## 陈小龙 20140031 计算机2班

## 实验4 死锁避免—银行家算法的实现

### 一、实验目的

1、掌握死锁产生的原因和必要条件。

2、掌握银行家算法的实现

### 二、实验理论基础及教材对应关系

1、处理机调度与死锁。

2、死锁的产生与预防。

3、银行家算法。

### 三、实验内容与步骤

1、创建C语言工程项目，按照教材上的有关说明，定义相应的数据结构。

int AllocMatrix[5][4] // 已经分配资源矩阵

int RequestMatrix[5][4] // 需求矩阵

int AvailResource[4] // 可用资源向量

int TryProcess[5] // 尝试序列

2、给各个数据结构设定合适的初始值。

给上述数据结构设定初始值。

如：int AllocMatrix[5][4]={ // 已经分配资源矩阵

{0,0,3,2},

{1,0,0,0},

{1,3,5,4},

{0,3,3,2},

{0,0,1,4}

};

注意：步骤1、2可同时进行，即利用C语言中的定义变量就可同时初始化的方式进行数值初设。

3、依据银行家算法的描述依次进行资源的试探性分配，直至成功或失败，成功则说明当前状态是安全的；失败后，还应该将资源回到初始状态，并进行另一次试探；只有所有的试探都失败了，才能说明当前状态是不安全的。

通常，这种试探性算法采用递归的方法是很合适的，程序也是很简洁的。

代码片段：

if( AvailResource[0] >= RequestMatrix[k][0]

&& AvailResource[1] >= RequestMatrix[k][1]

&& AvailResource[2] >= RequestMatrix[k][2]

&& AvailResource[3] >= RequestMatrix[k][3])//若资源能使进程ProcessID结束

for(int i=0;i<4;i++)

AvailResource[i] -= AllocMatrix[k][i]; // 恢复资源

int SearchSecurity(int level) //递归函数

{

if(level == 4){

……

}

else{

SearchSecurity(level+1); // 有条件递归调用自己

}

……

} 其余由大家自行完成。

### 四、实验材料的提交与成绩评定

本实验的实验报告一份（电子版，格式参考学院统一实验报告）

#include<STRING.H>

#include<stdio.h>

#include<stdlib.h>

#include<CONIO.H>/\*用到了getch()\*/

#define M 5 /\*进程数\*/

#define N 3 /\*资源数\*/

#define FALSE 0

#define TRUE 1 /\*M个进程对N类资源最大资源需求量\*/

int MAX[M][N]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};/\*系统可用资源数\*/

int AVAILABLE[N]={10,5,7};/\*M个进程已分配到的N类数量\*/

int ALLOCATION[M][N]={{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0}};/\*M个进程已经得到N类资源的资源量\*/

int NEED[M][N]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};/\*M个进程还需要N类资源的资源量\*/

int Request[N]={0,0,0};

void main(){

int i=0,j=0;

char flag;

void showdata();

void changdata(int);

void rstordata(int);

int chkerr();

showdata();

enter:

{

printf("请输入需申请资源的进程号（从0到");

printf("%d",M-1);

printf("）:");

scanf("%d",&i);

}

if(i<0||i>=M)

{

printf("输入的进程号不存在，重新输入!\n");

goto enter;

}

err:

{

printf("请输入进程");

printf("%d",i);

printf("申请的资源数\n");

printf("类别:ABC\n");

printf("");

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

{

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

if(Request[j]>NEED[i][j])

{

printf("%d",i);

printf("号进程");

printf("申请的资源数>进程");

printf("%d",i);

printf("还需要");

printf("%d",j);

printf("类资源的资源量!申请不合理，出错!请重新选择!\n");

goto err;

}

else

{

if(Request[j]>AVAILABLE[j])

{

printf("进程");

printf("%d",i);

printf("申请的资源数大于系统可用");

printf("%d",j);

printf("类资源的资源量!申请不合理，出错!请重新选择!\n");

goto err;

}

}

}

}

changdata(i);

if(chkerr())

{

rstordata(i);

showdata();

}else

showdata();

printf("\n");

printf("按'y'或'Y'键继续,否则退出\n");

flag=getch();

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

{

goto enter;

}else

{

exit(0);

}

}/\*显示数组\*/

void showdata()

{

int i,j;

printf("系统可用资源向量:\n");

printf("\*\*\*Available\*\*\*\n");

printf("资源类别:ABC\n");

printf("资源数目:");

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

{

printf("%d",AVAILABLE[j]);

}

printf("\n");

printf("\n");

printf("各进程还需要的资源量:\n");

printf("\*\*\*\*\*\*Need\*\*\*\*\*\*\n");

printf("资源类别:ABC\n");

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

{

printf("");

printf("%d",i);

printf("号进程:");

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

{

printf("%d",NEED[i][j]);

}

printf("\n");

