Chapter 32. Boost.Asio

Table of Contents

This chapter introduces the library Boost.Asio. Asio stands for asynchronous input/output. This library makes it possible to process data asynchronously. Asynchronous means that when operations are initiated, the initiating program does not need to wait for the operation to end. Instead, Boost.Asio notifies a program when an operation has ended. The advantage is that other operations can be executed concurrently.

Boost.Thread is another library that makes it possible to execute operations concurrently. The difference between Boost.Thread and Boost.Asio is that with Boost.Thread, you access resources inside of a program, and with Boost.Asio, you access resources outside of a program. For example, if you develop a function which needs to run a time-consuming calculation, you can call this function in a thread and make it execute on another CPU core. Threads allow you to access and use CPU cores. From the point of view of your program, CPU cores are an internal resource. If you want to access external resources, you use Boost.Asio.

Network connections are an example of external resources. If data has to be sent or received, a network card is told to execute the operation. For a send operation, the network card gets a pointer to a buffer with the data to send. For a receive operation the network card gets a pointer to a buffer it should fill with the data being received. Since the network card is an external resource for your program, it can execute the operations independently. It only needs time – time you could use in your program to execute other operations. Boost.Asio allows you to use the available devices more efficiently by benefiting from their ability to execute operations concurrently.

Sending and receiving data over a network is implemented as an asynchronous operation in Boost.Asio. Think of an asynchronous operation as a function that immediately returns, but without any result. The result is handed over later.

In the first step, an asynchronous operation is started. In the second step, a program is notified when the asynchronous operation has ended. This separation between starting and ending makes it possible to access external resources without having to call blocking functions.