

操作系统

实 验 四 报 告

学院： 计算机科学学院

专业： 网络工程1701

姓名： 胡玉琛41709040102

姓名： 何一鸣41709040107

## 1.实验名称

银行家算法

## 2.实验目的

掌握银行家算法，了解银行家算法的具体实现过程。银行家算法是避免死锁的一种重要方法，加深了解有关资源申请、避免死锁等概念，并体会和了解死锁和避免死锁的具体实施方法。

## 3.实验内容

(1) 输入的形式和输入值的范围；

#define MaxNum 20

static int n;//行，进程数

static int m;//列，资源数

static char ID[MaxNum];//进程程名

static int Max[MaxNum][MaxNum];//进程所需所有资源

static int Allocation[MaxNum][MaxNum];//进程已分配资源

static int Need[MaxNum][MaxNum];//进程还需要资源

static int Available[MaxNum];////系统可分配资源

static int Request[MaxNum];//指定执行进程所需资源

static int SafeOrder[MaxNum];//安全执行顺序

(2) 输出的形式；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| precessName | Max | Allocation | Need | Available |
|  | A B C | A B C | A B C | A B C |
| P0 |  | 0 1 0 | 7 4 3 | 3 3 2 |

(3) 程序所能达到的功能；

输入进程已分配资源（Allocation）、仍需资源(Need)、系统可分配资源（Available），计算出进程所需最大资源（Max），当前状态是否安全，若安全输出安全序列，可指定优先分配给某进程资源并执行，判断输入是否符合要求，若如何要求则判断执行后状态是否安全，若安全则输出安全序列。

(4) 测试数据，包括正确的输入及其输出结果和含有错误的输入及其输出结果。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Max | Allocation | Need | Available |
|  | A B C | A B C | A B C | A B C |
| P0 | 7 5 3 | 0 1 0 | 7 4 3 | 3 3 2 |
| P1 | 3 2 2 | 2 0 0 | 1 2 2 |  |
| P2 | 9 0 2 | 3 0 2 | 6 0 0 |  |
| P3 | 2 2 2 | 2 1 1 | 0 1 1 |  |
| P4 | 4 3 3 | 0 0 2 | 4 3 1 |  |

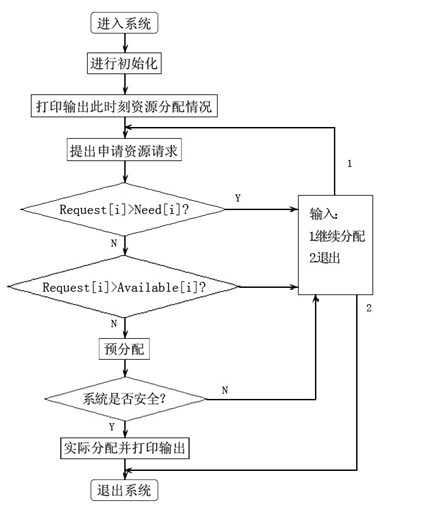
安全序列：P1->P3->P0->P2->P4

执行P1，分配给P1资源1 0 2后（蓝色标出的是变化后的值）

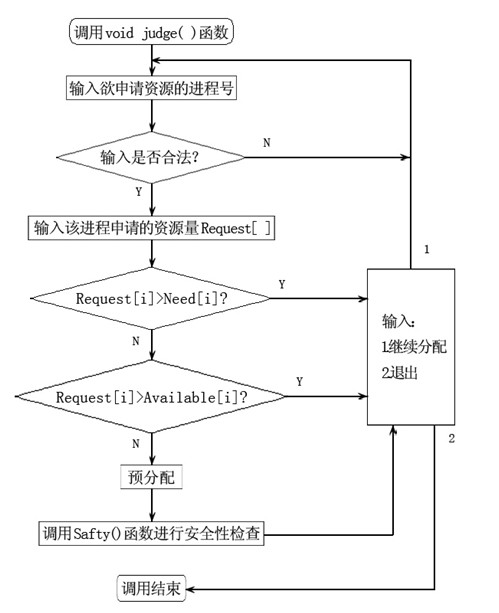
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Max | Allocation | Need | Available |
|  | A B C | A B C | A B C | A B C |
| P0 | 7 5 3 | 0 1 0 | 7 4 3 | 2 3 0 |
| P1 | 3 2 2 | 3 0 2 | 0 2 0 |  |
| P2 | 9 0 2 | 3 0 2 | 6 0 0 |  |
| P3 | 2 2 2 | 2 1 1 | 0 1 1 |  |
| P4 | 4 3 3 | 0 0 2 | 4 3 1 |  |

