res;
}
```

2.2.3 重载运算符 -

重载运算符 - ，使得可以直接使用 vector 容器进行向量减法运算。

```cpp
// 重载vector减法
vector<int> operator-(vector<int> &a, vector<int> &b) {
    if (a.size() != b.size()) {
        cout << "vector运算的两个操作数大小不同！\n";
        exit(-1);
    }
    int n = a.size();
    vector<int> res(n);
    for (int i = 0; i < n; i ++) {
        res[i] = a[i] - b[i];
    }
    return res;
}
```

2.2.4 重载运算符 <=

重载运算符 <= ，使得可以直接使用 vector 容器进行向量比较运算（用于判断请求量与现有资源的大小关系）。

```cpp
// 重载vector比较<=，用于判断是否资源数小于等于关系
bool operator<=(vector<int> const &a, vector<int> const &b) {
    if (a.size() != b.size()) {
        cout << "vector运算的两个操作数大小不同！\n";
        exit(-1);
    }
    int n = a.size();
    for (int i = 0; i < n; i ++) {
        if (a[i] > b[i]) return false;
    }
    return true;
}
```

2.3 初始化系统状态

读入资源种类数 `resource_num` 和进程数 `task_num`，并初始化 `Available`、`Max`、`Allocation`、`Need` 等数据结构。

```
// 初始化，获取用户输入，初始化各个变量(资源数，进程数，各个矩阵)
void Init() {
    string name; int num;
    cout << "*****************单处理机系统进程调度实现*****************"<< '\n';
    cout << "请首先输入系统可供资源种类的数量:";
    cin >> resource_num;
    resource_name.resize(resource_num);
    Available.resize(resource_num);
    for (int i = 0; i < resource_num; i++) {
        cout << "资源" << i+1 << "的名称:";
        cin >> resource_name[i];
        cout << "资源的数量:";
        cin >> Available[i];
    }
    cout << '\n';
    cout << "请输入作业的数量:";
    cin >> task_num;
    Max = Allocation = Need =
        vector<vector<int>>(task_num, vector<int>(resource_num, 0));

    printf("请输入各进程的最大需求量(%d*%d矩阵)[Max]:\n", task_num, resource_num);
    for (int i = 0; i < task_num; i++)
        for (int j = 0; j < resource_num; j++)
            cin >> Max[i][j];

    printf("请输入各进程已经申请的资源量(%d*%d矩阵)[Allocation]:\n", task_num,
resource_num);
    for (int i = 0; i < task_num; i++)
        for (int j = 0; j < resource_num; j++)
            cin >> Allocation[i][j];
    GetNeed();  // 由Max和Allocation计算Need
    cout << '\n';
    ListResouceStatus();  // 显示当前资源分配情况
    CheckStatusSecurity();  // 检查当前状态安全性
    cout << "\n\n";
}
```

2.4 显示当前资源分配情况(操作1)

显示当前资源分配情况，包括 `Available` 、 `Max` 、 `Allocation` 、 `Need` 等数据结构。

```
// 显示当前资源分配情况
// 操作1，显示当前资源分配情况
void ListResouceStatus() {
    cout << "系统目前可用的资源[Avaliable]: " << '\n';
    cout << resource_name << '\n';
    cout << Available << '\n';

    cout << "             Max      Allocation     Need" << '\n';
    cout << "进程名     ";
    for (int i = 0; i < 3; i++)
        cout << resource_name << "       ";
    cout << '\n';

    for (int i = 0; i < task_num; i++) {
        cout << " " << i << "         ";
        cout << Max[i] << "       ";
        cout << Allocation[i] << "       ";
        cout << Need[i] << '\n';
    }
}
```

2.5 检查当前系统是否安全(操作2)

检查当前系统是否安全，如果安全，显示安全序列，否则显示不安全。

算法如下：

1. 令 `Work = Available` ；令 `Finish[i] = false` ，表示进程i未完成；
2. 从进程集合中找到一个满足以下条件的进程i：
    - `Finish[i] == false` ;
    - `Need[i] <= Work` ;
3. 找如果到这样的进程i，则执行以下操作：
    1. `Work = Work + Allocation[i]` ;

2. `Finish[i] = true ;`

3. 转到步骤3；

4. 如果找不到这样的进程i，则转到步骤4;

4. 如果所有进程都满足 `Finish[i] == true` ，则系统是安全的，且安全序列为 `Finish` 中为true的进程序列；否则系统是不安全的。

