

XI`AN TECHNOLOGICAL UNIVERSITY

课程设计报告

课程名称 操作系统

专 业： 计算机科学与技术

班 级： 17060722

姓 名： 王 舟

学 号： 17060103115

指导教师： 王淑蓉

成 绩：

2020年 1 月 6 日

**银行家算法**

# 1 需求分析

（1）题目描述：在进程中的资源分配问题上，由于根本上系统的资源数目不够的问题，在某些场景下会导致死锁问题——所有的进程要请求的资源均被其他进程所占有，在没有外力作用的情况下，所有进程不会向下推进。

其中解决死锁问题有这么几种方法：预防死锁、避免死锁、检测与解除死锁。本次实验我们要解决的问题是避免死锁，在这里，我们用到的算法是银行家算法，通过资源的是分配与安全性检测来验证资源是否能够分配给某进程，从而避免了产生死锁的情况。

（2）输入形式以及输入值的范围：首先输入的的是进程的数目，再来资源的数目，最后需要输入两个矩阵，一个是每一个进程的已分配资源情况矩阵；另一个是每一个进程的资源需求矩阵，最后需要输入的是当前可利用的各类资源数目。

（3）输出的形式：

1）安全性检测：如果该时刻状态安全，则输出一个安全序列；

2）如果不安全：输出不安全的原因。

（4）程序的功能：能够动态的检测某时刻的安全性以及动态的检测能否为某进程分配资源。

在该部分中根据设计题目的要求，充分地分析和理解问题，叙述系统的功能要求，明确问题要求做什么？以及限制条件是什么？

# 2 概要设计

（1）数据结构：Banker类：它的成员有进程数目、资源数目、已分配矩阵、资源需求矩阵、当前可用资源数目。

（2）程序模块：

1）菜单模块：显示菜单供用户选择

2）安全性检测：用来进行某时刻系统资源分配的安全性状况

3）银行家算法：调用安全性检测，判断当前是否可以满足某进程对资源的请求。

4）Process：调用菜单函数。

5）input：录入进程信息函数。

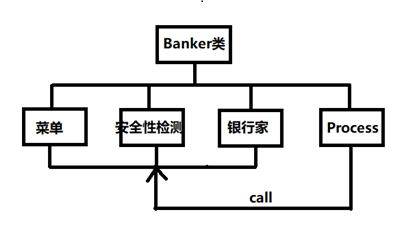
（3）各模块之间的调用关系：

1）menu函数：调用安全性检测算法和银行家算法以及input函数。

2）Process函数：调用menu函数。

3）银行家算法：调用安全性检测。

# 3 详细设计



（1）安全性检测算法：

bool securityCheck() {

int rest = 0;

vector<int> work(available); //工作向量初始值 == avaliable

for (auto& e : work) {

cout << "work" << e << " ";

}

cout << endl;

vector<bool> finish(pro);

for (int i = 0; i < pro; ++i) {

finish[i] = false;

}

while (rest != pro) {

++rest;

for (int i = 0; i < pro; ++i) {

int res\_temp = 0; //记录当前某进程有几类资源可以得到满足

if (!finish[i]) {

for (int j = 0; j < res; ++j) {

if (work[j] >= need[i][j]) {

res\_temp++;

}

}

if (res\_temp == res) {

finish[i] = true;

security.push\_back(i);

for (int m = 0; m < res; ++m) {

work[m] += allocation[i][m];

}

}

}

}

}

//遍历finish数组，如果有不满足要求的就直接返回false；

for (int i = 0; i < pro; ++i) {

if (!finish[i]) {

return false;

}

}

cout << "该状态安全，安全序列为：";

for (int i = 0; i < pro; ++i) {

cout << "p" << security[i] << " ";

}

cout << endl;

return true;

}

（2）资源分配算法

void Request() {

int id; //请求资源的进程编号

vector<int> request;

request.resize(res);

cout << "请输入请求资源的进程编号：";

cin >> id;

cout << "请输入请求的各类资源数目：";

for (int i = 0; i < res; ++i) {

cin >> request[i];