## 4.流程图

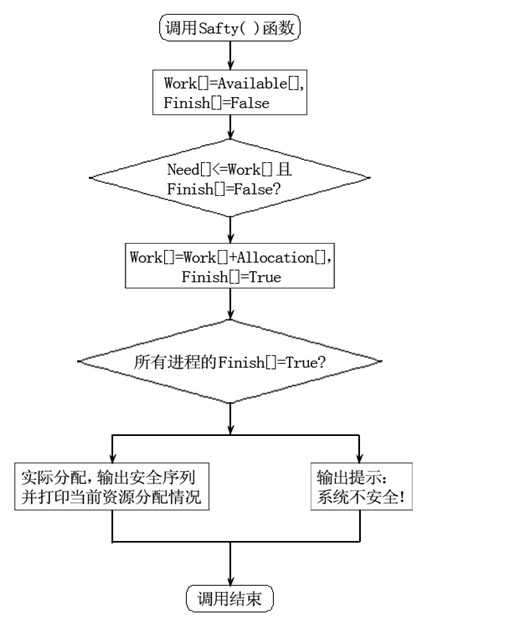
（1）系统主要过程流程图



（2）银行家算法流程图



（3）安全性算法流程图



## 5.实验过程

**1、算法思路：**先对用户提出的请求进行合法性检查，即检查请求是否大于需要的，是否大于可利用的。若请求合法，则进行预分配，对分配后的状态调用安全性算法进行检查。若安全，则分配；若不安全，则拒绝申请，恢复到原来的状态，拒绝申请。

实现程序模块的具体算法。

//初始化

void Enter(){

int i,j;

cout<<"Enter n: ";

cin>>n;

cout<<"Enter m: ";

cin>>m;

//初始化进程名

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

ID[i]=i+65;

}

cout<<"Enter Available: ";

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

cin>>Available[i];

}

cout<<"Enter Allocation: "<<endl;

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

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

cin>>Allocation[i][j];

}

}

cout<<"Enter Need: "<<endl;

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

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

cin>>Need[i][j];

}

}

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

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

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

}

}

display();

}

//判断当前状态是否安全

bool Safe(){

int work[MaxNum];

bool Finish[MaxNum];//标记是否安全

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

work[i] = Available[i];

}

//初始化安全序列

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

Finish[i] = false;

SafeOrder[i] = -1;

}

int FinishNumebr = 0;

int isSafe;

int i=0,j;

while(i<n){

isSafe = 0;

//判断进程对应的所有资源的数量是否都满足没有完成、所需资源小于可提供资源

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

if (Finish[i]==false && Need[i][j]<=work[j])

isSafe++;

else

break;

}

//当且仅当进程对应的所有资源的数量都满足的时候才将该资源执行

if (isSafe == m){

Finish[i] = true;//标记该资源已执行

SafeOrder[FinishNumebr] = i;//记录已执行的安全的资源的编号

FinishNumebr++;

//安全的资源执行，释放其所需资源

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

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

}

i=0; //找到满足条件的进程并执行后，再从头开始再寻找满足条件的资源

}

else

i++; //如果没有满足条件的资源则查看下一个资源

//资源全部安全执行则输出执行顺序

if (FinishNumebr==n){

cout<<"Before allocating resources is Safe"<<endl;;

cout<<"Security sequence: P"<<SafeOrder[0];

for (i=1;i<n;i++){

cout<<" -> P"<<SafeOrder[i] ;

}

cout<<endl;

return true;

}

}

//若没有执行完所有程序则是不安全的

cout<<"Before allocating resources is unSafe"<<endl;

return false;

}

//执行银行家算法，检测执行某一进程后是否安全

void banker(){

int i,j;

int chooseP;

char isContinue;

while(true){

bool RequestNeed = false;

bool RequestAvailable = false;

cout<<"Enter chooseP: ";

cin>>chooseP;

cout<<"Enter process number: ";

while(RequestNeed==false){

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

cin>>Request[i];

}

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

//判断Request是否大于Need

if (Request[i]>Need[chooseP][i]){

cout<<"Request is more over Need, please enter again"<<endl;

RequestNeed = false;

continue;

}

else

RequestNeed = true;

//判断Request是否大于Available

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

cout<<"Request is more over Available, please enter again"<<endl;

RequestAvailable = false;

continue;

}

else

RequestNeed = true;

}

}

//将要调度的资源执行

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

Available[j] -=Request[j];

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

Need[chooseP][j] -= Request[j];

}

//如果不安全，则恢复到原来的值并警告

if (!Safe()) {

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

Available[j] +=Request[j];

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

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

}

//当输入无误却不安全是，证明系统已不能再分配资源进入死锁状态

if (RequestAvailable&&RequestNeed){

cout<<"Unsafe, Avaiable can't satisfy any process's Need"<<endl;

}

}

cout<<"After allocating resources to P"<<chooseP<<": "<<endl;

display();

cout<<"Whether to continue request test (Y/N): ";

cin>>isContinue;

if (isContinue=='Y'||isContinue=='y')//input();//实现多次请求（保留上一次请求的状态）

continue;

else if (isContinue=='N'||isContinue=='n')

break;

