

hzau-1

**实验2**

**死锁**

姓 名：陈立宇

学 号: 2017310210305

专业班级：信息1703班

课程名：操作系统实验

指导教师：任继平

完成日期：2019.6.15

**一、实验目的**

学生应独立的采用高级语言编写一个动态分配系统资源的程序，模拟死锁现象，观察死锁发生的条件，并采用适当的算法，有效地防止死锁的发生。学生应通过本次实验，更直观的了解死锁发生的原因，初步掌握防止死锁、解除死锁的简单方法，加深理解教材中有关死锁的内容。

**2. 要求**

1. 设计一个几个并发进程共享m个系统资源的系统。进程可动态的申请资源和释

放资源，系统按各进程的申请动态的分配资源。

1. 系统应能显示各进程申请和释放资源的过程，能显示系统动态分配资源的过程，便于观察和分析。
2. 系统应能选择是否采用防止死锁的算法，应设计多种防止死锁的算法，并能任意选择其中任何一种投入运行，同时要求，在不采用防止死锁算法时观察死锁现象发生的过程，在使用防止死锁的算法时，了解在同样条件下防止死锁发生的过程。
3. 本次实验的上机时间为2～4学时。
4. 所有程序均应采用C语言编写程序，特别需要的部分可采用汇编语言便程序。

**二、实验内容**

本次实验采用银行算法防止死锁的发生。设有3个并发进程共享10个系统资源。在3个进程申请系统资源之和不超过10时，当然不可能发生死锁，因为各个进程资源都能满足。在有一个进程申请的系统资源数超过10时，必然会发生死锁。应该排队这两种情况。程序采用人工输入各进程的申请资源序列。如果随机给各进程分配资源，就可能发生死锁，这也就是不采用防止死锁算法的情况。假如，按照一定的规则，为各进程分配资源，就可以防止死锁的发生。示例采用银行算法。这是一种犹如“瞎子爬山”的方法，即探索一步，前进一步，行不通，再往其它方向试探，直至爬上山顶。这种方法是比较保守的，所花的代价也不小。

**三、实验设计**

**任务分析：**

*银行算法*，顾名思义是来源于银行的借贷业务，一定数量的本金要应付各种客户的借贷周转，为了防止银行因资金无法周转而倒闭，对每一笔贷款，必须考察其最后是否能归还。研究死锁现象时就碰到类似的问题，有限资源为多个进程共享，分配不好就会发生每个进程都无法继续下去的死锁僵局。

银行算法的原理是先假定某一次分配成立，然后检查由于这次分配是否会引起死锁，即剩下的资源是不是能满足任一进程完成的需要。如这次分配是安全的（不会引起死锁），就实施本次分配，再作另一种分配试探，一直探索到各进程均满足各自的资源请求，防止了死锁的发生。

**概要设计：**

定义一个进程的数据结构和对应的两种进程数组

struct Process{

int Had[15];//已分配资源数

int Max[15];//最大资源分配数

int Need[15];//还需资源数

bool IsEnd;}Processes[15];//进程数组

Process tmpProcesses[15];//假设允许分配的进程数组

定义两个数组分别表示剩余资源数和当前进程申请的资源数

int Res[15];//剩余资源数

int Req[15];//申请的资源数

定义功能函数模板：

bool IsEnd()//判断所有进程是否结束

bool IstmpEnd()//判断所有假设允许分配的进程是否结束

void Print()//打印第i个进程占有第j种资源的数目和最多需要第j种资源的数目和每种资源所剩的数目

bool Banker()//银行家算法

主程序流程：

初始化各数据结构和数组->选择是否采用防死锁算法->采用防死锁算法，则调用银行家算法判断是否会安全，若安全则同意申请并打印，否则拒绝并报错；不采用防死锁算法，则一直同意申请并打印，直到申请大于剩余量或需求量而报错。

**程序框图：**

输出出错信息

找出一个未完成的

进程作为现行进程

现行进程

已完成所有

请求

收回该进程占用的资源，标志该进程结束，如果全部进程结束，置ADVANCE为‘flase’，否则’true’

