Thread Lifetime Management: Warnings and Tips

Some caveats and tips on the lifetime of threads in C++ coming our way...

WE'LL COVER THE FOLLOWING ^

- Warnings
- Tips

Warnings



The Challenge of detach: Of course we can use t.detach()

instead of t.join() in the last program. The thread t is not joinable anymore; therefore, its destructor didn't call std::terminate. But now, we have another issue. The program behavior is undefined because the main program may complete before the thread t has time to complete its work package; therefore, its lifetime is too short to display the ID.

Tips



scoped_thread by Anthony Williams

If it's too bothersome to manually take care of the lifetime of our threads, we can encapsulate an std::thread in our own wrapper class. This class should automatically call join in its destructor. Of course, we can go the other way and call detach, but there is an issue with detach.

Anthony Williams created a very useful class and presented it in his

scoped_thread. scoped_thread gets a thread t in its constructor and checks if t is still joinable. If the thread t passed into the constructor is not joinable anymore, there is no need for the scoped_thread. If t is joinable, the constructor calls t.join(). Because the copy constructor and copy assignment operator are declared as delete, instances of scoped_thread can not be copied to or assigned from.

```
// scoped_thread.cpp

#include <iostream>
#include <thread>
#include <utility>

class scoped_thread{
std::thread t;
public:
    explicit scoped_thread(std::thread t_): t(std::move(t_)){
        if (!t.joinable()) throw std::logic_error("No thread");
    }
    ~scoped_thread(){
        t.join();
    }
    scoped_thread(scoped_thread&) = delete;
    scoped_thread& operator=(scoped_thread const &) = delete;
};
```

In the next lesson, we'll discuss how to pass arguments to threads in C++, both by copy and by reference.