学号 E21514033 专业 软件工程 姓名 何铭春

实验日期 教师签字 成绩

实验报告

【实验名称】 银行家算法和死锁检测算法对比

【实验目的】

1. 了解多道程序系统中多个进程并发执行的资源分配方法；

2. 掌握死锁产生的原因和处理死锁的基本方法；

3. 掌握银行家算法，并设计实验数据验证其避免死锁资源分配策略。

【实验原理】

银行家算法：

每一个新的进程在进入系统时，他必须申明在运行过程中，可能需要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。当进程请求一组资源时，系统必须首先确定是否有足够的资源分配给该进程。若有，再进一步计算在将这些资源分配给进程后，是否会使系统处于不安全状态。如果不会，才将资源分配给它，否则让进程等待。

死锁检测算法：

①可利用资源向量Available，表示了每一类资源的可用数目。

②把不占资源的进程记入L表中。

③寻找是否有能运行完成的进程，如果有释放其资源，并记入L表。

④若不能把所有进程都记入L表，即发生死锁。

【实验内容】

设计一个n个并发进程共享m个系统资源的程序实现银行家算法。要求包含：

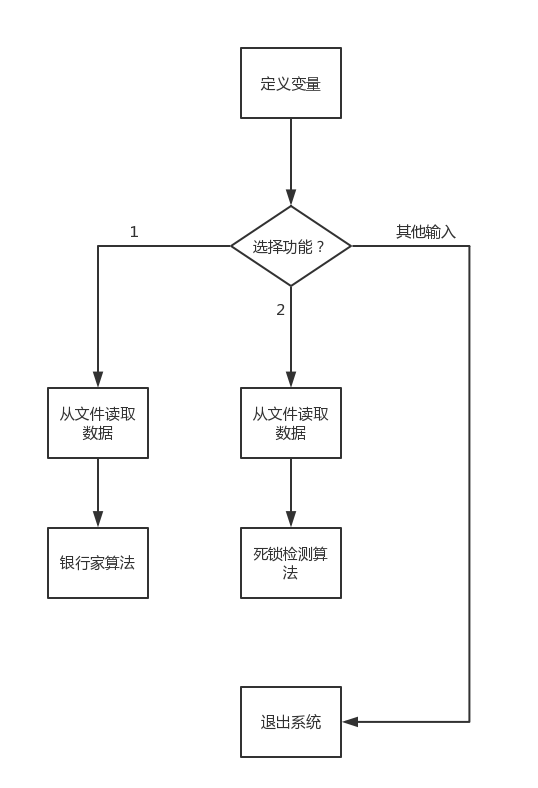
(1) 简单的选择界面；

(2) 能显示当前系统资源的占用和剩余情况；

(3) 进程提出资源请求，如何可以满足则分配，如果不可以满足则阻塞。

同时使用银行家算法、死锁检测算法比较算法运行结果。

程序流程图：



函数定义与功能：

openFile：从指定TXT文件中读取数据给一维数组赋值。

openFile(文件地址,一位数组指针,元素个数);

openFile("Available.txt",Available,m);

openFilePlus：从指定TXT文件中读取数据给一维数组赋值。openFilePlus(文件地址,二维数组指针,行数,列数);

openFilePlus("Max.txt",Max,n,m);

getDate：调用上述函数来为数组赋值

getDate(Available,Allocation,Max,need,m,n);