找出一个未完成的进程作为现行进程

申请量

超过最大需

求量

死锁处理程序

进程已全部结束？

开始

输入选择标志

OPTION

输入各进程动态申请

资源数组AP

输入各进程最资源需求

量数组VPMAXCLAIM

最大资源

需求量超过系统

资源数

输出出错信息

关闭输出文件

结束

是

否

否 是

否

是

否

是

银行家算法流程1

剩余资源

能否满足现行进程？

请求

开始

当前分配

状态安全吗？

（safe）

承认已假定分配给现行进程的资源输出信息6，置ADVANCE=true

采用死锁预防算法？

收回已假定分配给现行进程的资源，输出信息4，置ADVANCE=true

承认已假定分配给现行进程的资源，输出信息5，置ADVANCE=true

结束

收回已假定分配给现行进程的资源输出信息3

如不采用死锁预防算法，将现行进程死锁位置1

采用死锁预防算法，如已有进程死锁位置ADVANCE=false

否

是

是

否

是

否

银行家算法流程2

置未完成进程的T标志为1

置已完成进程的T标志为0

PROGRESS是否为“ture”？

开始

置局部变量PROGRESS为“ture”

（即是否有进程标志位从1变0）

判断是否还存在不能问成的进程，若不存在，置SAFE 为“ture”，否则为“false”

结束

置PROGRESS为 “false”

全部进程已经完成？（即各进程T标志均为0）

现行进程所占资源与剩余资源之和是否超过该资源的最大申请量？

现行进程能够完成，把其所占资源加到剩余资源中去，置‘0’标志T（表示该进程能完成）置PROGRESS为 ‘true’

否

是

否

是

否

是

银行家算法流程3

银行家算法流程3

**源代码：**

#include <iostream>

#include <cstring>

using namespace std;

int Res[15];//剩余资源数

int n,m;//进程数和资源数

struct Process{

int Had[15];//已分配资源数

int Max[15];//最大资源分配数

int Need[15];//还需资源数

bool IsEnd;}Processes[15];//进程数组

int curProcess;//申请资源的进程

int Req[15];//申请的资源数

Process tmpProcesses[15];//假设允许分配的进程数组

bool IsEnd()//判断所有进程是否结束

{

bool R = true;

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

{

if(Processes[i].IsEnd == false)

{

R = false;

break;

}

}

return R;

}

bool IstmpEnd()//判断所有假设允许分配的进程是否结束

{

bool R = true;

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

{

if(tmpProcesses[i].IsEnd == false)

{

R = false;

break;

}

}

return R;

}

bool Legal= false;

bool Banker()//银行家算法

{

Legal = true;

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

{

if(Req[i] > Processes[curProcess].Need[i])

{

Legal = false;

break;

}

}

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

{

if(Req[i] > Res[i])

{

Legal = false;

break;

}

}

int tmp[15];

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

{

tmp[i] = Res[i];

}

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

{

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

{

tmpProcesses[i].Max[j] = Processes[i].Max[j];

tmpProcesses[i].Need[j] = Processes[i].Need[j];

tmpProcesses[i].Had[j] = Processes[i].Had[j];

tmpProcesses[i].IsEnd = Processes[i].IsEnd;

}

}

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

{

tmpProcesses[curProcess].Had[i] +=Req[i];

tmpProcesses[curProcess].Need[i] -= Req[i];

tmp[i]-=Req[i];

}

bool IsFind = false;

do{

IsFind = false;

int curRel = -1;

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

{

if(tmpProcesses[i].IsEnd == false)

{

bool IsFF = true;

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

{

if(tmpProcesses[i].Need[j] > tmp[i])

{

IsFF = false;

break;

}

}

IsFind = IsFF;

if(IsFind == true)

{

curRel = i;

break;

}

}

}

if(IsFind)

{

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

{

tmp[i]+=tmpProcesses[curRel].Had[i];

tmpProcesses[curRel].IsEnd = true;

tmpProcesses[curRel].Had[i] = 0;

tmpProcesses[curRel].Need[i] = 0;

}

}

}while(!IstmpEnd() && IsFind == true);

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

{

if(tmpProcesses[i].IsEnd == false)

{

Legal = false;

break;

}

}

return Legal;

}

void Print()//打印第i个进程占有第j种资源的数目和最多需要第j种资源的数目和每种资源所剩的数目

{

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

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

cout<<"第"<<i<<"个进程占有第"<<j<<"种资源的数目和所需的最大数目："<<Processes[i].Had[j]<<" "<<Processes[i].Max[j]<<endl;

}

cout<<endl;

}

cout<<"每种资源所剩数目:"<<endl;

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

cout<<Res[i]<<" ";

}

cout<<endl;

}

int main()

