操作系统

实验四 银行家算法
实验报告

姓名：张凯

班级：无 97

学号：2019011159

项目代码链接：*https://github.com/zhangkai0425/OS*

一、实验目的

1.理解银行家算法的基本思想内容。

2.编程模拟实现银行家算法下的动态资源分配过程，避免死锁的发生。

二、实验内容

1.对实现的算法通过流程图进行说明

2.设计不少于三组测试样例，需包括资源分配成功和失败的情况

3.能够展示系统资源占用和分配情况的变化及安全性检测的过程

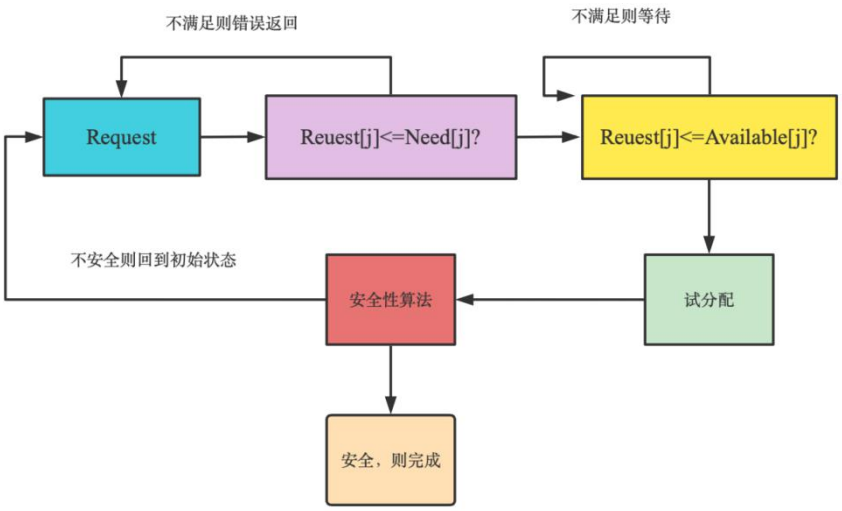
4.结合操作系统课程中银行家算法理论对实验结果进行分析，验证结果的正确性

5.分析算法的鲁棒性及算法效率

三、实验设计

实验较为容易，主要内容即为模拟和实现银行家算法。

算法流程完全按照课件所讲流程进行编写，没有其他的加工和改动，流程图如下：



具体来说，实际上只需要参考课件的矩阵过程表示即可，如下：

银行家算法运行实例

■T0时刻可以找到一个安全序列<P1, P3, P4, P2, P0>，系统是安全的

资源 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	3	3	2	1	2	2	2	0	0	5	3	2	true
P3	5	3	2	0	1	1	2	1	1	7	4	3	true
P4	7	4	3	4	3	1	0	0	2	7	4	5	true
P2	7	4	5	6	0	0	3	0	2	10	4	7	true
P0	10	4	7	7	4	3	0	1	0	10	5	7	true

基本上，按照此矩阵的生成步骤进行代码的编写，即可完成银行家算法。

四、代码实现

实验代码全部采用 **C++11** 编写，由于只是模拟银行家算法的动态分配过程，不需要真正去实现多线程系统调用，所以代码中不含系统调用的内容，在 **Mac/Linux/Windows** 上均可运行。

代码中主要实现和封装了一个 Banker 类，其成员变量和功能如下所示：

类型	变量	描述
std::vector	Safe	安全序列
	Available	资源向量
	Max	最大需求矩阵
	Allocation	分配矩阵
	Need	当前需求矩阵
	Work	动态可分配资源
	Work_Allocation	进程结束后释放资源向量
	Finish	是否成功分配向量
构造函数	Banker::Banker	构造函数
成员函数	Banker::Request	模拟某进程请求的函数
	Banker::SafeAlgorithm	安全性算法函数
	Banker::Ans	打印输出函数