}

vector<vector<int>> allo\_temp(allocation);

vector<vector<int>> need\_temp(need);

vector<int> avail\_temp(available);

for (int i = 0; i < res; ++i) {

if (request[i] > available[i]) {

cout << "请求资源大于现有资源数目，不能满足请求！！！" << endl;

return;

}

}

//现有资源数目，足够满足请求数目，修改allo和need矩阵

for (int i = 0; i < res; ++i) {

allo\_temp[id][i] += request[i];

int minus = need\_temp[id][i] - request[i]; //判断需求 - 请求是否变成了负数

if (minus < 0) {

cout << "请求资源数目大于所需，故不分配！！！" << endl;

return;

}

need\_temp[id][i] = minus;

avail\_temp[i] -= request[i];

}

Banker ban\_temp = \*this;

ban\_temp.allocation = allo\_temp;

ban\_temp.need = need\_temp;

ban\_temp.available = avail\_temp;

bool flag = ban\_temp.securityCheck();

//如果安全性检测通过，就要为其分配资源

if (flag) {

for (int j = 0; j < res; ++j) {

allocation[id][j] += request[j];

need[id][j] = need\_temp[id][j];

security[j] = ban\_temp.security[j];

available[j] = avail\_temp[j];

}

return;

}

cout << "经过安全检测后，不能找到一个安全序列，故不能分配！！！" << endl;

