C++ONLINE

ANDRÉ BRAND

TALK:

IF WRITING COROUTINES GOES OVER MY HEAD, WHERE DO I EVEN BEGIN WITH DEBUGGING?

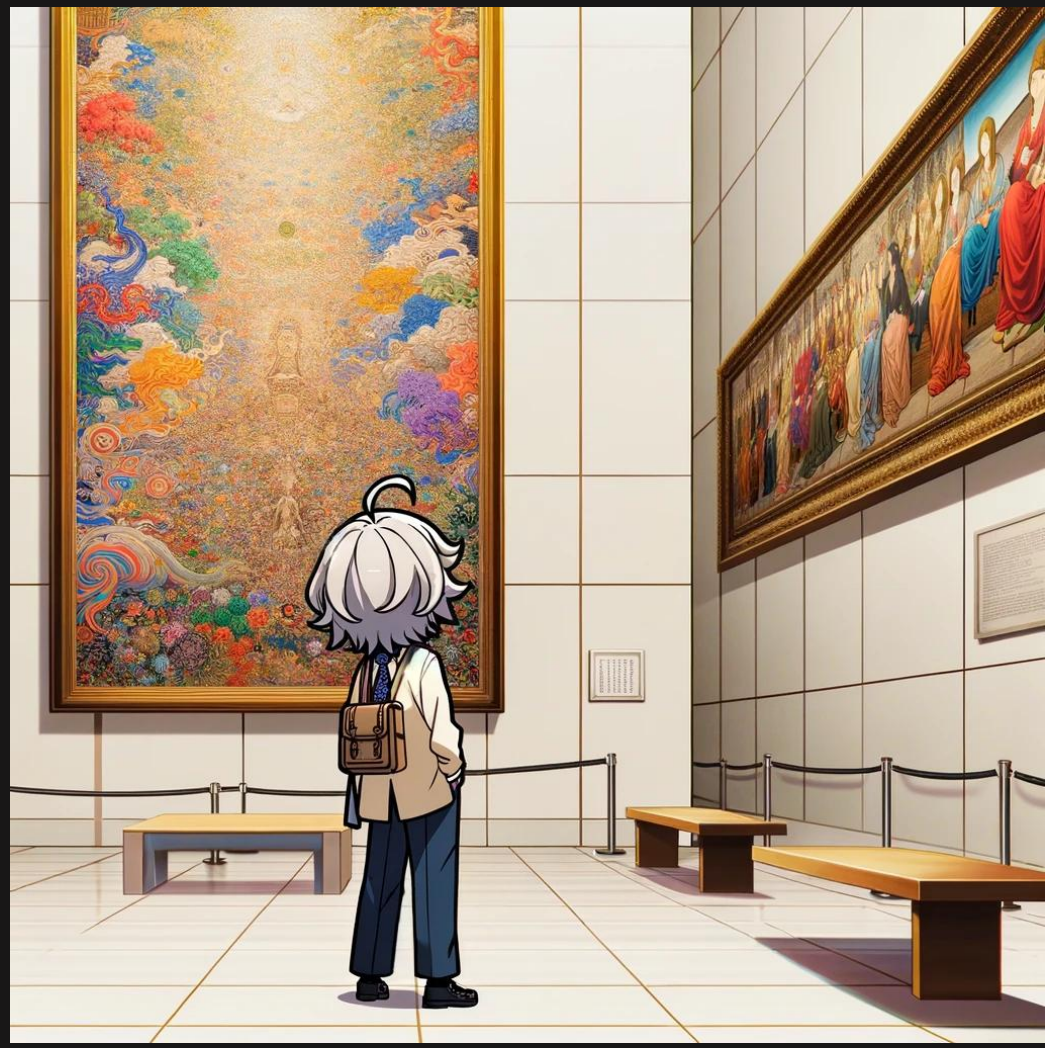
2025

THE BIG PICTURE

Coroutines are computer program components that allow execution to be suspended and resumed, generalizing subroutines for cooperative multitasking.
(Wikipedia)