}

}

//显示结果

void display(){

int i,j;

int processName[MaxNum];

for (i=0;i<MaxNum;i++){

processName[i]=i;

}

cout<<endl;

cout<<"processName"<<"\t";

cout<<"Max"<<"\t\t";

cout<<"Allocation"<<"\t";

cout<<"Need"<<"\t\t";

cout<<"Available"<<endl;

cout<<"\t\t";

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

cout<<ID[i]<<" ";

}

cout<<"\t";

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

cout<<ID[i]<<" ";

}

cout<<"\t";

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

cout<<ID[i]<<" ";

}

cout<<"\t";

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

cout<<ID[i]<<" ";

}

cout<<endl;

bool FirstLine = true; //Avaliable数组只输出第一行

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

cout<<"P"<<processName[i]<<"\t\t";

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

cout<<Max[i][j]<<" ";

}

cout<<"\t";

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

cout<<Allocation[i][j]<<" ";

}

cout<<"\t";

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

cout<<Need[i][j]<<" ";

}

cout<<"\t";

if (FirstLine){

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

cout<<Available[j]<<" ";

}

FirstLine = false;

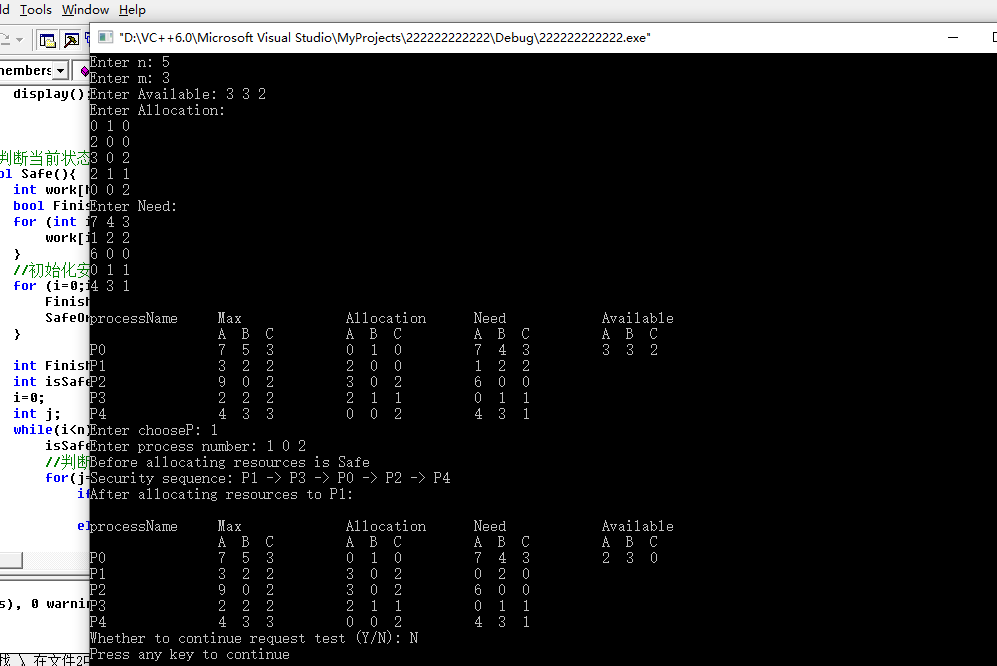
}

cout<<endl;

}

}

## 6.实验结果



## 7.实验总结

使用简单的数组、for循环、if判断实现，代码难度不大，只是需要十分细心，实现过程中多次遇到效果和预期不一样的情况，多次修改查找才解决了问题。

多个进程同时运行时，系统根据各类系统资源的最大需求和各类系统的剩余资源为进程安排序列，使得系统能快速且安全的进行进程，不至于发生死锁。银行家算法是避免死锁的主要算法。

## 附录

#include <iostream>

#include <iomanip>

#include <fstream>

using namespace std;

#define MaxNum 20

static int n;//行，进程数

static int m;//列，资源数

static char ID[MaxNum];//进程程名

static int Max[MaxNum][MaxNum];//进程所需所有资源

static int Allocation[MaxNum][MaxNum];//进程已分配资源

static int Need[MaxNum][MaxNum];//进程还需要资源

static int Available[MaxNum];////系统可分配资源

static int Request[MaxNum];//指定执行进程所需资源

static int SafeOrder[MaxNum];//安全执行顺序

void Enter();//初始化，输入进程已分配资源、仍需资源、系统可分配资源，计算出进程所需最大资源

bool Safe();//判断当前是否安全

void banker();//执行银行家算法，检测执行某一进程后是否安全

void display();//显示结果

int main(){

Enter();

banker();

return 0;