```cpp
// 操作2，检查当前状态安全性，若安全则输出安全序列；否则输出不安全
// 返回值：是否安全
bool CheckStatusSecurity() {
    vector<int> Work = Available;
    vector<bool> Finish(task_num, false);
    vector<int> SafeSequence;
    for (int count = 0; count < task_num; count ++) {
        bool flag = false;
        for (int i = 0; i < task_num; i++) {
            if (Finish[i] == false && Need[i] <= Work) {
                Work = Work + Allocation[i];
                Finish[i] = true;
                SafeSequence.push_back(i);
                flag = true;
                break;
            }
        }
        if (flag == false) {
            cout << "系统处于不安全状态，无法分配！" << '\n';
            return false;
        }
    }
    cout << "系统是安全的!" << '\n';
    cout << "安全序列为：";
    for (int i = 0; i < task_num; i++)
        cout << SafeSequence[i] << (i == task_num - 1 ? "" : "->");
    cout << '\n';
    return true;
}
```

2.6 分配资源(操作3)

读入进程号 `task_id` 和请求量 `Request` ，判断是否安全，如果安全，显示安全序列，将资源分配给该进程，否则该进程等待。
判断是否安全的算法如下：

1. 判断 `Request <= Need[task_id]` ;
2. 判断 `Request <= Available` ;
3. 尝试将资源分配给进程 `task_id` ，计算分配后状态：
    - `Available = Available - Request` ;
    - `Allocation[task_id] = Allocation[task_id] + Request` ;
    - `Need[task_id] = Need[task_id] - Request` ;
4. 判断分配后的状态是否安全
    - 这里可以直接调用操作2的函数；
5. 根据判断结果，执行以下操作：
    - 如果安全，显示安全序列，将资源分配给该进程；
    - 否则，回滚分配，不进行此次分配。

```
// 操作3，分配资源
// 先判断是否安全，若安全则分配，否则不分配（回滚）
void AllocateResource() {
    int task_id;
    cout << "请输入要求分配的资源进程号0-" << task_num - 1 << ":";
    cin >> task_id;
    vector<int> Request(resource_num);
    cout << "请输入进程 " << task_id << " 申请的资源:\n";
    for (int i = 0; i < resource_num; i++)
    {
        cout << resource_name[i] << ":";
        cin >> Request[i];
    }
    if (Request > Need[task_id]) {
        cout << "进程" << task_id << "申请的资源大于其需要量 分配出错，不予分配!"
             << '\n';
        return;
    }
    if (Request > Available) {
```

```
        cout << "进程" << task_id << "申请的资源大于系统现在可利用的资源 分配出错，不予分配!"
             << '\n';
        return;
    }
    Available = Available - Request;
    Allocation[task_id] = Allocation[task_id] + Request;
    Need[task_id] = Need[task_id] - Request;

    cout << '\n';
    ListResouceStatus();
    if (CheckStatusSecurity() == false)
    {
        // 回滚
        Available = Available + Request;
        Allocation[task_id] = Allocation[task_id] - Request;
        Need[task_id] =**** Need[task_id] + Request;

        cout << "系统目前可用的资源[Avaliable]仍然为:" << '\n';
        cout << resource_name << '\n';
        cout << Available << '\n';
    }
}
```

2.7 新建进程(操作4)

新建进程，读入新进程的 Max 和 Allocation ，并计算 Need ，将信息添加到全局 Max 、 Allocation 、 Need 中。

```
// 操作4，创建进程
void CreateProcess() {
    // 获取新进程信息
    vector<int> process_max(resource_num);
    cout << "请输入进程最大需求量:\n";
    for (int i = 0; i < resource_num; i++) {
        cout << resource_name[i] << ":";
        cin >> process_max[i];
    }

    vector<int> process_allocation(resource_num);
    cout << "请输入进程已经申请的资源量:\n";
    for (int i = 0; i < resource_num; i++) {
```

```
        cout << resource_name[i] << ":";
        cin >> process_allocation[i];
    }

    vector<int> process_need = process_max - process_allocation;

    // 更新各个矩阵
    Max.push_back(process_max);
    Allocation.push_back(process_allocation);
    Need.push_back(process_need);

    // 更新进程数
    task_num++;

