The Path to Coroutines

A Journey into Asynchronous Programming in C++

May 2024
Reza Jahanbakhshi

Introduction

- Asynchronous programming and its importance in modern software development.
  - Crucial in modern software development
  - Efficient non-blocking operation
  - Efficient utilization of platform resources
  - Improved responsiveness and resource utilization
  - Particularly valuable in scenarios involving I/O-bound operations
  - Handling multiple tasks simultaneously
  - Ensuring smooth user experiences
  - Better scalability
  - Improved overall system performance.
- Coroutines concept and their role in facilitating asynchronous programming in C++.
  - Structured approach
  - Functions that can be suspended and resumed
  - Code that looks synchronous but behaves asynchronously
  - More readable and maintainable code

Traditional Single-Flow Program

- Synchronous program
- Benefits
  - Simple
  - Easy to follow logic (Procedural)
  - Localized
  - Native exception handling
- Limitations
  - Blocking behavior
  - Inefficient

Misusing threads

- Multi-threaded program
- Benefits
  - Simple
  - Easy to follow logic (Procedural)
  - Localized
  - Native exception handling
  - Can serve multiple clients (semi-non-blocking)
- Drawbacks
  - Inefficient utilization of platform resources
  - Limited scalability
  - Measurable high latency for new connections

A Reactor with Active Objects

- Reactor using callbacks
- Benefits
  - Efficient utilization of platform resources
  - Non-blocking
  - Highly scalable
  - Lower latency for new connections (in comparison the thread only approach)
- Drawbacks
  - Requires objects
  - Non-localized storage (intermittent states needs to be kept throughout object lifetime)
  - Non-localized logic (the logic will be spreads over callbacks)
  - Handling exceptions is not straightforward
  - Hard to maintain

Coroutines

- Server using coroutines
- Benefits
  - Simple
  - Easy to follow logic (Procedural)
  - Localized
  - Native exception handling
  - Even more efficient utilization of platform resources (in comparison to the reactor pattern)
  - Non-blocking
  - Highly scalable
  - Lower latency for new connections (in comparison to the thread only approach)
- Drawbacks
  - Relatively new
  - Lack of coroutine support library in the standard library

Thank you for your time.