Современные средства разработки ПО

Сопрограммы.

Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru

Сопрограмма
Определение 1

- **Сопрограмма** (корутина, англ. coroutine) — блок кода, который работает по очереди с вызывающим кодом.
  - Выполнение в одном блоке кода приостанавливается в определённой точке, сохраняя полное состояние (включая стек вызовов и счётчик команд), и передаётся управление другому блоку кода, тот, в свою очередь, выполняет задачу и передаёт управление обратно, сохраняя свои стек и счётчик.

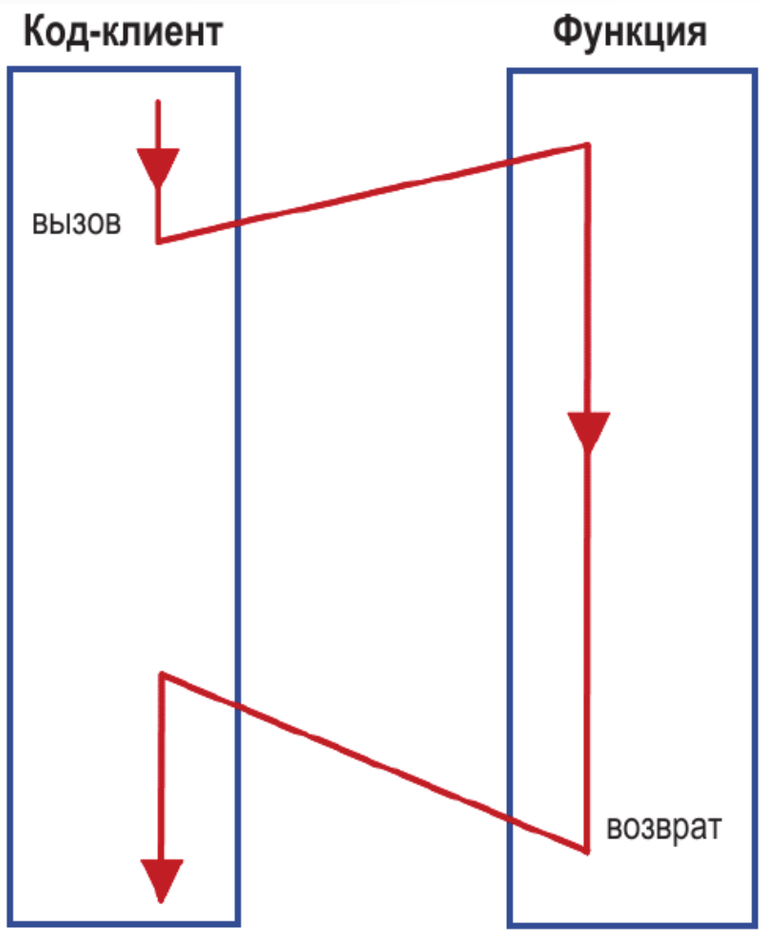
Сопрограмма
Определение 2

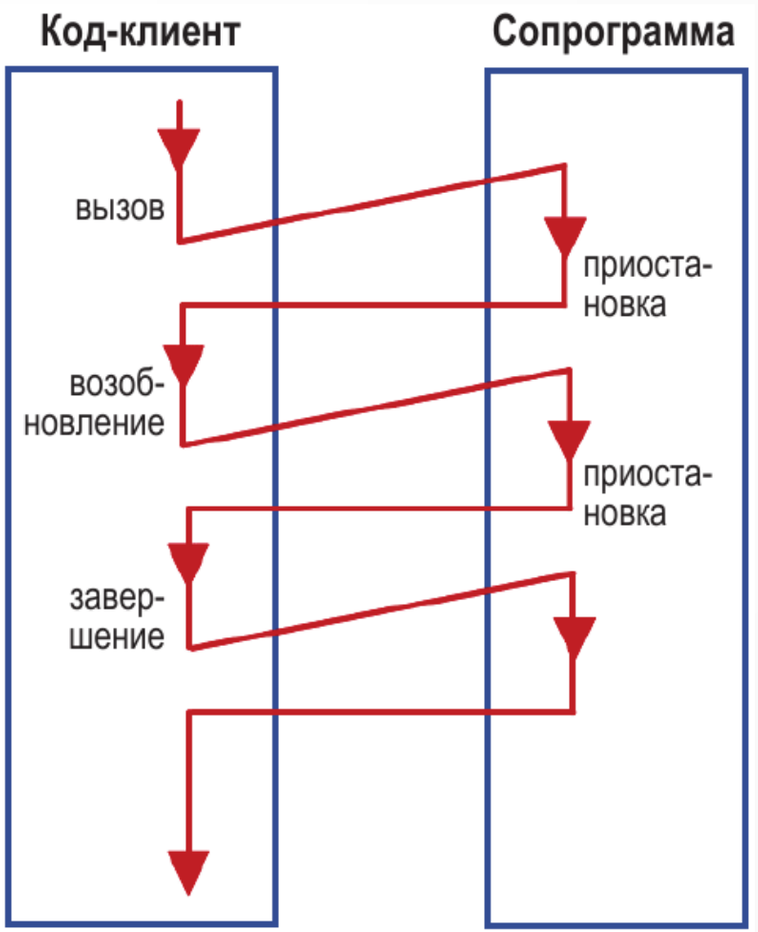
- **Сопрограмма** (корутина, англ. coroutine) — это функция, которая может приостановить выполнение, чтобы возобновить его позже.
  - https://en.cppreference.com/w/cpp/language/coroutines

