**武汉大学计算机学院**

**本科生课程设计报告**

**银行家算法总体设计与实现**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：操作系统实验

指 导 教 师 ：宋伟 职称 副教授

学 生 学 号 ：2016301500017

学 生 姓 名 ：伍丹梅

二○一八年十一月

**郑 重 声 明**

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名： 伍丹梅 日期： 2018.11.29

摘 要

银行家算法实验的实验目的是熟悉银行家算法，对预防死锁有更深刻的认识。

实验设计主要遵循银行家算法原理

实验内容主要包括：编写实现银行家算法，实现资源的安全分配。

实验结论为通过该算法，我对预防死锁有了更直观的认识。

**关键词：**银行家算法；安全性检查

**目 录**

**1** **实验目的和意义**

1.1 实验目的 ………………………………………………………………………1

1.2 实验意义………………………………………………………………………..1

**2** **实验设计**

2.1 概述………………………………………………………………………………1

2.2 实验原理…………………………………………………………………………1

2.3 实验方案…………………………………………………………………………2

**结论** …………………………………………………………………………………10

**附录** …………………………………………………………………………………11

**1 实验目的和意义**

**1.1** **实验目的**

本实验目的是通过本实验熟悉银行家算法，对预防死锁有更深刻的认识。

**1.2** **实验意义**

本实验意义是了解死锁产生的条件和原因，并采用银行家算法有效得防止死锁的发生，以加深对课堂上所授知识的理解。

**2 实验设计**

**2.1** **概述**

本实验内容为：初始状态下，设置数据结构存储可利用资源向量（Available），最大需求矩阵（MAX），分配矩阵（Allocation），需求矩阵（Need），输入待分配进程队列和所需资源。

设计安全性算法，设置工作向量表示系统可提供进程继续运行的可利用资源数目。

如果进程队列可以顺利执行打印输出资源分配情况，如果进程队列不能顺利执行打印输出分配过程，提示出现死锁位置。

**2.2** **实验原理**

2.2.1**银行家算法原理**

　 银行家算法最初为银行系统设计，以确保银行在发放现金贷款时，不会发生不能满足所有客户需要的情况。在OS设计中，银行家算法可以用它来预防死锁。   
　 为实现银行家算法，每个新进程在进入系统时它必须申明在运行过程中，可能需要的每种资源类型的最大单元数目，其数目不应超过系统所拥有的资源总量。当某一进程请求时，系统会自动判断请求量是否小于进程最大所需，同时判断请求量是否小于当前系统资源剩余量。若两项均满足，则系统预先分配资源并执行安全性检查算法。如果通过安全性检验，那么分配成功，如果不通过，则认为该分配序列不安全，不给予分配。

**2.2.2安全性检查算法**

　 安全性检查算法用于检查系统进行资源分配后是否安全，若安全系统才可以执行此次分配；若不安全，则系统不执行此次分配。   
　　安全性检查算法原理为：在系统预先分配资源后，算法从现有进程列表寻找出一个可执行的进程进行执行，执行完成后回收进程占用资源；进而寻找下一个可执行进程。当进程需求量大于系统可分配量时，进程无法执行。当所有进程均可执行，则产生一个安全执行序列，系统资源分配成功。若进程无法全部执行，即无法找到一条安全序列，则说明系统在分配资源后会不安全，所以此次分配失败。

**2.3** **实验方案**

**2.3.1 数据结构**

可利用资源向量Availble：含有m个元素的数组，其中的每一个元素代表一类可用的资源数目。Available[j]=k，则表示系统中现有第j类资源K个。

最大需求矩阵（MAX）：为n×m矩阵，定义了系统中n个进程中的每一个进程对m类资源的最大需求。Max[i,j]=K，则表示进程i需要第j类资源最大数目为K个。

分配矩阵（Allocation）：为n×m矩阵，定义了系统中每一类资源当前已分配给每一进程的资源数。Allocation[i,j]=K，则表示进程i当前已分配的第j类资源的数目为K。