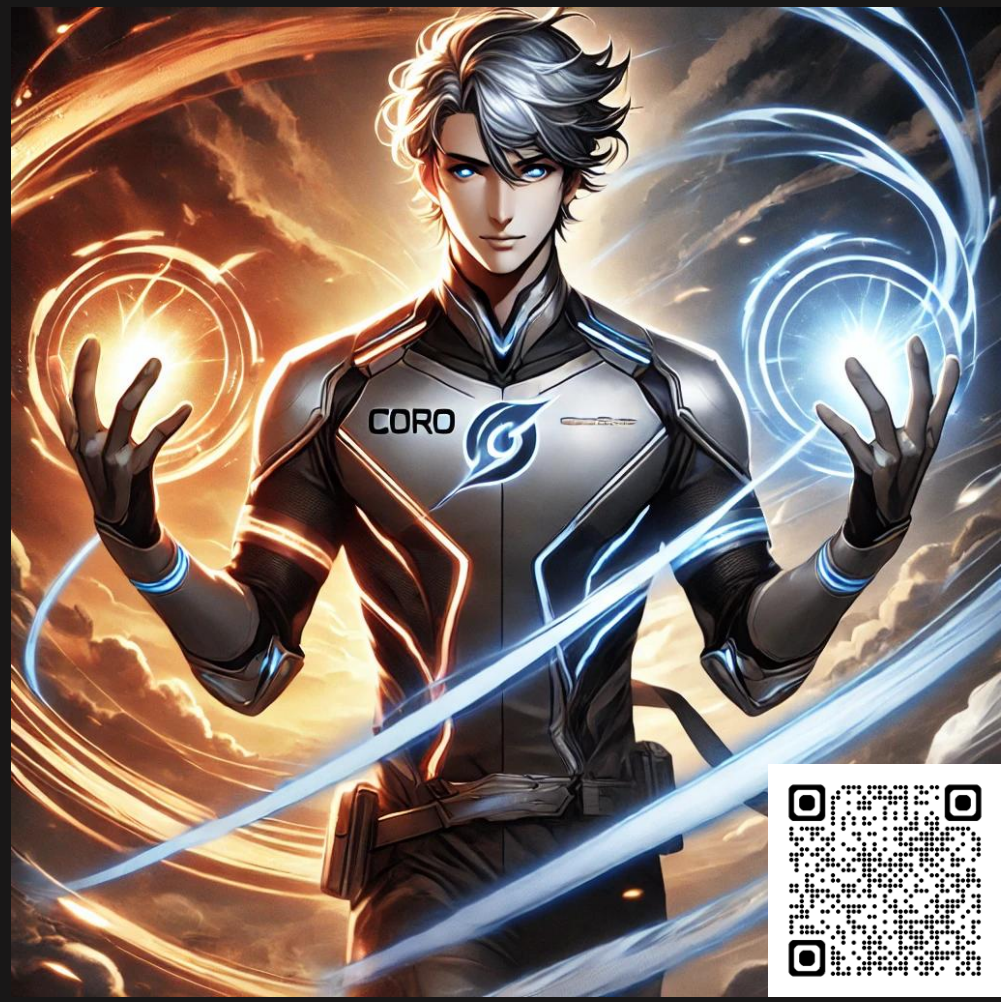
A function is a coroutine if its function-body encloses a coroutine-return-statement an await-expression, or a yield-expression.
[dcl.fct.def.coroutine]

Slightly less formal: (probably not a good fit for standardization)
A coroutine is a function in which you can unlock special powers through the magical **co_await** keyword.



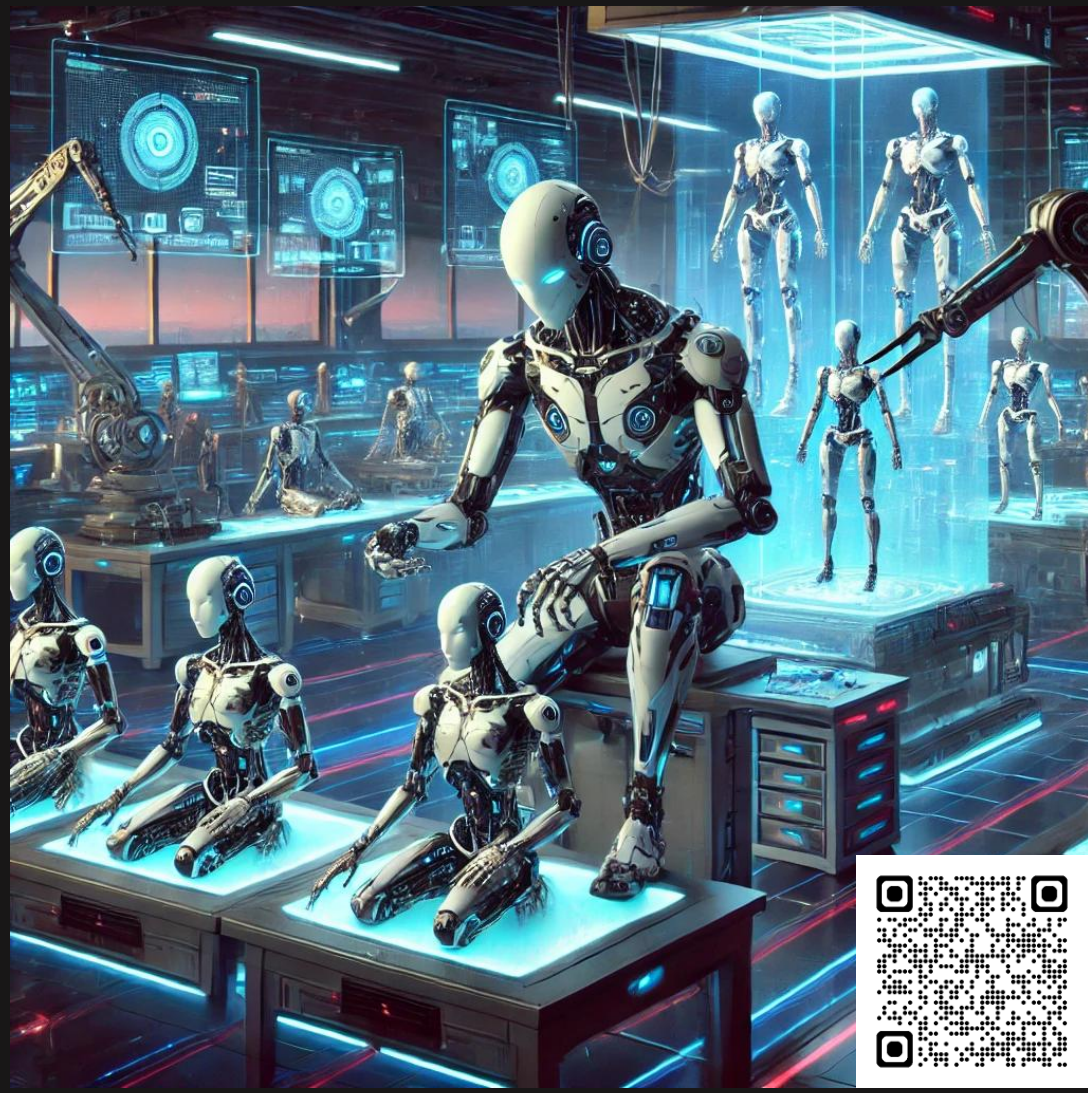
THE TWO SUPERPOWERS OF A COROUTINE

```cpp
namespace ex = stdexec;

exec::task<double> OtherTask(int i) {
  // ...
  co_return 42.;
}

exec::task<int> Loop() {
  auto StopToken = co_await ex::read_env(ex::get_stop_token);
  int i = 0;
  while (!StopToken.stop_requested()) {
    double Result = co_await OtherTask(i);
    // ...
    ++i;
  }
  co_return i;
}
exec::task<void> Timeout(std::chrono::seconds Duration) {
  std::this_thread::sleep_for(Duration);
  co_await ex::just_stopped();
}

int main() {
  // ... [setup scheduler and async_scope]
  ex::sync_wait(ex::starts_on(Scheduler, ex::when_all(Loop(),
                std::move(TimeoutFuture))));
}
```



