|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 数学与信息科学学院 | | | | | | |  |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | **实验报告** | | | | | | |  |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | **课程名称：** | | **计算机操作系统** | | | | |  |
|  | **姓 名：** | | **任希恒** | | | | |  |
|  | **学 号：** | | **541910010217** | | | | |  |
|  | **专业班级：** | | **信息与计算科学专业19-02班** | | | | |  |
|  | **指导教师：** | | **时海亮** | | | | |  |
|  |  | |  | | | | |  |
|  |  | |  | | | | |  |
|  |  | **2021-2022** | | **学年第** | **2** | **学期** |  |  |

**实验3 银行家算法**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验日期：** | **2022年04月30日** | **实验类型：** | **设计型** | **实验成绩：** |  |

**一、实验目的**

通过本实验加深对死锁、安全状态、安全序列等概念的理解，熟悉死锁避免方法的经典算法银行家算法的设计。

**二、实验内容**

1．模拟实现利用银行家算法进行资源分配。

**三、实验代码及结果**

## 1.银行家算法

代码如下所示：

#include <iostream.h>

#define MAXPROCESS 10 /\*最大进程数\*/

#define MAXRESOURCE 100 /\*最大资源数\*/

int Available[MAXRESOURCE]; /\*可用资源数组\*/

int Max[MAXPROCESS][MAXRESOURCE]; /\*最大需求矩阵\*/

int Allocation[MAXPROCESS] [MAXRESOURCE]; /\*分配矩阵\*/

int Need[MAXPROCESS][MAXRESOURCE]; /\*需求矩阵\*/

int Request[MAXPROCESS][MAXRESOURCE]; /\*进程需要资源数\*/

bool Finish[MAXPROCESS]; /\*系统是否有足够的资源分配\*/

int P[MAXPROCESS]; /\*记录序列\*/

int m; /\*m个资源, n个进程\*/

int n;

void Init ();

bool Safe ();

void Bank ();

void main ()

{

Init ();

Safe();

Bank ();

}

void Init () //初始化算法

{

int i,j;

cout<<"请输入进程的数目: ";

cin>>n;

cout<<"请输入资源的种类: ";

cin>>m;

cout<<"请输入每个进程最多所需的各资源数，按照"<<n<<"x"<<m<<"矩阵输入"<<endl;

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

{

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

{

cin>>Max[i][j];

}

}

cout<<"请输入每个进程已分配的各资源数，也按照"<<n<<"x"<<m<<"矩阵输入"<<endl;

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

{

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

{

cin>>Allocation[i][j];

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

if(Need[i][j]<0)

{

cout<<"您输入的第"<<i+1<<"个进程所拥有的第"<<j+1<<"类资源数错误,请重新输入: "<<endl;

j--;

continue;

}

}

}

}

void Bank () /\*银行家算法\*/

{

int i,j,pno;

char again;

bool input\_again = false;

while(1)

{

cout<<"请输入要申请资源的进程号（注:第1个进程号为0，依次类推) "<<endl;

cin>>pno;

cout<<"请输入进程所请求的各类资源的数量"<<endl;

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

{

if(Request[pno][j]>Need[pno][j])

{

cout<<"您输入的请求数超过进程的需求量!请重新输入!"<<endl;

input\_again=true;

break;

}

if(Request[pno][j]>Available[j])

{

cout<<"您输入的请求数超过系统有的资源数!请重新输入! "<<endl;

input\_again=true;

break;

}

}

if(input\_again)

{

continue;

}

//试探分配资源

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

{

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

Allocation[pno][j]+=Request[pno][j];

Need[pno][i]-=Request[pno][3];

}

//检查系统安全性

if(Safe ())

{

cout<<"同意分配请求! "<<endl;

}

else

{

cout<<"您的请求被拒绝! "<<endl;

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

{

Available[j]+=Request[pno][j];

Allocation[pno][j]-=Request[pno][j];

Need[pno][j]+=Request[pno][j];

}

}

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

{

Finish[i]=false;

}

cout<<"您还想再次请求分配吗?是请按y/Y,否请按其它键"<<endl;

cin>>again;

if(again=='y'||again=='Y')

{

continue;

}

break;

}

}

bool Safe() /\*安全性算法\*/

{

//初始化,设置两个向量

int i,j,k,l=0;

int work [MAXRESOURCE] ;

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

{

work [j]=Available[j];

}

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

{

Finish [ i]=false;

}

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

{

if(Finish [i ]==true)

{

continue;

}

else

{

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

{

if (Need[i][j]>work[j])

{

break;

}

}

if(j==m)

{

Finish[i]=true;

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

{

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

}

P[l++]=i;

i=-1;

}//重新开始查找符合条件的进程

else

{

continue; //查询下一个进程

}

}

if (l==n)

{

cout<<"系统是安全的"<<endl;

cout<<"安全序列:"<<endl;

for (i=0;i<l; i++)

{

cout<<P[i] ;

if(i!=l-1)

{

cout<<"-->";

}

}

cout<<""<<endl;

return true;

}

}

cout<<"系统是不安全的! "<<endl;

return false;

}

实验结果如图所示：

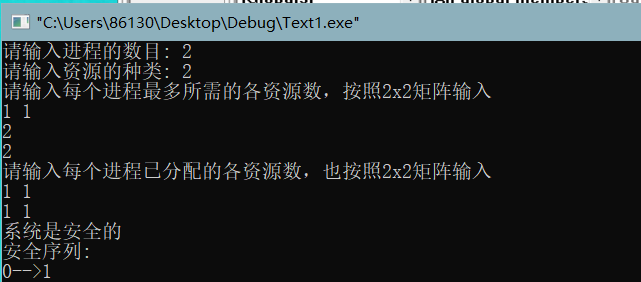


图1

**四、实验总结**

1、死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

2、在避免死锁的方法中，允许进程动态地申请资源，系统在进行资源分配之前，先计算资源分配的安全性。若此次分配不会导致系统进入不安全状态，则将资源分配给进程；否则，进程等待。

3、安全序列:一个进程序列(P1，…，Pn]是安全的，如果对于每一个进程Pi(1≤i≤n)，它以后尚需要的资源量不超过系统当前剩余资源量与所有进程Pj(j<i )当前占有资源量之和。

4、寻找一个安全序列：从进程P0开始遍历,一个一个尝试,一遍下来后,第二遍可以向上依次遍历或者从上往下遍历,直到找出一个安全序列。找所有进程中need最少的,然后看available是否满足,满足后此进程排第一，接着在找第二个need最少的,然后依次。如果过程中发现不能满足need<available,则换一条路。

5、可以把操作系统看作是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。

为保证资金的安全，银行家规定：

(1) 当一个顾客对资金的最大需求量不超过银行家现有的资金时就可接纳该顾客；

(2) 顾客可以分期贷款，但贷款的总数不能超过最大需求量；

(3) 当银行家现有的资金不能满足顾客尚需的贷款数额时，对顾客的贷款可推迟支付，但总能使顾客在有限的时间里得到贷款；

(4) 当顾客得到所需的全部资金后，一定能在有限的时间里归还所有的资金.

操作系统按照银行家制定的规则为进程分配资源，当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。当进程在执行中继续申请资源时，先测试该进程本次申请的资源数是否超过了该资源所剩余的总量。若超过则拒绝分配资源，若能满足则按当前的申请量分配资源，否则也要推迟分配。