需求矩阵（Need）：为n×m矩阵，表示每个进程尚需的各类资源数，Need[i,j]=K，则表示进程i还需要第j类资源数目为K个。

**2.3.2 银行家算法**

设Request i是进程Pi的请求矢量，如果Request i[j]=K，表示进程Pi需要第j类资源K个。当Pi发出资源请求后，系统按下述步骤进行检测：

1. 如果Request i[j] ≤Need[i,j]，便转向步骤2，否则认为出错，因为它所需的资源数超过了它所宣布的最大值。
2. 如果Request i[j] ≤Available[i,j]，便转向步骤3，否则，表示尚无足够资源，Pi须等待
3. 系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：

Available[j]=Availabe[j]-Request i[j]

Allocation[i,j]=Allocation[i,j]+Request i[j]

Need[i,j]=Need[i,j]-Request i[j]

1. 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程Pi，以完成本次分配

**2.3.2 安全性检查算法**

1. 设置两个矩阵。

工作矩阵Work；它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work=Available；

Finish：它表示系统是否有足够的资源分配给进程，使之运行完成。开始时Finish[i]=false；当有足够资源分配给进程Pi时，再令Finish[i]=true;

1. 从进程集合中找到一个能满足下述条件的进程：

Finish[i]=false;

Need[i,j]=Work[j];

若找到，执行下一步骤，否则，执行步骤4。

1. 当进程Pi获得资源后，可顺利执行，直至完成，并释放分配给它的资源

Work[j]=Work[j]+Allocation[i,j];

Finish[i]=true;