模拟实现银行家算法的过程中，直接在主函数代码中赋初值即可，然后调用

Banker 类的成员函数，模拟请求操作与安全性算法操作，并输出相应的结果。

具体实现代码见附录 main.cpp。

五、实验结果

直接采用课件和作业中的输入样例。

1.第一组

■T0时刻P_1发出请求Request(1, 0, 2)，执行银行家算法

资源 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3 2	3 3	2 0
P_1	3	2	2	2 3	0 0	0 2	1 0	2 2	2 0			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

输入数据：

```
int main()
{
    //设置全局变量
    int n = 5; //线程数
    int m = 3; //资源数
    // Available 资源向量
    vector<int> Available = {3, 3, 2};
    // Max 最大需求矩阵
    vector<vector<int>> Max = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};
    // Allocation 分配矩阵
    vector<vector<int>> Allocation = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}};
    // Need 需求矩阵
    vector<vector<int>> Need = {{7, 4, 3}, {1, 2, 2}, {6, 0, 0}, {0, 1, 1}, {4, 3, 1}};
    Banker banker(n, m, Available, Max, Allocation, Need);
    vector<int> R = {1, 0, 2};
    banker.Request(1, R);
}
```

结果如下：

```
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ g++ -std=c++11 main.cpp -o mainn
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ ./mainn
开始试分配...
系统新状态安全,分配完成!
算法执行流程:
试分配后Available变量: 2 3 0
进程号  |      Work    |      Need    |      Allocation      |      Work + Allocation      |      Finish
1       |      2 3 0   |      0 2 0   |      3 0 2           |          5 3 2              |      True
3       |      5 3 2   |      0 1 1   |      2 1 1           |          7 4 3              |      True
0       |      7 4 3   |      7 4 3   |      0 1 0           |          7 5 3              |      True
2       |      7 5 3   |      6 0 0   |      3 0 2           |          10 5 5             |      True
4       |      10 5 5  |      4 3 1   |      0 0 2           |          10 5 7             |      True
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$
```

课件结果如下：

■执行安全性算法，可以找到一个安全序列{P1, P3, P4, P0, P2}，系统是安全的，可以将P1请求资源分配给它

资源/进程	Work			Need			Allocation			Work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	2	3	0	0	2	0	3	0	2	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_0	7	4	5	7	4	3	0	1	0	7	5	5	true
P_2	10	4	7	6	0	0	3	0	2	10	5	7	true

对比可知，虽然得到的安全序列顺序和课件有所不同，但安全序列和过程都是对的。

2.第二组

■P4发出请求Request(3, 3, 0)，执行银行家算法

■Available=2 3 0

■不能通过算法第2步（ Requesti[j]≤Available[j] ），所以P4等待

资源/进程	Max			Allocation			Need		
	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3
P_1	3	2	2	3	0	2	0	2	0
P_2	9	0	2	3	0	2	6	0	0
P_3	2	2	2	2	1	1	0	1	1
P_4	4	3	3	0	0	2	4	3	1

Available		
A	B	C
2	3	0

输入：

```
int main()
{
    //设置全局变量
    int n = 5; //线程数
    int m = 3; //资源数
    // Available 资源向量
    vector<int> Available = {2, 3, 0};
    // Max 最大需求矩阵
    vector<vector<int>> Max = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};
    // Allocation 分配矩阵
    vector<vector<int>> Allocation = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}};
    // Need 需求矩阵
    vector<vector<int>> Need = {{7, 4, 3}, {1, 2, 2}, {6, 0, 0}, {0, 1, 1}, {4, 3, 1}};
    Banker banker(n, m, Available, Max, Allocation, Need);
    vector<int> R = {3, 3, 0};
    banker.Request(4, R);
}
```

结果为：

```
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ g++ -std=c++11 main.cpp -o mainn
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ ./mainn
Request[i]>Available[i],请求向量无法满足 线程:4 等待
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$
```

算法运行无误。

3.第三组

银行家算法运行实例

- P0发出请求Request(0, 2, 0)，执行银行家算法
- Available{2,1,0}已不能满足任何进程需要，所以系统进入不安全状态，P0的请求不能分配

