Talk proposal for

using std::cpp 2025

Cancellations in Asio: a tale of coroutines and timeouts

Rubén Pérez Hidalgo

Boost.Asio is a library to build platform-independent, asynchronous networking applications in C++. It's been along for almost 20 years, but it's still in constant evolution. Asio includes out-of-the-box support for C++20 coroutines, making I/O programs much easier to write. It also features a powerful cancellation mechanism, called per-operation cancellation, that can be used to selectively cancel asynchronous operations. This mechanism is the heart of many powerful high-level features, like timeouts and parallel groups. This talk explores how C++20 coroutines and per-operation cancellation can be used together to solve real-world problems in the asynchronous programming world.

Audience

This talk is relevant for anyone working with C++ code involving **asynchronous operations**, networking and I/O. It can be considered mid-level. I will provide a gentle introduction to Asio and C++20 coroutines so everyone can catch up. The concepts in the talk are applicable to both Boost.Asio and standalone Asio.

The talk has an eminently practical approach. Its ultimate goal is to help the audience understand how they can benefit from per-operation cancellation as library users to solve day to day problems. Attendees without prior Asio experience will be able to follow the talk.

The use case

This talk builds on a simple - yet real world - example: an asynchronous HTTP server that invokes other I/O operations to satisfy client requests. Classic examples of such intermediate operations are database queries and third-party REST API calls.

Handling a request involves multiple I/O operations, and is thus considered a composed asynchronous operation. As the number of intermediate I/O operations grows, expressing the logic using traditional asynchronous programming structures, like callbacks, gets more convoluted. C++20 coroutines are a great tool to keep code clean.

Robust servers should be able to gracefully handle problems like misbehaving clients or network faults. One of the most common requirements is setting timeouts to operations involving I/O. Timeouts can be specified at different levels. Common setups include setting timeouts to individual socket reads and writes, database queries or even to the overall request handling process.

The talk will use a server program that accesses a database to compose its client response. We will use Asio's C++20 coroutine support for logic, and per-operation cancellation to implement timeouts at the session and database operation level. This repository contains a sketch of the code that will be shown during the talk. Please note that this repository is still a work in progress.

Concepts to cover

The talk starts with a brief introduction to asynchronous programming and Asio. We will cover the differences between Boost.Asio and standalone Asio, and mention the higher-level libraries that build on top of it (including Boost.Beast, Boost.MySQL and Boost.Redis).

I will then provide a brief introduction to C++20 coroutines from an I/O perspective, and compare them with callbacks. I will show how to use C++20 coroutines to write a first sketch of the server. This first version will help the audience understand how to create coroutines in Asio and how to use library I/O functions as coroutines. We will briefly cover execution contexts and executors.

Next, I will address how to set timeouts to I/O operations using `asio::cancel_after`, a recent addition to Asio that makes setting timeouts really easy. Since `cancel_after` is a completion token, I will explain completion tokens from a user perspective, and their relation to callbacks and coroutines. If time allows it, I will mention other useful tokens, like `asio::as_tuple`.

We will then learn how to use `asio::cancel_after` to set a timeout to an entire coroutine. I will explain the sequence of events that happen when a cancellation occurs, eventually delving into the per-operation cancellation mechanism. I will cover cancellation signals and slots, and how they propagate with composed asynchronous operations.

Finally, we will cover the three cancellation types (terminal, partial and total), and their relation with the guarantees offered by different asynchronous operations after a cancellation happens. If we have the time, we will end up mentioning other Asio features involving per-operation cancellation, like parallel groups.

About the author

My name is Ruben Perez (https://github.com/anarthal/). I am the author of **Boost.MySQL**, an async MySQL client that follows Asio's universal async model. I've been working with C++ for 8 years in very different environments, including IoT, security and finance.

I had the pleasure of being a speaker at using std::cpp 2024, where I talked about Asio's universal async model (https://github.com/anarthal/usingstdcpp-2024). Unfortunately, due to a technical problem, the talk wasn't recorded. Compared to last year's talk, the proposed one focuses more on C++20 coroutines and covers a different set of concepts.