}

printf("\n");

printf("各进程已经得到的资源量:\n");

printf("\*\*\*Allocation\*\*\*\n");

printf("资源类别:ABC\n");

for(i=0;i<M;i++){printf("");

printf("%d",i);

printf("号进程:");/\*printf(":\n");\*/

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

{

printf("%d",ALLOCATION[i][j]);

}

printf("\n");

}

printf("\n");}/\*系统对进程请求响应，资源向量改变\*/

void changdata(int k){

int j;

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

{

AVAILABLE[j]=AVAILABLE[j]-Request[j];

ALLOCATION[k][j]=ALLOCATION[k][j]+Request[j];

NEED[k][j]=NEED[k][j]-Request[j];

}

}/\*资源向量改变\*/

void rstordata(int k)

{

int j;

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

{

AVAILABLE[j]=AVAILABLE[j]+Request[j];

ALLOCATION[k][j]=ALLOCATION[k][j]-Request[j];

NEED[k][j]=NEED[k][j]+Request[j];

}

}/\*安全性检查函数\*/

int chkerr()//在假定分配资源的情况下检查系统的安全性

{

int WORK[N],FINISH[M],temp[M];//temp[]用来记录进程安全执行的顺序

int i,j,m,k=0,count;

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

FINISH[i]=FALSE;

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

WORK[j]=AVAILABLE[j];//把可利用资源数赋给WORK[]

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

{

count=0;

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

if(FINISH[i]==FALSE && NEED[i][j]<=WORK[j])

count++;

if(count==N)//当进程各类资源都满足NEED<=WORK时

{

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

WORK[m]=WORK[m]+ALLOCATION[i][m];

FINISH[i]=TRUE;

temp[k]=i;//记录下满足条件的进程

k++;i=-1;

}

}

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

if(FINISH[i]==FALSE)

{

printf("系统不安全!!!本次资源申请不成功!!!\n");

return 1;

}

printf("\n");

printf("经安全性检查，系统安全，本次分配成功。\n");

printf("\n");

printf("本次安全序列：");

for(i=0;i<M;i++)//打印安全系统的进程调用顺序

{

printf("进程");

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

if(i<M-1)

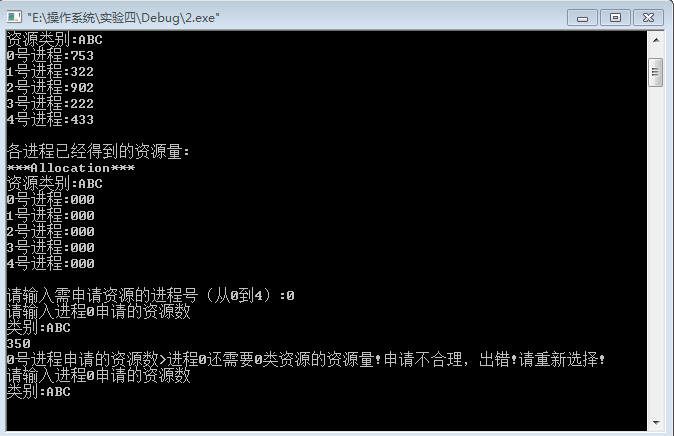
printf("->");

}

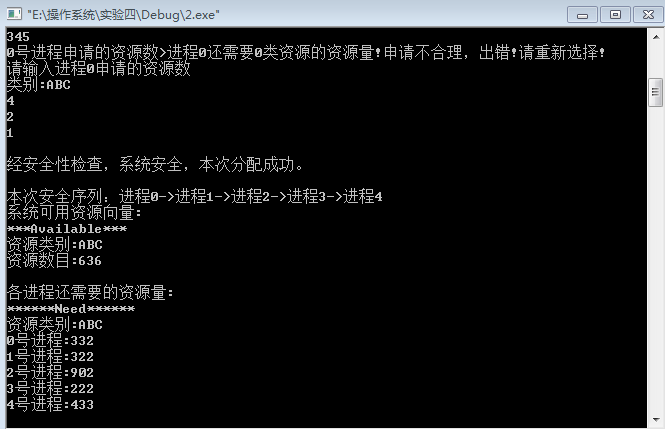
printf("\n");

return 0;

