# 实验06 银行家算法

## 一．实验目的

1. 理解银行家算法。
2. 掌握进程安全性检查的方法与资源分配的方法。

## 二．实验内容与基本要求

编制模拟银行家算法的程序，并以下面给出的例子验证所编写的程序的正确性。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | 已占资源 | | | | 最大需求数 | | | |
| A | B | C | D | A | B | C | D |
| P0 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 |
| P2 | 1 | 3 | 5 | 4 | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 |
| P4 | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 |

现在系统中A、B、C、D 4类资源分别还剩1、5、2、0个，请按银行家算法回答：

1. 现在系统是否处于安全状态？
2. 如果现在进程P1提出需要（0、4、2、0）个资源的请求，系统能否满足它的请求？

## 三．实验原理

## 1、死锁概念

在多道程序系统中，虽可借助于多个进程的并发执行，来改善系统的资源利用率，提高系统的吞吐量，但可能发生一种危险**—**死锁。所谓死锁(Deadlock),是指多个进程在运行中因争夺资源而造成的一种僵局(Deadly\_Embrace),当进程处于这种僵持状态时，若无外力作用，它们都将无法再向前推进。一组进程中，每个进程都无限等待被该组进程中另一进程所占有的资源，因而永远无法得到的资源，这种现象称为进程死锁，这一组进程就称为死锁进程。

## 2、关于死锁的一些结论：

 参与死锁的进程最少是两个

 （两个以上进程才会出现死锁）

 参与死锁的进程至少有两个已经占有资源

 参与死锁的所有进程都在等待资源

 参与死锁的进程是当前系统中所有进程的子集

注：如果死锁发生，会浪费大量系统资源，甚至导致系统崩溃。

## 3、资源分类。

永久性资源：

可以被多个进程多次使用（可再用资源）

 可抢占资源

 不可抢占资源

临时性资源：只可使用一次的资源；如信号量,中断信号，同步信号等（可消耗性资源）

　　　　　　“申请--分配--使用--释放”模式

4、产生死锁的四个必要条件：

互斥使用（资源独占）、不可强占（不可剥夺）、请求和保持（部分分配，占有申请）、循环等待。

1) 互斥使用（资源独占）

一个资源每次只能给一个进程使用。

2) 不可强占（不可剥夺）

资源申请者不能强行的从资源占有者手中夺取资源，资源只能由占有者自愿释放。

3) 请求和保持（部分分配，占有申请）

一个进程在申请新的资源的同时保持对原有资源的占有（只有这样才是动态申请，动态分配）。

4) 循环等待 ­­­

存在一个进程等待队列

{P1 , P2 , … , Pn},

其中P1等待P2占有的资源，P2等待P3占有的资源，…，Pn等待P1占有的资源，形成一个进程等待环路。

## 5、死锁预防:

定义:在系统设计时确定资源分配算法，保证不发生死锁。具体的做法是破坏产生死锁的四个必要条件之一。

①破坏“不可剥夺”条件

在允许进程动态申请资源前提下规定，一个进程在申请新的资源不能立即得到满足而变为等待状态之前，必须释放已占有的全部资源，若需要再重新申请

②破坏“请求和保持”条件。

要求每个进程在运行前必须一次性申请它所要求的所有资源，且仅当该进程所要资源均可满足时才给予一次性分配。

③破坏“循环等待”条件

采用资源有序分配法：

把系统中所有资源编号，进程在申请资源时必须严格按资源编号的递增次序进行，否则操作系统不予分配。

## 6．安全状态与不安全状态

安全状态：

如果存在一个由系统中所有进程构成的安全序列P1，…Pn，则系统处于安全状态。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，系统处于安全状态 (安全状态一定是没有死锁发生的)

