C++ Coroutine

Dian-Lun Lin

Department of Electrical and Computer Engineering

University of Utah, Salt Lake City, UT

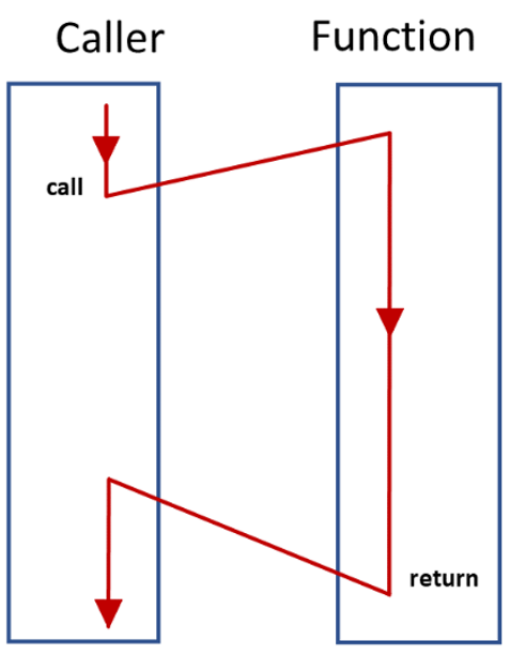
Outline

- What is coroutine
- Why coroutine
- C++ coroutine (rough idea)
  - Coroutine
  - Promise
  - Awaitable
  - Coroutine handle
- An example of scheduler implementation using C++ coroutines

What is Coroutine

suspend and resume!

- A coroutine is a function that can suspend itself and resume by caller
- A “typical” function is a subset of coroutine