资源 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0 0	1 3	0 0	7 7	4 2	3 0	2 2	3 1	0 0
P_1	3	2	2	3	0	2	0	2	0			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

输入：

```
int main()
{
    //设置全局变量
    int n = 5; //线程数
    int m = 3; //资源数
    // Available 资源向量
    vector<int> Available = {2, 3, 0};
    // Max 最大需求矩阵
    vector<vector<int>> Max = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};
    // Allocation 分配矩阵
    vector<vector<int>> Allocation = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}};
    // Need 需求矩阵
    vector<vector<int>> Need = {{7, 4, 3}, {1, 2, 2}, {6, 0, 0}, {0, 1, 1}, {4, 3, 1}};
    Banker banker(n, m, Available, Max, Allocation, Need);
    vector<int> R = {0, 2, 0};
    banker.Request(0, R);
}
```

结果：

```
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ g++ -std=c++11 main.cpp -o mainn
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ ./mainn
开始试分配...
安全性算法检查失败！
算法执行流程：
试分配后Available变量: 2 1 0
进程号   |       Work    |       Need    |       Allocation      |       Work + Allocation       |       Finish
0        |               |       7 2 3   |       0 3 0           |                               |       False
1        |               |       1 2 2   |       2 0 0           |                               |       False
2        |               |       6 0 0   |       3 0 2           |                               |       False
3        |               |       0 1 1   |       2 1 1           |                               |       False
4        |               |       4 3 1   |       0 0 2           |                               |       False
恢复原状态,进程等待
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$
```

算法依然无误。

4.第四组

作业题：

4. 某时刻，系统的资源分配状态如下。系统是否安全？如果安全，请给出安全序列。

进程	已分配资源			仍需分配			可用资源		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	2	0	0	0	0	1	0	2	1
P_1	1	2	0	1	3	2			
P_2	0	1	1	1	3	1			
P_3	0	0	1	2	0	0			

答:

无法找到安全序列，过程如下：

资源 / 进程	Work			Need			Allocation			Work+Allocation			Finish
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3	
P0	0	2	1	0	0	1	2	0	0	2	2	1	true
P3	2	2	1	2	0	0	0	0	1	2	2	2	true
P1	2	2	2	1	3	2	1	2	0				false
P2				1	3	1	0	1	1				false

由于可用资源的限制，系统只能先执行 P0 进程，之后只能执行 P1 进程，但 P0 和 P1 之后，无论是 P2 还是 P3 都无法正常获得所需要的资源，无法执行，所以不是安全的状态。

输入:

```
int main()
{
    //设置全局变量
    int n = 4; //线程数
    int m = 3; //资源数
    // Available 资源向量
    vector<int> Available = {0, 2, 1};
    // Max 最大需求矩阵
    vector<vector<int>> Max = {{2, 0, 1}, {2, 5, 2}, {1, 4, 2}, {2, 0, 1}};
    // Allocation 分配矩阵
    vector<vector<int>> Allocation = {{2, 0, 0}, {1, 2, 0}, {0, 1, 1}, {0, 0, 1}};
    // Need 需求矩阵
    vector<vector<int>> Need = {{0, 0, 1}, {1, 3, 2}, {1, 3, 1}, {2, 0, 0}};
    Banker banker(n, m, Available, Max, Allocation, Need);
    vector<int> R = {0, 0, 0};
    banker.Request(0, R);
}
```

结果:

```
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ g++ -std=c++11 main.cpp -o mainn
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$ ./mainn
开始试分配...
安全性算法检查失败！
算法执行流程:
试分配后Available变量: 0 2 1
进程号  |      Work    |      Need    |      Allocation      |      Work + Allocation      |      Finish
0       |      0 2 1   |      0 0 1   |      2 0 0           |          2 2 1              |      True
3       |      2 2 1   |      2 0 0   |      0 0 1           |          2 2 2              |      True
1       |              |      1 3 2   |      1 2 0           |                             |              False
2       |              |      1 3 1   |      0 1 1           |                             |              False
恢复原状态,进程等待
(base) zhangkai@server2:/hdd1/zhangkai/OS/exp3$
```

对齐不太好，但结果还是对的。

如果尝试更多的可能，算法也是能够正常工作的，这里就不再多尝试了，总之算法是正确的。

六、实验思考题

1.银行家算法在实现过程中需注意资源分配的哪些事项才能避免死锁?

