**银行家算法**

**实验报告**

姓名： 余泽晨

学号： 71115112

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering

College of Software Engineering

Southeast University

二0 17 年 5 月

一、实验目的

通过实验，加深对多实例资源分配系统中死锁避免方法——银行家算法的理解，掌握 Windows 环境下银行家算法的实现方法，同时巩固利用 Windows API 进行共享数据互斥访问和多线程编程的方法。

二、实验内容

1. 在 Windows 操作系统上，利用 Win32 API 编写多线程应用程序实现银行家算法。

2. 创建 n 个线程来申请或释放资源，只有保证系统安全，才会批准资源申请。

3. 通过 Win32 API 提供的信号量机制，实现共享数据的并发访问。

三、实验步骤

1. 环境

Win10

2. 在 Win10 操作系统上，利用 Win32 API 提供的信号量机制，编写多线程应用程序实现银行家算法.

3. **银行家算法:**设进程run\_id提出请求Request [id]，则银行家算法按如下规则进行判断。

(1)如果Request [id] [i]<= Need[id][i]，则转(2)；否则，返回请求错误。

(2)如果Request [id] [i]<= Avaliable[i]，则转(3)；否则，返回请求等待。

(3)系统试探分配资源，修改相关数据：

Avaliable [i]-= Request [cusneed][i];

Allocation[id][i]+= Request [cusneed][i];

Need[id][i]-= Request [cusneed][i];

(4)系统执行安全性检查，如安全，则分配成立；否则试探性分配作废，系统恢复原状，请求等待。

**安全性检查算法:**

(1)设置两个工作向量Work= Avaliable;Finish

(2)从进程集合中找到一个满足下述条件的进程，

Finish ==false;

Need<=Work;

如找到，执行(3)；否则，执行(4)

(3)设进程获得资源，可顺利执行，直至完成，从而释放资源。

Work+= Allocation;

Finish=true;

GOTO (2)

(4)如所有的进程Finish= true，则表示安全；否则系统不安全。

四、主要数据结构及其说明

1. 可利用资源向量Available

含有m个元素的[数组](http://baike.baidu.com/item/%E6%95%B0%E7%BB%84)，其中的每一个元素代表一类可利用的资源数目。如果Available[j]=K，则表示系统中现有Rj类资源K个。

1. 最大需求[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)Max

n×m的[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max[i,j]=K，则表示进程i需要Rj类资源的最大数目为K。

1. 分配[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)Allocation

n×m的[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)，它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。如果Allocation[i,j]=K，则表示进程i当前已分得Rj类资源的 数目为K。

1. 需求[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)Need

n×m的[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)，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Rj类资源K个，方能完成其任务。

Need[i,j]=Max[i,j]-Allocation[i,j]

1. 请求矩阵 Request

n×m的[矩阵](http://baike.baidu.com/item/%E7%9F%A9%E9%98%B5)，用以表示每一个进程一次请求的各类资源数。请求的资源数不可大于Need的值

1. Finish向量

bool类型数组,用于辅助判断系统是否处在安全状态

1. ss向量

int类型数组,用于记录系统的一个安全序列

1. Mutex

互斥信号量Mutex,实现对临界区的互斥访问

1. int n

记录线程数量

1. int m

记录资源类型数量

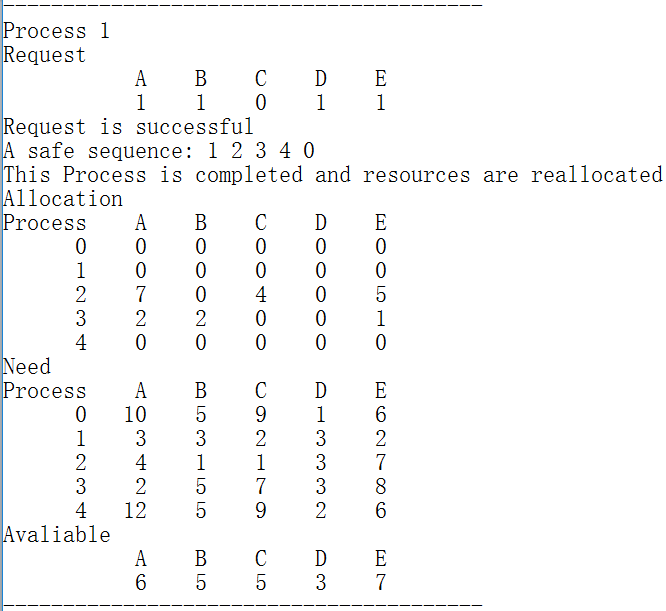
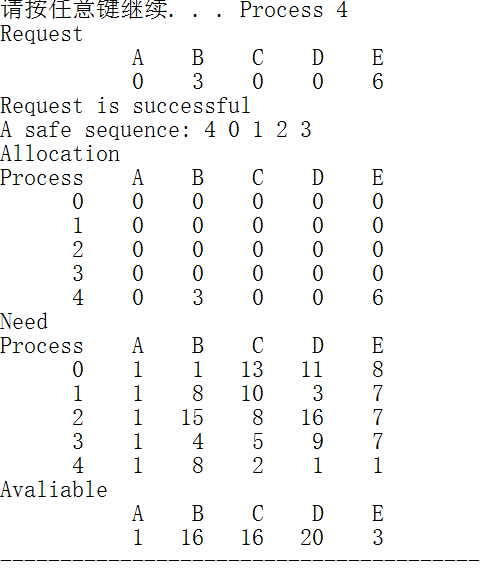
1. PThreadId,CThreadId

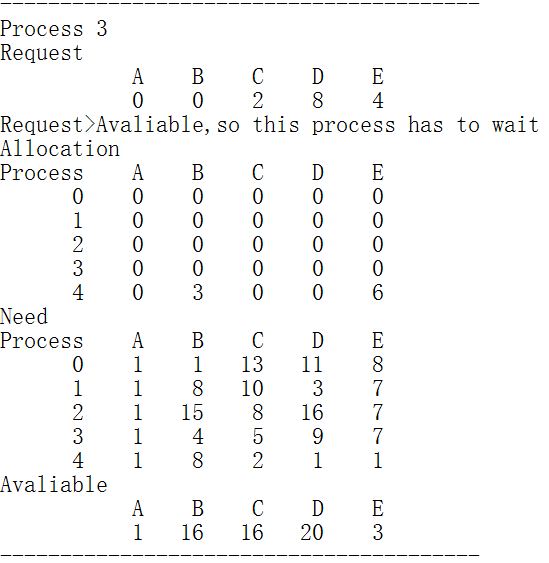
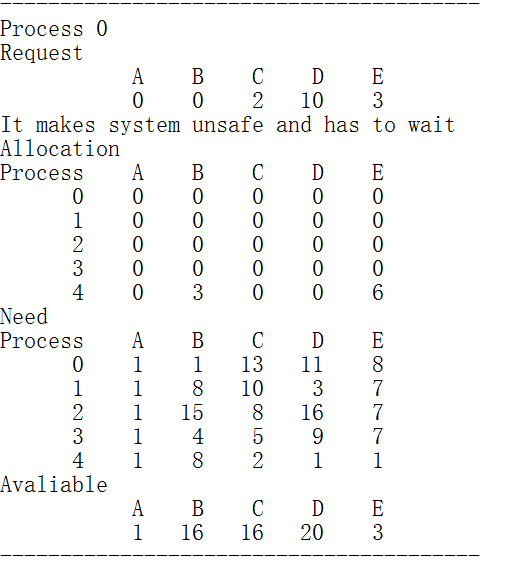
运行线程的id

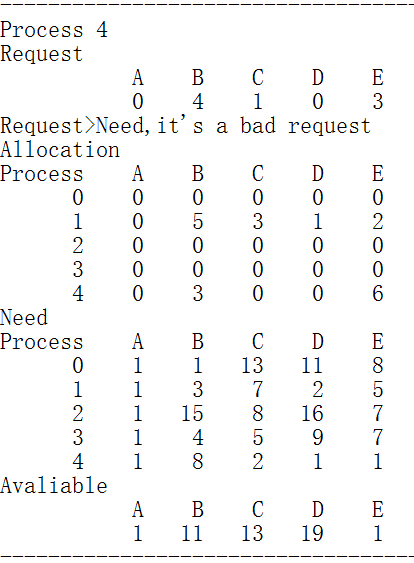
1. PThreadHandle

运行的线程

五、程序运行结果（系统截图）







六、实验体会

1. 在实验中,熟悉了利用 Windows API进行多线程编程,加深了对临界区问题和进程同步机制的理解的认识,对用信号量机制解决一些问题也有了更深的体会.

2. 编程实现银行家算法,加深了对银行家算法,安全性检查算法和死锁避免的认识.

3. 编程实现时中要注意对各个数据结构的使用,认清每个变量的含义.两个数组比较时注意结果是否正确(比较结果是否符合实际的含义)

七、源程序并附上注释

#include<iostream>

#include<Windows.h>

#include<stdlib.h>

#include<time.h>

#include<iomanip>

#include<string>

using namespace std;

int n;//进程数

int m;//资源类型数

int Max[10][10];//各个进程需要的各种资源的最大值

int Allocation[10][10];//已分配给各个进程的各种资源的值

int Need[10][10];//各个进程还需要的各种资源数量

int Request[10][10];//各个进程新提出的对各种资源的请求

int Available[10];//系统当前可分配的各种资源的值

bool Finish[10];//各个进程是否能分配到足够的资源

int ss[10];//记录安全序列

enum Ret

{

Bad\_Request,

Wait,

UnSafe,

Safe

};//枚举进程请求的反馈结果

HANDLE Mutex;//互斥锁,实现各个线程对临界资源的互斥访问

void output(string s, int a[][10]) {//输出二维矩阵

cout << s << endl;

cout << "Process";

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

cout << setw(5) << (char)('A' + i);

cout << endl;

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

cout << setw(7) << i;

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

cout << setw(5) << a[i][j];

cout << endl;

