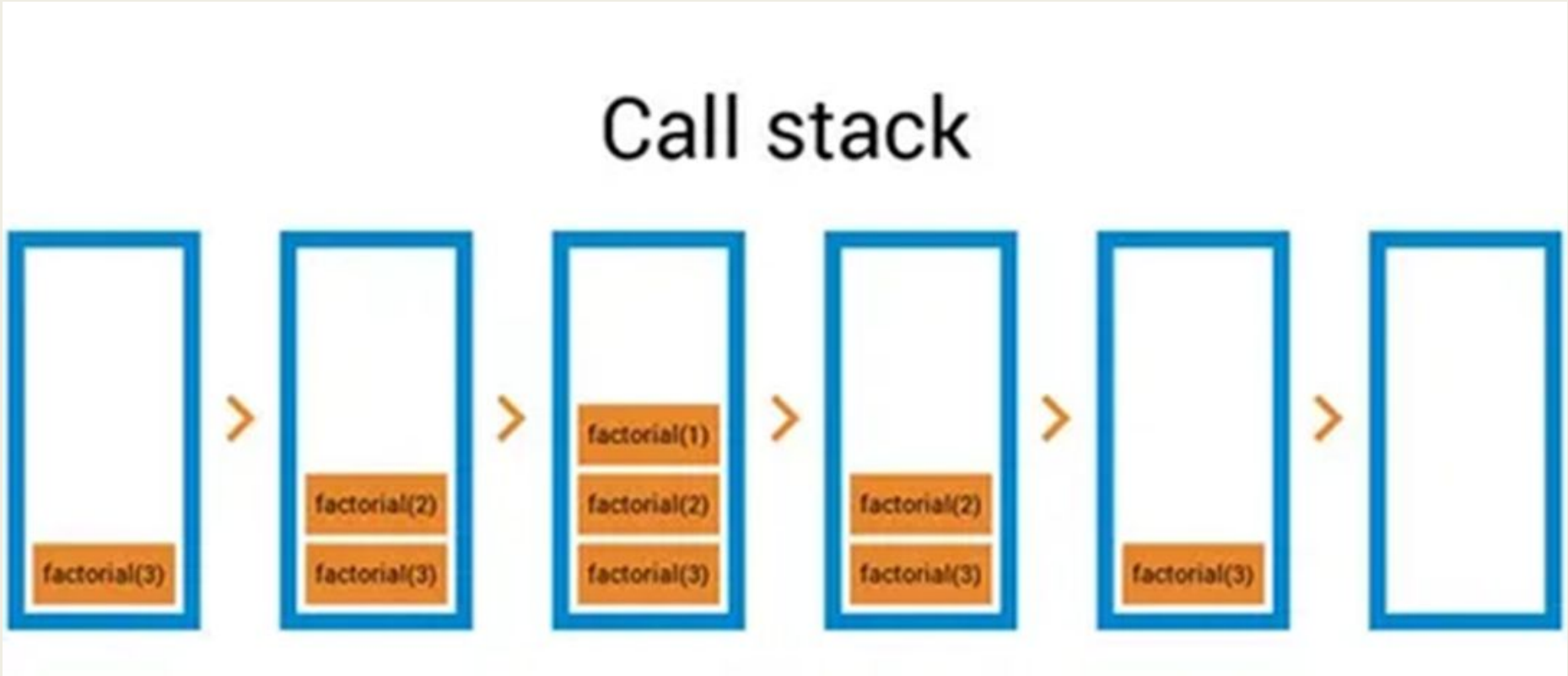
СОЗДАНИЕ И УПРАВЛЕНИЕ ПОТОКАМИ, ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ПОТОКОВ

Класс thread

Потоки

- Поток – абстрактная единица выполнения кода программы, создаваемая ядром ОС. Одному процессу может принадлежать несколько потоков, разделяющих его ресурсы, например, память. Все потоки, принадлежащие процессу разделяют его адресное пространство, последовательность инструкций (код), стек вызовов и его контекст (данные). Процессу принадлежит минимум один поток, называемый главным потоком.

