co_routines

A cursory look.

Peter Lorimer

Asynchronous Introduction

- Serial vs. Parallel Computing
- Peter Lorimer
- Single Threaded vs. Multi Threaded
- B.Sc. Computing Science
- Synchronous vs. Asynchronous
- Boulderer @ <3, Software Scientist @ Mind

Native Coroutine Support

- Aikido
- AngelScript
- Ballerina
- BCPL
- Pascal (Borland Turbo Pascal 7.0 with uThreads module)
- BETA
- BLISS
- C++ (Since C++20)
- C# (Since 2.0)
- ChucK
- CLU
- D
- Dynamic C
- Erlang
- F#
- Factor
- GameMonkey Script
- GDScript (Godot's scripting language)
- Go
- Haskell[9][10]
- High Level Assembly[11]
- Icon
- Io
- JavaScript
- Julia[13]
- Kotlin (since 1.1)[14]
- Limbo
- Lua[15]
- Lucid
- µC++
- MiniD
- Modula-2
- Nemerle
- Perl 5 (using the Coro module)
- PHP (with HipHop, native since PHP 5.5)
- Picolisp
- Prolog
- Python (since 2.5,[16] with improved support since 3.3 and with explicit syntax since 3.5[17])
- Raku[18]
- Ruby
- Sather
- Scheme
- Self
- Simula 67
- Smalltalk
- Squirrel
- Stackless Python
- SuperCollider[19]
- Tcl (since 8.6)
- urbiscript

Where’s JAVA???

Coroutines: What?

Coroutines are computer program components that generalize subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed. Coroutines are well-suited for implementing familiar program components such as cooperative tasks, exceptions, event loops, iterators, infinite lists and pipes. --Wikipedia

Coroutines: What?

```
std::future<void> do_something_async() {
    co_await tcp_write(“Hello World”);
    auto val = co_await tcp_read();
}
```

Coroutines: Why?

- Let’s take a walk, through hell. Callback hell.

Synchronous

```
template<typename SocketT>
int foo_handshake(SocketT &socket)
{
    const std::string value{"Hello World"};
    write(socket, value);

    char buffer[11] = {0};
    const size_t size = read(socket, buffer);
    if(std::string(buffer, buffer+read) == "Hello World")
    {
        return 0;
    }
    else
        return -1;

}
```

Callbacks, Callbacks, Callbacks.

```
template <typename SocketT, typename CallbackT>
void foo_handshake_async(SocketT &socket, CallbackT &&on_complete)
{
    const std::string value{"Hello World"};
    async_write(socket, value, [on_complete, socket](size_t written, int ec)
    {
        char buffer[11] = {0};
        async_read(socket,buffer,[on_complete, buffer](size_t read, int ec)
        {
            if(std::string(buffer, buffer+read) == "Hello World")
            {
                on_complete(0);
            }
            else
            {
                on_complete(-1);
            }
        });
    });
}
```

Callbacks, Callbacks, Callbacks.

```
template <typename SocketT, typename CallbackT>
void foo_handshake_async(SocketT &socket, CallbackT &&on_complete)
{
    const std::string value{"Hello World"};
    async_write(socket, value, [on_complete, socket](size_t written, int ec)
    {
        std::shared_ptr<char[]> buffer(new char[11]);
        async_read(socket,buffer.get(),[on_complete, buffer](size_t read, int ec)
        {
            if(std::string(buffer.get(), buffer.get()+read) == "Hello World")
            {
                on_complete(0);
            }
            else
            {
                on_complete(-1);
            }
        });
    });
}
```

Enter co_routines

```
template <typename SocketT>
std::future<int> foo_handshake_coasync(SocketT &socket)
{
    const std::string value{"Hello World"};
    co_await async_write(socket, value);
    char buffer[12] = {0};
    const size_t size = co_await async_read(socket, buffer);
    if(std::string(buffer, buffer+read) == "Hello World")
        co_return 0;
    else
        co_return -1;
}
```

Coroutines: Why?

- Easier to read / understand
- Easier to reason about
- Safer – exceptions & memory

Coroutines: How?

- Coroutines TS
  - New C++ keywords, co_await, co_yield, co_return
  - New types in std::experimental
  - Language features to support coroutines

Keywords

- co_await – await resumption
- co_yield – yields a value from a coroutine
- co_return – returns a value from a coroutine

Main Concepts

- Promise
  - The structure of the coroutine
- Awaitable
  - Controls suspension & resumption of the task

co_await

- Generates a Promise type, based on the coroutine return type
- Gets an awaiter from the expression
  - Directly
  - operator co_await()(expr)
  - Promise::await_transform(expr)
- Calls p.initial_suspend();
- Return object is p.get_return_object() decltype

A simple Coroutine

```
htk::co_task<void> something_async()
{
    std::cout << “This happens” << std::endl;
    co_await std::experimental::suspend_always
{};
    std::cout << “Then this happens” <<
std::endl;
}
```

Promises…

- Defines the behavior of the coroutine eager vs. lazy
- Coroutine payload storage
- Heap-allocated

Promises…

- {awaitable} initial_suspend() -> defines what the coroutine should do initially
- {awaitable} final_suspend() -> defines what the coroutine should do when it finishes
- {coroutine} get_return_object() -> defines the return type for the coroutine i.e. conversion from handle to coroutine. The object that tracks the coroutine.

Promises…

- void return_void() -> called when the coroutine either co_return void or finishes without a co_return
- void return_value({value_type} v) -> called when the coroutine has a co_return statement
- void unhandled_exception() -> called when an exception is thrown during execution

A simple Coroutine

```
htk::co_task<void> something_async()
{
    std::cout << “This happens” << std::endl;
    co_await std::experimental::suspend_always
{};
    std::cout << “Then this happens” <<
std::endl;
}
```

What can we co_await?

- co_await is done on an expression that
  - Is an Awaitable
  - Can be converted to an Awaitable
  - Has operator co_await

Awaitable Concept

- await_ready – the first check to determine coroutine state. Are you ready?
- void await_suspend({coroutine handle} handle) -> called on coroutine suspension
- {type} await_resume() -> called after resumption, chains the result

A simple Awaitable

- Awaitable to resume on background thread.

References & Example

- CPP Reference: Coroutines
- Lewis Baker Async Blog
- Lewis Baker cppcoro Library
- mvpete htk examples