执行步骤2

1. 如果所有进程的Finish[i]=true满足，则表示系统处于安全状态；否则系统处于不安全状态

**2.3.4 算法流程图**

银行家算法流程图：

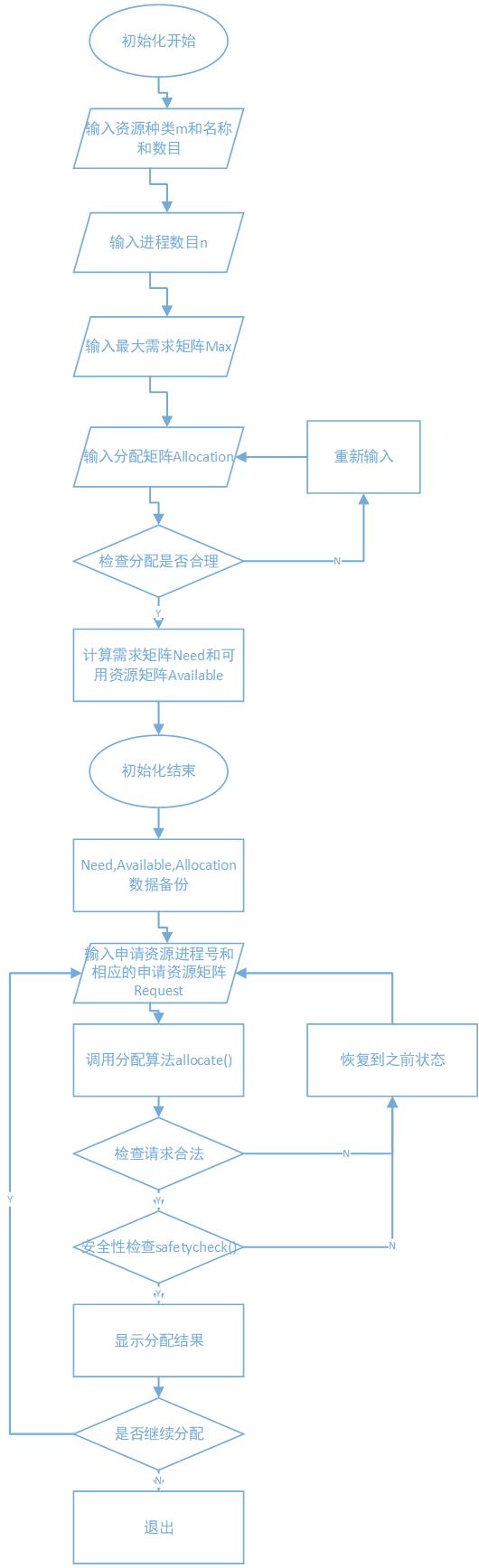


图2-1

分配算法allocate()流程图：

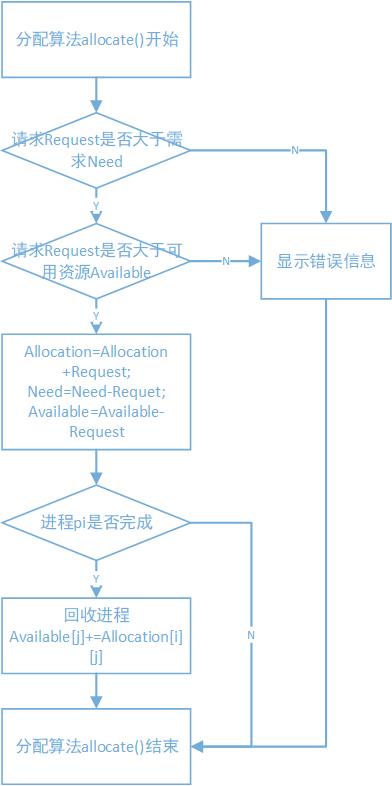


图2-2

安全检查算法safetycheck()：

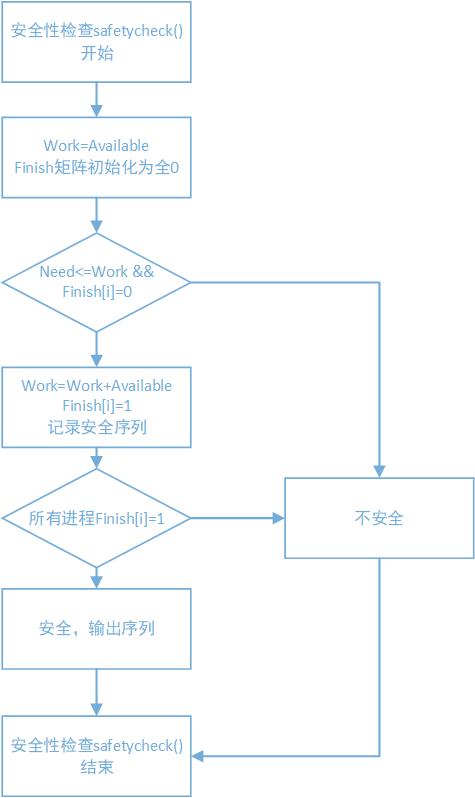
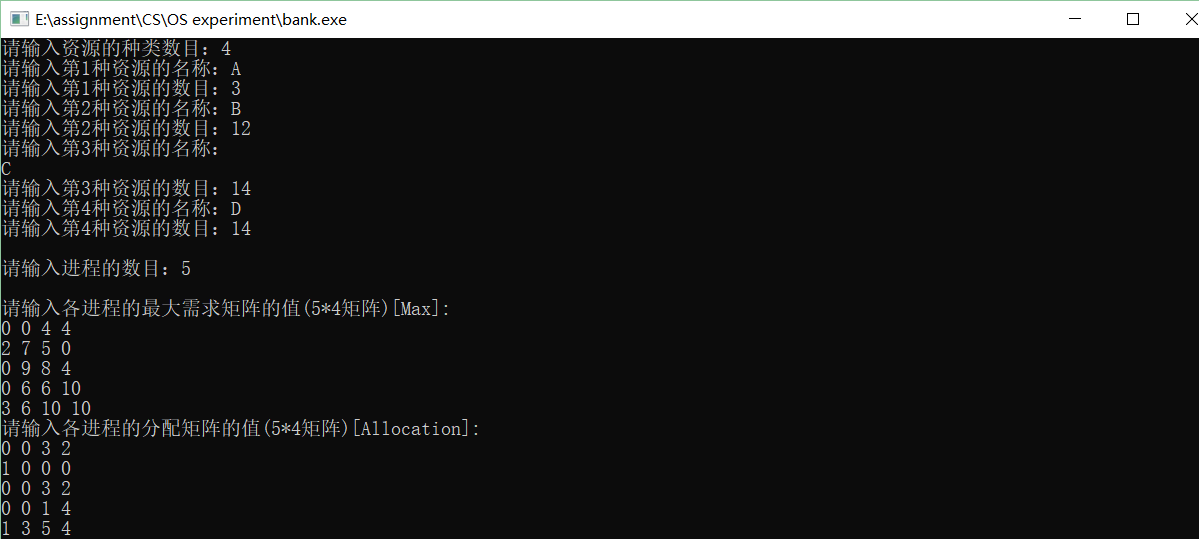