bank：银行家算法

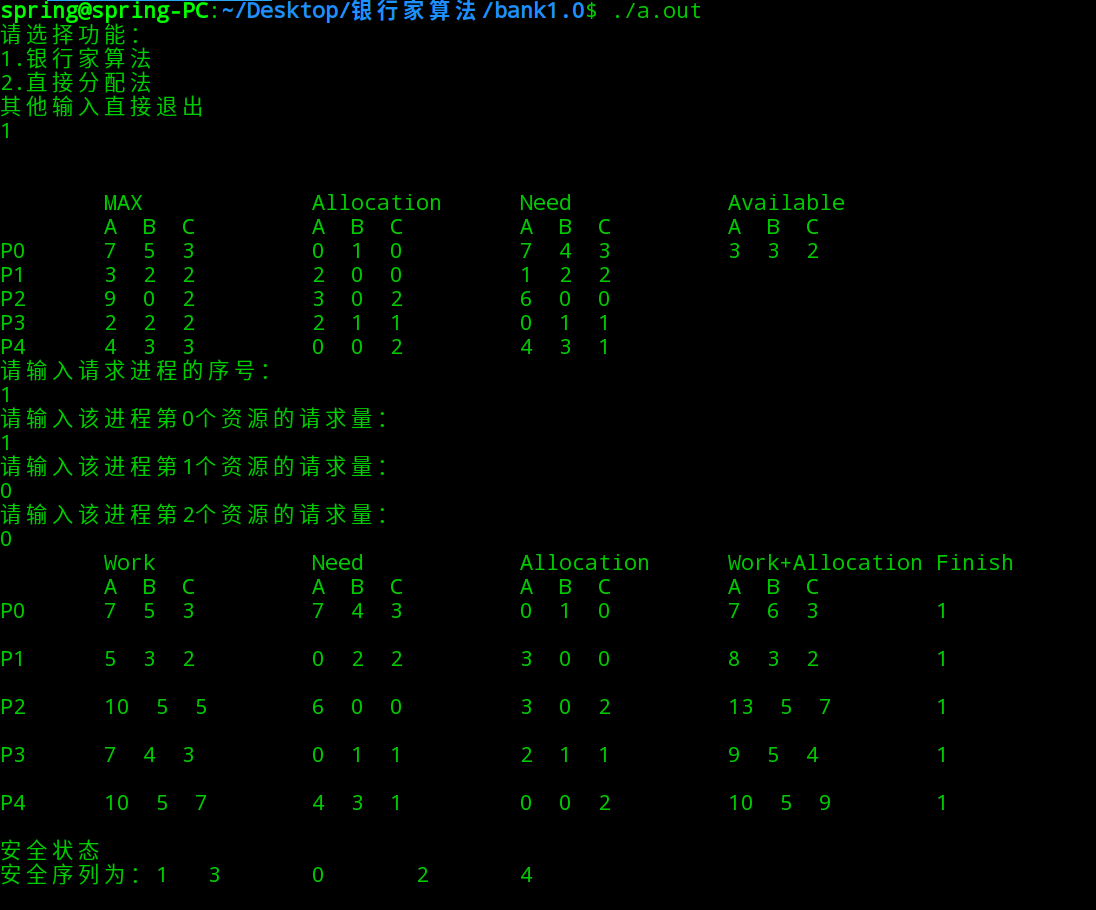
bank(Available,Allocation,Max,need,m,n);

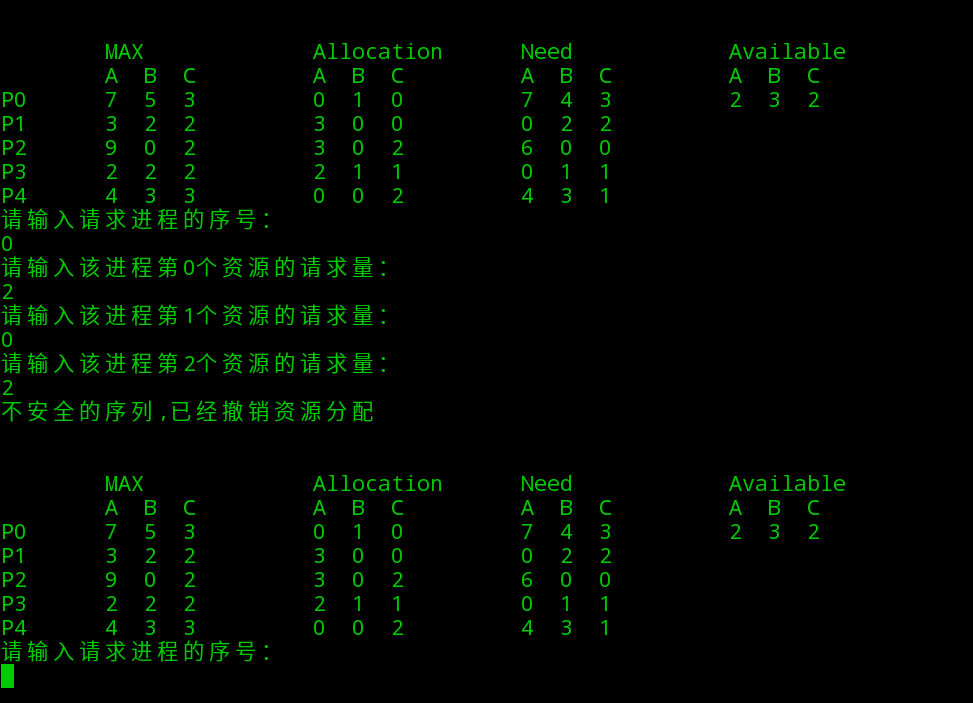
bankFor：死锁检测算法

bankFor(Available,Allocation,Max,need,m,n);

