Information
Tutorials
Reference
Articles
Forum

Reference
C library:
Containers:
Input/Output:
Multi-threading:
<atomic>
<condition_variable>
<future>
<mutex>
<thread>
Other:

<thread>
classes:
thread
namespaces:
this_thread

thread
thread::thread
thread::~thread
member functions:
thread::detach
thread::get_id
thread::join
thread::joinable
thread::native_handle
thread::operator=
thread::swap
member types:
thread::id
thread::native_handle_type
static member functions:
thread::hardware_concurrency
non-member overloads:
swap (thread)

public member function

std::thread::thread

<thread>

default (1)	`thread() noexcept;`
initialization (2)	`template <class Fn, class... Args>` `explicit thread (Fn&& fn, Args&&... args);`
copy [deleted] (3)	`thread (const thread&) = delete;`
move (4)	`thread (thread&& x) noexcept;`

Construct thread

Constructs a `thread` object:

(1) default constructor
Construct a `thread` object that does not represent any thread of execution.

(2) initialization constructor
Construct a `thread` object that represents a new *joinable* thread of execution.
The new thread of execution calls *fn* passing *args* as arguments (using *decay copies* of its *lvalue* or *rvalue* references).
The completion of this construction synchronizes with the beginning of the invocation of this copy of *fn*.

(3) copy constructor
Deleted constructor form (`thread` objects cannot be copied).

(4) move constructor
Construct a `thread` object that acquires the thread of execution represented by *x* (if any). This operation does not affect the execution of the moved thread in any way, it simply transfers its handler.
The *x* object no longer represents any thread of execution.

`thread` objects that are *joinable* shall either be *joined* or *detached* before they are *destroyed*.

Parameters

`fn`
A pointer to function, pointer to member, or any kind of *move-constructible* function object (i.e., an object whose class defines `operator()`, including *closures* and `function` objects).
The return value (if any) is ignored.

`args...`
Arguments passed to the call to *fn* (if any). Their types shall be *move-constructible*.
If *fn* is a *member pointer*, the first argument shall be an object for which that member is defined (or a reference, or a pointer to it).

x
`thread` object whose state is moved to the constructed object.

`Fn` and `Args...` are template parameters: if implicitly deduced, these are the proper *lvalue* or *rvalue* reference type to bind the arguments to. Note though, that on the call to *fn* in the new thread, *decay copies* of *fn* and *args...* are always used (see `std::ref` for a wrapper class that makes references copyable).

Example

```
// constructing threads
#include <iostream>       // std::cout
#include <atomic>         // std::atomic
#include <thread>         // std::thread
#include <vector>         // std::vector

std::atomic<int> global_counter (0);

void increase_global (int n) { for (int i=0; i<n; ++i) ++global_counter; }

void increase_reference (std::atomic<int>& variable, int n) { for (int i=0; i<n; ++i) ++variable; }

struct C : std::atomic<int> {
  C() : std::atomic<int>(0) {}
  void increase_member (int n) { for (int i=0; i<n; ++i) fetch_add(1); }
};

int main ()
{
  std::vector<std::thread> threads;

  std::cout << "increase global counter with 10 threads...\n";
  for (int i=1; i<=10; ++i)
    threads.push_back(std::thread(increase_global,1000));

  std::cout << "increase counter (foo) with 10 threads using reference...\n";
  std::atomic<int> foo(0);
  for (int i=1; i<=10; ++i)
    threads.push_back(std::thread(increase_reference,std::ref(foo),1000));

  std::cout << "increase counter (bar) with 10 threads using member...\n";
  C bar;
  for (int i=1; i<=10; ++i)
    threads.push_back(std::thread(&C::increase_member,std::ref(bar),1000));

  std::cout << "synchronizing all threads...\n";
  for (auto& th : threads) th.join();

  std::cout << "global_counter: " << global_counter << '\n';
  std::cout << "foo: " << foo << '\n';
  std::cout << "bar: " << bar << '\n';

  return 0;
}
```

Output:

```
increase global counter using 10 threads...
increase counter (foo) with 10 threads using reference...
increase counter (bar) with 10 threads using member...
synchronizing all threads...
global_counter: 10000
foo: 10000
bar: 10000
```

Data races

The *move constructor (4)* modifies *x*.

Exception safety

The initialization constructor *(2)* throws an exception on the following conditions:

exception type	error condition	description
`system_error`	`errc::resource_unavailable_try_again`	The system is unable to start a new thread

It also throws if the construction of any of the copies it makes (of the *decay types* of Fn and `Args...`) throws.
Depending on the library implementation, this constructor may also throw exceptions to report other situations (such as `bad_alloc` or `system_error` with other *error conditions*).

Note that if an exception is thrown from the function invocation (i.e., from *fn* itself), it is handled by the new thread. If this invocation terminates with an uncaught exception, `terminate()` is called.

See also

thread::operator=	Move-assign thread (public member function)
thread::join	Join thread (public member function)