Multithreading

Now, we will talk about how C++11 supports and implements multithreaded programming.

C++ gets with the 2011 published C++ standard a multithreading library. This library has basic building blocks like atomic variables, threads, locks, and condition variables. That's the base on which future C++ standards can build higher abstractions. But C++11 already knows tasks, which provide a higher abstraction than the cited basic building blocks.

At a low level, C++11 provides for the first time a memory model and atomic variables. Both components are the foundation for *well-defined* behavior in multithreading programming.

Threads

A new thread in C++ will immediately start its work. It can run in the foreground or background and gets its data by copy or reference.

Shared Variables

The access to shared variables between threads has to be coordinated. This coordination can be done in different ways with mutexes or locks. But often it's sufficient to protect the initialization of the data as it will be immutable during its lifetime.

Thread-Local Variable

Declaring a variable as thread-local ensures that a thread gets its own copy, so there is no conflict.

Condition variables

Condition variables are a classic solution to implement sender-receiver workflows. The key idea is that the sender notifies the receiver when it's done with its work so that the receiver can start.

Tasks

Tasks have a lot in common with threads. But while a programmer explicitly creates a thread, a task will be implicitly created by the C++ runtime. Tasks are like data channels. The promise puts data into the data channel; the future picks the value up. The data can be a value, an exception or simply a notification.

Now, let's talk about how we can use libraries in C++.