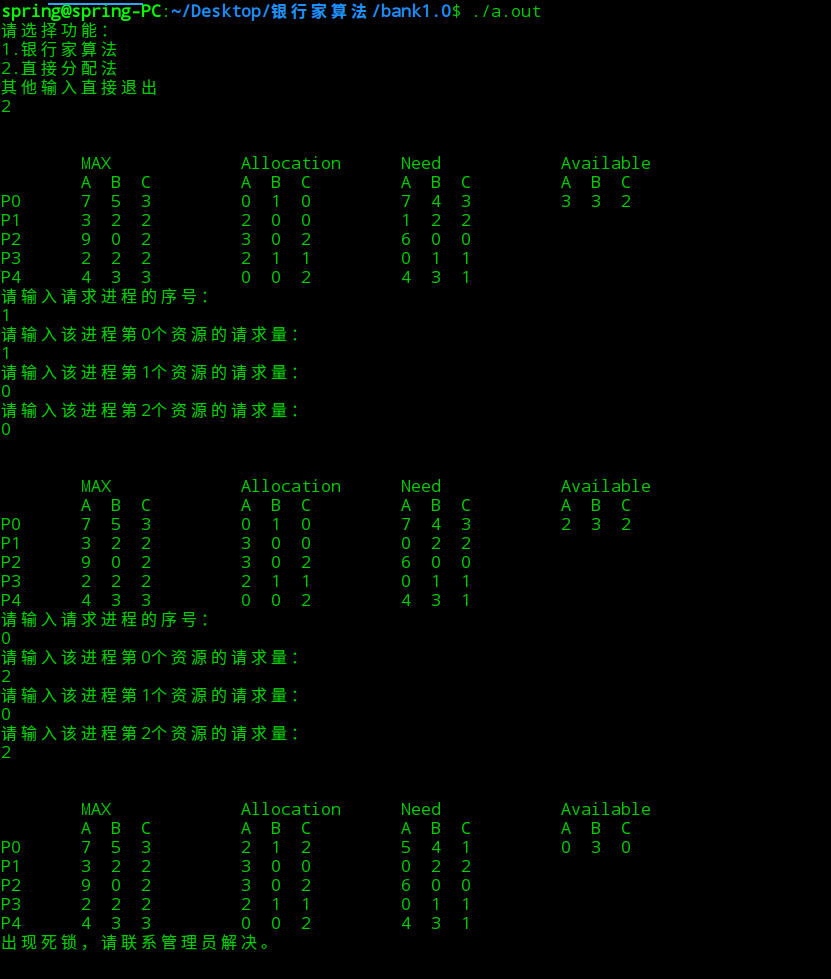
实验截图：

银行家算法：





死锁检测算法：



【小结或讨论】

1.银行家算法能够在一个进程请求资源时判断是否可以给该进程分配资源，从而避免死锁发生。

2.死锁检测算法则直接分配资源，然后查看是否所有进程仍能运行完毕。不能则死锁警报。

3.银行家算法和死锁检测有相同之处，都要判断是否所有进程仍能运行完毕。

4.不同之处在于银行家算法和死锁检测算法的检测时间。

附录：实验源代码

Bank1.0.cpp

主文件，用于调用其他头文件

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> File Name: bank1.0.cpp

> Author: spring

> Mail: 1143515591@qq.com

> Created Time: 2017年11月04日 星期六 15时30分08秒

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

#include <fstream>

#include <string>

#include"openFile.h"

#include"openFilePlus.h"

#include"safety.h"

#include"printAll.h"

#include"bank.h"

#include"bankFor.h"

using namespace std;

void getDate(int\* Available,int\*\* Allocation,int\*\* Max,int\*\* need,int m,int n)

{

openFile("Available.txt",Available,m);

openFilePlus("Max.txt",Max,n,m);

openFilePlus("Allocation.txt",Allocation,n,m);

openFilePlus("need.txt",need,n,m);

}

int main(void)

{

int m=3;

int n=5;

//读取Available文件，为Available数组赋值。

int \*Available=new int(m);

//读取Max文件，为Max数组赋值

int \*\*Max=new int\*[n];

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

{

Max[i]=new int(m);

}

//读取Allocation文件，为Allocation数组赋值

int \*\*Allocation=new int\*[n];

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

{

Allocation[i]=new int(m);

}

//读取need文件，为need数组赋值

int \*\*need=new int\*[n];

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

{

need[i]=new int(m);

}

int xyz;

while(1)