Сопрограммы являются более гибкими и обобщёнными, чем подпрограммы

- В сравнении с подпрограммой, имеющей всегда одну входную точку,
  - сопрограмма имеет стартовую точку входа и размещённые внутри последовательность возвратов и следующих за ними точек входа.
- Подпрограмма может возвращаться только однажды,
  - сопрограмма может возвращать управление несколько раз.
- Время работы подпрограммы определяется принципом LIFO (последняя вызванная подпрограмма завершается первой),
  - время работы сопрограммы определяется её использованием и необходимостью.

Сопрограмма
Принцип работы





Сопрограммы
История

- Появление понятия сопрограммы относят к конструкции, применённой Мелвином Конвеем в 1958 году в практике программирования на языке ассемблера.
- В 1960-е — 1970-е годы сопрограммы практиковались в некоторых высокоуровневых языках (Клу, Симула, Модула-2).
- Но заметное распространение получили лишь в 2000-е годы, когда появились многочисленные библиотеки поддержки сопрограмм в популярных языках программирования, а в некоторые новые языки (такие как Lua, Ruby, Go, Julia) встроены изначально.

Обозначим операции над сопрограммой следующим образом:

- handle = spawn(СП); — запуск сопрограммы,
- yield; — приостановка текущей сопрограммы,
- resume(handle); — возобновление сопрограммы.

Порядок работы с сопрограммой определяется структурой её кода

```
// СП1          |  // СП2
{               |  {
  f1();         |     g1();
  f2();         |     yield;
  yield;        |     g2();
  f3();         |     g3();
  f4();         |     yield;
  yield;        |     g4();
  f5();         |     g5();
}               |  }
```

```
// Системный поток   |  Выполняемый код
c1 = spawn(СП1);     |  f1();
                     |  f2();
c2 = spawn(СП2);     |        g1();
resume(c1);          |  f3();
                     |  f4();
resume(c2);          |        g2();
                     |        g3();
resume(c1);          |  f5();
resume(c2);          |        g4();
                     |        g5();
```

Использование сопрограмм
Асинхронные операции

- Один из основных сценариев применения сопрограмм — это асинхронные операции, такие как ввод-вывод и анимации в UI.
- После начала асинхронной операции, сопрограмма может сделать **yield** и продолжиться уже после завершения этой операции.
- При этом поток, на котором она выполнялась, не засыпает вместе с сопрограммой, а остается свободен для других сопрограмм.

Использование сопрограмм проще для понимания (пример)

```
// Использование коллбеков

Socket src, dst;
byte buffer[1024];
void copy() {
  async_read(src, buffer, on_read);
}
void on_read(int n) {
  if (n <= 0) return;
  async_write(dst, buffer, n, copy);
}
```

```
// Использование сопрограмм

void copy(Socket src, Socket dst) {
  byte buffer[1024];
  for (;;) {
    int n = await async_read(src, buffer);
    if (n <= 0) return;
    await async_write(dst, buffer, n);
  }
}
```

Во фрагменте кода слева цикл копирования основан на рекурсивном вызове, что не всегда очевидно и плохо читается.

Использование сопрограмм
Генераторы

```
generator<int> fib() {
  int a = 0, b = 1;
  for (;;) {
    yield a;
    a = a + b;
    yield b;
    b = a + b;
  }
}

int main() {
  generator<int> g = fib();
  // печатаем первые 5
  for (int i = 0; i != 5; ++i) {
    g.next();
    print(g.value());
  }
}
```

«Генераторы» генерируют последовательности однотипных объектов, например, последовательности чисел.

Стековые сопрограммы
Определение

- **Стековые (stackful) сопрограммы** — это сопрограммы, у которых есть свой стек (как у потоков).
  - Основное отличие таких сопрограмм от обычных потоков — это то, что они выполняются в том же потоке, что вызывающая сторона и переключаются вручную.
  - По сравнению со скоростью переключения потоков, переключение сопрограмм практически бесплатно (сотни тактов процессора).
  - Однако, из-за того что для каждой сопрограммы надо выделять отдельный стек и прочие служебные структуры данных — их создание и существование не дешевле, чем создание потока.