GENERATOR

```cpp
std::generator<int> fibonacci() {
  int a = 0, b = 1;
  while (true) {
    co_yield a;
    std::swap(a, b);
    b += a;
  }
}

std::generator<int> bar() {
  co_yield 43;
  co_yield 44;
}

std::generator<int> foo() {
  co_yield 42;
  co_yield std::ranges::elements_of(bar());
  auto fib = fibonacci() | std::views::drop(3)
                         | std::views::take(5);
  co_yield std::ranges::elements_of(fib);
  co_yield 42;
  co_yield -1;
}

int main() {
  for (auto i : foo()) std::println("Generated number: {}.", i);
}
```



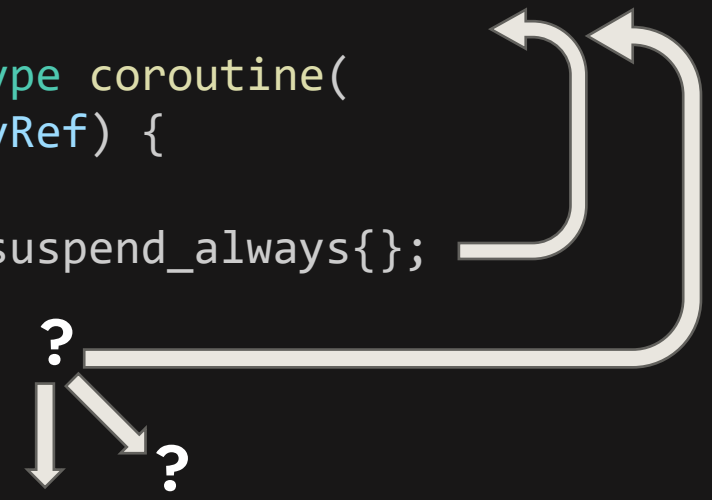
WITHOUT SYNTACTIC SUGAR

```
class Coroutine {
public:
  std::string resume(double x) {
    if (SuspendPoint == 0) {
      SuspendPoint = 1;
      // [code to be called when resumed for the first time]
      return "Maybe resume me again!";
    } else if (SuspendPoint == 1) {
      SuspendPoint = 2;
      // [code to be called when resumed for the second time]
      return "Please don't resume me again";
    }
  }
private:
  int SuspendPoint{};
  // [class data members persist across suspend and resume]
};

int main() {
  auto c = new Coroutine();
  std::cout << c->resume(1.) << '\n';
  std::cout << c->resume(2.) << '\n';
}
```

In a suite of generic async algorithms that are expected to be callable from hot code paths, the extra allocations and indirections are a deal-breaker. It is for these reasons that we consider coroutines a poor choice for a basis of all standard async. (P2300R10, std::execution)

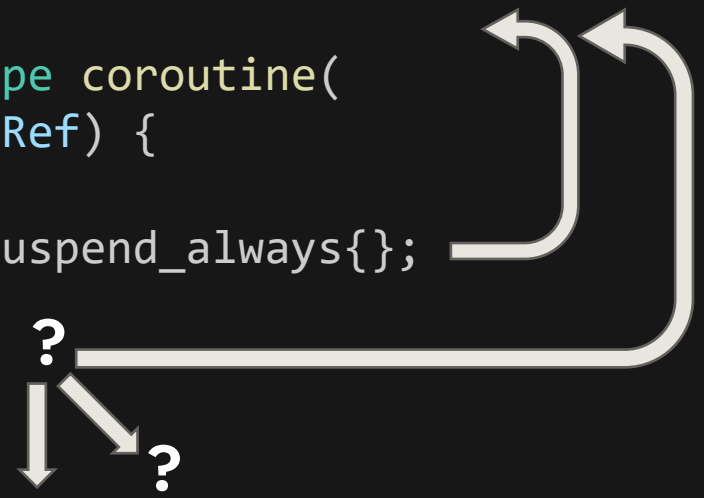
THE BUILDING BLOCKS

```
struct Awaitable {
  bool await_ready() noexcept;
  auto await_suspend(std::coroutine_handle<>);
  auto await_resume() noexcept;
};


struct Foo {};

CoroutineReturnType coroutine(
  int &CapturedByRef) {
  // ...
  co_await std::suspend_always{};
  // ...
  co_await Foo{};
  // ...
}
```