    // 显示当前进程及资源分配情况
    cout << "进程创建成功!" << '\n';
    cout << "当前进程及资源分配情况:" << '\n';
    ListResouceStatus();
}
```

2.8 撤销进程(操作5)

撤销进程，读入进程号 `task_id` ，将该进程的 `Max` 、 `Allocation` 、 `Need` 信息从全局变量中删除，同时回收该进程的资源到 `Available` 。

```
// 操作5，删除进程
void DeleteProcess() {
    // 获取要删除的进程号
    int process_id;
    cout << "请输入要删除的进程号0-" << task_num - 1 << ":";
    cin >> process_id;

    // 更新各个矩阵
    Available = Available + Allocation[process_id];
    Max.erase(Max.begin() + process_id);
    Allocation.erase(Allocation.begin() + process_id);
    Need.erase(Need.begin() + process_id);

    // 更新进程数
    task_num--;

    // 显示当前进程及资源分配情况
```

```
    cout << "进程删除成功!" << '\n';
    cout << "当前进程及资源分配情况:" << '\n';
    ListResouceStatus();
}
```

2.9 操作选择与主函数

根据用户输入的操作号，执行相应的操作(调用上述操作函数)。

```
// 获取用户输入的选项
int GetOption() {
    cout << "**********银行家算法演示**********" << '\n';
    cout << "*               1:显示当前资源分配情况" << '\n';
    cout << "*               2:检查当前状态安全性" << '\n';
    cout << "*               3:分配资源" << '\n';
    cout << "*               4:创建进程" << '\n';
    cout << "*               5:删除进程" << '\n';
    cout << "*               6:离开" << '\n';
    cout << "************************************************" << '\n';
    cout << "请选择功能号: ";
    int op;
    cin >> op;
    return op;
}
int main() {
    Init();
    int op;
    while (op = GetOption()) {
        switch (op) {
            case 1:
                ListResouceStatus();  // 显示当前资源分配情况
                break;
            case 2:
                CheckStatusSecurity();  // 检查当前状态安全性
                break;
            case 3:
                AllocateResource();  // 分配资源
                break;
            case 4:
                CreateProcess();  // 创建进程
                break;
            case 5:
```

```
                DeleteProcess();  // 删除进程
                break;
            case 6:
                return 0;       // 退出
            default:
                cout << "输入错误，请重新输入！" << '\n';
        }
        cout << '\n' << '\n';
    }
    return 0;
}
```

03 实验测试与结果

使用 银行家测试数据例子.txt 数据，对操作1~3进行测试，同时自选数据测试了操作4、5（创建进程和删除进程）功能。

测试结果如下：
（注：由于测试数据较多，这里略去多余空行及部分操作选择提示）

```
PS D:\files\Course\2023Fal-操作系统\lab\lab5> g++ .\banker.cpp
PS D:\files\Course\2023Fal-操作系统\lab\lab5> .\a.exe
*****************单处理机系统进程调度实现*****************
请首先输入系统可供资源种类的数量:3
资源1的名称:a
资源的数量:3
资源2的名称:b
资源的数量:3
资源3的名称:c
资源的数量:2
请输入作业的数量:5
请输入各进程的最大需求量(5*3矩阵)[Max]:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
请输入各进程已经申请的资源量(5*3矩阵)[Allocation]:
0 1 0
2 0 0
```

```
3 0 2
2 1 1
0 0 2
系统目前可用的资源[Avaliable]:
a b c
3 3 2
              Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         7 5 3       0 1 0       7 4 3
 1         3 2 2       2 0 0       1 2 2
 2         9 0 2       3 0 2       6 0 0
 3         2 2 2       2 1 1       0 1 1
 4         4 3 3       0 0 2       4 3 1
系统是安全的!
安全序列为: 1->3->0->2->4
**********************银行家算法演示*****************
*               1:显示当前资源分配情况
*               2:检查当前状态安全性
*               3:分配资源
*               4:创建进程
*               5:删除进程
*               6:离开
**************************************************
请选择功能号: 1
系统目前可用的资源[Avaliable]:
a b c
3 3 2
              Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         7 5 3       0 1 0       7 4 3
 1         3 2 2       2 0 0       1 2 2
 2         9 0 2       3 0 2       6 0 0
 3         2 2 2       2 1 1       0 1 1
 4         4 3 3       0 0 2       4 3 1
**********************银行家算法演示*****************
请选择功能号: 2
系统是安全的!
安全序列为: 1->3->0->2->4
**********************银行家算法演示*****************
请选择功能号: 3
请输入要求分配的资源进程号0-4:1
请输入进程 1 申请的资源:
a:1
b:0
c:2
```

