银行家算法的仿真

**一、实验目的**

理解死锁的概念和消除方法；

**二、实验设备**

硬件设备：PC机；

软件：Windows操作系统；Vc++高级语言开发环境。

**三、实验内容**

编程实现银行家算法进行资源分配以避免死锁的发生。

**四、实验步骤**

1．复习死锁的概念和发生的必要条件。

2．写出利用银行家算法进行资源分配的伪码。（实验思路）

设Requesti是进程Pi的请求向量，如果Requesti［j］=K，表示进程Pi需要K个第j类型的资源。当Pi发出资源请求后，系统按下述步骤进行检查：

(1) 如果对所有j, Requesti［j］≤Need［i,j］，便转向步骤2；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2) 如果对所有j, Requesti［j］≤Available［j］，便转向步骤(3)；否则， 表示尚无足够资源，Pi须等待。

(3) 系统试探着把资源分配给进程Pi，并修改下面数据结构中的数值：

Available［j］∶=Available［j］- Requesti［j］

Allocation［i,j］∶=Allocation［i,j］+Requesti［j］;

Need［i,j］ ∶= Need［i,j］- Requesti［j］

1. 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程Pi，以完成本次分配；否则， 将本次的试探分配作废，恢复原来的资源分配状态，让进程Pi等待
2. 安全性算法 

(1) 设置两个向量

① 工作向量Work: 它表示系统可提供给进程继续运行所需的各类资源数目，它含有m个元素，在执行安全算法开始时，Work∶=Available;

② Finish: 它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做Finish［i］∶=false; 当有足够资源分配给进程时， 再令Finish［i］∶=true。

（2）从进程集合中找到一个能满足下述条件的进程：

① Finish［i］=false;

② 如果对所有j，如果 Need［i,j］≤Work［j］， 执行步骤(3)， 否则，执行步骤(4)

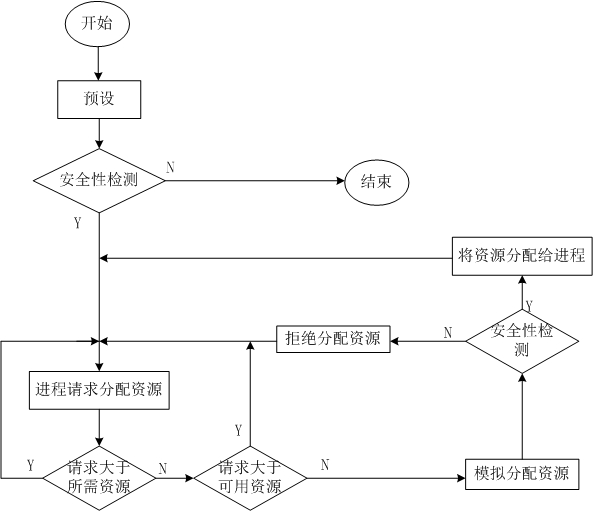
(3) 当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：对所有j Work［j］∶=Work［j］+Allocation［i,j］; Finish［i］∶=true; go to step 2;

