课程名称 计算机操作系统

实验成绩

指导教师 曹勇



实 验 报 告

院系 信息工程学院

班级 物联网工程

学号 1601141019

姓名 晏沈威

日期 2018/10/15 地点：505

**­** 实验二、银行家算法

一、 目的和要求

银行家算法是由 Dijkstra 设计的最具有代表性的避免死锁的算法。本实验要求用高级语言编写一

个银行家的模拟算法。通过本实验可以对预防死锁和银行家算法有更深刻的认识。

二、 实验内容

1、 设置数据结构

包括可利用资源向量 （ Availiable ）,最大需求矩阵 （Max ）,分配矩阵（Allocation ）,需求矩阵（Need）

2、 设计安全性算法

设置工作向量 Work 表示系统可提供进程继续运行可利用资源数目， Finish 表示系统是否有足

够的资源分配给进程

三、 实验环境

1、 pc

2、 vc++

四、程序源代码：

/\* 子函数声明 \*/

/\* 子函数声明 \*/

int Isprocessallover(); //判断系统中的进程是否全部运行完毕

void Systemstatus(); //显示当前系统中的资源及进程情况

int Banker(int ,int \*); //银行家算法

void Allow(int ,int \*); //若进程申请不导致死锁，用此函数分配资源

void Forbidenseason(int ); //若发生死锁，则显示原因

/\* 全局变量 \*/

int Availiable[3]={3,3,2}; //初始状态，系统可用资源量

int Max[5][3]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};

//各进程对各资源的最大需求量

int Allocation[5][3]={{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};

//初始状态，各进程占有资源量

int Need[5][3]={{7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1}};

//初始状态时，各进程运行完毕，还需要的资源量

int over[5]={0,0,0,0,0}; //标记对应进程是否得到所有资源并运行完毕

#include <iostream>

using namespace std;

/\* 主函数 \*/

int main()

{

int process=0; //发出请求的进程

int decide=0; //银行家算法的返回值

int Request[3]={0,0,0}; //申请的资源量数组

int sourcenum=0; // 申请的各资源量

/\*判断系统中进程是否全部运行完毕 \*/

step1: if(Isprocessallover()==1)

{

cout<<" 系统中全部进程运行完毕！ ";

return 0;

}

/\*显示系统当前状态 \*/

Systemstatus();

/\*人机交互界面 \*/

step2: cout<<"\n 输入发出请求的进程（输入 “0”退出系统） : ";

cin>>process;

if(process==0)

{

cout<<" 放弃申请 ,退出系统！ ";

return 0;

}

if(process<1||process>5||over[process-1]==1)

{

cout<<" 系统无此进程！ \n";

goto step2;

}

cout<<" 此进程申请各资源（ A ，B， C）数目： \n";

for(int h=0;h<3;h++)

{

cout<<char(65+h)<<" 资源 :";

cin>>sourcenum;

Request[h]=sourcenum;

}

/\*用银行家算法判断是否能够进行分配 \*/

decide=Banker(process,Request);

if (decide==0)

{

/\* 将此进程申请资源分配给它 \*/

Allow(process,Request);

goto step1;

}

else

{

/\* 不能分配，显示原因 \*/

Forbidenseason(decide);

goto step2;

}

}

/\* 子函数 Isprocessallover( )的实现 \*/

int Isprocessallover()

{

int processnum=0;

for(int i=0;i<5;i++)

{

/\* 判断每个进程是否运行完毕 \*/

if(over[i]==1)

processnum++;

}

if(processnum==5)

/\* 系统中全部进程运行完毕 \*/

return 1;

else

return 0;

}

/\* 子函数 Systemstatus( )的实现 \*/

void Systemstatus()

{

cout<<" 此刻系统中存在的进程： \n";

for(int i=0;i<5;i++)

{

if(over[i]!=1)

cout<<"P"<<i+1<<" ";

}

cout<<endl;

cout<<" 此刻系统可利用资源（单位：个） ：\n";

cout<<"A B C\n";

for(int a=0;a<3;a++)

{

cout<<Availiable[a]<<" ";

}

cout<<endl;

cout<<" 此刻各进程已占有资源如下（单位：个） : \n"

<<" A B C\n";

for(int b=0;b<5;b++)

{

if(over[b]==1)

continue;

cout<<"P"<<b+1<<" ";

for(int c=0;c<3;c++)

cout<<Allocation[b][c]<<" ";

cout<<endl;