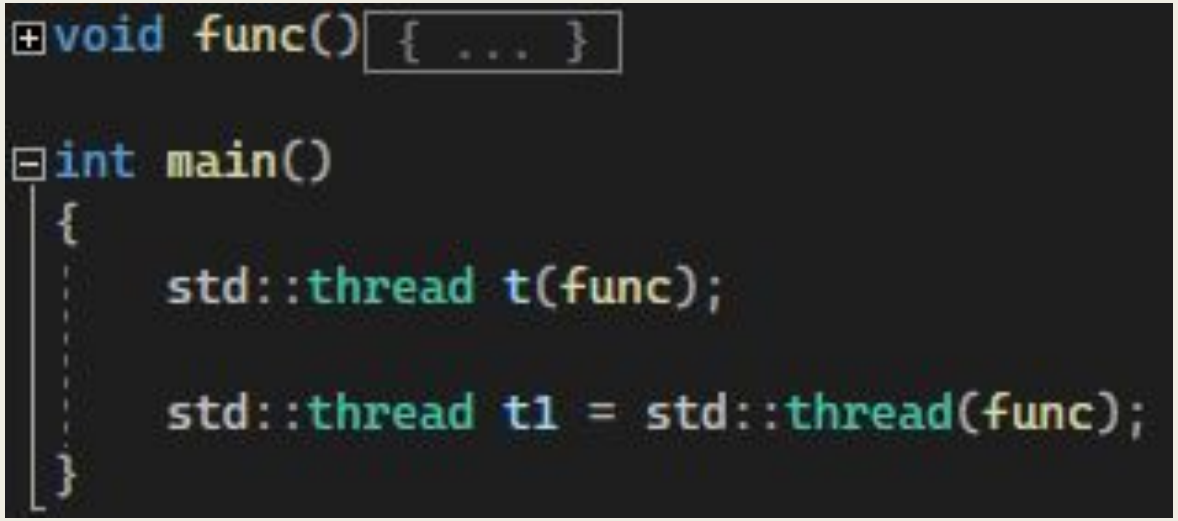
- Стек вызовов – структура данных, организованная по принципу LIFO(Last in First out – последний пришел первый вышел), хранящая адреса для возврата управления из подпрограмм в подпрограмму(при рекурсивном вызове функции) или в основную программу.



Класс thread

- thread - Определяет объект, который позволяет наблюдать за потоком выполнения в приложении и управлять этим потоком. Для работы с ним необходимо включить заголовочный файл thread. Находится в пространстве имен std.

Пример объявления и инициализации:



Класс thread

Для управления выполнением потока используются следующие функции:

- join( );
- detach( );

join( ) – заставляет вызывающий функцию поток дождаться конца выполнения указанного потока в момент вызова – блокирует его.

detach( ) – указывает, что дожидаться конца выполнения не обязательно – отсоединяет указанный поток от объекта thread. ОС становится ответственной за освобождение ресурсов.

При создании потока обязательным является указать один из способов управления созданного потока, в противном случае будет вызвано исключение.

Примеры применения функций управления потоком:

