

C++

# Урок 27

# **Многопоточность ч 1**

# multithreading

**Многозадачность (multitasking)** — свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких задач.

**Многопоточность (multithreading)** — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть **без предписанного порядка во времени**. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

# multithreading

**Процесс** — экземпляр программы во время выполнения.

**Потоки** — ветви кода, выполняющиеся «параллельно», то есть без предписанного порядка во времени.

# multithreading

## Состояние потока:

- Выполняемый (Executing) — поток, который выполняется в текущий момент на процессоре.
- Готовый (Runnable) — поток ждет получения кванта времени и готов выполнять назначенные ему инструкции. Планировщик выбирает следующий поток для выполнения только из готовых потоков.
- Ожидающий (Waiting) — работа потока заблокирована в ожидании блокирующей операции.

# multithreading

## Применение

zoom

- Многопоточность широко используется в приложениях с пользовательским интерфейсом. В этом случае за работу интерфейса отвечает один поток, а какие-либо вычисления выполняются в других потоках. Это позволяет пользовательскому интерфейсу не подвисать, когда приложение занято другими вычислениями.
- Многие алгоритмы легко разбиваются на независимые подзадачи, которые можно выполнять в разных потоках для повышения производительности. Например, при фильтрации изображения разные потоки могут заниматься фильтрацией разных частей изображения.
- Если некоторые части приложения вынуждены ждать ответа от сервера/пользователя/устройства, то эти операции можно выделить в отдельный поток, чтобы в основном потоке можно было продолжать работу, пока другой поток ждёт ответа.



# multithreading

## Достоинства

- Упрощение программы в некоторых случаях, за счёт вынесения механизмов чередования выполнения различных слабо взаимосвязанных подзадач, требующих одновременного выполнения, в отдельную подсистему многопоточности.
- Повышение производительности процесса за счёт распараллеливания процессорных вычислений и операций ввода-вывода.
- Меньшие относительно процесса временные затраты на создание потока.

# multithreading

## Недостатки

- Состояние гонки (race condition)
- Взаимная блокировка (deadlock)
- Голодание потоков — это ситуация, в которой поток не может получить доступ к общим ресурсам, потому что на эти ресурсы всегда претендуют какие-то другие потоки, которым отдаётся предпочтение.



# multithreading

Пришло из boost в C++11

## Потоки в C++

По умолчанию в C++ есть один поток — **поток**, выполняющий функцию **main()**.

Программа может запустить дополнительные потоки, точкой входа в которые служит другая функция. После чего эти потоки и начальный поток выполняются **одновременно**.

Аналогично завершению программы при выходе из `main()` основной поток завершается при возвращении из функции, указанной в качестве точки входа.

**Дополнительный поток** должен завершаться **до завершения** основного потока `main` или открепляться и завершаться самостоятельно.

# multithreading

## Потоки в C++

### Member classes

**id** represents the *id* of a thread  
(public member class)

### Member functions

(constructor)	constructs new thread object (public member function)
(destructor)	destructs the thread object, underlying thread must be joined or detached (public member function)
<b>operator=</b>	moves the thread object (public member function)

### Observers

<b>joinable</b>	checks whether the thread is joinable, i.e. potentially running in parallel context (public member function)
<b>get_id</b>	returns the <i>id</i> of the thread (public member function)
<b>native_handle</b>	returns the underlying implementation-defined thread handle (public member function)
<b>hardware_concurrency</b> [static]	returns the number of concurrent threads supported by the implementation (public static member function)

### Operations

<b>join</b>	waits for the thread to finish its execution (public member function)
<b>detach</b>	permits the thread to execute independently from the thread handle (public member function)
<b>swap</b>	swaps two thread objects (public member function)