}

cout<<" 各进程运行完毕还需各资源如下（单位：个） :\n"

<<" A B C\n";

for(int f=0;f<5;f++)

{

if(over[f]==1)

continue;

cout<<"P"<<f+1<<" ";

for(int g=0;g<3;g++)

cout<<Need[f][g]<<" ";

cout<<endl;

}

}

/\* 子函数 Banker(int ,int &) 的实现 \*/

int Banker(int p,int \*R)

{

int num=0; //标记各资源是否能满足各进程需要

int Finish[5]={0,0,0,0,0}; //标记各进程是否安全运行完毕

int work[5]={0,0,0,0,0}; //用于安全检查

int AvailiableTest[3]; //用于试分配

int AllocationTest[5][3]; //同上

int NeedTest[5][3]; //同上

/\*判断申请的资源是否大于系统可提供的资源总量 \*/

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

{

if(\*(R+j)>Availiable[j])

/\* 返回拒绝分配原因 \*/

return 1;

}

/\*判断该进程申请资源量是否大于初始时其申明的需求量 \*/

for(int i=0;i<3;i++)

{

if(\*(R+i)>Need[p-1][i])

/\* 返回拒绝原因 \*/

return 2;

}

/\* 为检查分配的各数据结构赋初值 \*/

for(int t=0;t<3;t++)

{

AvailiableTest[t]=Availiable[t];

}

for(int u=0;u<5;u++)

{

for(int v=0;v<3;v++)

{

AllocationTest[u][v]=Allocation[u][v];

}

}

for(int w=0;w<5;w++)

{

for(int x=0;x<3;x++)

{

NeedTest[w][x]=Need[w][x];

}

}

/\* 进行试分配 \*/

for(int k=0;k<3;k++)

//修改 NeedTest[]

{

AvailiableTest[k]-=\*(R+k);

AllocationTest[p-1][k]+=\*(R+k);

NeedTest[p-1][k]-=\*(R+k);

}

/\* 检测进程申请得到满足后，系统是否处于安全状态 \*/

for(int l=0;l<3;l++)

{

work[l]=AvailiableTest[l];

}

for(int m=1;m<=5;m++)

{

for(int n=0;n<5;n++)

{

num=0;

/\* 寻找用此刻系统中没有运行完的进程 \*/

if(Finish[n]==0&&over[n]!=1)

{

for(int p=0;p<3;p++)

{

if(NeedTest[n][p]<=work[p])

num++;

}

if(num==3)

{

for(int q=0;q<3;q++)

{

work[q]=work[q]+AllocationTest[n][q];

}

Finish[n]=1;

}

}

}

}

for(int r=0;r<5;r++)

{

if(Finish[r]==0&&over[r]!=1)

/\* 返回拒绝分配原因 \*/

return 3;

}

return 0;

}

/\* 子函数 Allow(int ,int &) 的实现 \*/

void Allow(int p,int \*R)

{

cout<<" 可以满足申请！ ";

static int overnum;

/\*对进程所需的资源进行分配 \*/

for(int t=0;t<3;t++)

{

Availiable[t]=Availiable[t]-\*(R+t);

Allocation[p-1][t]=Allocation[p-1][t]+\*(R+t);

Need[p-1][t]=Need[p-1][t]-\*(R+t);

}

/\*分配后判断其是否运行完毕 \*/

overnum=0;

for(int v=0;v<3;v++)

{

if(Need[p-1][v]==0)

overnum++;

}

if(overnum==3)

{

/\* 此进程运行完毕，释放其占有的全部资源 \*/

for(int q=0;q<3;q++)

Availiable[q]=Availiable[q]+Allocation[p-1][q];

/\* 标记该进程运行完毕 \*/

over[p-1]=1;

cout<<" 进程 P"<<p<<" 所需资源全部满足，此进程运行完毕！ \n";

}

}

/\* 子函数 Forbidenseason(int )的实现 \*/

void Forbidenseason(int d)

{

cout<<" 不能满足申请，此进程挂起，原因为： \n";

switch (d)

{

case 1:cout<<"申请的资源量大于系统可提供的资源量！ ";break;

case 2:cout<<"申请的资源中有某种资源大于其声明的需求量！ ";break;