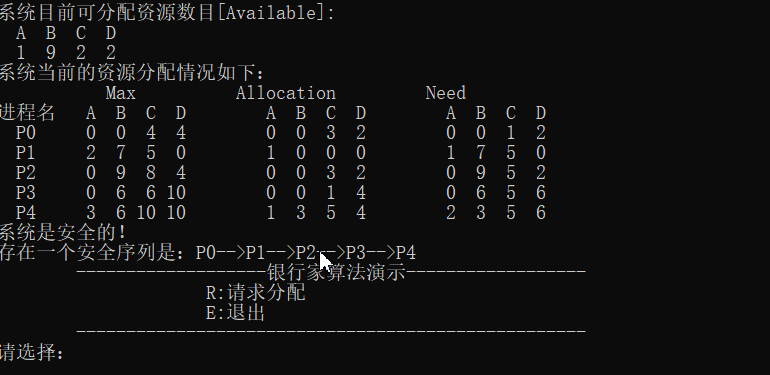
图2-3

**2.3.5 测试**

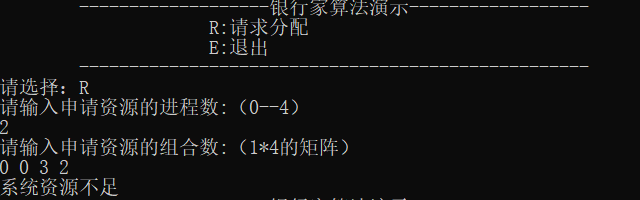
输入资源种类及其名称数量，进程数量，最大需求矩阵和分配矩阵



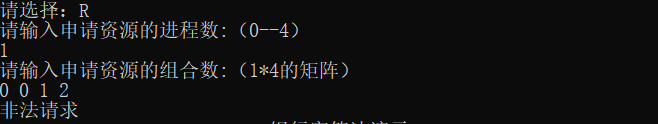
得到可分配矩阵和安全序列检查结果：



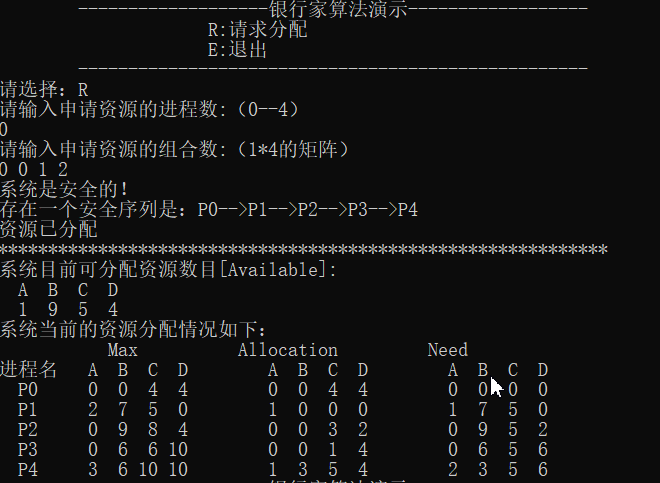
