C++ 11 线程：让您的（多任务）生活更轻松。

迈克尔·乔达基斯

2013 年 2 月 6 日 警察

C++ 11 线程

下载 c11threads.zip - 1.1 KB

介绍

本文旨在帮助有经验的 Win32 程序员了解 C++ 11 线程与同步对象、Win32 线程与同步对象之间的异同。

在 Win32 中，所有同步对象句柄都是全局句柄。它们可以共享，甚至可以在进程之间复制。在 C++ 11 中，所有同步对象都是堆栈对象，这意味着它们必须被“分离”（如果支持分离）才能被堆栈帧破坏。如果您不分离许多对象，它们将撤消它们的操作并可能会终止您的计划（如果您是 Win32 程序员，则这些计划是面向“全局句柄”的）。

所有 C++ 11 同步对象都有一个 native_handle() 成员，它返回特定于实现的句柄（在 Win32 中，一个 HANDLE）。

在我所有的例子中，我都给出了 Win32 伪代码。玩得开心！

背景颜色

0x00000000。也就是说，什么都没有。我也是 C++ 11 线程新手。您需要了解有关 Win32 同步的方法。这不是关于正确同步技术的教程，而是对 C++11 机制的快速介绍，以执行您已经在脑海中计划好的事情。

简单使它完美

简单的例子：启动一个线程，然后等待它完成。

复制代码

```
void foo()
  {
  }
void func()
  {
  std::thread t(foo); // Starts. Equal to CreateThread.
  t.join();  // Equal to WaitForSingleObject to the thread handle.
  }
```

然而，不像 Win32 线程，在这里你可以有参数。

复制代码

```
void foo(int x,int y)
  {
  // x = 4, y = 5.
  }
void func()
  {
  std::thread t(foo,4,5); // Acceptable.
  t.join();
  }
```

通过将隐藏的“this”指针传递给**std::thread**，这使得将成员函数作为线程变得容易。

如果**std::thread**被破坏并且您没有调用**join** ()，它将调用 abort。要让线程在没有 C++ 包装器的情况下运行：

复制代码

```
void foo()
  {
  }
void func()
  {
  std::thread t(foo);
  t.detach(); // C++ object detached from Win32 object. Now t.join() would throw
std::system_error().
  }
```

随着**加入**（）和**分离**（），也有**可连接**（），**get_id**（），**sleep_for**（），**sleep_until**（）。它们的使用应该是不言自明的。

使用那个互斥锁

一个**的std ::互斥**类似于一个Win32临界区。**lock** () 就像**EnterCriticalSection**， unlock() 就像**LeaveCriticalSection**，**try_lock** () 就像**TryEnterCriticalSection**。

复制代码

```
std::mutex m;
int j = 0;
void foo()
  {
  m.lock();
  j++;
  m.unlock();
  }
void func()
  {
  std::thread t1(foo);
  std::thread t2(foo);
  t1.join();
  t2.join();
 // j = 2;
}
```

和以前一样，您必须在锁定后解锁**std::mutex**，并且如果您已经锁定了**std::mutex**，则不得锁定它。这与 Win32 不同，在 Win32 中，当您已经在临界区中时**EnterCriticalSection**不会失败，而是增加一个计数器。

嘿，别走。有**std::recursive_mutex**（谁发明了这些名称？）的行为与临界区完全一样：

复制代码

```
std::recursive_mutex m;
void foo()
  {
```

```
  m.lock();
  m.lock(); // now valid
  j++;
  m.unlock();
  m.unlock(); // don't forget!
  }
```

除了这些类之外，还有**std::timed_mutex**和**std::recursive_timed_mutex**也提供了**try_lock_for / try_lock_until**。这些允许您等待锁定，直到特定超时或特定时间。

线程本地存储

与**TLS**类似，此功能允许您使用**thread_local**修饰符声明全局变量。这意味着每个线程都有它自己的变量实例，具有通用的全局名称。再次考虑前面的例子：

复制代码

```
int j = 0;
void foo()
  {
  m.lock();
  j++;
  m.unlock();
  }
void func()
  {
  j = 0;
  std::thread t1(foo);
  std::thread t2(foo);
  t1.join();
  t2.join();
 // j = 2;
}
```

但是现在看看这个：

复制代码

```
thread_local int j = 0;
void foo()
  {
  m.lock();
  j++; // j is now 1, no matter the thread. j is local to this thread.
  m.unlock();
  }
void func()
  {
  j = 0;
  std::thread t1(foo);
  std::thread t2(foo);
  t1.join();
  t2.join();
 // j still 0. The other "j"s were local to the threads
}
```

Visual Studio 尚不支持线程本地存储。

神秘变量

条件变量是使线程能够等待特定条件的对象。在 Windows 中，这些对象是用户模式的，不能与其他进程共享。在 Windows 中，条件变量与临界区相关联以获取或释放锁。出于同样的原因，**std::condition_variable**与**std::mutex**相关联。

