Coroutines

Vlad Hanciuta, Software Engineer @ Arista Networks

October 2, 2017

Agenda

- What are coroutines?
- Existing implementations
- Coroutines TS
- Examples
- Further reading

What are coroutines?

- Functions that have multiple entry points allowing for suspending and resuming the execution at certain points

What are coroutines?

- Functions that have multiple entry points allowing for suspending and resuming the execution at certain points
- They can be used to implement cooperative multitasking, generators, infinite lists, event loops, state machines

Execution transfer mechanism

- An asymmetric coroutine remembers the invoker and passes the control specifically to it with the `yield` instruction

Execution transfer mechanism

- An asymmetric coroutine remembers the invoker and passes the control specifically to it with the `yield` instruction
- A symmetric coroutine can pass the control to any other coroutine, and it has to be explicitly specified

Stackful vs stackless coroutines

- A stackful coroutine can be suspended from a nested stack frame

Stackful vs stackless coroutines

- A stackful coroutine can be suspended from a nested stack frame
- In stackless coroutine only the top level function can suspend

First-class continuations

- It means that the continuation can be used as value (passed to functions as an argument, stored or returned)

Existing implementations

- Boost.Coroutine - stackful symmetric and asymmetric coroutines
- Boost.Coroutine2 - modernized version, stackful asymmetric coroutines
- CO2 - stackless coroutines

Coroutines TS

- Draft TS N4680
- Supported in Clang 5.0 and Visual Studio 2015
- Stackless coroutines
- It introduces 3 new keywords: `co_return`, `co_yield`, `co_await`

Examples

Further reading

- Coroutines TS
- CppCoro - abstractions on top of coroutines

Questions?