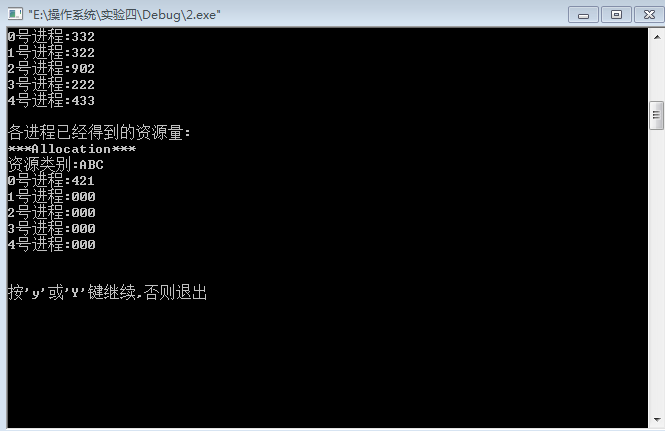
}



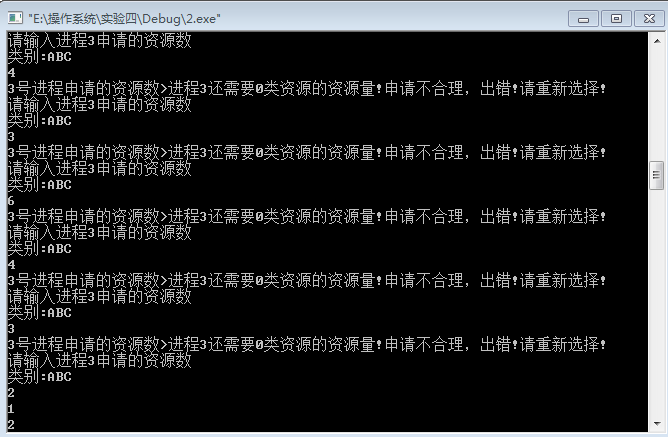
0号进程资源分配不成功



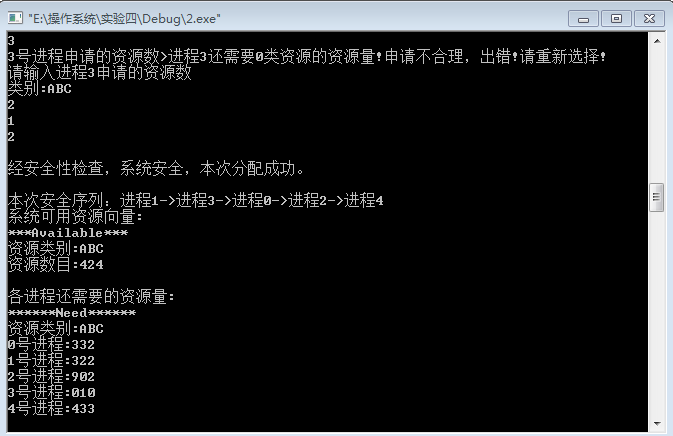
系统安全，0号进程资源分配成功



0号进程已有的资源数为421，还差332个



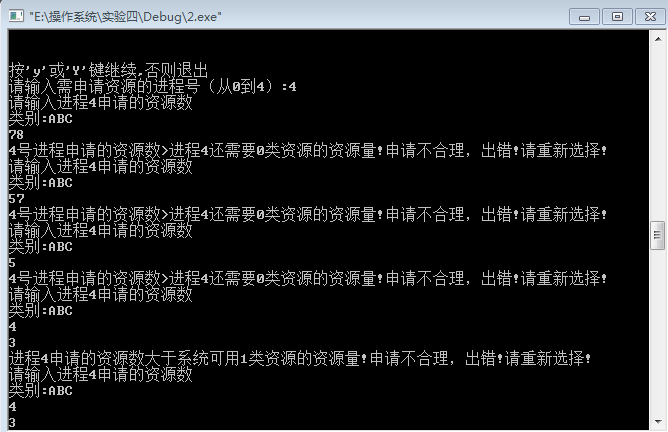
3号进程连输了几个资源分配都不成功



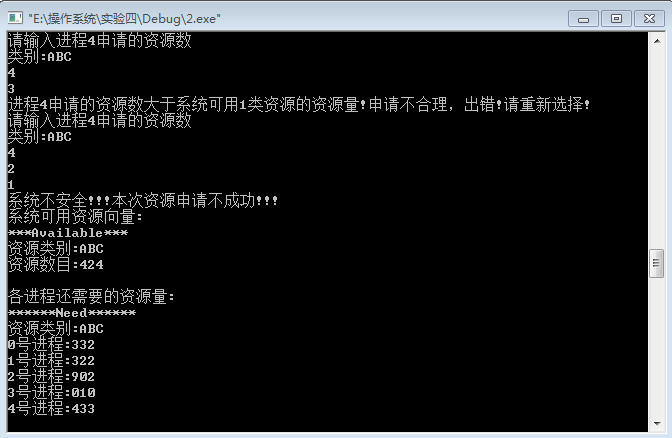
系统安全的，3号进程资源分配成功，已有资源数212，资源数还差10个，总资源数还有424个



0号进程和3号进程成功申请得到资源量



4号进程资源分配不成功，申请数量不合理



由于系统不安全，4号资源申请分配不成功，总的资源数还有424个



只有0号进程和3号进程资源申请分配成功，系统是安全的，其他几号资源分配不成功（由于申请数量不合理或者系统不安全）