{

memset(Processes,0,sizeof(Processes));

memset(Res,0,sizeof(Res));

memset(Req,0,sizeof(Req));

cout<<"请输入进程总数和资源种数:"<<endl;

cin>>n>>m;

cout<<"请输入每种资源所剩数目:"<<endl;

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

cin>>Res[i];

}

cout<<"请输入第i个进程占有第j种资源的数目和最多需要第j种资源的数目:"<<endl;

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

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

cin>>Processes[i].Had[j]>>Processes[i].Max[j];

}

}

int option = -1;

cout<<"是否采用防止死锁算法（0为采用）:"<<endl;

cin>>option;

if(option == 0){

do{

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

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

Processes[i].Need[j] = Processes[i].Max[j] - Processes[i].Had[j];

}

}

cout<<"请输入需要申请资源的进程ID:"<<endl;

cin>>curProcess;

cout<<"请输入该进程所需要每种资源的数目:"<<endl;

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

cin>>Req[i];

}

Legal = false;

Banker();

if(Legal){cout<<"同意申请！"<<endl;

Processes[curProcess].IsEnd = true;

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

Processes[curProcess].Had[i]+=Req[i];

Processes[curProcess].Need[i]-=Req[i];

Res[i]-=Req[i];

if(Processes[curProcess].Had[i] < Processes[curProcess].Max[i]){Processes[curProcess].IsEnd = false;}

else{Res[i]+=Processes[curProcess].Had[i];

Processes[curProcess].Had[i]=0;

Processes[curProcess].Need[i]=0;

cout<<"进程"<<curProcess<<"已经完成所有需求！"<<endl;

}

}

Print();

}

else{cout<<"拒绝申请！"<<endl;}

}while(!IsEnd());

cout<<"所有进程都已完成所有需求！";

}

else{

while(1){

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

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

Processes[i].Need[j] = Processes[i].Max[j] - Processes[i].Had[j];

}

}

cout<<"请输入需要申请资源的进程ID:"<<endl;

cin>>curProcess;

cout<<"请输入该进程所需要每种资源的数目:"<<endl;

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

cin>>Req[i];

}

Legal=true;

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

{

if(Req[i] > Processes[curProcess].Need[i] || Req[i] > Res[i])

{

Legal=false;

break;

}

}

if(Legal)

{ cout<<"同意申请！"<<endl;

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

Processes[curProcess].Had[i]+=Req[i];

Processes[curProcess].Need[i]-=Req[i];

Res[i]-=Req[i];

if(Processes[curProcess].Had[i] == Processes[curProcess].Max[i])

{

Processes[curProcess].Had[i]=0;

Processes[curProcess].Need[i]=0;

Res[i]+=Processes[curProcess].Had[i];

cout<<"进程"<<curProcess<<"已经完成所有需求"<<endl;