复制代码

```
std::condition_variable c;
std::mutex mu; // We use a mutex rather than a recursive_mutex because the lock has to be
acquired only and exactly once.
void foo5()
   {
   std::unique_lock lock(mu); // Lock the mutex
   c.notify_one(); // WakeConditionVariable. It also releases the unique lock
   }
void func5()
   {
   std::unique_lock lock(mu); // Lock the mutex
   std::thread t1(foo5);
   c.wait(lock); // Equal to SleepConditionVariableCS. This unlocks the mutex mu and allows
foo5 to lock it
   t1.join();
   }
```

这并不像看起来那么无辜。C。即使在**c.wait** () 时也可能返回。**notify_one** () 未被调用（一种称为**虚假唤醒的情况**-http://msdn.microsoft.com/en-us/library/windows/desktop/ms686301(v=vs.85).aspx）。通常，您将 c.wait() 放在 while 循环中，该循环还会检查外部变量以验证通知。

条件变量仅在 Vista 或更高版本中受支持。

承诺未来

考虑这种情况。你想要一个线程做一些工作并返回一个结果。同时你想做其他可能需要也可能不需要一些时间的工作。您希望其他线程的结果在某个时刻可用。

在 Win32 中，您将：

- 使用 CreateThread() 启动线程。
- 在线程内部，完成工作并在准备好时设置事件，同时将结果存储到全局变量中。
- 在主代码中，当你想要结果时，做其他工作然后 WaitForSingleObject。

在 C++ 11 中，这可以通过使用**std::future**轻松完成，并返回任何类型，因为它是一个模板。

复制代码

```
int GetMyAnswer()
   {
   return 10;
   }
int main()
  {
  std::future<int> GetAnAnswer = std::async(GetMyAnswer);  // GetMyAnswer starts background
execution
  int answer = GetAnAnswer.get(); // answer = 10;
  // If GetMyAnswer has finished, this call returns immediately.
  // If not, it waits for the thread to finish.
  }
```

你也有**std::promise**。这个对象可以提供**std::future**稍后会请求的东西。如果你在任何东西被放入承诺之前调用**std::future::get(**），get 会一直等到承诺的值出现。若**std::promise::set_exception()**被调用，则**std::future::get()** 抛出该异常。如果**std::promise**被销毁并且你调用**std::future::get()**，你会得到一个**broken_promise**异常。

复制代码

```
std::promise<int> sex;
void foo()
  {
  // do stuff
  sex.set_value(1); // After this call, future::get() will return this value.
  sex.set_exception(std::make_exception_ptr(std::runtime_error("broken_condom"))); // After
this call, future::get() will throw this exception
  }
int main()
```

```
{
future<int> makesex = sex.get_future();
std::thread t(foo);

// do stuff
try
  {
  makesex.get();
  hurray();
  }
catch(...)
  {
  // She dumped us :(
  }
}
```

代码

随附的 CPP 文件包含我们迄今为止在带有 2012 年 11 月 CTP 编译器（TLS 机制除外）的准备编译的 Visual Studio 12 中所说的所有内容。

下一步是什么？

有很多东西值得包含，例如：

- 信号量
- 命名对象
- 跨进程的可共享对象。
- [...]

你该怎么办？一般来说，在编写新代码时，如果标准对你来说足够了，那就更喜欢这些标准。对于现有代码，我会保留我的 Win32 调用，当我需要将它们移植到另一个平台时，我会使用 C++ 11 函数实现 CreateThread、SetEvent 等。

祝你好运。

历史

- 5 - 2 - 2013 年：首次发布。

执照

分享

关于作者

迈克尔·乔达基斯

软件开发人员

希腊

手表该会员

I'm working in C++, PHP , Java, Windows, iOS, Android and Web (HTML/Javascript/CSS).

I 've a PhD in Digital Signal Processing and Artificial Intelligence and I specialize in Pro Audio and AI applications.

My home page: https://www.turbo-play.com

Comments and Discussions

Add a Comment or Question

Email Alerts

Search Comments

First Prev Next

My vote of 5
imagiro 15-Mar-14 3:11

My vote of 5
Ștefan-Mihai MOGA 12-Mar-13 15:48

syntax error
megaadam 9-Feb-13 0:33

Re: *syntax error*
Michael Chourdakis 10-Feb-13 17:54

Nice but unreadable...
megaadam 7-Feb-13 16:54

Re: Nice but unreadable...
Stone Free 24-May-13 22:34

Good article but hard to read
H.Brydon 7-Feb-13 1:00

Re: Good article but hard to read
Michael Chourdakis 7-Feb-13 18:09

顺便说一句：检查您的签名是否有拼写错误
Andreas Gieriet 6-Feb-13 21:11

我的5票
Ahmed Ibrahim Assaf 6-Feb-13 15:25

我的5票
Andreas Gieriet 6-Feb-13 3:45

又好又好笑
Marius Bancila 6-Feb-13 3:26

不错的文章，但格式不正确
Andreas Gieriet 6-Feb-13 2:31

回复：不错的文章，但格式不正确
Michael Chourdakis 6-Feb-13 3:09

我的5票
Paulo Zemek 6-Feb-13 2:10

回复：我的5票
Michael Chourdakis 6-Feb-13 3:22

刷新 1

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。