操作系统实验三:银行家算法

姓名:魏远卓

学号:71113211

提交日期:2015.6.15

【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。

【实验环境】

ubuntu12.04

【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

【实验过程】  
linux下：

#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>

#include <pthread.h>

#include <iostream>

#include <unistd.h>

using namespace std;

#define pthnum 5

#define resource 3

#define SAFE true

#define UNSAFE false

int Max[pthnum][resource]={

{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}

};

int allocation[pthnum][resource]={

{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}

};

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

int finished[pthnum]={0};

int need[pthnum][resource]={0};

sem\_t mutex;

bool compare\_work(int \*b,int i){

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

if(need[i][j]<=b[j])

continue;

else

return false;

}

return true;

}

bool compare\_need(int \*request,int pthnumber){

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

{

if(request[i]<=need[pthnumber][i])

continue;

else

return false;

}

return true;

}

bool compare\_available(int \*request){

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

{

if(request[i]<=available[i])

continue;

else

return false;

}

}

bool safety(){

int work[resource]={0};

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

{

work[j]=available[j];

}

for(int k=0;k<pthnum;k++){

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

{

if(finished[i]==0&&compare\_work(work,i)==true){

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

{

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

finished[i]=true;

}

}

}

}

int count=0;

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

{

if(finished[i]==true)

count++;

}

if(count==pthnum)

return SAFE;

else

return UNSAFE;

}

void \*request\_thread(void \*value){

sem\_wait(&mutex);

int \*value1=(int\*)value;

srand((unsigned int)time(NULL));

int request[resource]={0};

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

request[i]=rand()%3;

}

usleep(1000000);

cout<<"Thread "<<\*value1<<" request "<<"A:"<<request[0]<<" B:"<<request[1]

<<" C:"<<request[2]<<"\t";

if(compare\_need(request,\*value1))

{

cout<<"request<=need"<<"\t";

if(compare\_available(request))

{

cout<<"request<=available"<<"\t";

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

{

available[i]=available[i]-request[i];

allocation[\*value1][i]=allocation[\*value1][i]+request[i];

need[\*value1][i]=need[\*value1][i]-request[i];

}

if(safety())

{

cout<<"Allocation done.Thread "<<\*value1<<" got need."<<"SUCCESS!"<<endl;

}

else

{

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

{

available[i]=available[i]+request[i];

allocation[\*value1][i]=allocation[\*value1][i]-request[i];

need[\*value1][i]=need[\*value1][i]+request[i];

}

cout<<"Not safe.Thread "<<\*value1<<" must wait.NOT SUCCESS!"<<endl;

}

}

else

cout<<"Thread must wait.There is no available resource.NOT SUCCESS!"<<endl;

}

else

cout<<"Error!request>need.NOT SUCCESS!"<<endl;

sem\_post(&mutex);

}

int main()

{

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

{

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

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

}

}

pthread\_t pid[pthnum];

pthread\_attr\_t attr[pthnum];

int thread\_count[pthnum]={0};

sem\_init(&mutex,0,1);

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

pthread\_attr\_init(&attr[i]);

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

thread\_count[i]=i;

//usleep(1000000);

pthread\_create(&pid[i],&attr[i],request\_thread,(void\*)&thread\_count[i]);

}

pthread\_join(pid[0],NULL);

pthread\_join(pid[1],NULL);

pthread\_join(pid[2],NULL);

