62 Threads in C++

本节课讲的是*threads*（线程），也就是讲我们如何进行*parallelization*（并行化）。现在大多数计算机或处理器等设备都有不止一个逻辑处理线程，当我们进入更复杂的项目时，将某些工作移动到两个不同的执行线程会对我们非常有益。不仅仅是为了提高性能，也是我们还能用它做些什么事。

比如，我们用`std::cin.get()`请求用户的输入，但是我们在等待输入的时候什么都做不了，直到用户按下Enter，我们的线程就像被阻塞了一样，但如果我们可以做些其它事情呢？比如向控制台打印一些东西等等。

1. 用线程实现

C++

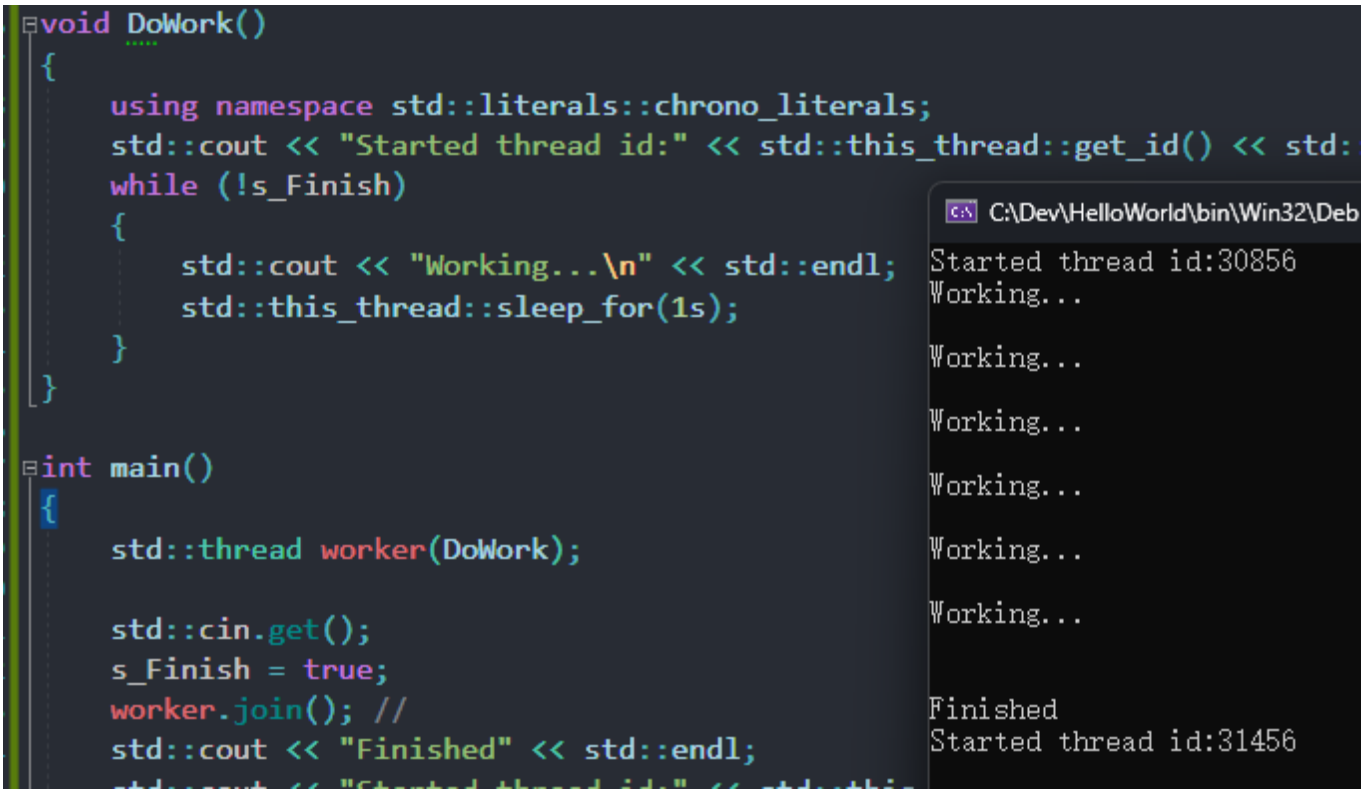
```
#include <iostream>
#include <thread>

static bool s_Finish = false;

void DoWork()
{
    using namespace std::literals::chrono_literals; //引入chrono库的字面量。它使得程序能够更方便地表示时间单位。
    std::cout << "Started thread id:" << std::this_thread::get_id() << std::endl;
    while (!s_Finish)
    {
        std::cout << "Working...\n" << std::endl;
        std::this_thread::sleep_for(1s); // 这里直接用1s表示一秒
    }
}

int main()
{
    std::thread worker(DoWork);

    std::cin.get();
    s_Finish = true;
//  worker.join(); //
    std::cout << "Finished" << std::endl;
    std::cout << "Started thread id:" << std::this_thread::get_id() << std::endl;
    std::cin.get();
}
```



这就是一个简单的C++多线程例子。代码的主要工作流程如下：

1. 全局的`s_Finish`标记声明为`static`，以限制其在当前源文件中的作用范围。
2. `DoWork`函数是一个线程例程。它进入一个无限循环，每秒打印一次"Working..."，直到`s_Finish`标志被设置为`true`。
3. 在`main`函数中，创建了一个名为`worker`的新线程。`DoWork`函数被作为参数传递给线程的构造函数，表示应在新创建的线程中运行此函数。
4. `std::cin.get();`语句是一个阻塞调用，它等待用户按回车键。
5. 一旦按下回车键，`s_Finish`标志被设置为`true`，这导致`DoWork`函数跳出其循环并返回。
6. `worker.join();`语句用于等待线程完成其执行，然后程序才能继续。确保线程在主线程（在这种情况下，是程序）结束之前完成执行是至关重要的。如果程序在`worker`线程仍在运行时结束，那么它将被突然终止，这可能导致各种问题，如资源未被正确释放。

线程很重要，它们对于加速程序非常有用，主要目的是优化，还可以做例如上面例子中这些事情。