<https://en.cppreference.com/w/cpp/thread/thread>

# multithreading

## Потоки в C++

- Объект класса представляет собой **один** поток выполнения.
- Новый поток начинает выполнение сразу же после построения объекта `std::thread`. Выполнение начинается с функции верхнего уровня, которая передаётся в качестве аргумента в конструктор `std::thread`.
- Передать возвращаемое значение или исключение из нового потока наружу можно через `std::promise` или через глобальные переменные (работа с которыми потребует синхронизации)
- Объекты `std::thread` также могут быть не связаны ни с каким потоком (после default construction, move from, detach или join), и поток выполнения может быть не связан ни с каким объектом `std::thread` (после detach).
- Никакие два объекта `std::thread` не могут представлять один и тот же поток выполнения; `std::thread` нельзя копировать (не является CopyConstructible или CopyAssignable), но можно перемещать (является MoveConstructible и MoveAssignable).

# multithreading

## Потоки в C++

- Поток – объект
- В конструктор потока можно передать функтор.

# multithreading

## Методы текущего потока

- `std::this_thread::yield()` подсказывает планировщику потоков перепланировать выполнение, приостановив текущий поток и отдав преимущество другим потокам.
- `std::this_thread::get_id()` работает аналогично `std::thread::get_id()`.
- `std::this_thread::sleep_for(sleep_duration)` блокирует выполнение текущего потока на время `sleep_duration`.
- `std::this_thread::sleep_until(sleep_time)` блокирует выполнение текущего потока до наступления момента времени `sleep_time`.

# multithreading

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void printHello()
{
    cout << "Beginin of printHello thread" << endl;
    cout << "printHello_ID - " << this_thread::get_id() << endl;
    cout << "End of printHello thread" << endl;
}

void Hello()
{
    cout << "Beginin of Hello thread" << endl;
    cout << "Hello_ID - " << this_thread::get_id() << endl;
    cout << "End of Hello thread" << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    thread first_th(Hello); // создание потока first_th и передача в него ф-ии
    thread second_th(printHello); // создание потока second_th и передача в него ф-ии

    int a = 42;
    a++;

    printHello();

    first_th.join(); // дожидается закрытия потока
    second_th.join();

    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

```
Начало работы потока системы - 0x2045d2500
Beginin of printHello thread
printHello_ID - 0x2045d2500
End of printHello thread
Beginin of Hello thread
Hello_ID - 0x16f137000
End of Hello thread
Beginin of printHello thread
printHello_ID - 0x16f1c3000
End of printHello thread
Конец работы потока системы - 0x2045d2500
```

# multithreading

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void printHello()
{
    cout << "Beginin of printHello thread" << endl;
    cout << "printHello_ID - " << this_thread::get_id() << endl;
    cout << "End of printHello thread" << endl;
}

void Hello()
{
    cout << "Beginin of Hello thread" << endl;
    cout << "Hello_ID - " << this_thread::get_id() << endl;
    cout << "End of Hello thread" << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    thread first_th(Hello); // создание потока first_th и передача в него ф-ии
    thread second_th(printHello); // создание потока second_th и передача в него ф-ии

    int a = 42;
    a++;

    printHello();

    first_th.join(); // дожидается закрытия потока
    second_th.join();

    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

```
Начало работы потока системы - 0x2045d2500
Beginin of printHello thread
printHello_ID - 0x2045d2500
End of printHello thread
Beginin of Hello thread
Hello_ID - 0x16f137000
End of Hello thread
Beginin of printHello thread
printHello_ID - 0x16f1c3000
End of printHello thread
Конец работы потока системы - 0x2045d2500
```

# multithreading

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void printHello()
{
    cout << "Beginin of printHello thread" << endl;
    cout << "printHello_ID - " << this_thread::get_id() << endl;
    cout << "End of printHello thread" << endl;
}

void Hello()
{
    cout << "Beginin of Hello thread" << endl;
    cout << "Hello_ID - " << this_thread::get_id() << endl;
    cout << "End of Hello thread" << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    thread first_th(Hello); // создание потока first_th и передача в него ф-ии
    thread second_th(printHello); // создание потока second_th и передача в него ф-ии

    int a = 42;
    a++;

    printHello();

    first_th.join(); // дожидается закрытия потока
    second_th.join();