不安全状态:不存在一个安全序列，不安全状态一定导致死锁。

## 7. 银行家算法的基本思想

银行家算法的基本思想是分配资源之前，判断系统是否是安全的；若是，才分配。每分配一次资源就测试一次是否安全，若不是，资源全部就位后才测试，注意理解函数的循环顺序。

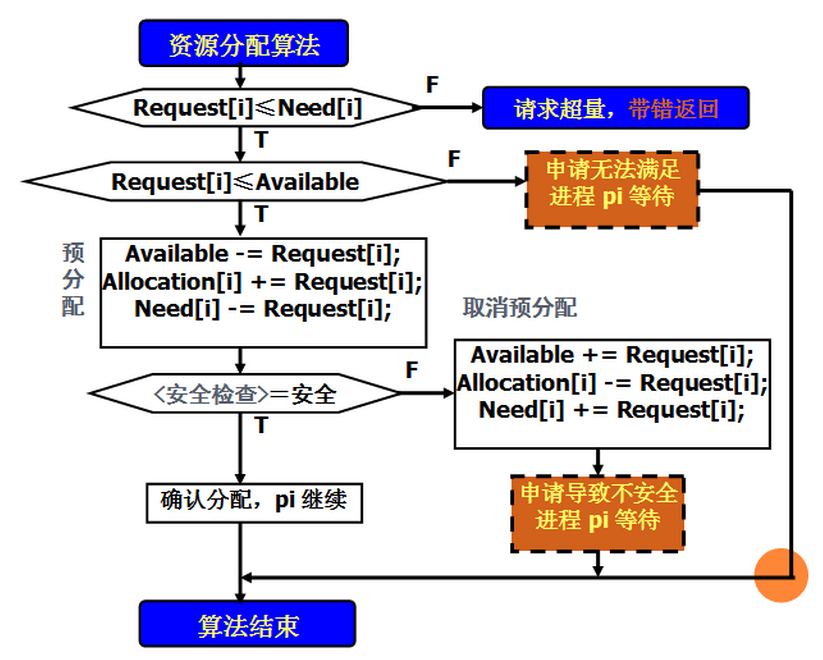
我们可以把操作系统看作是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。

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(试探性分配)
2. 顾客可以分期贷款，但贷款的总数不能超过最大需求量(可能一次并不能满足所需要的全部资源)
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(不存在死锁)
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(运行后释放)

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，若能存在安全状态，则按当前的申请量分配资源，否则也要推迟分配。

## 四．实验程序流程框图



## 五．实验程序

#include <stdio.h>

#define M 100

#define N 50 //全局变量定义

int Available[M]; //可利用资源数组

int Max[N][M]; //最大需求矩阵

int Allocation[N][M]; //分配矩阵

int Need[N][M]; //需求矩阵

int Request[N][M]; //M个进程还需要N类资源的资源量

int Finish[N];

int p[N];

int m,n; //M个进程,N类资源

//安全性算法

int Safe()

{

int i,j,k,l=0;

int Work[M]; //可利用资源数组

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

Work[i]=Available[i];

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

Finish[i]=0;

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

{

if (Finish[i]==1)

continue;

else

{

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

{

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

break;

}

if (j==n)

{

Finish[i]=1;

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

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

p[l++]=i;

i=-1;

}

else continue;

}

if (l==m)

{

printf("系统是安全的\n");

printf("系统安全序列是:\n");

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

{

printf("%d",p[i]);

if (i!=l-1)

printf("-->");

}

printf("\n");

return 1;

}

}

return 0;

}

//银行家算法

int main()

{

int i,j,mi;

printf("请输入进程的数目:\n");

scanf("%d",&m);

printf("请输入资源的种类:\n");

scanf("%d",&n);

printf("输入每个进程对资源的最大需求量,按照%d\*%d矩阵输入\n",m,n);

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

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

scanf("%d",&Max[i][j]);

printf("输入每个进程资源的目前占有量,按照%d\*%d矩阵输入\n",m,n);

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

{

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

{

scanf("%d",&Allocation[i][j]);

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

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

{

printf("你输入的第%d个进程所拥有的第%d个资源错误请重新输入:\n",i+1,j+1);

j--;

continue;

}

}

}

printf("请输入系统剩余资源量:\n");

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

scanf("%d",&Available[i]);

Safe();

while (1)

{

printf("输入要申请的资源的进程号（第一个进程号为0、第二个进程号为1，依此类推）\n"); scanf("%d",&mi);

printf("输入进程所请求的各个资源的数量\n");

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

{

scanf("%d",& Request[mi][i]);

｝

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

{

if (Request[mi][i]>Need[mi][i])

{

printf("所请求资源数超过进程的需求量!\n");

return 0;

}

if (Request[mi][i]>Available[i])

{

printf("所请求资源数超过系统所有的资源数!\n");

return 0;

}

}

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

{

Available[i]-=Request[mi][i];

Allocation[mi][i]+=Request[mi][i];

Need[mi][i]-=Request[mi][i];

}

if (Safe())

printf("同意分配请求\n");

else

{

printf("SORRY你的请求被拒绝\n");

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

{

Available[i]+=Request[mi][i];

Allocation[mi][i]-=Request[mi][i];

Need[mi][i]+=Request[mi][i];

}

}

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

Finish[i]=0;

char Flag; //标志位

printf("是否再次请求分配是请按Y/y否请按N/n");

while (1)

{

scanf("%c",&Flag);

if (Flag=='Y'||Flag=='y'||Flag=='N'||Flag=='n')

break;

else

{

printf("请按要求重新输入:\n");

continue;

}

}

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

continue;

else break;

}

return 0;

}

## 六．运行结果与结论

请把程序代码读懂并按题目中的要求运行，截图运行结果，并提交实验报告。