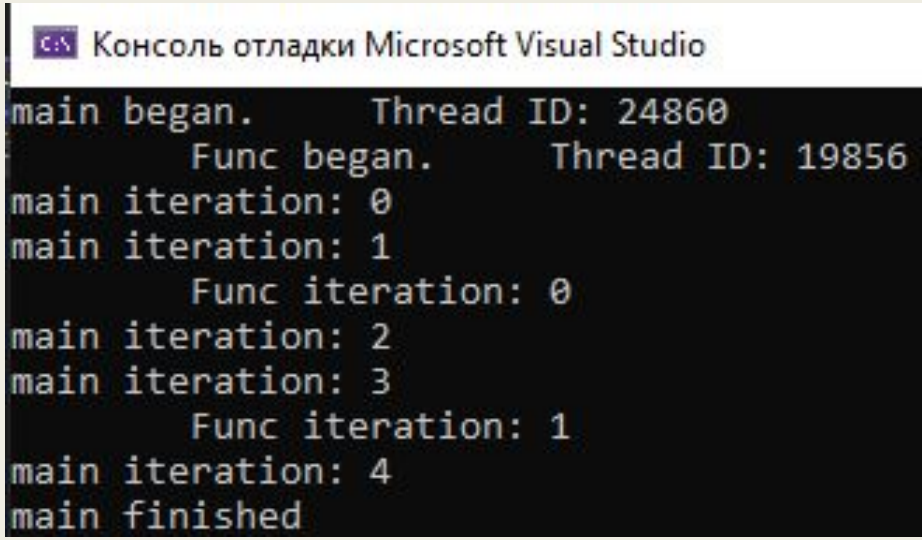
```cpp
#include <iostream>
#include <thread>

void func() {

    std::cout << "\tFunc began.";
    std::cout << "\tThread ID: " << std::this_thread::get_id() << std::endl;

    for (int i = 0; i < 10; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
        std::cout << "\tFunc iteration: " << i << std::endl;
    }

    std::cout << "\tFunc finished" << std::endl;
}

int main()
{
    std::cout << "main began.";
    std::cout << "\tThread ID: " << std::this_thread::get_id() << std::endl;
    std::thread t(func);

    for (int i = 0; i < 5; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(200));
        std::cout << "main iteration: " << i << std::endl;
    }

    t.join();

    std::cout << "main finished" << std::endl;
}
```

Консоль отладки Microsoft Visual Studio

```
main began.     Thread ID: 2008
        Func began.     Thread ID: 28868
main iteration: 0
main iteration: 1
        Func iteration: 0
main iteration: 2
main iteration: 3
        Func iteration: 1
main iteration: 4
        Func iteration: 2
        Func iteration: 3
        Func iteration: 4
        Func iteration: 5
        Func iteration: 6
        Func iteration: 7
        Func iteration: 8
        Func iteration: 9
        Func finished
main finished
```

Примеры применения функций управления потоком:

```cpp
#include <iostream>
#include <thread>

void func() {

    std::cout << "\tFunc began.";
    std::cout << "\tThread ID: " << std::this_thread::get_id() << std::endl;

    for (int i = 0; i < 10; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
        std::cout << "\tFunc iteration: " << i << std::endl;
    }

    std::cout << "\tFunc finished" << std::endl;
}

int main()
{
    std::cout << "main began.";
    std::cout << "\tThread ID: " << std::this_thread::get_id() << std::endl;
    std::thread t(func);

    t.detach();

    for (int i = 0; i < 5; i++) {
        std::this_thread::sleep_for(std::chrono::milliseconds(200));
        std::cout << "main iteration: " << i << std::endl;
    }

    std::cout << "main finished" << std::endl;
}
```

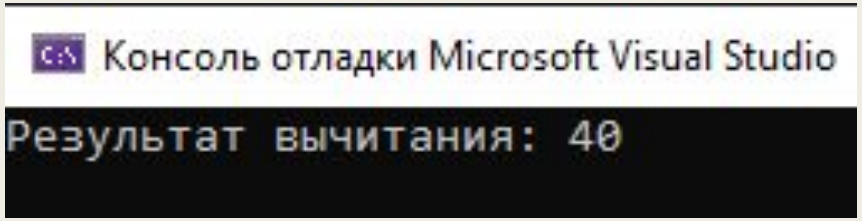


Передача параметров в функцию, выполняемую в другом потоке

Входные параметры для функции передаются в инициализатор потока через запятую.