}

# 4 测试

（1）测试安全性检测算法

1）安全状态的序列：

给出已分配矩阵：

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

需求矩阵：

7 4 3

1 2 2

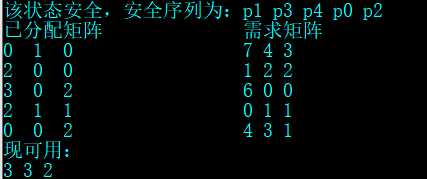
6 0 0

0 1 1

4 3 1

当前可利用资源：3 3 2

运行结果：



2）不安全状态：

给出已分配矩阵：

2 1 1

3 1 1

需求矩阵：

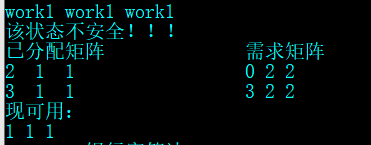
0 2 2

3 2 2

当前可利用资源：

1 1 1

运行结果：

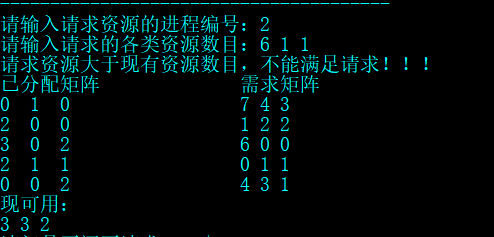


（2）资源请求：

1）请求 > 所需

比如：进程2请求资源（6，1，1）

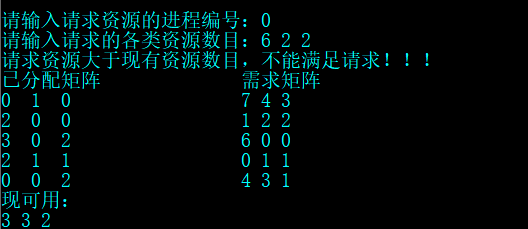
运行结果：



2）请求 <= 所需；但是请求 > 当前可用

比如：进程0发出请求（6，2，2）

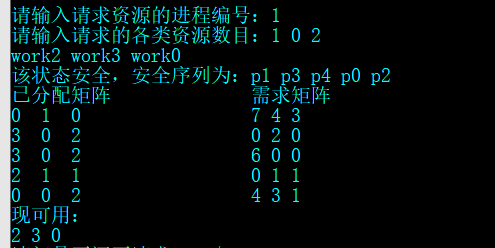
运行结果：



3）请求 <= 所需；并且经过安全性检测后安全，可以分配

比如：进程1发出请求（1，0，2）

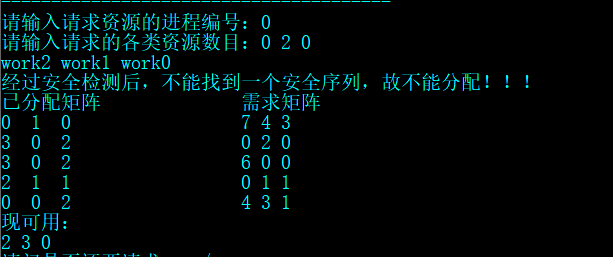
运行结果：



4）请求 <= 所需；但是经过安全性检测后，不安全，并不能分配

比如：进程0发出请求（0，2，0）

运行结果：



# 5 总结

（1）通过本次实验我了解到了死锁的解决办法中的银行家算法的原理以及实现。

（2）在做测试的时候要考虑到所有的情况，利用黑盒测试完成各个功能的测试。

（3）在做该程序时，遇到的问题有，使用C++编写，经常会出现内存越界，要明确各个变量的含义，以及控制条件，最后使用单步调试解决了内存越界问题。

（4）对于资源分配算法，要调用安全性检测算法，如果安全，要分配，并且修改当前每一个矩阵的情况；但如果不安全，就要将试分配的资源再还回去，此时我利用的是将之前的状态保存下来，如果安全，就直接修改了，如果不安全就回溯到安全状态。

（5）了解了银行家算法后，为了后续的死锁的解决办法，RAG图化简法原理其实与银行家算法大同小异，原理相同。当然我们现在写的银行家算法只是能够达到安全的目的，但是还有一个更重要的东西--效率，为了达到安全状态使用了自己写的银行家算法确实解决了死锁问题，但是该算法的开销却没有尽量的考虑。

问题解决之后，更应该考虑的是效率。

# 附录：源代码

#include <iostream>

#include <vector>

#include<stdio.h>

#include<windows.h>

using namespace std;

//设置文字颜色

// 0-黑 1-蓝 2-绿 3-浅绿 4-红 5-紫 6-黄 7-白 8-灰 9-淡蓝

//10-淡绿 11-淡浅绿  12-淡红 13-淡紫 14-淡黄 15-亮白

void SetColor(int color);

//颜色函数

void SetColor(int color)

{

    HANDLE winHandle; //句柄

    winHandle = GetStdHandle(STD\_OUTPUT\_HANDLE);

    //设置文字颜色

    SetConsoleTextAttribute(winHandle, color | 1);

}

class Banker {

private:

    vector<vector<int>> allocation;     //已分配矩阵

    vector<vector<int>> need;           //需求矩阵

    vector<int> available;              //可用资源数

    vector<int> security;               //安全序列

    int pro;        //进程数目

    int res;        //资源数目

public:

    //0.菜单

    void menu() {

        cout << "--------银行家算法---------" << endl;

        cout << "   1.录入信息" << endl;

        cout << "   2.安全性检测" << endl;

        cout << "   3.进程资源请求" << endl;

        cout << "   4.退出" << endl;

        cout << "--------------------------" << endl;

    }

    void Process() {

        while (1) {

            int fun = 0;

            menu();

            cout << "请选择功能：";

            cin >> fun;

            if (fun == 1) {

                cout << "---------------------------------------" << endl;

                input();

                print();

            }

            else if (fun == 2) {

                cout << "---------------------------------------" << endl;

                securityCheck();

                print();

            }

            else if (fun == 3) {

                while (1) {

                cout << "---------------------------------------" << endl;

                    Request();

                    print();

                    cout << "请问是否还要请求：y / n" << endl;

                    char ret;

                    cin >> ret;

                    if (ret == 'y') {

                        continue;

                    }

                    else {

                        break;

                    }

                }

            }

            else if (fun == 4) {

                cout << "谢谢使用！！！" << endl;

                cout << "---------------------------------------" << endl;

                return;

            }

        }

    }

    //1.录入进程信息

    void input() {

        cout << "录入进程数目：";

        cin >> pro;

        cout << "录入资源数目";

        cin >> res;

        allocation.resize(pro,vector<int>(res,0));

        need.resize(pro,vector<int>(res,0));

        available.resize(res);

        cout << "录入已分配矩阵：";

        for (int i = 0; i < pro; ++i) {

            for (int j = 0; j < res; ++j) {

                cin >> allocation[i][j];