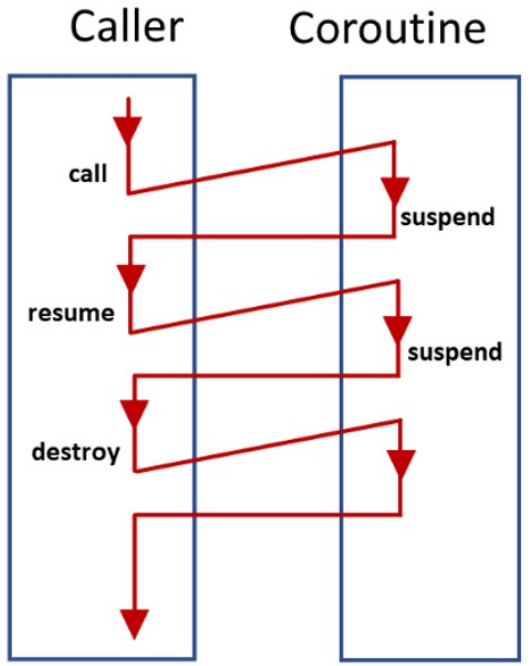


Image credit: Moderns C++

Why Coroutine

- Imaging you want to do two things when you go home…

1. Boil the water

2. Take a shower

Suppose each thing is a function
Suppose you are single… 🙁

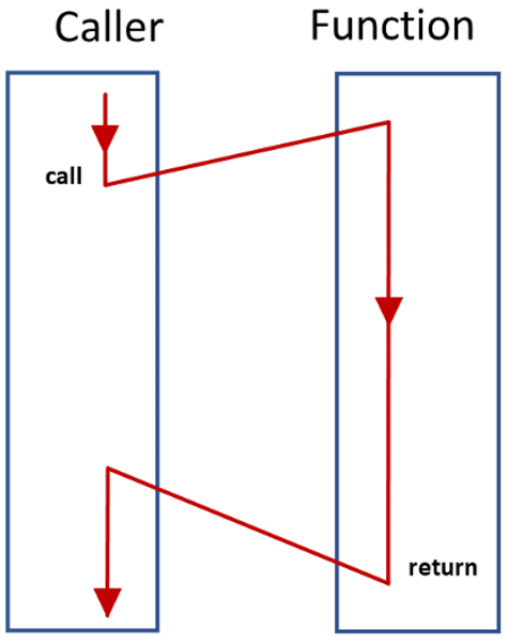


Image credit: Moderns C++

wait, wait, and wait…

Why Coroutine

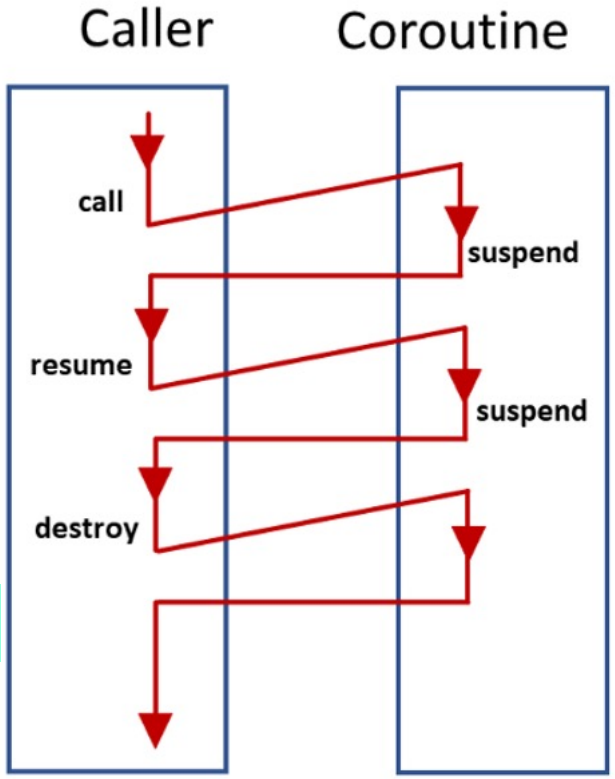
- Imaging you want to do two things when you go home…

1. Boil the water

2. Take a shower

take a shower

enjoy you hot water



turn on the oven

turn off the oven

Image credit: Moderns C++

Why Coroutine

- Coroutine is very useful if you have an ~~oven~~

other computing resource!
GPU, TPU, request data from internet, …

Without coroutine

```cpp
void cpu_work() {
  cpu_matmul(matA, matB, ...);
}

void gpu_work() {
  gpu_matmul<<<8, 256, 0, stream>>>(matA, matB, ...);
  cudaStreamSynchronize(stream);
}


// suppose we only have one CPU thread
// cpu_work and gpu_work are independent to each other
int main() {
  cpu_work();
  gpu_work();

  // or alternatively
  gpu_work();
  cpu_work();
}
```

With coroutine

```cpp
void cpu_work() {

  cpu_matmul(matA, matB, ...);
}

// a coroutine
Coroutine gpu_work() {

  gpu_matmul<<<8, 256, 0, stream>>>(matA, matB, ...);
  while(cudaStreamQuery(stream) != cudaSucess) {
    co_await std::suspend_always;
  }
}

// suppose we only have one CPU thread
// cpu_work and gpu_work are independent to each other
int main() {
  coro = gpu_work();
  cpu_work();
  while(!coro.done()) { coro.resume(); };
}
```

C++ coroutine

- Introduced in C++ 20

- While coroutine concept is simple, C++ coroutine is not easy…
  - Lots of customization points
  - Lack of examples
  - Not that straightforward

- Implementing a C++ coroutine requires:
  - **Coroutine**
  - **Promise**
  - **Awaitable**
  - **Coroutine handle**

Coroutine

- **co_await**
- co_yield
- co_return

```
1
2  void cpu_work() {
3    cpu_matmul(matA, matB, ...);
4  }
5
6  // a coroutine
7  Coroutine gpu_work() {
8
9    gpu_matmul<<<8, 256, 0, stream>>>(matA, matB, ...);
10   while(cudaStreamQuery(stream) != cudaSucess) {
11     co_await;
12   }
13 }
14
15
16 // suppose we only have one CPU thread
17 // cpu_work and gpu_work are independent to each other
18 int main() {
19   coro = gpu_work();
20   cpu_work();
21   coro.resume();
22 }
```

Coroutine

- User needs to define the coroutine class
- A coroutine class needs to define the promise type

Promise defines…

- **The suspension of beginning and end of a coroutine**

- The creation of return object
- Allocation/Deallocation of the stackframe
- Exception handling

I promise this coroutine's behavior is like…

Awaitable

Awaitable controls suspension point behavior

```
 8 Coroutine gpu_work() {
 9
10   gpu_matmul<<<8, 256, 0, stream>>>(matA, matB, ...);
11   while(cudaStreamQuery(stream) != cudaSucess) {
12     co_await std::suspend_always;
13   }
14 }
```

Awaitable

```
1
2  co_await std::suspend_always;
3
4
5  // meaning
6  auto&& awaiter = std::suspend_always;
7
8  if(!awaiter.await_ready()) {
9    awaiter.await_suspend(std::coroutine_handle);
10   // suspend the coroutine here
11 }
12 awaiter.await_resume();
```

Awaitable

Awaitable controls suspension point behavior

```
namespace std {

    struct suspend_never {
        constexpr bool await_ready() const noexcept { return true; }
        constexpr void await_suspend(coroutine_handle<>) const noexcept {}
        constexpr void await_resume() const noexcept {}
    };

    struct suspend_always {
        constexpr bool await_ready() const noexcept { return false; }
        constexpr void await_suspend(coroutine_handle<>) const noexcept {}
        constexpr void await_resume() const noexcept {}
    };

}
```

Promise v.s. Awaitable

- Promise controls coroutine behavior
  - initial_suspend, final_suspend, exception handling, …
- Awaitable controls suspension point behavior
  - co_await std::suspend_always

Coroutine Handle

- Just Like a handle
- You can access promise and coroutine via coroutine handle

- https://en.cppreference.com/w/cpp/coroutine/coroutine_handle

A Scheduler Implementation Example

- Live coding…