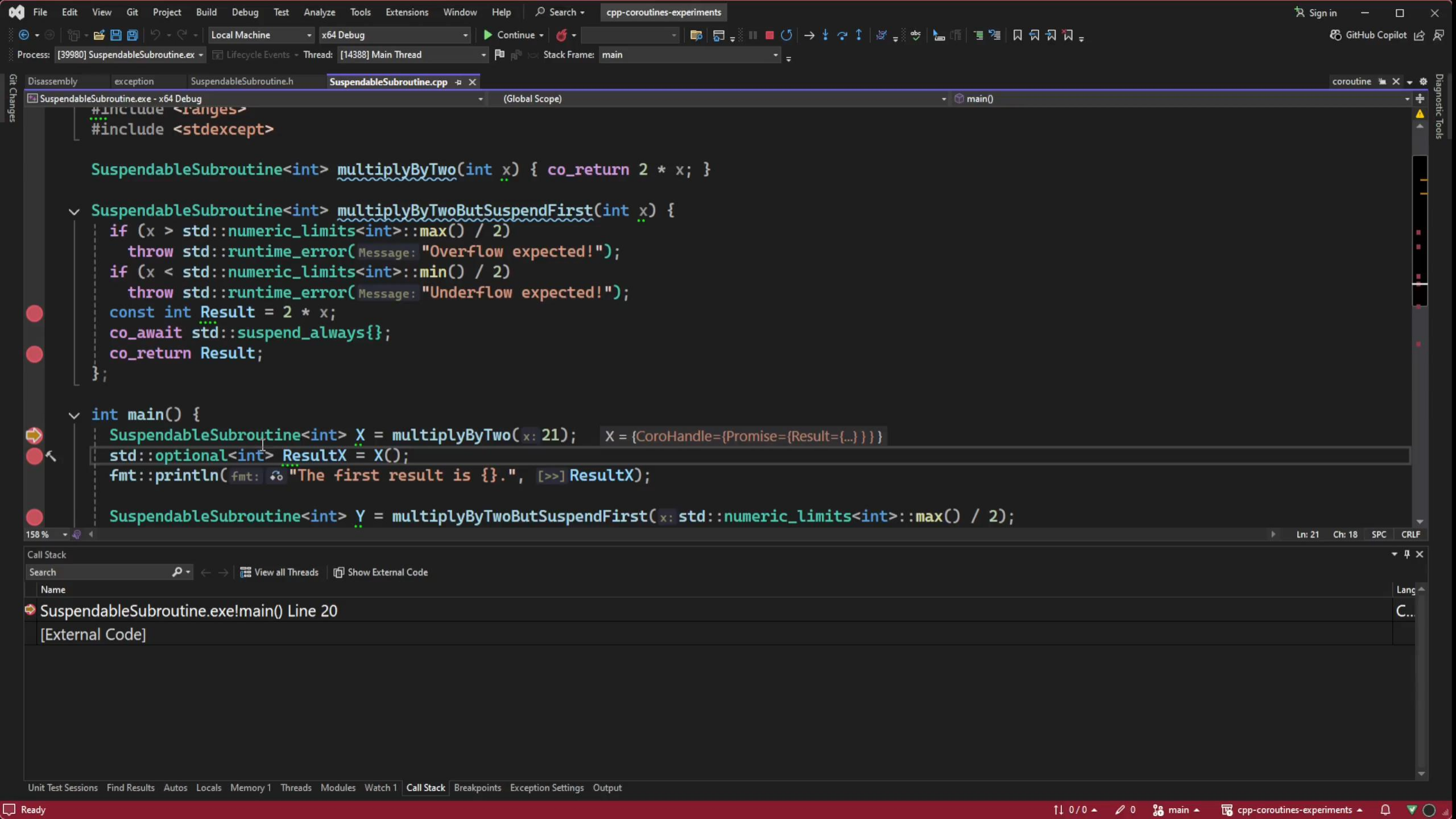


THE BUILDING BLOCKS

```
struct Awaitable {
  bool await_ready() noexcept;
  auto await_suspend(std::coroutine_handle<>);
  auto await_resume() noexcept;
};


struct Foo {};

CoroutineReturnType coroutine(
  int &CapturedByRef) {
  // ...
  co_await std::suspend_always{};
  // ...
  co_await Foo{};
  // ...
}
```




```
struct CoroutineReturnType {
  struct promise_type {
    CoroutineReturnType get_return_object();
    Awaitable initial_suspend();
    Awaitable final_suspend() noexcept;
    void unhandled_exception();
    void return_void();

    // optional
    Awaitable await_transform(Foo);
    // + additional overloads
  };
};
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
cpp-coroutines-experiments
