C++多线程

//多线程异步执行

thread t1(run2);//C++的多线程类

thread t2(run2);

thread t3(run2);

t1.join();

t2.join();

t3.join();//主线程必须等待我退出才能退出

t1.detach();

t2.detach();

t3.detach();//主线程退出，子线程也退出

输出电脑的核心数和线程数，线程数最好是核心数的整数倍,可以使得cpu达到最平衡的运行状态

cout << thread::hardware\_concurrency() << "核心" << endl;

得到线程的id

using namespace std::this\_thread;

auto fun = [](){cout << get\_id() << endl; };

fun();

C++多线程和lambda表达式:

以后的多线程函数都使用Lambda表达式

//必须是const char\* str

auto fun = [](const char\* str){MessageBoxA(0, str, str, 0); };

thread t1(fun, "Hello");

thread t2(fun, "World");

多线程在栈和堆上：

auto fun = [](const char\* str){MessageBoxA(0, str, str, 0); };

thread t1(fun, "hello");//不常用的线程放在栈上

thread\* t2 = new thread(fun, "Hello");//常用的，常驻的线程放在堆上

thread t11[2] = { thread(fun, "Hello"), thread(fun, "world") };//栈上的线程数组

堆上的线程数组

thread\* p = new thread[2];

p[0] = thread(fun, "Hello");

p[1] = thread(fun, "World");

堆上的多线程没有自动初始化函数

thread\* p2 = new thread{ thread(fun, "Hello"), thread(fun, "world") };

线程安全：

使用的头文件：

#include <mutex>

#include <atomic>

三种方法：

1.int num = 0;//多线程使用全局变量，线程数量较多的时候可能会出现访问冲突

2.mutex mlock;//全局变量的锁，实现锁定 但是会增加内存的访问时间 开锁解锁浪费时间

mlock.lock();

num += 1;

mlock.unlock();

3.atomic\_int ato\_num;//原子变量，一定可以保证线程安全

有多种类型可以使用