输入R请求分配：可看出请求资源大于系统已有资源会报错



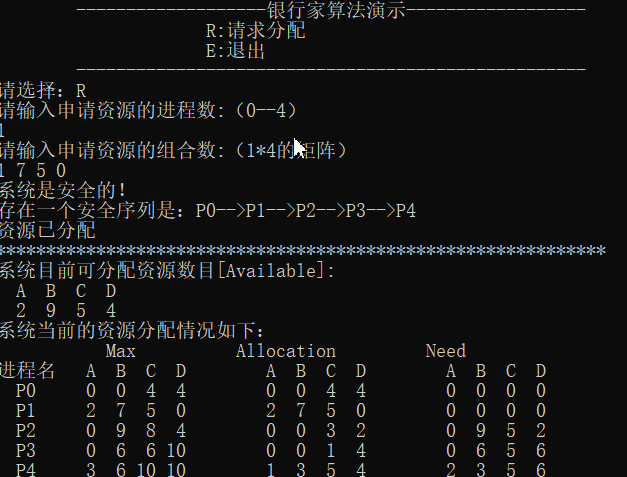
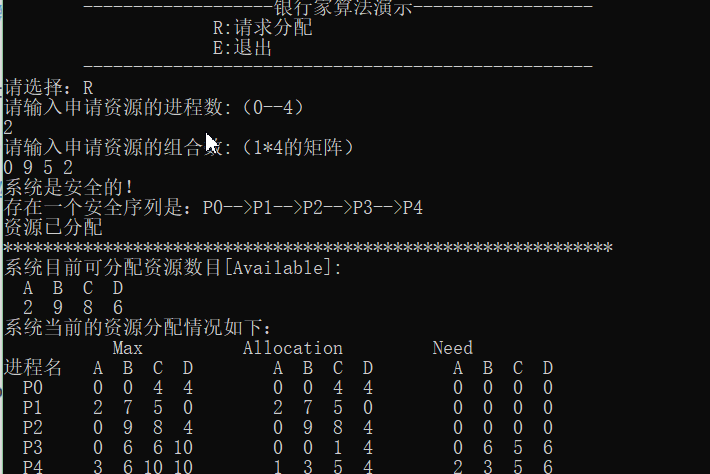
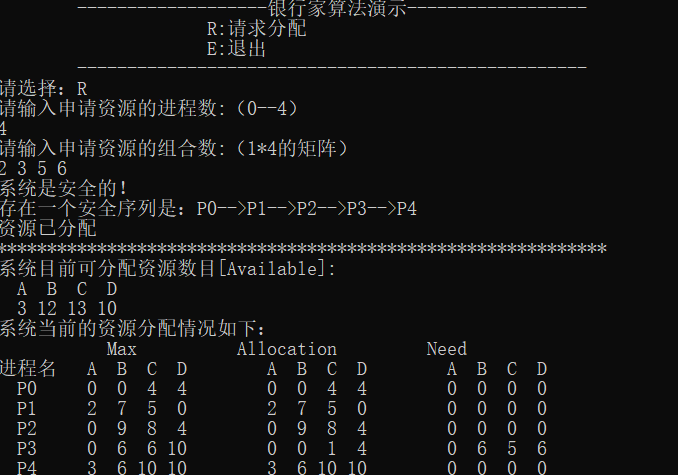
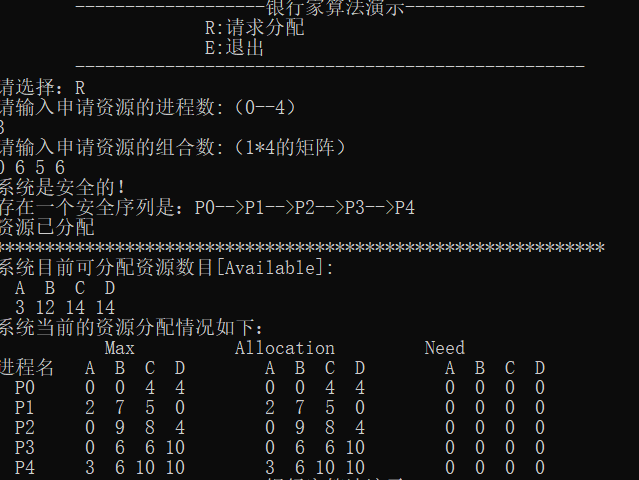
输入R请求分配：可看出请求资源大于需求会报错：