(4) 如果所有进程的Finish［i］=true都满足， 则表示系统处于安全状态，否则处于不安全状态。

3．编程实现银行家算法过程。

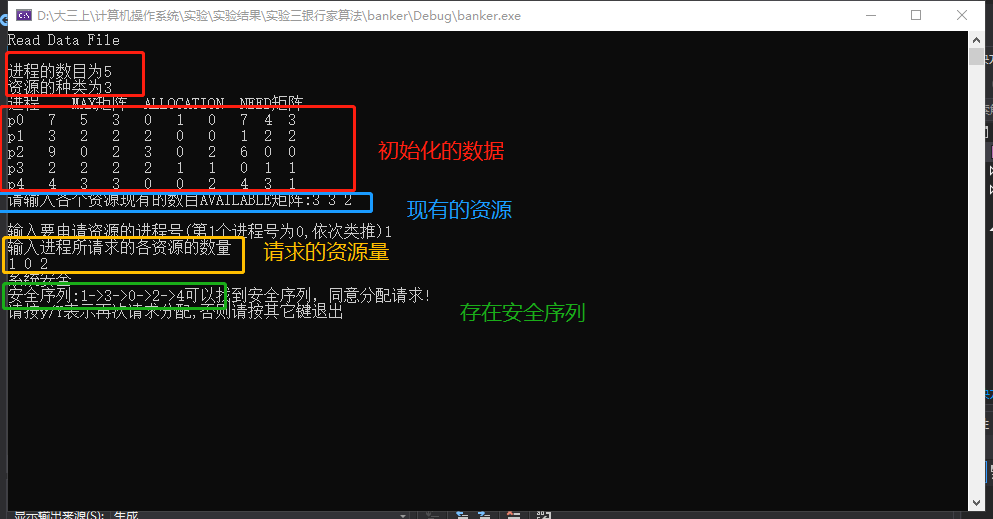
建立相应的数据结构（系统中每类资源的总量，进程对各类资源的最大需求量矩阵，资源已分配情况矩阵）从文件中读入MAX和ALLOCATION矩阵，并根据这两个矩阵计算出NEED矩阵，完成数据的初始化，随机输入用户的资源需求申请，调用安全性算法得出此次资源申请是否安全的结论，若安全，输出安全序列，否则，不予分配。

实验流程：

****

1. **实验结果**

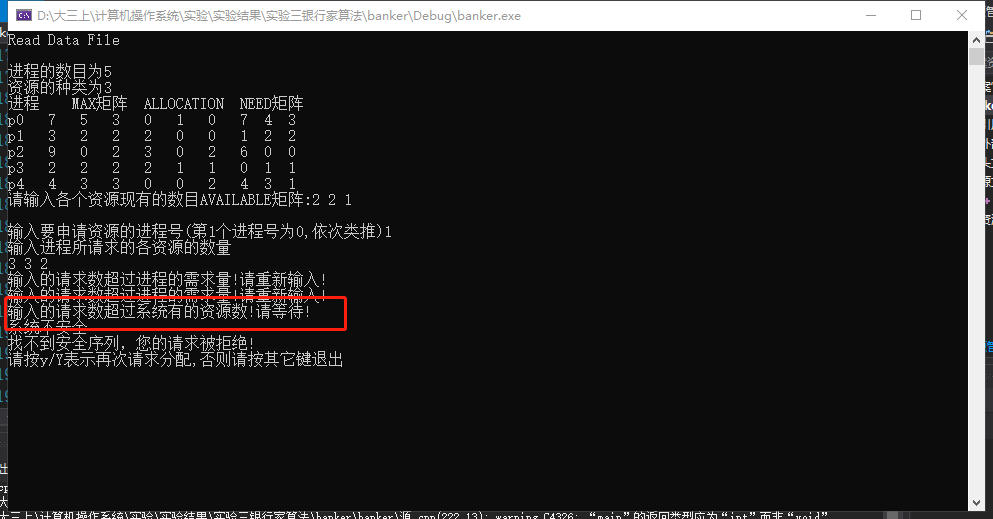
（1）文件中读入进程数目为5，资源种类为3，各进程的MAX和AVAILABLE矩阵如下图，并输入现有的资源，以及请求的进程和资源，得到了一个安全序列，说明是安全的。