    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

Начало работы потока системы - 0x2045d2500  
Beginin of printHello thread  
printHello\_ID - 0x2045d2500  
End of printHello thread  
Beginin of Hello thread  
Hello\_ID - 0x16f137000  
End of Hello thread  
Beginin of printHello thread  
printHello\_ID - 0x16f1c3000  
End of printHello thread  
Конец работы потока системы - 0x2045d2500



# multithreading

Т.к. конструктор потока принимает функтор, то логичны и такие конструкции

```
thread three([]()  
    { cout << "\n\nFunctor thread: " << this_thread::get_id() << "\n"  
      << endl; });
```

# multithreading

Использование `.detach()` позволяет “откреплять” поток

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void printHello()
{
    cout << "Beginin of printHello thread" << endl;
    cout << "printHello_ID - " << this_thread::get_id() << endl;
    cout << "End of printHello thread" << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    thread second_th(printHello); // создание потока second_th и передача в него ф-ии
    printHello();

    thread three([]()
    { cout << "\n\nFuncionr thread: " << this_thread::get_id() << "\n"
      << endl; });

    second_th.join();
    three.detach(); // запуск потока в фоновом режиме

    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

```
Начало работы потока системы - 0x2045d2500
Beginin of printHello thread
printHello_ID - 0x2045d2500
End of printHello thread
Beginin of printHello thread
printHello_ID - 0x16b2e7000
End of printHello thread

Funcionr thread: Конец работы потока системы - 0x16b373000
0x2045d2500
```

# multithreading

Запустили процесс, но не следим за его выполнением

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void printHello()
{
    cout << "Beginin of printHello thread" << endl;
    cout << "printHello_ID - " << this_thread::get_id() << endl;
    cout << "End of printHello thread" << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    thread second_th(printHello); // создание потока second_th и передача в него ф-ии
    printHello();

    thread three([]()
    {
        int i = 100000, sum = 0;
        while (i--);
        cout << "\n\nFunctor thread: " << this_thread::get_id() << "\n"
            << endl; });

    second_th.join();
    three.detach(); // запуск потока в фоновом режиме

    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

```
Начало работы потока системы - 0x2045d2500
Beginin of printHello thread
printHello_ID - 0x2045d2500
End of printHello thread
Beginin of printHello thread
printHello_ID - 0x16ba2b000
End of printHello thread
Конец работы потока системы - 0x2045d2500
```

# multithreading

## Передача переменных в поток

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void show_val(int &val)
{
    cout << val << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    int temp = 42;
    thread first_th(show_val(temp));

    first_th.join();
    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

Скомпилируется программа?

# multithreading

## Передача переменных в поток

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void show_val(int &val)
{
    cout << val << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    int temp = 42;
    thread first_th(show_val(temp));

    first_th.join();
    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

```
t << endl;

void show_val(int &val)
no instance of constructor "std::_1::thread::thread" matches the argument list C/C++
(289)
Начало
27_03.cpp(15, 21): argument types are: (void)
42; View Problem (\CF8) Quick Fix... (#)
st_th(show_val(temp));
join();
```

Скомпилируется программа? **НЕТ**

# multithreading

## Передача переменных в поток

```
cout << "Начало работы потока системы: " << this_thread::get_id() << endl;
int temp = 42;
thread first_th(show_val, temp, &temp);
```

thread<\_Fp, \_Args..., <unnamed>>(\_Fp &&\_\_f, \_Args  
&&...\_\_args)

Передача параметров

# multithreading

## Передача переменных в поток

```
cout << "Начало работы потока системы - " << this_thread::get_id() << endl;
int temp = 42;
thread first_th(show_val, temp,);
```

```
#include <iostream>
#include <thread> // библиотека C++
using namespace std;

void show_val(int &val)
{
    cout << val << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    int temp = 42;
    thread first_th(show_val, ref(temp));

    first_th.join();
    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

Передача параметров

# Задача

**Задача:** реализуйте функцию, в которую поступает 2 целочисленных значения `from` и `to`, где `from` – число, с которого мы начинаем подсчет, а `to` – к которому нужно прийти.

Требуется дойти до числа `to` инкрементируя или декрементируя число `to`. Запустите ф-ию в своем потоке.



# multithreading