            }

        }

        cout << "录入需求矩阵：";

        for (int i = 0; i < pro; ++i) {

            for (int j = 0; j < res; ++j) {

                cin >> need[i][j];

            }

        }

        cout << "录入可用资源情况：";

        for (int i = 0; i < res; ++i) {

            cin >> available[i];

        }

    }

    //2.安全性检测

    bool securityCheck() {

        int rest = 0;

        vector<int> work(available);    //工作向量初始值 == avaliable

        for (auto& e : work) {

            cout << "work" << e << " ";

        }

        cout << endl;

        vector<bool> finish(pro);

        for (int i = 0; i < pro; ++i) {

            finish[i] = false;

        }

        while (rest != pro) {

            ++rest;

            for (int i = 0; i < pro; ++i) {

                int res\_temp = 0;       //记录当前某进程有几类资源可以得到满足

                if (!finish[i]) {

                    for (int j = 0; j < res; ++j) {

                        if (work[j] >= need[i][j]) {

                            res\_temp++;

                        }

                    }

                    if (res\_temp == res) {

                        finish[i] = true;

                        security.push\_back(i);

                        for (int m = 0; m < res; ++m) {

                            work[m] += allocation[i][m];

                        }

                    }

                }

            }

        }

        //遍历finish数组，如果有不满足要求的就直接返回false；

        for (int i = 0; i < pro; ++i) {

            if (!finish[i]) {

                return false;

            }

        }

        cout << "该状态安全，安全序列为：";

        for (int i = 0; i < pro; ++i) {

            cout << "p" << security[i] << " ";

        }

        cout << endl;

        return true;

    }

    //3.进程资源请求

    void Request() {

        int id;     //请求资源的进程编号

        vector<int> request;

        request.resize(res);

        cout << "请输入请求资源的进程编号：";

        cin >> id;

        cout << "请输入请求的各类资源数目：";

        for (int i = 0; i < res; ++i) {

            cin >> request[i];

        }

        vector<vector<int>> allo\_temp(allocation);

        vector<vector<int>> need\_temp(need);

        vector<int> avail\_temp(available);

        for (int i = 0; i < res; ++i) {

            if (request[i] > available[i]) {

                cout << "请求资源大于现有资源数目，不能满足请求！！！" << endl;

                return;

            }

        }

        //现有资源数目，足够满足请求数目，修改allo和need矩阵

        for (int i = 0; i < res; ++i) {

            allo\_temp[id][i] += request[i];

            int minus = need\_temp[id][i] - request[i];  //判断需求 - 请求是否变成了负数

            if (minus < 0) {

                cout << "请求资源数目大于所需，故不分配！！！" << endl;

                return;

            }

            need\_temp[id][i] = minus;

            avail\_temp[i] -= request[i];

        }

        Banker ban\_temp = \*this;

        ban\_temp.allocation = allo\_temp;

        ban\_temp.need = need\_temp;

        ban\_temp.available = avail\_temp;

        bool flag = ban\_temp.securityCheck();

        //如果安全性检测通过，就要为其分配资源

        if (flag) {

            for (int j = 0; j < res; ++j) {

                allocation[id][j] += request[j];

                need[id][j] = need\_temp[id][j];

                security[j] = ban\_temp.security[j];

                available[j] = avail\_temp[j];

            }

            return;

        }

        cout << "经过安全检测后，不能找到一个安全序列，故不能分配！！！" << endl;

    }

    void print() {

        cout << "已分配矩阵 \t\t需求矩阵" << endl;

        for (int i = 0; i < pro; ++i) {

            for (int j = 0; j < res; ++j) {

                cout << allocation[i][j] << "  ";

            }

            cout << "\t\t";

            for (int j = 0; j < res; ++j) {

                cout << need[i][j] << " ";

            }

            cout << endl;

        }

        cout << "现可用：" << endl;

        for (int i = 0; i < res; ++i) {

            cout << available[i] << " ";

        }

        cout << endl;

    }

};

int main() {

    SetColor(10);

    Banker ban;

    ban.Process();

    system("pause");

    return 0;

};