#### 实验二 死锁避免的模拟

## 实验目的

1）深入了解死锁的原因和必要条件。

2）掌握死锁的处理方式。

3）实现死锁避免的模拟

## 实验预备知识

1）系统安全状态

2）银行家算法

## 实验内容

验证银行家算法避免系统死锁。

假设系统中有三类资源A、B、C和五个进程P1、P2、P3、P4、P5。然后设定每种资源的数量。之后设定每个进程对各类资源的最大需求，最后假定在某一时刻，系统已经给各个进程分配多少资源。要求检查该时刻系统是否处于安全状态。

## 实验要求

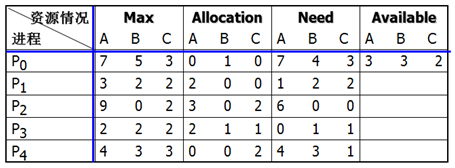
1）  给出程序流程和源程序（附有详细注释）

2）  程序运行截图

3）  收获、体会及对该实验的改进意见和见解

## 实验提示

由于是实验，没有真正的避免死锁。所以在实验中首先假定一种系统状态，假设T0时刻资源的分配情况如下所示：



要求实现：

（1）确定系统在T0时刻的安全性。

（2）P1发出资源请求向量Request1(1,0,2)，按照银行家算法确定能否将资源分配给P1。

（3）在（2）的基础上， P4发出请求向量Request4(3,3,0)，按照银行家算法确定能否将资源分配给P1。

（4）再（3）的基础上，P0发出请求向量Request0(0,2,0)，按照银行家算法确定能否将资源分配给P1。

[收起](javascript:void(0))

截止 2016-05-17

#include<stdio.h>  
  
typedef struct Node  
{  
int no;  
int flag;  
}Node;  
  
Node p[5]={{1,0},{2,0},{3,0},{4,0},{5,0}};  
int max[5][3]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};  
int alloc[5][3]={{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};  
int need[5][3]={0};  
int avi[3]={3,3,2};  
int sub(int \*p,int \*q,int len)  
{  
int i = 0;  
while(i<len)  
{  
if(p[i]-q[i]<0)  
return 0;  
i++;  
}  
return 1;  
}  
  
void fneed()  
{  
int i = 0,j= 0;  
while (i<5)  
{  
j = 0;  
while(j<3)  
{  
need[i][j]=max[i][j]-alloc[i][j];  
j++;  
}  
i++;  
}  
}  
  
int check()  
{  
int work[3];  
int i = 0,t;  
int sum = 0;  
while(i<3)  
{  
work[i]=avi[i];  
i++;  
}  
for(t = 0;t<5;t++)  
{  
for(i=0;i<5;i++)  
{  
if(p[i].flag==0)  
if(sub(work,need[i],3))  
{  
int j = 0;  
while(j<3)  
{  
work[j]+=alloc[i][j];  
j++;  
}  
sum++;  
p[i].flag = 1;  
}   
}  
}   
if(5==sum)  
return 1;  
else   
return 0;  
}  
  
int main(void)  
{  
fneed();  
if(check())  
printf("安全\n");  
else  
printf("不安全\n");  
return 0;  
}