{

cout<<"请选择功能："<<endl;

cout<<"1.银行家算法"<<endl;

cout<<"2.直接分配法"<<endl;

cout<<"其他输入直接退出"<<endl;

cin>>xyz;

if(xyz==1)

{

getDate(Available,Allocation,Max,need,m,n);

bank(Available,Allocation,Max,need,m,n);

}

else if(xyz==2)

{

getDate(Available,Allocation,Max,need,m,n);

bankFor(Available,Allocation,Max,need,m,n);

}

else

{

break;

}

}

return 0;

}

openfile.h

保存openfile函数

using namespace std;

//读取文件的数据,并赋值给一维数组

void openFile(const char \*filePath,int \*p,int num)

{

ifstream file;

file.open(filePath,ios::in);

if(!file.is\_open())

cout<<"文件不存在，读取错误。"<<endl;

std::string strLine;

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

{

getline(file,strLine);

p[i]=std::stoi(strLine);

}

}

openFilePlus.h

保存openFilePlus函数

using namespace std;

//读取文件数据，并赋值给二维数组

void openFilePlus(const char \*filePath,int \*\*p,int n,int m)

{

ifstream file;

file.open(filePath,ios::in);

if(!file.is\_open())

cout<<"文件不存在，读取错误。"<<endl;

std::string strLine;

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

{

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

{

getline(file,strLine);

p[i][j]=std::stoi(strLine);

}

}

}

safety.h

安全性算法函数

using namespace std;

int safety(int \*Available,int \*\*Max,int \*\*Allocation,int \*\*need,int m,int n)

{

int \*order=new int(n);

int \*Work=new int(m);

int \*Finish=new int(n);

int \*\*workLoad=new int\*[n];

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

{

workLoad[i]=new int(m);

}

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

{

Work[i]=Available[i];

}

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

{

Finish[i]=0;

}

int counter=0;

while(1)

{

int choose;

choose=-1;

int ready;

ready=0;

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

{

if(Finish[i]==0)

{

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

{

if(need[i][j]>Work[j])

ready=-1;

}

if(ready==-1)

{

ready=0;

}

else

{

choose=i;

break;

}

}

}

if(choose==-1)

{

break;

}

order[counter]=choose;

counter++;

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

{

Work[i]=Work[i]+Allocation[choose][i];

workLoad[choose][i]=Work[i];

}

Finish[choose]=1;

}

int flag=0;

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

{

if(Finish[i]==0)

{

flag=1;

}

}

if(flag)

{

cout<<"不安全的序列,已经撤销资源分配"<<endl;

return 1;

}

else

{

cout<<'\t'<<"Work"<<'\t'<<'\t'<<"Need"<<'\t'<<'\t'<<"Allocation"<<'\t'<<"Work+Allocation"<<'\t'<<"Finish"<<endl;

cout<<'\t'<<"A B C"<<'\t'<<'\t'<<"A B C"<<'\t'<<'\t'<<"A B C"<<'\t'<<'\t'<<"A B C"<<endl;

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

{

cout<<"P"<<i<<'\t';

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

{

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

}

cout<<'\t';

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

{

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

}

cout<<'\t';

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

{

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

}

cout<<'\t';

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

{

cout<<workLoad[i][j]+Allocation[i][j]<<" ";

}

cout<<'\t';

cout<<Finish[i]<<endl;

cout<<endl;

}

cout<<"安全状态"<<endl;

cout<<"安全序列为：";

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

{

cout<<order[i]<<'\t';

}

cout<<endl;

}

return 0;

}

printAll.h

输出数组

using namespace std;

int printAll(int \*\*Max,int \*\*Allocation,int \*\*need,int \*Available,int m,int n)

{

cout<<endl;

cout<<endl;

cout<<'\t'<<"MAX"<<'\t'<<'\t'<<"Allocation"<<'\t'<<"Need"<<'\t'<<'\t'<<"Available"<<endl;

cout<<'\t'<<"A B C"<<'\t'<<'\t'<<"A B C"<<'\t'<<'\t'<<"A B C"<<'\t'<<'\t'<<"A B C"<<endl;

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

{

cout<<"P"<<i<<'\t';

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

{

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

}

cout<<'\t';

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

{

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

}

cout<<'\t';

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

{

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

}

cout<<'\t';

if(i==0)

{

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

{

cout<<Available[j]<<" ";

}

}

cout<<endl;

}

}

bank.h

调用safety函数，银行家算法主函数

using namespace std;

int bank(int \*Available,int \*\*Allocation,int \*\*Max,int \*\*need,int m,int n)