Пример:

```cpp
#include <iostream>
#include <thread>

void func() { ... }

void Subtract(int a, int b);

int main()
{
    setlocale(LC_ALL, "Russian");
    /* ... */

    std::thread t = std::thread(Subtract, 50, 10);

    t.join();
}

void Subtract(int a, int b) {
    int c = a - b;

    std::cout << "Результат вычитания: " << c << std::endl;
}
```



Получение результата выполнения

Получение результата выполнения функции из другого потока выполняется с помощью лямбда-функции(анонимного метода) по ссылке.

Пример:

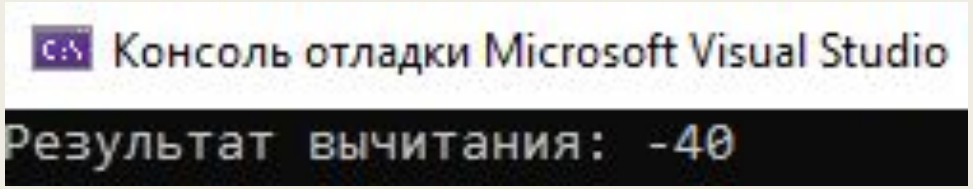
```cpp
#include <iostream>
#include <thread>

void func() { ... }

int Subtract(int a, int b);

int main()
{
    setlocale(LC_ALL, "Russian");

    int result = 0;
    /* ... */

    std::thread t;

    t = std::thread([&result]() {
        result = Subtract(10, 50);
        }
    );

    t.join();

    std::cout << "Результат вычитания: " << result << std::endl;
}

int Subtract(int a, int b) {
    int c = a - b;

    return c;
}
```

```
Консоль отладки Microsoft Visual Studio
Результат вычитания: -40
```

```cpp
#include <iostream>
#include <thread>

void func() { ... }

int Subtract(int a, int b);

int main()
{
    setlocale(LC_ALL, "Russian");

    int result = 0;
    /* ... */

    std::thread t;

    t = std::thread([&result]() {
        result = Subtract(10, 50);
        }
    );

    std::cout << "Результат вычитания: " << result << std::endl;

    t.join();
}

int Subtract(int a, int b) {
    int c = a - b;

    return c;
}
```

```
Консоль отладки Microsoft Visual Studio
Результат вычитания: 0
```

```cpp
#include <iostream>
#include <thread>

void func() { ... }

int Subtract(int a, int b);

int main()
{
    setlocale(LC_ALL, "Russian");

    int result = 0;
    /* ... */

    std::thread t;

    t = std::thread([&result]() {
        result = Subtract(10, 50);
        }
    );

    std::this_thread::sleep_for(std::chrono::milliseconds(50));

    std::cout << "Результат вычитания: " << result << std::endl;

    t.join();
}

int Subtract(int a, int b) {
    int c = a - b;

    return c;
}
```



Потокобезопасность

- При использовании многопоточного программирования часто случается так, что несколько потоков используют один и тот же ресурс или набор данных. В подобных случаях нередко возникает явление «гонки», когда каждый поток производит свои операции над теми же данными. Такая ситуация приводит к неожиданным результатам.

Потокобезопасность

Для предотвращения состояний «гонки» применяются несколько подходов:

- Использование мьютексов (mutex – mutual exclusion – взаимное исключение)

Мьютекс – примитив синхронизации, блокирующий выполнение участка кода. Имеет два состояния: разблокирован и заблокирован. Поток, заблокировавший мьютекс ответственен за его разблокировку.

- Атомарные операции – операции, выполняющиеся за один такт

Мьютексы

Чтобы использовать мьютексы необходимо подключить заголовочный файл mutex.

В языке программирования C++ существует несколько разновидностей мьютексов, разделенные по назначению:

- mutex – базовый тип мьютекса
- recursive_mutex – используется для рекурсивных вызовов подпрограмм
- lock_guard – «обёртка» базового мьютекса, разблокирует его при выходе за скобки
- unique_lock – более гибки вариант lock_guard