答:

(1)在 Request 后试分配之前，一定要检查 Request 是否合法，即是否小于 Need 向量和 Available 向量。

(2)试分配的过程中，分配完后需要运行安全性算法检查是否可以分配。

(3)如果新状态安全，则分配完成；如果新状态不安全，则恢复原状态，进程等待。

七、实验小结

本次实验较为简单，完全参照和实现课件所讲的算法步骤即可完成，总体上非常顺利，也没有进行很多 debug。为了让代码更加简洁，实验中讲各种功能封装为一个 Banker 类，比较直观。唯一的缺点是算法需要很多矩阵和向量操作，而 C++确实矩阵和向量操作很不方便，不如 Matlab 和 Python 直接，所以必须要遍历很多层数组，代码比较繁琐，当然，使用了 std::vector 能稍微简化一下操作。

总之，通过本次实验，也比较透彻地理解了银行家算法的思想和流程，也再次锻炼了一下 C++代码能力。

八、代码附录

```
//  main.cpp
//  实验3
//  Created by 张凯 on 2022/6/1.
//  Copyright © 2022 张凯. All rights reserved.

#include<iostream>
#include<fstream>
#include<queue>
#include<vector>
using namespace std;
class Banker{
public:
    Banker(int n,int m, vector<int> Available,vector<vector<int>>
Max,vector<vector<int>> Allocation,vector<vector<int>> Need);
    // 主要函数:处理请求
    void Request(int id,vector<int>R);
    // 安全性算法函数
    bool SafeAlgorithm();
    // 终端打印输出结果的函数
    void Ans();
private:
    const int M = 100;
    //全局变量
    int n = 0; //线程数
    int m = 0; //资源数
    //安全序列
    vector<int>Safe;
    // Available 资源向量
    vector<int> Available;
    // Max 最大需求矩阵
    vector<vector<int>> Max;
    // Allocation 分配矩阵
    vector<vector<int>> Allocation;
    // Need 需求矩阵
    vector<vector<int>> Need;
    // Work 动态可分配资源
    vector<vector<int>> Work;
    // Work+Allocation Matrix
    vector<vector<int>> Work_Allocation;
    // Finish 是否成功分配
    vector<bool> Finish;
};
Banker::Banker(int n,int m,vector<int> Available,vector<vector<int>>
Max,vector<vector<int>> Allocation,vector<vector<int>> Need){
    this->n = n;
    this->m = m;
    this->Available = Available;
    this->Max = Max;
```

```
    this->Allocation = Allocation;
    this->Need = Need;
    vector<bool>tmp(n,false);
    this->Finish = tmp;
}
void Banker::Request(int id,vector<int>R){
    bool flag = true;
    for (int i=0;i<R.size();i++) {
        if(this->Need[id][i]<R[i]){
            flag = false;
            cout<<"Request[i]>Need[i],请求向量不正确,程序错误返回 "<<endl;
            return;
        }
    }
    for (int i=0;i<R.size();i++) {
        if(this->Available[i]<R[i]){
            flag = false;
            cout<<"Request[i]>Available[i],请求向量无法满足 "<<"线程:"<<id<<" 等待"<<endl;
            return;
        }
    }
    if(flag){
        //试分配
        cout<<"开始试分配..."<<endl;
        for(int i=0;i<R.size();i++){
            this->Available[i] -= R[i];
            this->Allocation[id][i] += R[i];
            this->Need[id][i] -= R[i];
        }
        if (SafeAlgorithm()) {
            cout<<"系统新状态安全,分配完成! "<<endl;
            //输出程序运行结果步骤
            this->Ans();
        }
        else{
            cout<<"安全性算法检查失败! "<<endl;
            flag = false;
            this->Ans();
            //恢复原状态,进程等待
            for(int i=0;i<R.size();i++){
                this->Available[i] += R[i];
                this->Allocation[id][i] -= R[i];
                this->Need[id][i] += R[i];
            }
            cout<<"恢复原状态,进程等待"<<endl;
            return;
        }
    }
}
```

```
bool Banker::SafeAlgorithm(){
    vector<int> work = this->Available;
    vector<int> work_allocation = work;
    vector<bool> finish(this->n,false);
