# std::thread::thread

| | | |
|---|---|---|
| `thread();` | (1) | (since C++11) |
| `thread( thread&& other );` | (2) | (since C++11) |
| `template< class Function, class... Args > explicit thread( Function&& f, Args&&... args );` | (3) | (since C++11) |
| `thread(const thread&) = delete;` | (4) | (since C++11) |

Constructs new thread object.

1) Creates new thread object which does not represent a thread.
2) Move constructor. Constructs the thread object to represent the thread of execution that was represented by `other`. After this call `other` no longer represents a thread of execution.
3) Creates new `std::thread` object and associates it with a thread of execution. The new thread of execution starts executing

```
std::invoke(decay_copy(std::forward<Function>(f)), decay_copy(std::forward<Args>(args))...)
```

where `decay_copy` is defined as

```
template <class T>
std::decay_t<T> decay_copy(T&& v) { return std::forward<T>(v); }
```

Except that the calls to `decay_copy` are evaluated in the context of the caller, so that any exceptions thrown during evaluation and copying/moving of the arguments are thrown in the current thread, without starting the new thread.

The completion of the invocation of the constructor *synchronizes-with* (as defined in `std::memory_order`) the beginning of the invocation of the copy of *f* on the new thread of execution.

This constructor does not participate in overload resolution if `std::decay_t<Function>` is the same type as `std::thread`. (since C++14)

4) The copy constructor is deleted; threads are not copyable. No two `std::thread` objects may represent the same thread of execution.

## Parameters

| | | |
|---|---|---|
| **other** | - | another thread object to construct this thread object with |
| **f** | - | `Callable` object to execute in the new thread |
| **args...** | - | arguments to pass to the new function |

## Postconditions

1) `get_id()` equal to `std::thread::id()` (i.e. `joinable` is `false`)
2) `other.get_id()` equal to `std::thread::id()` and `get_id()` returns the value of `other.get_id()` prior to the start of construction
3) `get_id()` not equal to `std::thread::id()` (i.e. `joinable` is `true`)

## Exceptions

1-2) `noexcept` specification: `noexcept`

3) `std::system_error` if the thread could not be started. The exception may represent the error condition `std::errc::resource_unavailable_try_again` or another implementation-specific error condition.

## Notes

The arguments to the thread function are moved or copied by value. If a reference argument needs to be passed to the thread function, it has to be wrapped (e.g. with `std::ref` or `std::cref`).

Any return value from the function is ignored. If the function throws an exception, `std::terminate` is called. In order to pass return values or exceptions back to the calling thread, `std::promise` or `std::async` may be used.

## Example

Run this code

```
#include <iostream>
#include <utility>
#include <thread>
#include <chrono>
#include <functional>
#include <atomic>

void f1(int n)
{
    for (int i = 0; i < 5; ++i) {
        std::cout << "Thread 1 executing\n";
        ++n;
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }
}

void f2(int& n)
{
    for (int i = 0; i < 5; ++i) {
        std::cout << "Thread 2 executing\n";
        ++n;
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }
}

int main()
{
    int n = 0;
    std::thread t1; // t1 is not a thread
    std::thread t2(f1, n + 1); // pass by value
    std::thread t3(f2, std::ref(n)); // pass by reference
    std::thread t4(std::move(t3)); // t4 is now running f2(). t3 is no longer a thread
    t2.join();
    t4.join();
    std::cout << "Final value of n is " << n << '\n';
}
```

Possible output:

```
Thread 1 executing
Thread 2 executing
Thread 1 executing
Thread 2 executing
Thread 1 executing
Thread 2 executing
Thread 1 executing
Thread 2 executing
Thread 2 executing
Thread 1 executing
Final value of n is 5
```

## References

- C++11 standard (ISO/IEC 14882:2011):
  - 30.3.1.2 thread constructors [thread.thread.constr]