```
#include <iostream>
#include <thread> // библиотека C++
#include <chrono>

using namespace std;

void repeat_untill(int &val1, int &val2)
{
    int cnt = 0;
    while (val1 != val2)
    {
        (val1 > val2) ? val1-- : val1++;
        // остановка потока на 1000 миллисекунд
        cout << val1 << endl;
        this_thread::sleep_for(chrono::milliseconds(1000));

        cnt++;
    }
    cout << "Всего " << cnt << " итераций" << endl;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    int from, to;
    cin >> from >> to;

    thread first_th(repeat_untill, ref(from), ref(to));

    first_th.join();
    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

Листинг кода

# Задача

**Задача:** создайте второй поток, который увеличивает значение переменной `from` в 2 раза во время работы первого потока

# multithreading

```
#include <iostream>
#include <thread> // библиотека C++
#include <chrono>

using namespace std;

void repeat_untill(int &val1, int &val2)
{
    int cnt = 0;
    while (val1 != val2)
    {
        (val1 > val2) ? val1-- : val1++;
        // остановка потока на 1000 миллисекунд
        cout << val1 << endl;
        this_thread::sleep_for(chrono::milliseconds(1000));

        cnt++;
    }
    cout << "Всего " << cnt << " итераций" << endl;
}

// задача: создать второй поток, который удваивает значение переменной from в 2 ра
// за во время работы первого потока

void change_val(int &val)
{
    val <= 1;
}

int main()
{
    cout << "Начало работы потока системы - " << this_thread::get_id() << endl;

    int from, to;
    cin >> from >> to;

    thread first_th(repeat_untill, ref(from), ref(to));
    thread sec_th(change_val, ref(from));

    first_th.join();
    sec_th.join();
    cout << "Конец работы потока системы - " << this_thread::get_id() << endl;
}
```

Листинг кода

# Задача

**Задача:** создайте второй поток, который 4 раза увеличивает значение переменной `from` в 2 раза каждые 2 секунды.

A hand-drawn blue oval frame with a slightly irregular, sketchy border, centered on the page. The word "Mutex" is written inside this frame.

**Mutex**

# Mutex

## Определение

**Мьютекс** («взаимное исключение») — это базовый механизм синхронизации. Он предназначен для организации взаимоисключающего доступа к общим данным для нескольких потоков с использованием барьеров памяти.

## Идея

В программе наступает момент **барьерной синхронизации** (построение потоков). Для этого построения и нужен Mutex.

Mutex = регулировщик, который в определенный момент поднимает ключ и говорит "стоять" остальным потокам. Как только поток завершил свое действие, он сообщает регулировщику, что остальные потоки могут продолжать.

Mutex – объект для синхронизации потоков.

# Mutex

## Основные действия:

- Объявление | `std::mutex mutex_name;`
- Захват мьютекса | `mutex_name.lock();`  
Поток запрашивает монопольное использование общих данных, защищаемых мьютексом. Далее два варианта развития событий: происходит захват мьютекса этим потоком (и в этом случае ни один другой поток не сможет получить доступ к этим данным) или поток блокируется (если мьютекс уже захвачен другим потоком).
- Метод `try_lock` пытается получить права владения мьютексом без блокировки. Его возвращаемое значение можно преобразовать в `bool` и оно является `true`, если метод получает права владения; в противном случае — `false`.
- Освобождение мьютекса | `mutex_name.unlock();`  
Когда ресурс больше не нужен, текущий владелец должен вызвать функцию разблокирования `unlock`, чтобы и другие потоки могли получить доступ к этому ресурсу. Когда мьютекс освобождается, доступ предоставляется одному из ожидающих потоков.