{

//声明请求的变量

int requesetProcess;

int \*requeset=new int(m);

//循环所需的提示变量

int flag1;

int flag2;

int flag3;

int spring;

while(1)

{

printAll(Max,Allocation,need,Available,m,n);

//循环运行函数

while(1)

{

//将提示变量赋值为0

flag1=0;

flag2=0;

flag3=0;

cout<<"请输入请求进程的序号："<<endl;

cin>>requesetProcess;

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

{

cout<<"请输入该进程第"<<i<<"个资源的请求量："<<endl;

cin>>requeset[i];

}

//银行家算法第一步

//如果存在请求大于Need，flag1=1，出错。

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

{

if (requeset[i]>need[requesetProcess][i])

{

flag1=1;

}

}

//判断flag1是否为1，如果是1，为flag2，flag3赋值为1，也就是跳过下面所有语句

if (flag1==1)

{

flag2=1;

flag3=1;

cout<<"所需要资源超过他宣布的最大值"<<endl;

}

//如果flag1=0，开始银行家算法第二步

if (flag1==0)

{

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

{

if (requeset[i]>Available[i])

{

flag2=1;

}

}

if (flag2==1)

{

flag3=1;

cout<<"超过系统所能分配的资源"<<endl;

}

if(flag2==0)

{

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

{

Available[i]=Available[i]-requeset[i];

Allocation[requesetProcess][i]=Allocation[requesetProcess][i]+requeset[i];

need[requesetProcess][i]=need[requesetProcess][i]-requeset[i];

}

//安全性算法

spring=safety(Available,Max,Allocation,need,m,n);

int count=0;

if (spring)

{

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

{

Available[i]=Available[i]+requeset[i];

Allocation[requesetProcess][i]=Allocation[requesetProcess][i]-requeset[i];

need[requesetProcess][i]=need[requesetProcess][i]+requeset[i];

}

}

else

{

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

{

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

{

count=count+need[i][j];

}

if (count==0)

{

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

{

Available[k]=Available[k]+Allocation[i][k];

Allocation[i][k]=0;

}

}

count=0;

}

}

break;

}

}

//判断流程

}

}

return 0;

}

bankFor.h

死锁检测算法

using namespace std;

int bankFor(int \*Available,int \*\*Allocation,int \*\*Max,int \*\*need,int m,int n)

{

//声明请求的变量

int requesetProcess;

int \*requeset=new int(m);

//循环所需的提示变量

int flag1;

int flag2;

int flag3;

int dead;

int forCount;

int count;

while(1)

{

printAll(Max,Allocation,need,Available,m,n);

//循环运行函数

while(1)

{

dead=0;

forCount=0;

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

{

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

{

forCount=forCount+need[i][j];

}

if (forCount==0)

{

dead++;

}

}

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

{

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

{

if(need[i][j]>Available[j])

{

dead++;

break;

}

}

}

if (dead==n)

{

cout<<"出现死锁，请联系管理员解决。"<<endl;

cout<<endl;

cout<<endl;

break;

}

//将提示变量赋值为0

flag1=0;

flag2=0;

flag3=0;

cout<<"请输入请求进程的序号："<<endl;

cin>>requesetProcess;

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

{

cout<<"请输入该进程第"<<i<<"个资源的请求量："<<endl;

cin>>requeset[i];

}

//银行家算法第一步

//如果存在请求大于Need，flag1=1，出错。

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

{

if (requeset[i]>need[requesetProcess][i])

{

flag1=1;

}

}

//判断flag1是否为1，如果是1，为flag2，flag3赋值为1，也就是跳过下面所有语句

if (flag1==1)

{

flag2=1;

flag3=1;

cout<<"所需要资源超过他宣布的最大值"<<endl;

}

//如果flag1=0，开始银行家算法第二步

if (flag1==0)

{

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

{

if (requeset[i]>Available[i])

{

flag2=1;

}

}

if (flag2==1)

{

flag3=1;

cout<<"超过系统所能分配的资源"<<endl;

}

if(flag2==0)

{

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

{

Available[i]=Available[i]-requeset[i];

Allocation[requesetProcess][i]=Allocation[requesetProcess][i]+requeset[i];

need[requesetProcess][i]=need[requesetProcess][i]-requeset[i];

}

//安全性算法

count=0;

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

{

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

{

count=count+need[i][j];

}

if (count==0)

{

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

{

Available[k]=Available[k]+Allocation[i][k];

Allocation[i][k]=0;

}

}

count=0;

}

break;

}

}

}

//判断流程

if (dead==n)

{

break;

}

}

return 0;

}