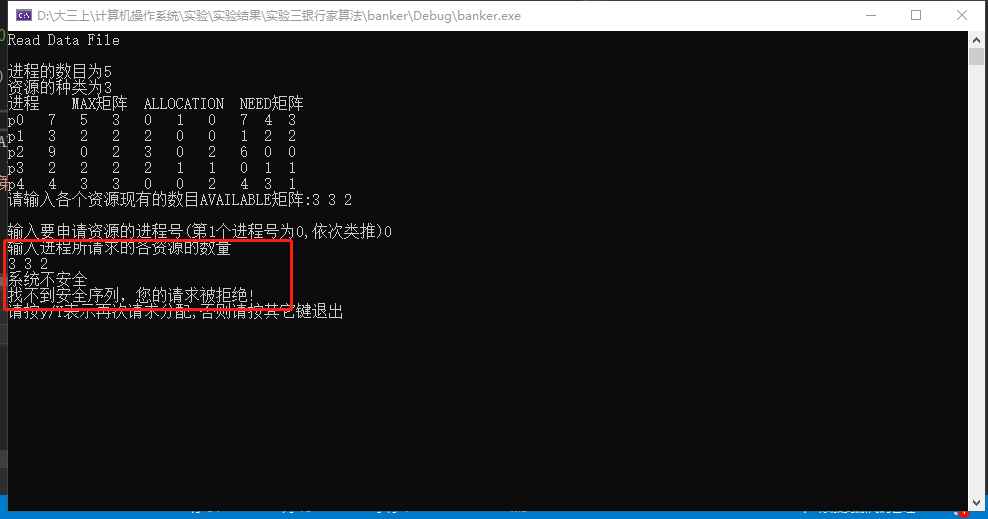
1. 若请求的资源超过最大需求量，则会报错



1. 如果请求的资源超过系统现有的资源量，也会报错



1. 若找不到安全序列则表示系统不安全，不能为该进程分配资源



1. **实验代码**

#include <stdio.h>

#define max\_Process 50 //最大进程数

#define max\_Resource 100 //最大资源数

#include <stdlib.h>

#include <fstream>

#pragma warning(disable:4996)

#include <io.h>

#include "windows.h"

using namespace std;

//数据结构

// m个进程，n类个资源

//(1)可利用资源向量Available。  含有n个元素的数组

// 每一元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用

//资源的数目，其数值随该类资源的分配和回收而动态地改变。

// 如Available［j］ = K，则表示系统中现有第j类资源K个。

//(2) 最大需求矩阵Max。  n×m矩阵

// 定义了系统中m个进程中的每一个进程对n类资源的最大需求

// 如Max［i, j］ = K，则表示进程i需要第j类资源的最大数目为K。 (3) 分配矩阵Allocation。  n×m矩阵

// 定义了系统中每一类资源当前已分配给每一进程的资源数

// Allocation［i, j］ = K，则表示进程i当前已分得第j类资源的数目为K。 (4) 需求矩阵Need

// n×m矩阵，用以表示每一个进程尚需的各类资源数。如果Need［i, j］ = K，则

//表示进程i还需要第j类资源K个，方能完成其任务。

int AVAILABLE[max\_Resource]; //可用资源数组

int MAX[max\_Process][max\_Resource]; //最大需求矩阵

int ALLOCATION[max\_Process][max\_Resource]; //分配矩阵

int NEED[max\_Process][max\_Resource]; //需求矩阵

int REQUEST[max\_Process][max\_Resource]; //进程需要资源数（输入的）

int Enough[max\_Process]; /\*系统是否有足够的资源分配\*/

int Safe[max\_Process]; //记录安全序列

int m, n; //m个进程,n个资源

void Input() {//初始化算法

ifstream inFile;

inFile.open("test.txt");

puts("Read Data File \n");

inFile >> m;

inFile >> n;

int i, j;

j = 0;

i = 0;

while (inFile)//从文件中读入数据（MAX和ALLOCATION矩阵）

{

inFile >> MAX[i][j];

j++;

inFile >> MAX[i][j];

j++;

inFile >> MAX[i][j];

j = 0;

inFile >> ALLOCATION[i][j];

j++;

inFile >> ALLOCATION[i][j];

j++;

inFile >> ALLOCATION[i][j];

j = 0;

i++;

}

//printf("%d", MAX[0][0]);

for (i = 0; i < m; i++) {//计算NEED矩阵

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

NEED[i][j] = MAX[i][j] - ALLOCATION[i][j];//NEED矩阵可以直接计算，不需要输入

if (NEED[i][j] < 0) {

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

j--;

continue;

}

}

}

printf("进程的数目为%d\n资源的种类为%d\n",m,n);

printf("进程 MAX矩阵 ALLOCATION NEED矩阵\n");