}

}

}

else{cout<<"出错,拒绝申请";return 0;}

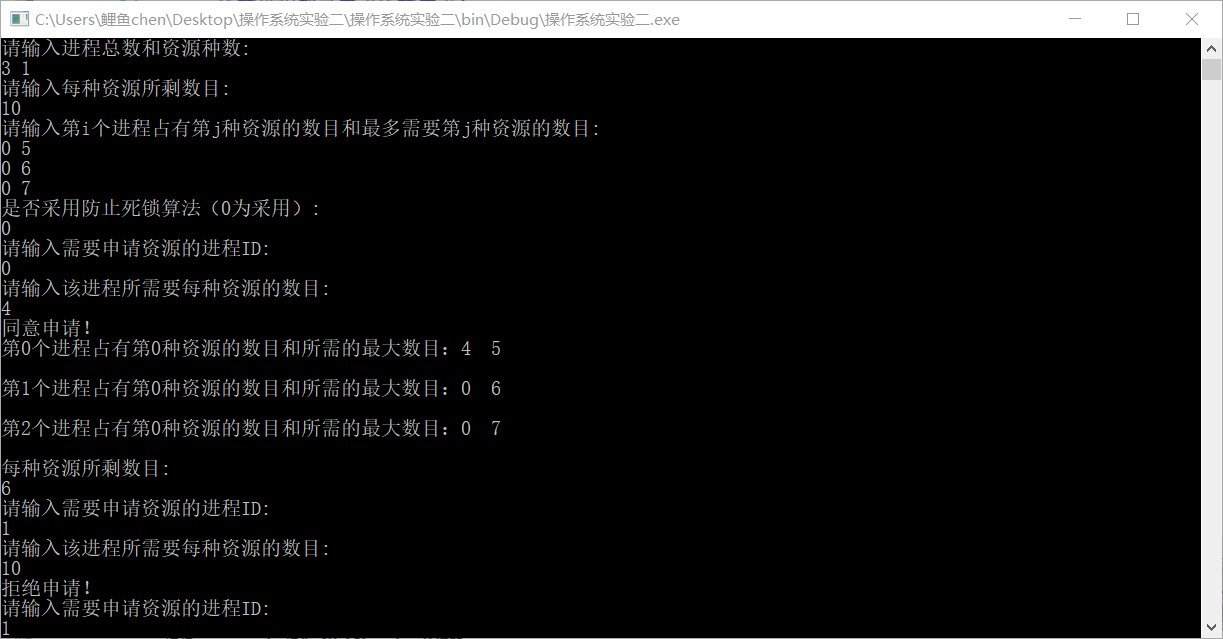
}

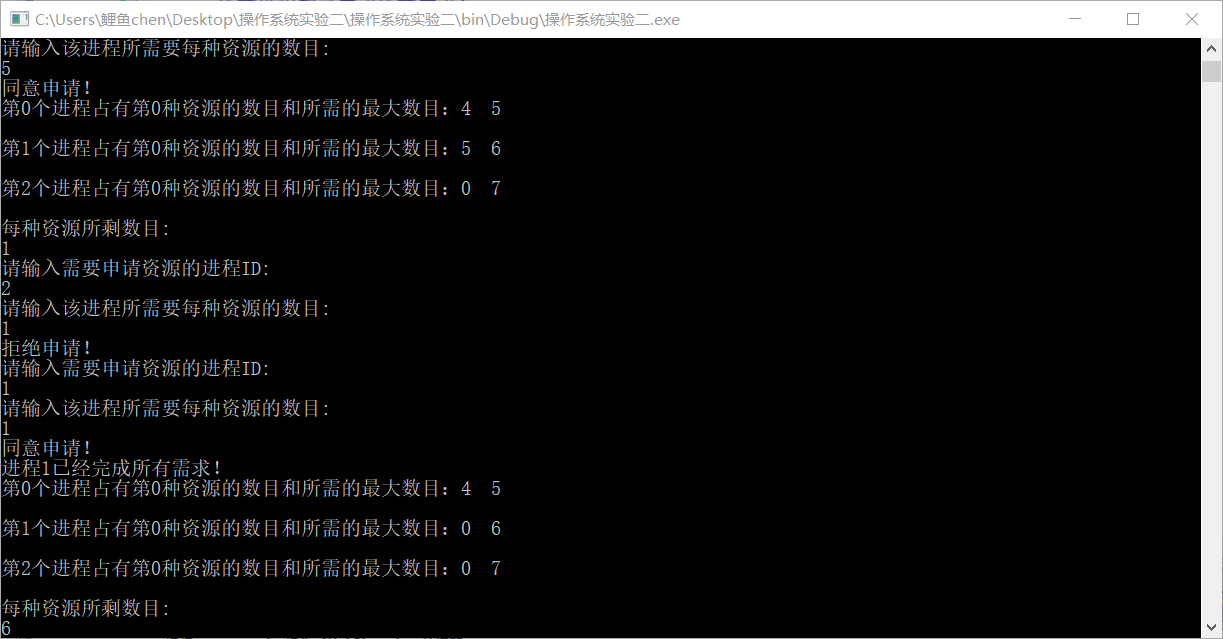
}

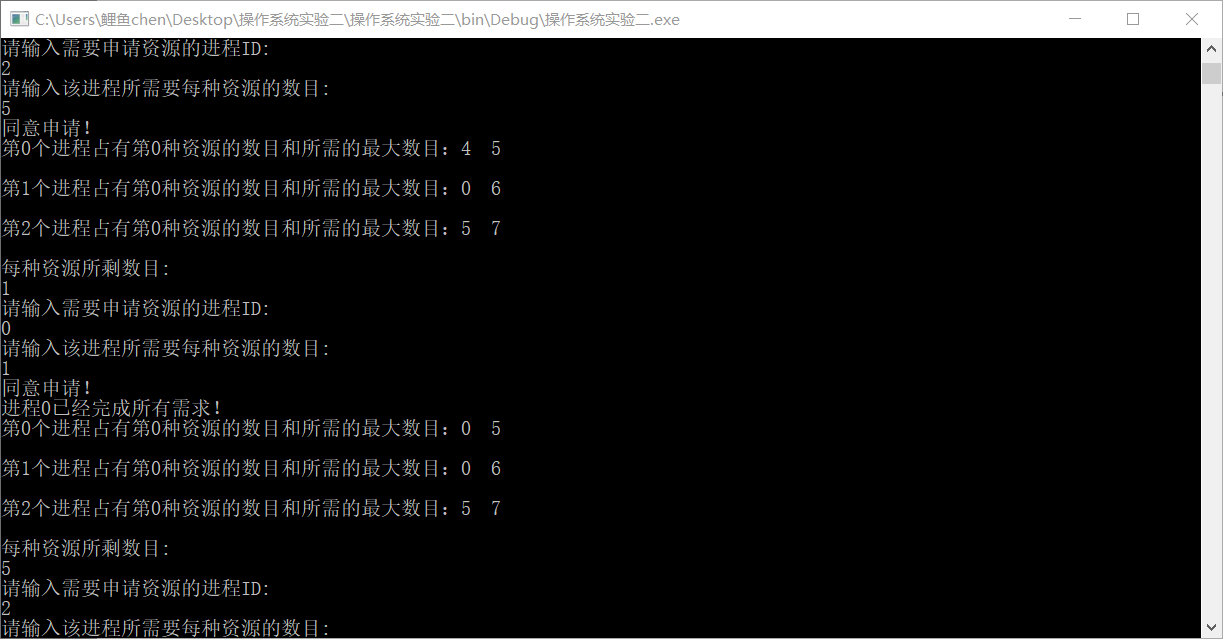
return 0;