Безстековые сопрограммы
Определение

- **Безстековые (stackless) сопрограммы** — никак не зависят от операционной системы, и реализуются исключительно средствами компилятора:
  - код сопрограммы переписывается компилятором в объект-конечный автомат, локальные переменные выделяются не на стеке, а становятся членами этого объекта.

Безстековые сопрограммы
Пример преобразования кода

```
// сопрограмма
generator<int> fib() {
  int a = 0, b = 1;
  for (;;) {
    yield a;
    a = a + b;
    yield b;
    b = a + b;
  }
}
```

```
// код, генерируемый компилятором
struct __fib {
  int a, b;
  int __result;
  int __state = 0;
  void next() {
    for (;;) switch (__state) {
    case 0: a = 0;
            b = 1;
    case 3: __result = a;
            __state = 1;
            return;
    case 1: a = a + b;
            __result = b;
            __state = 2;
            return;
    case 2: b = a + b;
            __state = 3;
            break;  // loop
    }
  }
  int value() { return __result; }
};
generator<int> fib() {
  return __fib();
}
```

- В отличие от stackful сопрограмм, в сопрограмме без стека **yield** может быть только в ее теле. Нельзя сделать **yield** из функции, вызванной сопрограммой.
- *Создание безстековой сопрограммы — это создание объекта, хранящего ее состояние.*

Сопрограммы
Применение

- Конечные автоматы в рамках одной подпрограммы,
  - где состояние определяется текущей точкой входа / выхода процедуры; это может привести к более удобочитаемому коду по сравнению с использованием goto, а также может быть реализовано посредством взаимной рекурсии с конечными вызовами.
- Акторная модель параллелизма, например, в видеоиграх.
  - У каждого участника есть свои собственные процедуры (это снова логически разделяет код), но они добровольно передают управление центральному планировщику, который выполняет их последовательно (это форма совместной многозадачности).
- Генераторы,
  - которые могут быть полезны для потоков – особенно ввода / вывода – и для общего обхода структур данных.
- Передача последовательных процессов, где каждый подпроцесс является сопрограммой.
  - Операции ввода / вывода канала и блокировки приводят к сопрограммам, и планировщик разблокирует их при событиях завершения.
  - В качестве альтернативы, каждый подпроцесс может быть родительским для следующего за ним в конвейере данных (или предшествующего ему, и в этом случае шаблон может быть выражен как вложенные генераторы).
- Обратная связь, обычно используемая в математическом программном обеспечении,
  - в которой такая процедура, как решатель, интегральный вычислитель, нуждается в процессе использования для выполнения вычислений, таких как оценка уравнения или подынтегрального выражения.

Реализация в C++20
Реализованы безстековые сопрограммы

- В C++20 реализованы безстековые сопрограммы.

Реализация в C++20
Определение сопрограммы

- Функция является сопрограммой, если её определение содержит одно из следующих новых ключевых слов:
  - использует оператор **co_await** в вызывающем коде для приостановки выполнения до его возобновления;
  - использует ключевое слово **co_yield**, чтобы приостановить выполнение, возвращая значение;
  - использует ключевое слово **co_return** для завершения выполнения, возвращая значение.

Реализация в C++20
Составные части

- **Обещающий объект (promise)**, управляемый изнутри сопрограммы. Сопрограмма отправляет свой результат или исключение через этот объект.
- **Дескриптор сопрограммы**, управляемый из-за пределов сопрограммы. Этот дескриптор, не являющийся владельцем, используемый для возобновления выполнения сопрограммы или для уничтожения фрейма сопрограммы.
- **Состояние сопрограммы**, которое является внутренним, выделенным в куче (если выделение не оптимизировано), объектом, который содержит:
  - обещающий объект;
  - параметры (все копируются по значению);
  - некоторое представление текущей точки приостановки, так что восстановление знает, где продолжить, а деструктор знает, какие локальные переменные были в области видимости;
  - локальные переменные и временные объекты, время жизни которых охватывает текущую точку приостановки.

Дополнительные источники по сопрограммам

- C++ reference
  - https://en.cppreference.com/w/cpp/language/coroutines
- Библиотека cppcoro от Льюиса Бейкера (Lewis Baker)
  - https://github.com/lewissbaker/cppcoro
- C++20: корутины cppcoro
  - https://habr.com/ru/company/otus/blog/672838/
- David Mazières, 2021 — Учебник по сопрограммам C++20
  - https://www.scs.stanford.edu/~dm/blog/c++-coroutines.html
- The C++ Asynchronous Framework
  - https://userver.tech/

Примеры кода

- L05/MoveToAnotherThread.cpp
- L05/Generator.cpp

Вопросы?

Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru