|  |
| --- |
| 71Y13110 陈建蓉  2015-6-8 |
|  |

|  |
| --- |
| **操作系统实验三：** |
| **银行家算法的实现** |

1. **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。

1. **实验内容**
2. 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。
3. 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。
4. 通过 Win32 API 提供的信号量机制，实现共享数据的并发访问。
5. **实验环境**

Windows 8 + VMware Workstation + Ubuntu14.04

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. 实验步骤
3. 安全性算法实现
4. 资源请求性算法实现
5. 源程序

#include<iostream>

#include<windows.h>

#include <time.h>

#include<stdio.h>

#include<stdlib.h>

using namespace std;

const int m = 3;

const int n = 5;

HANDLE mutex;

HANDLE mutex1;

int available[m] = { 3, 3, 2 };

int allocation[n][m] = {

{ 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[n][m] = {

{ 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int need[n][m] = {0};

bool compare(int a[][m], int\*b, int i)

{

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

{

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

return false;

}

return true;

}

bool compare(int a[][m], int b[][m], int i)

{

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

{

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

return false;

}

return true;

}

bool safe(int need[][m], int \* available, int allocation[][m])

{

int work[m] = { 0 };

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

{

work[i] = available[i];

}

bool finish[n] = { false };

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

{

int i;

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

{

if (finish[i] == false && compare(need, work, i) == true)

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

{

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

finish[i] = true;

}

}

i = 0;

}

int num = 0;

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

if (finish[i] == true)

num++;

if (num == n)

return true;

else

return false;

}

DWORD WINAPI pro(LPVOID param)

{

Sleep(1000);

srand((unsigned)time(NULL));

int j = rand() % n;

int request[n][m] = { 0 };

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

{

request[j][i] = rand() % 4;

}

WaitForSingleObject(mutex1, INFINITE);

cout << "--------------Process No." << j << " request A:" << request[j][0]

<< ", B:" << request[j][1] << ", C:"

<< request[j][2] << "."

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

ReleaseMutex(mutex1);

WaitForSingleObject(mutex, INFINITE);

int no = j;

if (compare(request, need, no))

{

if (compare(request, available, no))

{

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

{

available[i] = available[i] - request[no][i];

allocation[no][i] = allocation[no][i] + request[no][i];

need[no][i] = need[no][i] - request[no][i];

}

if (safe(need, available, allocation))

{

cout << "Process No." << no << " request completed." << endl;

cout << "Available:A:" << available[0]

<< ", B:" << available[1]

<< ", C:" << available[2] << ".\n" << endl ;

return 0;

}

else

{

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

{

available[i] = available[i] + request[no][i];

allocation[no][i] = allocation[no][i] - request[no][i];

need[no][i] = need[no][i] + request[no][i];

}

cout << "Error : Not safe.";

}

}

else

cout << "Error : request > available"<<endl;

}

else

cout << "Error : request > need" << endl;

cout << "Process No." << no << " request failed." << endl;

cout << "Available:A:" << available[0]

<< ", B:" << available[1]

<< ", C:" << available[2] << ".\n" << endl;

ReleaseMutex(mutex);

return 0;

}

int main()

{

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

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

need[i][j] = max[i][j] - allocation[i][j];

cout << "Available:A:" << available[0]

<< ", B:" << available[1]

<< ", C:" << available[2] << ".\n" << endl;

const int num = 10;

DWORD ThreadId[num];

HANDLE ThreadHandle[num];

mutex = CreateMutex(NULL, FALSE, NULL);

mutex1 = CreateMutex(NULL, FALSE, NULL);

for (int k = 0; k < num; k++)

{

Sleep(50);

ThreadHandle[k] = CreateThread(

NULL,

0,

&pro,

NULL,

0,

&ThreadId[k]);

Sleep(5000);

if (k-1>0&&ThreadHandle[k-1] != NULL)

{

TerminateThread(ThreadHandle[k-1], 0);

}

}

for (int k = 0; k < num; k++)

{

CloseHandle(ThreadHandle[k]);

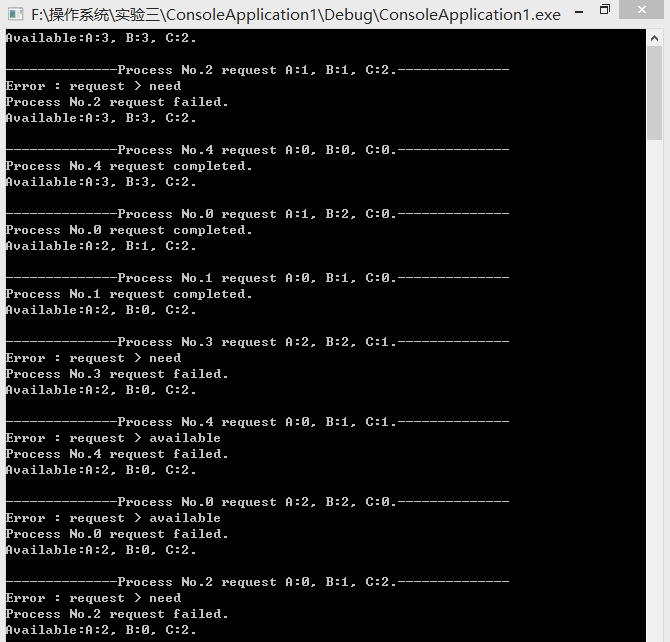
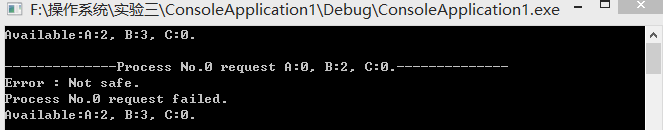
}

system("pause");

return 0;

}

1. 程序运行截屏

1. 实验体会

多个数组的大小比较和数组间赋值容易出错，导致算法出错。