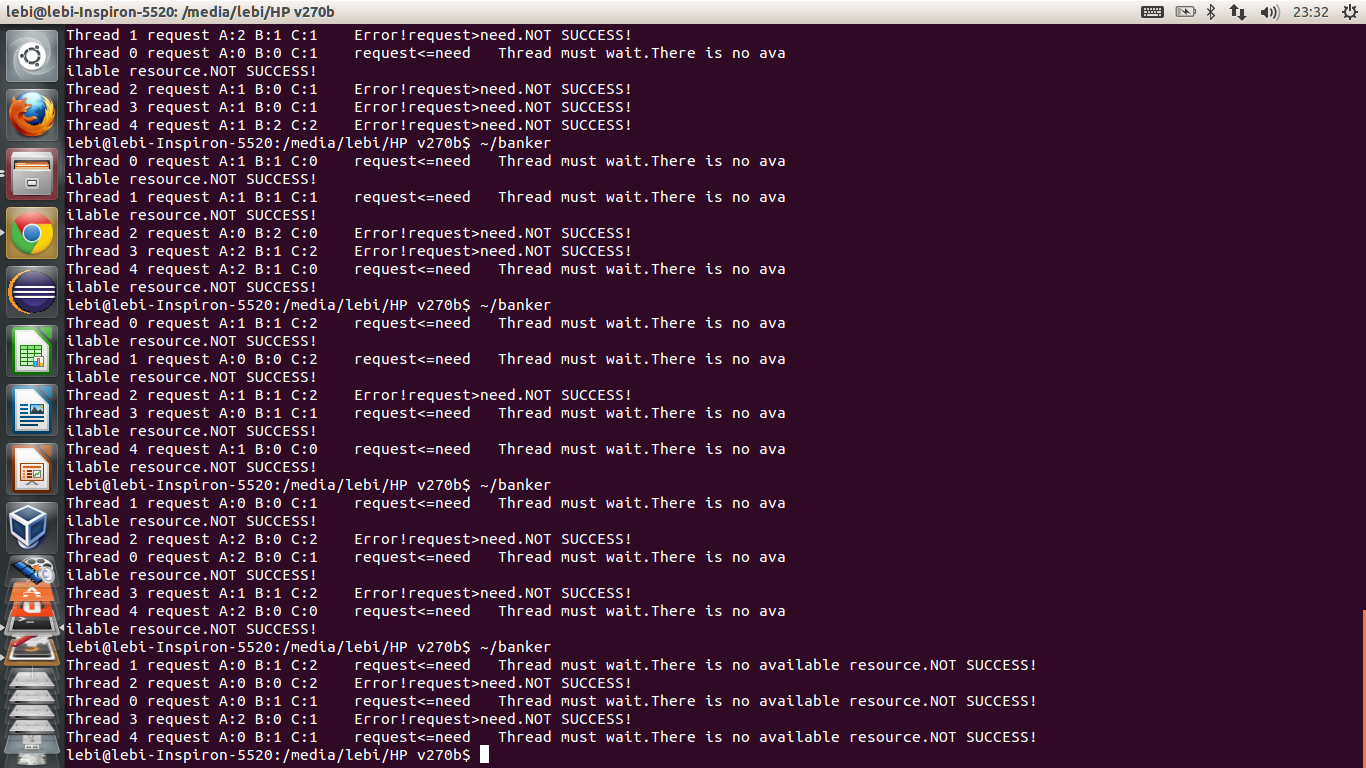
pthread\_join(pid[3],NULL);

pthread\_join(pid[4],NULL);

return 0;

}

运行结果：



【心得体会】

银行家算法有多种实现形式，我用的是随机请求一定资源的方式来检验银行家算法。主要用safety（）函数来判断申请是否安全，线程执行的操作写在request\_thread（）里。

遇到的问题及体会：

1.多线程的并发过程不完全按照代码写的样子来，会无法预测。（设置了互斥信号量控制表示资源的全局变量，但是第一个抢到锁的不一定是几号线程）

2.g++编译器的问题，在线程创建函数 pthread\_create(&pid[i],&attr[i],request\_thread,(void\*)&thread\_count[i])中的程序入口参数遇到过一些格式的问题，后来在传进的参数前加(void\*)解决此问题

3.银行家算法是解决死锁问题的重要算法，这次实验使我对它有了更深入地理解，也对多线程和信号量在实际编程中的应用有了更具体的认知。