输入R请求分配：请求成功



再继续请求分配：

最后分配完毕和，可看出和之前的资源总数保持一致

**结论**

通过这次实验，我更加了解了死锁产生的条件和原因，银行家算法是如何解决死锁问题的，本实验意义是了解死锁产生的条件和原因。

**附录**

**#include<iostream>**

**#include<iomanip>**

**using namespace std;**

**//定义变量并初始化**

**//进程的最大数**

**int N=10;**

**//资源的最大数**

**int M=50;**

**//可利用资源向量**

**int Available[50] = {0};**

**int Available\_A[50] = {0};**

**//最大需求量**

**int Max[10][50] = {0};**

**//已分配资源量**

**int Allocation[10][50] = {0};**

**int Allocation\_A[10][50] = {0};**

**//尚需资源数目**

**int Need[10][50] = {0};**

**int Need\_N[10][50] = {0};**

**//请求资源向量**

**int Request[100]={0};**

**//工作向量**

**int Work[50] = {0};**

**//分配向量**

**int Finish[50] = {0};**

**//安全序列**

**int Security[10] = {0};**

**//判断值**

**int Bool = 0;**

**//暂存矩阵**

**int Temp[50] = {0};**

**//资源名称**

**char Name[20] = {0};**

**//初始化资源数目M、输入每一种资源的名称和数目以及进程数目N**

**void input\_1(){**

**int m,n;**

**cout<<"请输入资源的种类数目：";**

**cin>>m;**

**if(m<=M){**

**M=m;**

**//输入每一种资源的名称和数目Available**

**for(int i=1;i<=M;i++){**

**cout<<"请输入第"<<i<<"种资源的名称：";**

**cin>>Name[i-1];**

**cout<<"请输入第"<<i<<"种资源的数目：";**

**cin>>Available[i-1];**

**}**

**cout<<endl;**

**}**

**else**

**cout<<"资源数目太多，请重新输入···"<<endl;**

**cout<<"请输入进程的数目：";**

**cin>>n;**

**if(n<=N){**

**N=n;**

**cout<<endl;**

**}**

**else**

**cout<<"进程太多，请重新输入···"<<endl;**

**}**

**//输入最大需求矩阵和分配矩阵**

**void input\_2(){**

**int i,j;**

**//最大需求矩阵**

**cout<<"请输入各进程的最大需求矩阵的值("<<N<<"\*"<<M<<"矩阵)[Max]:"<<endl;**

**for(i=0;i<N;i++){**

**for(j=0;j<M;j++){**

**cin>>Max[i][j];**

**}**

**}**

**//分配矩阵**

**cout<<"请输入各进程的分配矩阵的值("<<N<<"\*"<<M<<"矩阵)[Allocation]:"<<endl;**

**for(i=0;i<N;i++){**

**for(j=0;j<M;j++){**

**cin>>Allocation[i][j];**

**}**

**}**

**}**

**//计算系统目前可用资源和需求矩阵**

**void input\_4();**

**void input\_3(){**

**int i,j;**

**Bool=1;**

**//判断分配是否合理（看是否有分配大于最大需求）**

**for(i=0;i<N;i++){**

**for(j=0;j<M;j++){**

**if(Allocation[i][j]>Max[i][j]){**

**cout<<"不合理分配"<<endl;**

**Bool=0;**

**break;**

**}**

**}**

**}**

**//如果判断合理**

**if(Bool==1){**

**//计算系统目前可用资源**

**for(j=0;j<M;j++){**

**for(i=0;i<N;i++){**

**Temp[j] += Allocation[i][j];**

**}**

**Available[j] -= Temp[j];**

**}**

**//得到需求矩阵**

**for(i=0;i<N;i++){**

**for(j=0;j<M;j++){**

**Need[i][j] = Max[i][j]-Allocation[i][j];**

**}**

**}**

**}**

**//如果判断分配不合理**

**else**

**input\_4();**

**}**

**//重新输入**

**void input\_4(){**

**int i,j;**

**cout<<"请重新输入各进程的分配矩阵的值("<<N<<"\*"<<M<<"矩阵)[Allocation]:"<<endl;**