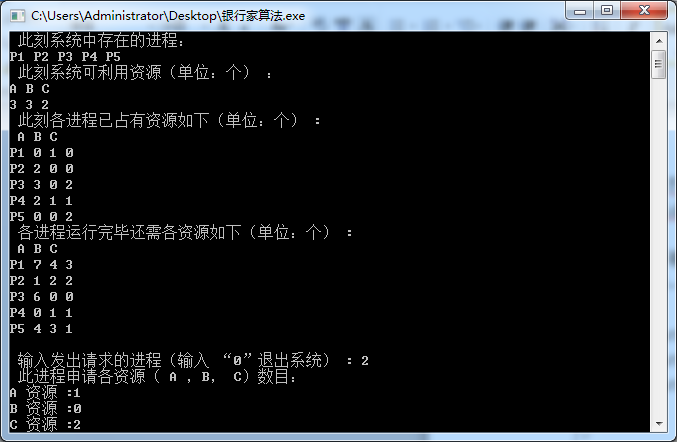
case 3:cout<<"若满足申请，系统将进入不安全状态，可能导致死锁！ ";

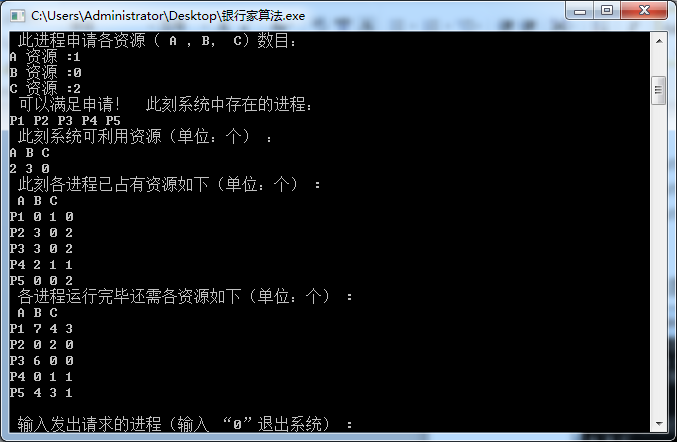
}

}

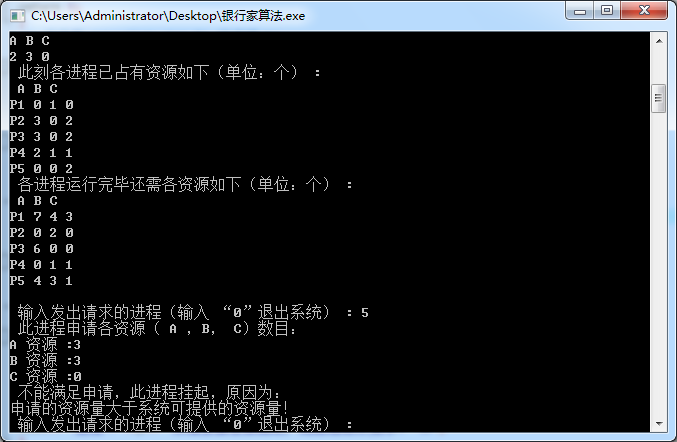
五、实验截图

第一种情况 p2发出请求Request2（1，2，2）

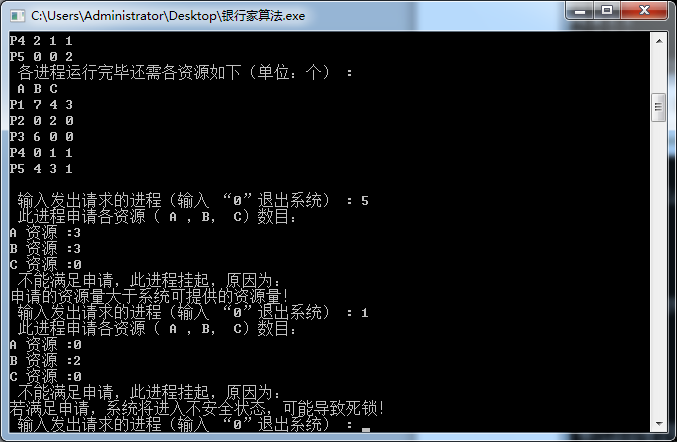




第二种情况P5请求资源 发出请求向量Request5（3，3，0）



第三种情况P1 发出请求向量Request（0，2，0）



六、实验总结

通过本实验我对预防死锁和银行家算法有更深刻的认识，并且了解的银行家算法的具体思想和运算过程