Sign in
Local Machine
x64 Debug
Continue
GitHub Copilot
Process: [39980] SuspendableSubroutine.ex
Lifecycle Events
Thread: [14388] Main Thread
Stack Frame: main
Disassembly
exception
SuspendableSubroutine.h
SuspendableSubroutine.cpp
coroutine
SuspendableSubroutine.exe - x64 Debug
(Global Scope)
main()
#include <ranges>
#include <stdexcept>
SuspendableSubroutine<int> multiplyByTwo(int x) { co_return 2 * x; }
SuspendableSubroutine<int> multiplyByTwoButSuspendFirst(int x) {
if (x > std::numeric_limits<int>::max() / 2)
throw std::runtime_error(Message: "Overflow expected!");
if (x < std::numeric_limits<int>::min() / 2)
throw std::runtime_error(Message: "Underflow expected!");
const int Result = 2 * x;
co_await std::suspend_always{};
co_return Result;
};
int main() {
SuspendableSubroutine<int> X = multiplyByTwo(x: 21);
X = {CoroHandle={Promise={Result={...}}}}
std::optional<int> ResultX = X();
fmt::println(fmt: "The first result is {}.", [>>]ResultX);
SuspendableSubroutine<int> Y = multiplyByTwoButSuspendFirst(x: std::numeric_limits<int>::max() / 2);
158 %
Ln: 21
Ch: 18
SPC
CRLF
Call Stack
Search
View all Threads
Show External Code
Name
Lang
SuspendableSubroutine.exe!main() Line 20
C...
[External Code]
Unit Test Sessions
Find Results
Autos
Locals
Memory 1
Threads
Modules
Watch 1
Call Stack
Breakpoints
Exception Settings
Output
Ready
0 / 0
0
main
cpp-coroutines-experiments

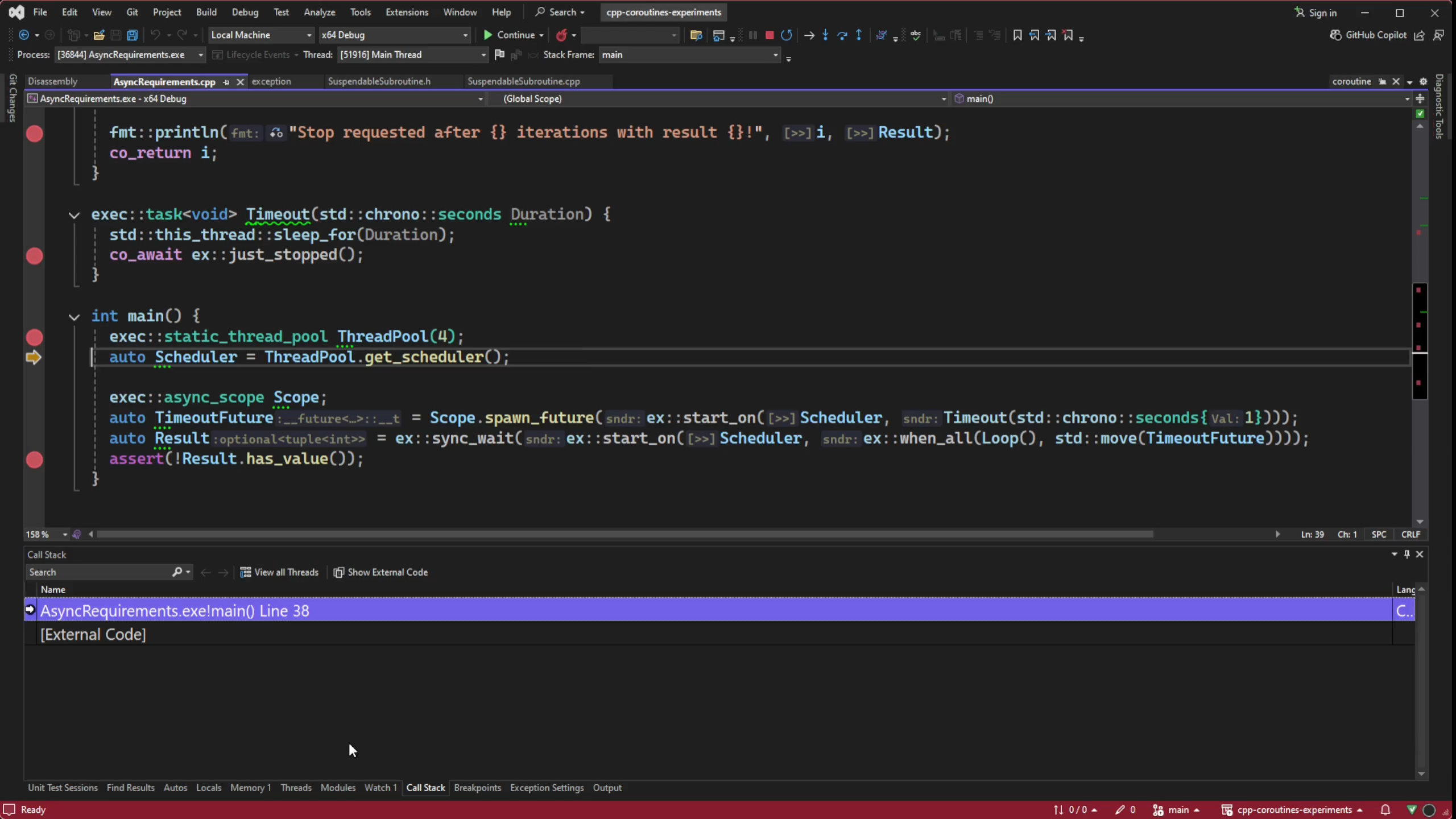
I LIKE BAD CODE

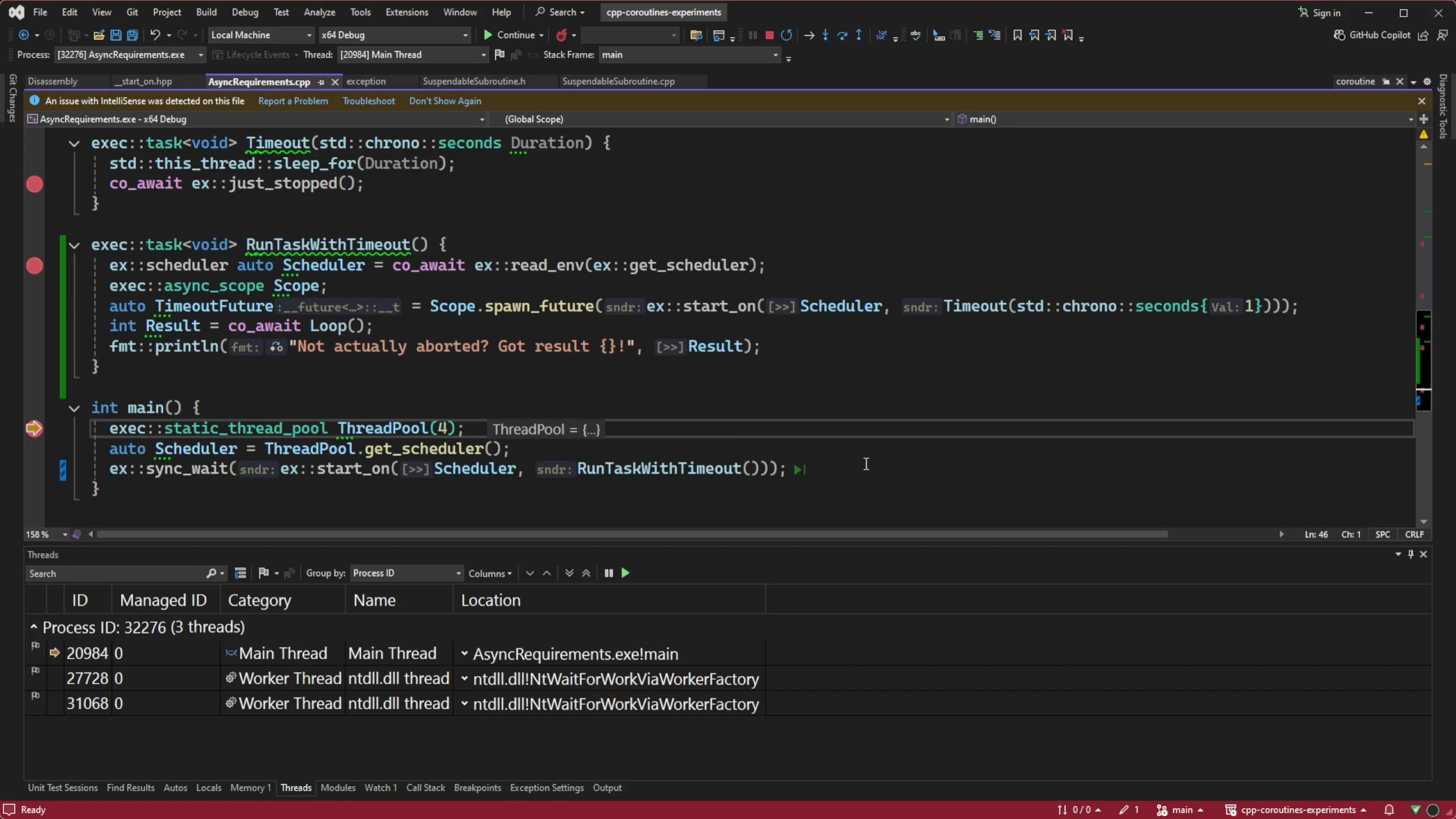
```cpp
#include <generator>