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

{

printf("p%d ", i);

{

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

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

}

{

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

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

}

{

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

{

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

}

}

printf("\n");

}

printf("请输入各个资源现有的数目AVAILABLE矩阵:");

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

scanf("%d", &AVAILABLE[i]);//测试为3 3 2

}

}

//安全性算法

//(1) 设置两个向量

// ① 工作向量Work : 它表示系统可提供给进程继续运行所需的各类资源

//数目，它含有m个元素，在执行安全算法开始时，Work∶ = Available;

// ② Finish : 它表示系统是否有足够的资源分配给进程，使之运行完成。

//开始时先做Finish［i］∶ = false; 当有足够资源分配给进程时， 再令

//Finish［i］∶ = true。 (2) 从进程集合中找到一个能满足下述条件的进程：

// ① Finish［i］ = false;

// ② 如果对所有j，如果 Need［i, j］≤Work［j］， 执行步骤(3)， 否

//则，执行步骤(4)。 (3) 当进程Pi获得资源后，可顺利执行，直至完成，并释放出分配

//给它的资源，故应执行：

// 对所有j Work［j］∶ = Work［j］ + Allocation［i, j］;  Finish［i］∶ = true;

// go to step 2;

//(4) 如果所有进程的Finish［i］ = true都满足， 则表示系统处于安

//全状态；否则，系统处于不安全状态

int Find\_\_safe() {//安全性算法,返回值:true,false

int i, j, k, l = 0;

int Work[max\_Resource]; //工作数组

for (i = 0; i < n; i++)//n个资源

Work[i] = AVAILABLE[i];

for (i = 0; i < m; i++) {//m个进程

Enough[i] = false;

}

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

if (Enough[i] == true) {

continue;

}

else {

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

if (NEED[i][j] > Work[j]) {

break;

}

}

if (j == n) {//所有need都<work

Enough[i] = true;//可分配

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

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

}

Safe[l++] = i;

i = -1;

}

else {

continue;

}

}

if (l == m) {//均为true

printf("系统安全\n");

printf("安全序列:");

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

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

if (i != l - 1) {

printf("->");

}

}

return true;

}

}

printf("系统不安全\n");

return false;

}

void Banker() /\*银行家算法\*/ {

int i, process\_Id;

char again;

while (1) {

printf("\n输入要申请资源的进程号(第1个进程号为0,依次类推)");

scanf("%d", &process\_Id);//测试1

printf("输入进程所请求的各资源的数量\n");

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

scanf("%d", &REQUEST[process\_Id][i]);//测试1 0 2

}

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

if (REQUEST[process\_Id][i] > NEED[process\_Id][i]) {

printf("输入的请求数超过进程的需求量!请重新输入!\n"); continue;

}

if (REQUEST[process\_Id][i] > AVAILABLE[i]) {

printf("输入的请求数超过系统有的资源数!请等待!\n"); continue;

}

}

for (i = 0; i < n; i++) {//系统试着把资源分配给pi

AVAILABLE[i] -= REQUEST[process\_Id][i];

ALLOCATION[process\_Id][i] += REQUEST[process\_Id][i];

NEED[process\_Id][i] -= REQUEST[process\_Id][i];

}

if (Find\_\_safe()) {

printf("可以找到安全序列，同意分配请求!\n");

}

else {

printf("找不到安全序列，您的请求被拒绝!\n");

for (i = 0; i < n; i++) {//恢复原来的资源分配

AVAILABLE[i] += REQUEST[process\_Id][i];

ALLOCATION[process\_Id][i] -= REQUEST[process\_Id][i];

NEED[process\_Id][i] += REQUEST[process\_Id][i];

}

}

for (i = 0; i < m; i++) {//恢复Enough序列

Enough[i] = false;

}

printf("请按y/Y表示再次请求分配,否则请按其它键退出\n");

scanf(" %c", &again);

if (again == 'y' || again == 'Y') {

continue;

}

break;

}

}

void main() {

Input();

Banker();

}