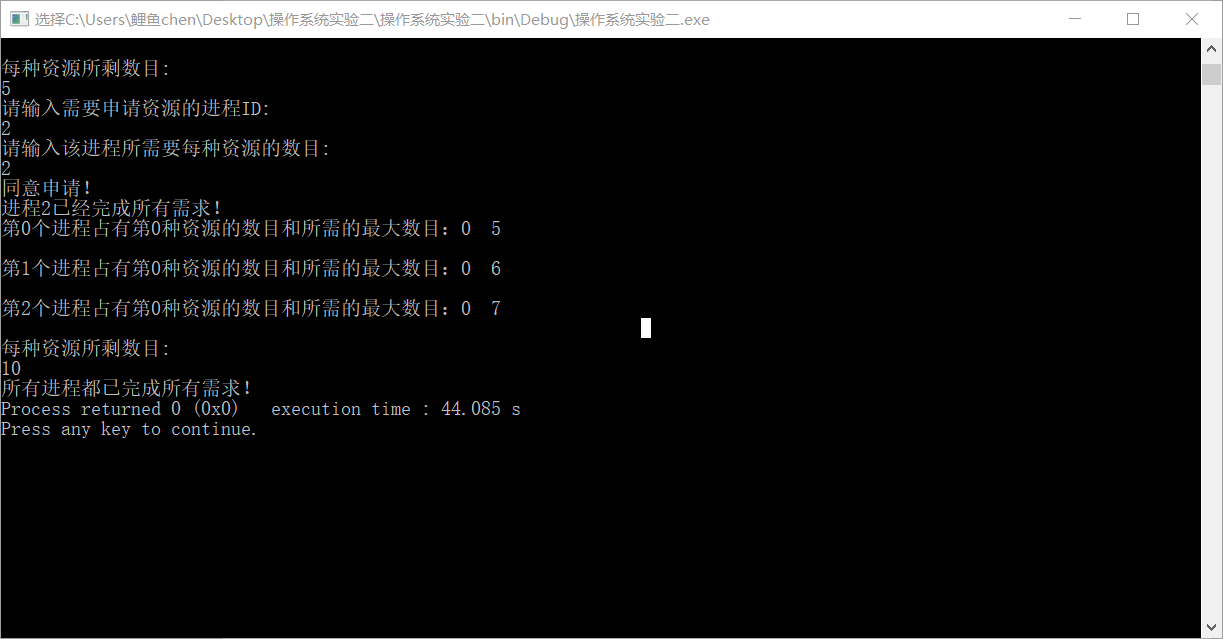
}

**输入输出示例：**









**四、实验心得**

多个进程同时运行时，系统根据各类系统资源的最大需求数和各类资源的剩余数为进程安排安全序列，使得系统能快速且安全地运行进程，不至于发生死锁。银行家算法是避免死锁的主要方法，其思路在很多方面都非常值得我们去学习借鉴。