template <std::ranges::input_range First, typename... Args>
std::generator<std::ranges::range_reference_t<First>> concat(First &&R1, Args &&...Rest) {
  co_yield std::ranges::elements_of(std::forward<First>(R1));
  (co_yield std::ranges::elements_of(concat(std::forward<Args>(Rest))), ...);
}
```

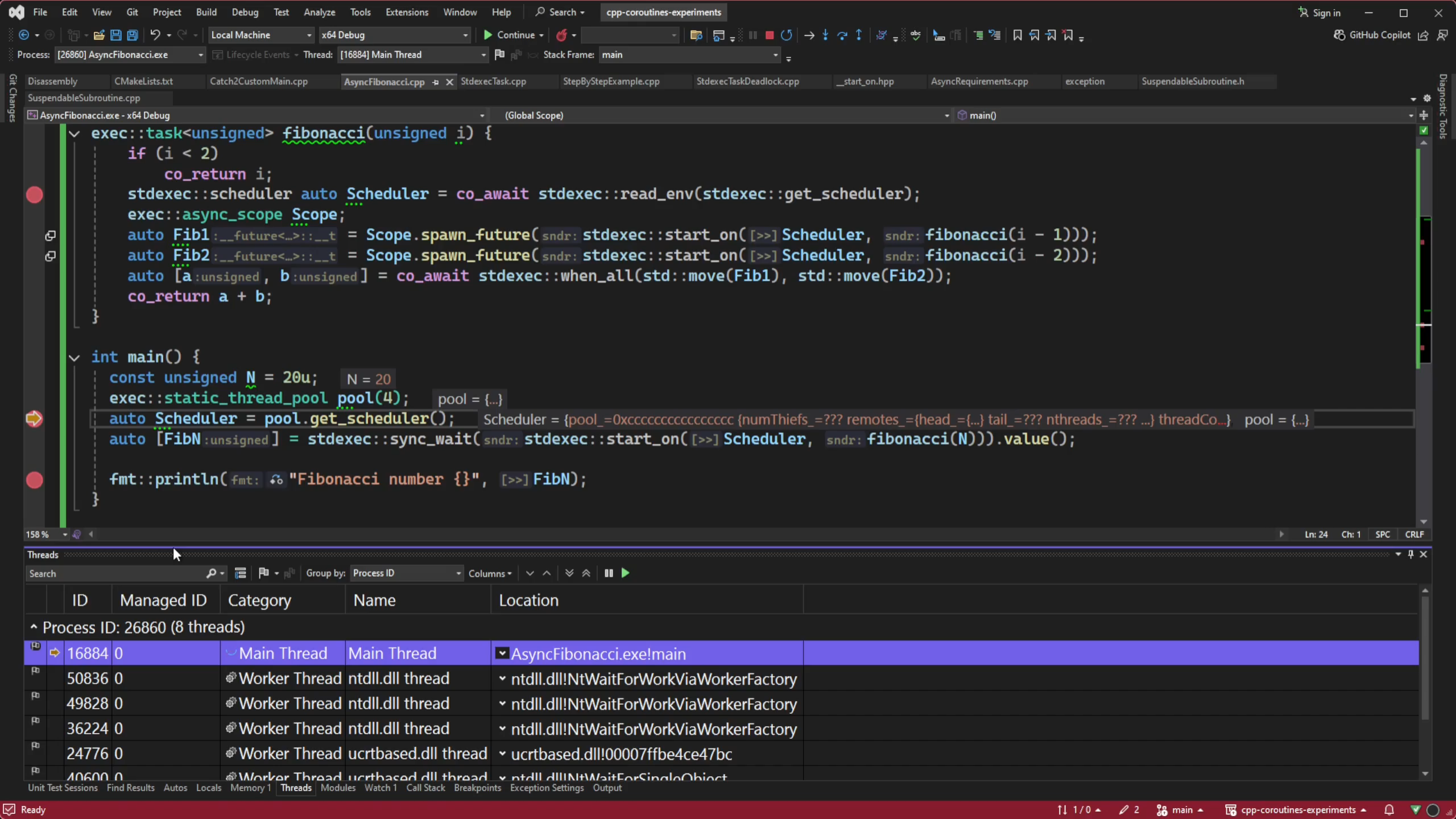
```cpp
#include <print>
#include <vector>