}

//初始化，输入进程已分配资源、仍需资源、系统可分配资源，计算出进程所需最大资源

void Enter(){

int i,j;

cout<<"Enter n: ";

cin>>n;

cout<<"Enter m: ";

cin>>m;

//初始化进程名

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

ID[i]=i+65;

}

cout<<"Enter Available: ";

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

cin>>Available[i];

}

cout<<"Enter Allocation: "<<endl;

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

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

cin>>Allocation[i][j];

}

}

cout<<"Enter Need: "<<endl;

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

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

cin>>Need[i][j];

}

}

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

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

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

}

}

display();

}

//判断当前状态是否安全

bool Safe(){

int work[MaxNum];

bool Finish[MaxNum];//标记是否安全

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

work[i] = Available[i];

}

//初始化安全序列

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

Finish[i] = false;

SafeOrder[i] = -1;

}

int FinishNumebr = 0;

int isSafe;

int i=0,j;

while(i<n){

isSafe = 0;

//判断进程对应的所有资源的数量是否都满足没有完成、所需资源小于可提供资源

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

if (Finish[i]==false && Need[i][j]<=work[j])

isSafe++;

else

break;

}

//当且仅当进程对应的所有资源的数量都满足的时候才将该资源执行

if (isSafe == m){

Finish[i] = true;//标记该资源已执行

SafeOrder[FinishNumebr] = i;//记录已执行的安全的资源的编号

FinishNumebr++;

//安全的资源执行，释放其所需资源

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

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

}

i=0; //找到满足条件的进程并执行后，再从头开始再寻找满足条件的资源

}

else

i++; //如果没有满足条件的资源则查看下一个资源

//资源全部安全执行则输出执行顺序

if (FinishNumebr==n){

cout<<"Before allocating resources is Safe"<<endl;;

cout<<"Security sequence: P"<<SafeOrder[0];

for (i=1;i<n;i++){

cout<<" -> P"<<SafeOrder[i] ;

}

cout<<endl;

return true;

}

}

//若没有执行完所有程序则是不安全的

cout<<"Before allocating resources is unSafe"<<endl;

return false;

}

//执行银行家算法，检测执行某一进程后是否安全

void banker(){

int i,j;

int chooseP;

char isContinue;

while(true){

bool RequestNeed = false;

bool RequestAvailable = false;

cout<<"Enter chooseP: ";

cin>>chooseP;

cout<<"Enter process number: ";

while(RequestNeed==false){

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

cin>>Request[i];

}

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

//判断Request是否大于Need

if (Request[i]>Need[chooseP][i]){

cout<<"Request is more over Need, please enter again"<<endl;

RequestNeed = false;

continue;

}

else

RequestNeed = true;

//判断Request是否大于Available

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

cout<<"Request is more over Available, please enter again"<<endl;

RequestAvailable = false;

continue;

}

else

RequestNeed = true;

}

}

//将要调度的资源执行

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

Available[j] -=Request[j];

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

Need[chooseP][j] -= Request[j];

}

//如果不安全，则恢复到原来的值并警告

if (!Safe()) {

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

Available[j] +=Request[j];

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

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

}

//当输入无误却不安全是，证明系统已不能再分配资源进入死锁状态

if (RequestAvailable&&RequestNeed){

cout<<"Unsafe, Avaiable can't satisfy any process's Need"<<endl;

}

}

cout<<"After allocating resources to P"<<chooseP<<": "<<endl;

display();

cout<<"Whether to continue request test (Y/N): ";

cin>>isContinue;

if (isContinue=='Y'||isContinue=='y')//input();//实现多次请求（保留上一次请求的状态）

continue;

else if (isContinue=='N'||isContinue=='n')

break;

}

}

//显示结果

void display(){

int i,j;

int processName[MaxNum];

for (i=0;i<MaxNum;i++){

processName[i]=i;

}

cout<<endl;

cout<<"processName"<<"\t";

cout<<"Max"<<"\t\t";

cout<<"Allocation"<<"\t";

cout<<"Need"<<"\t\t";

cout<<"Available"<<endl;

cout<<"\t\t";

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

cout<<ID[i]<<" ";

}

cout<<"\t";

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

cout<<ID[i]<<" ";

}

cout<<"\t";

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

cout<<ID[i]<<" ";

}

cout<<"\t";

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

cout<<ID[i]<<" ";

}

cout<<endl;

bool FirstLine = true; //Avaliable数组只输出第一行

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

cout<<"P"<<processName[i]<<"\t\t";

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

cout<<Max[i][j]<<" ";

}

cout<<"\t";

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

cout<<Allocation[i][j]<<" ";

}

cout<<"\t";

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

cout<<Need[i][j]<<" ";

}

cout<<"\t";

if (FirstLine){

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

cout<<Available[j]<<" ";

}

FirstLine = false;

}

cout<<endl;

}

}