**for(i=0;i<N;i++){**

**for(j=0;j<M;j++){**

**cin>>Allocation[i][j];**

**}**

**}**

**//继续判断**

**input\_3();**

**}**

**//显示信息**

**void show(){**

**cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;**

**cout<<"系统目前可分配资源数目[Available]:"<<endl;**

**for(int i=0;i<M;i++){**

**cout<<setw(3)<<Name[i];**

**}**

**cout<<endl;**

**for(int i=0;i<M;i++){**

**cout<<setw(3)<<Available[i];**

**}**

**cout<<endl;**

**cout<<"系统当前的资源分配情况如下："<<endl;**

**cout<<" Max Allocation Need"<<endl;**

**cout<<"进程名 ";**

**for(int j=0;j<3;j++){**

**for(int i=0;i<M;i++)**

**cout<<setw(3)<<Name[i];**

**cout<<" ";**

**}**

**cout<<endl;**

**//打印矩阵**

**for(int i=0;i<N;i++){**

**cout<<" P"<<i<<" ";**

**for(int j=0;j<M;j++)**

**cout<<setw(3)<<Max[i][j];**

**cout<<" ";**

**for(int j=0;j<M;j++)**

**cout<<setw(3)<<Allocation[i][j];**

**cout<<" ";**

**for(int j=0;j<M;j++)**

**cout<<setw(3)<<Need[i][j];**

**cout<<endl;**

**}**

**}**

**//判断此时系统是否是安全的**

**int safetycheck(){**

**int i,j;**

**for(j=0;j<M;j++){**

**Work[j]=Available[j];**

**}**

**int s=0;**

**//本层循环保证进行N次排查，使得前几次排查过程中不能满足的进程能继续判断**

**for(int a=0;a<N;a++){**

**//i=0;**

**//本层循环控制从0~N-1个进程的判断**

**for(i=0;i<N;i++){**

**//判断进程资源请求状态是否为未完成**

**if(Finish[i]==0){**

**//判断每一种资源的需求是否小于可用**

**j=0;**

**while(j<M){**

**if(Need[i][j]<=Work[j]){**

**//该条件满足时，说明符合分配规则，进行试分配**

**if(j==M-1){**

**//分配资源**

**for(int b=0;b<M;b++){**

**Work[b] += Allocation[i][b];**

**}**

**//修改状态**

**Finish[i]=1;**

**//记录安全顺序**

**Security[s]=i;**

**s++;**

**}**

**j++;**

**}**

**//只要有一种需求不能被满足，则跳过**

**else**

**break;**

**}**

**}**

**}**

**}**

**Bool=1;**

**//判断是否有没有得到分配的进程**

**//如果有，使Bool的值改为 0**

**for(j=0;j<N;j++){**

**if(Finish[j]==0){**

**Bool=0;**

**break;**

**}**

**}**

**//Bool=1表示所有进程的Finish都等于 1 ，既系统是安全的**

**if(Bool==1){**

**cout<<"系统是安全的！"<<endl;**

**cout<<"存在一个安全序列是：";**

**//输出安全序列**

**for(int j=0;j<N;j++){**

**cout<<"P"<<Security[j];**

**if(j<N-1)**

**cout<<"-->";**

**}**

**cout<<endl;**

**return Bool;**

**}**

**//Bool！=1；说明有没有得到资源的进程，系统不安全**

**else{**

**cout<<"系统不是安全的！"<<endl;**

**return Bool;**

**}**

**}**

**//分配资源**

**int allocate(int p){**

**int j=0;**

**Bool=1;**

**int flag=1;**

**//判断申请的合法性**

**for(j=0;j<M;j++){**

**//判断请求是否大于需求**

**if(Request[j]>Need[p][j]){**

**Bool=0;**

**//大于时将 0 值返回给主调函数**

**return Bool;**

**break;**

**}**

**//判断请求是否大于可用资源**

**else if(Available[j]<Request[j]){**

**Bool=-1;**

**//大于时将 -1 值返回给主调函数**

**return Bool;**

**break;**

**}**

**}**

**//当Bool值还是 1 ，说明请求合理，给予分配**

**if(Bool==1){**

**for(j=0;j<M;j++){**

**Allocation[p][j] += Request[j];**

**Need[p][j] -= Request[j];**

**Available[j] -= Request[j];**

**}**

**//判断进程是否完成，如果完成，回收进程，更新可用资源**

**for(j=0;j<M;j++){**

**if(Need[p][j]!