auto getNumbers() {
  std::vector<int> a{1, 2, 3};
  std::vector<int> b{4, 5, 6};
  std::vector<int> c{7, 8, 9};
  return concat(a, b, c);
}
```

```cpp
int main() {
  std::vector<int> a{1, 2, 3};
  std::vector<int> b{4, 5, 6};
  std::vector<int> c{7, 8, 9};
  for (auto &&i : concat(a, b, c))
    std::println("Generated number: {}.", i);

  for (auto &&i : getNumbers())
    std::println("Generated number: {}.", i);
}
```

PARALLELISM VS CONCURRENCY


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
cpp-coroutines-experiments
Sign in
Local Machine
x64 Debug
Continue
GitHub Copilot
Process: [36844] AsyncRequirements.exe
Lifecycle Events
Thread: [51916] Main Thread
Stack Frame: main
Disassembly
AsyncRequirements.cpp
exception
SuspendableSubroutine.h
SuspendableSubroutine.cpp
coroutine
AsyncRequirements.exe - x64 Debug
(Global Scope)
main()
fmt::println(fmt: "Stop requested after {} iterations with result {}!", i, Result);
co_return i;
}
exec::task<void> Timeout(std::chrono::seconds Duration) {
std::this_thread::sleep_for(Duration);
co_await ex::just_stopped();
}
int main() {
exec::static_thread_pool ThreadPool(4);
auto Scheduler = ThreadPool.get_scheduler();
exec::async_scope Scope;
auto TimeoutFuture :__future<...>::__t = Scope.spawn_future(sndr: ex::start_on(Scheduler, sndr: Timeout(std::chrono::seconds{Val: 1})));
auto Result :optional<tuple<int>> = ex::sync_wait(sndr: ex::start_on(Scheduler, sndr: ex::when_all(Loop(), std::move(TimeoutFuture))));
assert(!Result.has_value());
}
158 %
Ln: 39
Ch: 1
SPC
CRLF
Call Stack
Search
View all Threads
Show External Code
Name
Lang
AsyncRequirements.exe!main() Line 38
C...
[External Code]
Unit Test Sessions
Find Results
Autos
Locals
Memory 1
Threads
Modules
Watch 1
Call Stack
Breakpoints
Exception Settings
Output
Ready
0 / 0
0
main
cpp-coroutines-experiments


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
cpp-coroutines-experiments
Sign in
Local Machine
x64 Debug
Continue
GitHub Copilot
Process: [32276] AsyncRequirements.exe
Lifecycle Events
Thread: [20984] Main Thread
Stack Frame: main
Disassembly
_start_on.hpp
AsyncRequirements.cpp
exception
SuspendableSubroutine.h
SuspendableSubroutine.cpp
coroutine
An issue with IntelliSense was detected on this file
Report a Problem
Troubleshoot
Don't Show Again
AsyncRequirements.exe - x64 Debug
(Global Scope)
main()
exec::task<void> Timeout(std::chrono::seconds Duration) {
std::this_thread::sleep_for(Duration);
co_await ex::just_stopped();
}
exec::task<void> RunTaskWithTimeout() {
ex::scheduler auto Scheduler = co_await ex::read_env(ex::get_scheduler);
exec::async_scope Scope;
auto TimeoutFuture :__future<…>::__t = Scope.spawn_future(sndr: ex::start_on([>>] Scheduler, sndr: Timeout(std::chrono::seconds{Val: 1})));
int Result = co_await Loop();
fmt::println(fmt: "Not actually aborted? Got result {}!", [>>] Result);
}
int main() {
exec::static_thread_pool ThreadPool(4);
ThreadPool = {...}
auto Scheduler = ThreadPool.get_scheduler();
ex::sync_wait(sndr: ex::start_on([>>] Scheduler, sndr: RunTaskWithTimeout()));
}
158 %
Ln: 46
Ch: 1
SPC
CRLF
Threads
Search
Group by: Process ID
Columns
ID
Managed ID
Category
Name
Location
Process ID: 32276 (3 threads)
20984 0 Main Thread Main Thread AsyncRequirements.exe!main
27728 0 Worker Thread ntdll.dll thread ntdll.dll!NtWaitForWorkViaWorkerFactory
31068 0 Worker Thread ntdll.dll thread ntdll.dll!NtWaitForWorkViaWorkerFactory
Unit Test Sessions
Find Results
Autos
Locals
Memory 1
Threads
Modules
Watch 1
Call Stack
Breakpoints
Exception Settings
Output
Ready
0 / 0
1
main
cpp-coroutines-experiments


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
cpp-coroutines-experiments
Sign in
Local Machine
x64 Debug
Continue
GitHub Copilot
Process: [26860] AsyncFibonacci.exe
Lifecycle Events
Thread: [16884] Main Thread
Stack Frame: main
Disassembly
CMakeLists.txt
Catch2CustomMain.cpp
AsyncFibonacci.cpp
StdexecTask.cpp
StepByStepExample.cpp
StdexecTaskDeadlock.cpp
_start_on.hpp
AsyncRequirements.cpp
exception
SuspendableSubroutine.h
SuspendableSubroutine.cpp
AsyncFibonacci.exe - x64 Debug
(Global Scope)
main()
exec::task<unsigned> fibonacci(unsigned i) {
if (i < 2)
co_return i;
stdexec::scheduler auto Scheduler = co_await stdexec::read_env(stdexec::get_scheduler);
exec::async_scope Scope;
auto Fib1 :__future<…>::__t = Scope.spawn_future(sndr: stdexec::start_on([>>] Scheduler, sndr: fibonacci(i - 1)));
auto Fib2 :__future<…>::__t = Scope.spawn_future(sndr: stdexec::start_on([>>] Scheduler, sndr: fibonacci(i - 2)));
auto [a :unsigned, b :unsigned] = co_await stdexec::when_all(std::move(Fib1), std::move(Fib2));
co_return a + b;
}
int main() {
const unsigned N = 20u;
N = 20
exec::static_thread_pool pool(4);
pool = {...}
auto Scheduler = pool.get_scheduler();
Scheduler = {pool_=0xcccccccccccccccc {numThiefs_=??? remotes_={head_={...} tail_=??? nthreads_=??? ...} threadCo...}
pool = {...}
auto [FibN :unsigned] = stdexec::sync_wait(sndr: stdexec::start_on([>>] Scheduler, sndr: fibonacci(N))).value();
fmt::println(fmt: "Fibonacci number {}", [>>] FibN);
}
158 %
Ln: 24
Ch: 1
SPC
CRLF
Threads
Search
Group by: Process ID
Columns
ID
Managed ID
Category
Name
Location
Process ID: 26860 (8 threads)
16884 0 Main Thread Main Thread AsyncFibonacci.exe!main
50836 0 Worker Thread ntdll.dll thread ntdll.dll!NtWaitForWorkViaWorkerFactory
49828 0 Worker Thread ntdll.dll thread ntdll.dll!NtWaitForWorkViaWorkerFactory
36224 0 Worker Thread ntdll.dll thread ntdll.dll!NtWaitForWorkViaWorkerFactory
24776 0 Worker Thread ucrtbased.dll thread ucrtbased.dll!00007ffbe4ce47bc
Unit Test Sessions
Find Results
Autos
Locals
Memory 1
Threads
Modules
Watch 1
Call Stack
Breakpoints
Exception Settings
Output
Ready
main
cpp-coroutines-experiments

THANK YOU!