```
系统目前可用的资源[Avaliable]:
a b c
2 3 0
              Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         7 5 3       0 1 0       7 4 3
 1         3 2 2       3 0 2       0 2 0
 2         9 0 2       3 0 2       6 0 0
 3         2 2 2       2 1 1       0 1 1
 4         4 3 3       0 0 2       4 3 1
系统是安全的!
安全序列为: 1->3->0->2->4
**********************银行家算法演示*****************
请选择功能号: 3
请输入要求分配的资源进程号0-4:4
请输入进程 4 申请的资源:
a:3
b:3
c:0
进程4申请的资源大于系统现在可利用的资源 分配出错，不予分配!
**********************银行家算法演示*****************
请选择功能号: 3
请输入要求分配的资源进程号0-4:0
请输入进程 0 申请的资源:
a:0
b:2
c:0
系统目前可用的资源[Avaliable]:
a b c
2 1 0
              Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         7 5 3       0 3 0       7 2 3
 1         3 2 2       3 0 2       0 2 0
 2         9 0 2       3 0 2       6 0 0
 3         2 2 2       2 1 1       0 1 1
 4         4 3 3       0 0 2       4 3 1
系统处于不安全状态，无法分配!
系统目前可用的资源[Avaliable]仍然为:
a b c
2 3 0
**********************银行家算法演示*****************
请选择功能号: 2
系统是安全的!
安全序列为: 1->3->0->2->4
```

```
**********银行家算法演示**********
请选择功能号：1
系统目前可用的资源[Avaliable]：
a b c
2 3 0
             Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         7 5 3       0 1 0       7 4 3
 1         3 2 2       3 0 2       0 2 0
 2         9 0 2       3 0 2       6 0 0
 3         2 2 2       2 1 1       0 1 1
 4         4 3 3       0 0 2       4 3 1
**********银行家算法演示**********
请选择功能号：5
请输入要删除的进程号0-4:4
进程删除成功!
当前进程及资源分配情况：
系统目前可用的资源[Avaliable]：
a b c
2 3 2
             Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         7 5 3       0 1 0       7 4 3
 1         3 2 2       3 0 2       0 2 0
 2         9 0 2       3 0 2       6 0 0
 3         2 2 2       2 1 1       0 1 1
**********银行家算法演示**********
请选择功能号：5
请输入要删除的进程号0-3:0
进程删除成功!
当前进程及资源分配情况：
系统目前可用的资源[Avaliable]：
a b c
2 4 2
             Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         3 2 2       3 0 2       0 2 0
 1         9 0 2       3 0 2       6 0 0
 2         2 2 2       2 1 1       0 1 1
**********银行家算法演示**********
请选择功能号：2
系统是安全的!
安全序列为：0->2->1
**********银行家算法演示**********
请选择功能号：4
```

```
请输入进程最大需求量:
a:3
b:3
c:3
请输入进程已经申请的资源量:
a:1
b:1
c:1
进程创建成功!
当前进程及资源分配情况:
系统目前可用的资源[Avaliable]:
a b c
2 4 2
              Max      Allocation     Need
进程名     a b c       a b c       a b c
 0         3 2 2       3 0 2       0 2 0
 1         9 0 2       3 0 2       6 0 0
 2         2 2 2       2 1 1       0 1 1
 3         3 3 3       1 1 1       2 2 2
**********************银行家算法演示*****************
请选择功能号: 2
系统是安全的!
安全序列为: 0->2->1->3
**********************银行家算法演示*****************
请选择功能号: 6

PS D:\files\Course\2023Fal-操作系统\lab\lab5>
```

04 实验总结

本次实验主要是对银行家算法的实现。银行家算法是一种避免死锁的算法，通过在系统的资源分配前进行安全性检查，从而避免死锁的发生。而在安全性检查中，重要的如何是判断系统是否处于安全状态，即能否找到一个安全路径，使得所有进程能够完成任务。

对于本次实验的实现，通过C++的vector容器对银行家算法的数据结构进行了封装，使得可以直接使用 `vector` 容器进行向量加减法运算，从而简化了代码的编写。