}

}

void output(string s, int a[]) {//输出一维矩阵

cout << s << endl << setw(7) << " ";

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

cout << setw(5) << (char)('A' + i);

cout << endl << setw(7) << " ";

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

cout << setw(5) << a[j];

cout << endl;

}

void Init() {//初始化Max,Need,Allocation,Available,Request矩阵

srand(time(0));

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

Available[i] = rand() % 20 + 1;

}

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

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

srand(time(0) + 7 \* i + 3 \* j + i\*j);

Need[i][j] = Max[i][j] = rand() % Available[j] + 1;

}

memset(Allocation, 0, 100);

memset(Request, 0, 100);

}

bool leq(int a[], int b[]) {

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

if (a[i] > b[i])

return false;

}

return true;

}

bool geq(int a[], int b[]) {

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

if (a[i] < b[i])

return false;

}

return true;

}

void add(int d[], int s[]) {//数组相加

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

d[i] += s[i];

}

void Minus(int d[], int s[]) {//数组相减

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

d[i] -= s[i];

}

bool IsSave() {//判断系统是否处于安全状态,若处于安全状态,找出一个安全序列

int Work[10];

copy(Available, Available + 10, Work);

memset(Finish, 0, 10);

int count = 0, temp = -1;

while (count != temp&&count < n) {

temp = count;

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

if ((!Finish[i]) && geq(Work, Need[i])) {

Finish[i] = 1;

add(Work, Allocation[i]);

ss[count] = i;

count++;

if (count == n)

break;

}

}

}

return count == n ? true : false;

}

Ret Req(int id) {//判断系统当前状态对当前请求的响应

if (!leq(Request[id], Need[id]))//若Request中有资源数大于Need,则判断该请求错误

return Bad\_Request;

else if (!leq(Request[id], Available))//若Request中有资源数大于Available,则判断该请求需要等待

return Wait;

else {

Minus(Available, Request[id]);

Minus(Need[id], Request[id]);

add(Allocation[id], Request[id]);

if (IsSave())//若满足当前请求后,系统处于安全状态,则判断系统可以满足当前请求

return Safe;

else {//若满足当前请求会导致系统处在不安全状态,则判断该请求会导致系统处于不安全状态,需要等待

add(Available, Request[id]);

add(Need[id], Request[id]);

Minus(Allocation[id], Request[id]);

return UnSafe;

}

}

}

bool Complete(int id) {//判断进程是否获得全部需要的资源

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

if (Need[id][i])

return false;

return true;

}

void ReleaseResources(int id) {//释放进程占用的资源

add(Available, Allocation[id]);

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

Need[id][i] = Max[id][i];

Allocation[id][i] = 0;

}

}

int GetRandomRequest(int id) {//获得一个随机的Request(请求的资源值<Max)

int sum = 0;

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

if (Max[id][i])

Request[id][i] = (rand() % Max[id][i]);

else

Request[id][i] = 0;

sum += Request[id][i];

}

return sum;

}

DWORD WINAPI Run(LPVOID p)//运行的线程

{

srand(time(0) + GetCurrentThreadId());

int id = (int)p;

Ret ret = Safe;

do {

int random = rand() % 100 + 1;

Sleep(random);//随机等待1~100ms

WaitForSingleObject(Mutex, INFINITE);//实现对临界区的互斥访问

if (ret != UnSafe) {

if (!GetRandomRequest(id)) {

ReleaseMutex(Mutex);

continue;

}

}

cout << "Process " << id << endl;

output("Request", Request[id]);

ret = Req(id);

if (ret == Bad\_Request) {

cout << "Request>Need,it's a bad request" << endl;

}

else if (ret == UnSafe) {

cout << "It makes system unsafe and has to wait" << endl;

}

else if (ret == Wait) {

cout << "Request>Avaliable,so this process has to wait" << endl;

}

else {

cout << "Request is successful" << endl;

cout << "A safe sequence: ";

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

cout << ss[i] << " ";

cout << endl;

if (Complete(id))

{

ReleaseResources(id);

cout << "This Process is completed and resources are reallocated" << endl;

}

}

output("Allocation", Allocation);

output("Need", Need);

output("Avaliable", Available);

cout << "----------------------------------------" << endl;

ReleaseMutex(Mutex);

} while (true);

return 0;

}

int main() {

cout << "Input the number of threads" << endl;

cin >> n;

cout << "Input the number of kinds of resources" << endl;

cin >> m;

Mutex = CreateMutex(NULL, FALSE, NULL);

DWORD \*PThreadId = new DWORD[n];

HANDLE \*PThreadHandle = new HANDLE[n];

cout << "Init" << endl;

Init();

output("Allocation", Allocation);

output("Need", Need);

output("Avaliable", Available);

cout << "----------------------------------------" << endl;

for (unsigned short i = 0; i < n; i++)//创建运行线程

{

PThreadId[i] = i;

PThreadHandle[i] = CreateThread(NULL, 0, Run, (LPVOID)i, 0, &PThreadId[i]);

}

//Sleep(2000);//主线程睡眠输入时间

cout << "End" << endl;

system("pause");

return 0;

}