银行家算法linux多线程实现

#include<cstring>

#include<pthread.h>

#include<cstdio>

#include<unistd.h>

#include<iostream>

#include<stdlib.h>//atoi

#include <sys/ipc.h> //shm

#include <sys/shm.h> //shm

#include<sys/types.h>

#include<sys/stat.h> //S\_IRUSR | S\_IWUSR

#include<sys/wait.h>

#include<semaphore.h>

#include<vector>

using namespace std;

const int Maxpronum =50; //最大线程的数目

using namespace std;

int res; //用户输入的资源数

int pronum; //输入的线程数

int restthread; //剩余线程数目

int vis[Maxpronum]; //线程有没有访问过

int threadFinished[Maxpronum]; //线程是否已经结束

vector<int> resMax[Maxpronum]; //最大需求量

vector<int> Allocation[Maxpronum];

vector<int> Need[Maxpronum];

vector<int> Available;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

bool safe(int id){ //安全性算法

vector<int> tmpAvailable(Available); //临时分配量

vector<int> threadSafeSequence;

memset(vis, 0, sizeof(vis));

while(threadSafeSequence.size() < restthread){

int find = 0; //find==true表示找到了可以完成的线程

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

if(!vis[i] && !threadFinished[i]){

int j=0;

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

if(Need[i][j] > tmpAvailable[j])

break;

find =1;

vis[i] = 1;

threadSafeSequence.push\_back(i);

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

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

}

if(!find)

break;

}

if(threadSafeSequence.size() == restthread){

cout<<"此时系统处于安全状态，存在线程安全序列如下:"<<endl;

for(int i=0; i<threadSafeSequence.size(); ++i)

cout<<threadSafeSequence[i]<<" ";

cout<<endl;

return true;

}

else {

cout<<"此时系统处于不安全状态，Request被拒绝，试分配作废。进程"<<id<<"等待!"<<endl;

return false;

}

}

void reset(int id){ //资源回收

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

{

Available[i] += Allocation[id][i];

Allocation[id][i]=0;

}

}

int Banker(int id, vector<int>Req){ //银行家算法 //thread id对 Reqid 的请求数量为k

//return 1则线程等待。return 0分配成功。

for(int i=0; i<res;i++){ //步骤一，检查Request是否大于need或Available

int m = Req[i];

if(m <= Need[id][i])

{

if(m > Available[i]){

cout<<"ERROR!!!线程"<<id<<"请求"<<i<<"类资源数目大于该类剩余资源的数目!"<<endl<<endl;

return 1; }

}

else{

cout<<"ERROR!线程"<<id<<"请求"<<i<<"类资源数目大于线程所需要的该类资源的数目!"<<endl<<endl;

return 1;

}

}

//预分配

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

int k = Req[i];

Available[i] -= k; Allocation[id][i] += k; Need[id][i] -= k;

}

//安全性算法

if(!safe(id)){ //预分配作废

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

int k = Req[i];

Available[i] += k;

Allocation[id][i] -= k;

Need[id][i] += k;

}

return 1;

}

cout<<"线程"<<id<<"获得资源:";

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

cout<<" "<<i<<"类:"<<Req[i];

cout<<endl;

return 0;

}

void \*threadprocess(void \*arg){

int id=\*(int\*)arg;

srand((int)time(0));

//设置第一次的Request。 这里每次Request都请求res类。如果想要每次请求一类，可以先rand一类，再rand请求资源数。

vector<int> Req;

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

int m = Need[id][i] == 0 ? 0 : (rand() % Need[id][i]+1);

Req.push\_back(m);

}

sleep(pronum-id); //等待所有线程创建完毕

while(true){

pthread\_mutex\_lock(&mutex);

/\* bool ptfinished = true;//该线程是否已经将资源请求完毕

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

if(Need[id][i] != 0){

ptfinished = false;

break;

}

if(ptfinished){

cout<<"线程"<<id<<"得到需要的全部资源!"<<endl;

sleep(1);

cout<<"线程"<<id<<"执行完毕，线程结束"<<endl<<endl;

restthread--; //全局变量

threadFinished[id] = 1;//线程结束

reset(id);

pthread\_cond\_broadcast(&cond);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}\*/

cout<<"线程"<<id;

for(int l=0;l<res;l++)

cout<<"请求"<<l<<"类资源"<<Req[l]<<"个"<<" ";

cout<<endl;

switch(Banker(id, Req)){ //Banker

case 1:

pthread\_cond\_wait(&cond, &mutex); //Banker失败，等待 //这里犯了错误。一开始用的sem\_init mutex。

break;

case 0: //Request成功 重新Req

Req.clear();

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

int m = Need[id][i] == 0 ? 0 : rand() % Need[id][i]+1;

Req.push\_back(m);

}

break;

}

bool ptfinished = true;//该线程是否已经将资源请求完毕

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

if(Need[id][i] != 0){

ptfinished = false;

break;

}

if(ptfinished){

cout<<"线程"<<id<<"得到需要的全部资源!"<<endl;

sleep(1);

cout<<"线程"<<id<<"执行完毕，线程结束"<<endl<<endl;

restthread--; //全局变量

threadFinished[id] = 1;//线程结束

reset(id);

pthread\_cond\_broadcast(&cond);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

pthread\_mutex\_unlock(&mutex); //效果更明显

sleep(1);

}

}

void init() //初始化res，pronum，Allocation，Need，resMax，Available，restthread

{

memset(threadFinished, 0, sizeof(threadFinished));

for(int i=0;i<50;i++)

{

Allocation[i].clear();

Need[i].clear();

resMax[i].clear();

}

Available.clear();

cout<<"输入线程的数目和资源的种类数目:"<<endl;

cout<<"线程数不超过50"<<endl;

cin>>pronum>>res;

restthread = pronum;

cout<<"请输入每种资源的数目:" <<endl;

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

int k;

cin>>k;

Available.push\_back(k);

}

int k=0;

cout<<"按顺序输入每个线程对某类资源的需求量:"<<endl;

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

cout<<"线程"<<i<<"需要的资源:"<<endl;

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

cin>>k;

resMax[i].push\_back(k);

}

}

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

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

{

Need[i].push\_back(resMax[i][j]);

Allocation[i].push\_back(0);

}

}

int main(int argc, char\*argv[]){

init();

pthread\_t tid[pronum+2];

pthread\_mutex\_init(&mutex, NULL);

pthread\_cond\_init(&cond, NULL);

int m[pronum];

if(safe(0)==false) //先跑一次安全算法

return 0;

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

{

m[i]=i;

pthread\_create(&tid[i], NULL, threadprocess, (void\*)&m[i]);

}

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

pthread\_join(tid[i], NULL);

sleep(15);

return 0;

}