    while(this->Safe.size()<n){
        bool select = false;
        int select_id=0;
        for (int i=0; i<n; i++) {
            if(finish[i]) continue;
            select = true;
            for (int j=0; j<work.size(); j++) {
                if(work[j]<this->Need[i][j]){
                    select = false;
                    break;
                }
            }
            if(select){
                select_id = i;
                break;
            }
        }
        if(!select)
            return false;
        //可分配给此进程
        else{
            //加入安全序列
            finish[select_id] = true;
            this->Finish[select_id] = true;
            this->Safe.push_back(select_id);
            this->Work.push_back(work);
            for (int j=0; j<work.size(); j++)
                work_allocation[j] += this->Allocation[select_id][j];
            this->Work_Allocation.push_back(work_allocation);
            //更新work向量
            work = work_allocation;
        }
    }
    return true;
}
void Banker::Ans(){
    cout<<"算法执行流程:"<<endl;
    cout<<"试分配后Available变量: ";
    for(auto x:this->Available)
        cout<<x<<" ";
    cout<<endl;
    cout<<"进程号"<<"\t"<<"|"<<"\t"<<"Work"<<"\t"<<"|"<<"\t"<<"Need"<<"\t"<<"|"<<"\t"
<<"Allocation"<<"\t"<<"|"<<"\t"<<"Work + Allocation" <<"\t"<<"|"<<"\t" << "Finish"
<<endl;
    for (int i=0; i<this->Safe.size(); i++) {
```

```
        auto id = this->Safe[i];
        cout<<id<<"     \t"<<"|"<<"\t";
        for(auto x:this->Work[i])
            cout<<x<<" ";
        cout<<"\t"<<"|"<<"\t";
        for(auto x:this->Need[id])
            cout<<x<<" ";
        cout<<"\t"<<"|"<<"\t";
        for(auto x:this->Allocation[id])
            cout<<x<<" ";
        cout<<"   \t"<<"|"<<"\t    ";
        for(auto x:this->Work_Allocation[i])
            cout<<x<<" ";
        cout<<"      \t"<<"|"<<"\t";
        cout<<"True ";
        cout<<endl;
    }
    for (int i=0; i<this->n; i++) {
        if(Finish[i]) continue;
        auto id = i;
        cout<<id<<"     \t"<<"|"<<"\t";
        for(int j=0;j<this->Work.size();j++)
            cout<<"  ";
        cout<<"      \t"<<"|"<<"\t";
        for(auto x:this->Need[id])
            cout<<x<<" ";
        cout<<"\t"<<"|"<<"\t";
        for(auto x:this->Allocation[id])
            cout<<x<<" ";
        cout<<"   \t"<<"|"<<"\t    ";
        for(int j=0;j<this->Work_Allocation.size();j++)
            cout<<"  ";
        cout<<"            \t"<<"|"<<"\t";
        cout<<"False";
        cout<<endl;
    }
}

int main(){
    //设置全局变量
    int n = 5; //线程数
    int m = 3; //资源数
    // Available 资源向量
    vector<int> Available = {3,3,2};
    // Max 最大需求矩阵
    vector<vector<int>> Max = {{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
    // Allocation 分配矩阵
    vector<vector<int>> Allocation = {{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
    // Need 需求矩阵
```

```
    vector<vector<int>> Need = {{7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1}};
    Banker banker(n,m,Available,Max,Allocation,Need);
    vector<int> R = {1, 0, 2};
    banker.Request(1,R);
}
```