=0)**

**flag=0;**

**break;**

**}**

**if(flag==1){**

**for(j=0;j<M;j++){**

**Available[j]+=Allocation[p][j];**

**}**

**}**

**return Bool;**

**}**

**}**

**//发出资源申请请求并检验请求的合理性**

**void bank(){**

**int i,j,p;**

**//数据备份**

**for(i=0;i<N;i++){**

**Available\_A[i]=Available[i];**

**for(j=0;j<M;j++){**

**Allocation\_A[i][j]=Allocation[i][j];**

**Need\_N[i][j]=Need[i][j];**

**}**

**}**

**//接收申请**

**cout<<"请输入申请资源的进程数:（0--"<<N-1<<"）"<<endl;**

**cin>>p;**

**cout<<"请输入申请资源的组合数:（1\*"<<M<<"的矩阵）"<<endl;**

**for(j=0;j<M;j++){**

**cin>>Request[j];**

**}**

**//调用分配算法**

**int value\_1=allocate(p);**

**//如果请求不合法**

**if(value\_1==0){**

**cout<<"非法请求"<<endl;**

**//退出该函数**

**return;**

**}**

**if(value\_1==-1){**

**cout<<"系统资源不足"<<endl;**

**//退出该函数**

**return;**

**}**

**//如果请求符合规范**

**if(value\_1==1){**

**//执行安全性算法**

**int value\_2=safetycheck();**

**//符合安全性算法**

**if(value\_2==1){**

**cout<<"资源已分配"<<endl;**

**show();**

**}**

**//不符合安全性算法**

**else{**

**cout<<"该请求导致系统不安全，不进行分配。"<<endl;**

**//数据还原**

**for(i=0;i<N;i++){**

**Available[i]=Available\_A[i];**

**for(j=0;j<M;j++){**

**Allocation[i][j]=Allocation\_A[i][j];**

**Need[i][j]=Need\_N[i][j];**

**}**

**}**

**show();**

**}**

**}**

**}**

**//主函数**

**int main(){**

**char choice;**

**int a=1;**

**//输入**

**input\_1();**

**input\_2();**

**input\_3();**

**//显示**

**show();**

**//判断输入的状态是否安全**

**safetycheck();**

**//功能选项**

**while(a==1){**

**cout<<"\t-------------------银行家算法演示------------------"<<endl;**

**cout<<" R:请求分配 "<<endl;**

**cout<<" E:退出 "<<endl;**

**cout<<"\t---------------------------------------------------"<<endl;**

**cout<<"请选择：";**

**cin>>choice;**

**switch(choice){**

**case 'R':**

**bank();**

**break;**

**case 'E':**

**a=0;**

**break;**

**default: cout<<"请正确选择!"<<endl;**

**break;**

**}**

**}**

**return 0;**

**}**

**/\*示例：**

**4**

**A**

**3**

**B**

**12**

**C**

**14**

**D**

**14**

**5**

**max：**

**0 0 4 4**

**2 7 5 0**

**0 9 8 4**

**0 6 6 10**

**3 6 10 10**

**allocate:**

**0 0 3 2**

**1 0 0 0**

**0 0 3 2**

**0 0 1 4**

**1 3 5 4**

**\*/**