Modern cpp 30 lectures — concurrency

Friday, December 25, 2020 8:33 AM

我们编译完执行的C++程序就是一个进程，而每个进程里可以有多个线程。每个进程有独立的地址空间，不与其他进程分享，一个进程里有多个线程，彼此共享同一个地址空间。
C++里的并发，主要讲的就是多线程。

基于thread的多线程开发：

```
#include <chrono>
#include <iostream>
#include <mutex>
#include <thread>

using namespace std;

mutex output_lock;

void func(const char* name)
{
  this_thread::sleep_for(100ms);
  lock_guard<mutex> guard{output_lock};
  cout << "I am thread " << name << '\n';
}

int main()
{
 thread t1{func, "A"};
 thread t2{func, "B"};
 t1.join();
 t2.join();
}
```

需要限制同时只有一个线程在使用cout。

Output

```
I am thread B
I am thread A
```

代码执行了下列操作：

- 传递参数，起两个线程
- 两个线程分别休眠100毫秒
- 使用互斥量(mutex)锁定cout，然后输出一行信息。
- 主线程等待这两个线程退出后程序结束。

注意：

- thread的构造函数的第一个参数是函数(对象)，后面跟的是这个函数所需的参数。
- thread要求在析构之前要么join，(阻塞直到线程退出)，要么detach(放弃对线程的管理)，否则程序会异常退出。
- sleep_for是this_thread名空间下的一个自由函数，表示当前线程休眠指定的时间。
- 如果没有output_lock的同步，输出会交错在一起。

Mutex

一个互斥量只能被一个线程锁定，用来保护某个代码块在同一时间只能被一个线程执行。

- lock：锁定，锁已经被其他线程获得时则阻塞执行。
- try_lock：尝试锁定，获得锁返回true，在锁被其它线程获得时返回false。
- unlock：解除锁定。
- 为避免手动加锁解锁的麻烦，一般应用lock_guard

执行任务，返回数据。

```
#include <chrono>
#include <future>
#include <iostream>
#include <thread>

using namespace std;

int work()
{
  //假装计算了很久
  this_thread::sleep_for(2s);
  return 42;
}

int main()
{
  auto fut = async(launch::async, work);
  ...
  cout << "I am waiting now\n";
  cout << "Answer: " << fut.get() << "\n";
}
```

- launch::async是运行策略，告诉函数模板async应当在新线程里异步调用目标函数。
- 在未来量future上调用get成员函数可获得其结果。