

Methods of Threads

This lesson lists and explains the commonly used methods of threads in C++.

WE'LL COVER THE FOLLOWING ^

- More on Swap

Here is the interface of `std::thread t` in a concise table. For additional details, please refer to cppreference.com.

Method	Description
<code>t.join()</code>	Waits until thread <code>t</code> has finished its executable unit.
<code>t.detach()</code>	Executes the created thread <code>t</code> independently of the creator.
<code>t.joinable()</code>	Returns true if thread <code>t</code> is still joinable.
<code>t.get_id()</code> and <code>std::this_thread::get_id()</code>	Returns the identity of the thread.
<code>std::thread::hardware_concurrency()</code>	Returns the number of cores, or 0 if the runtime can not determine the number. Indicates the number of threads that can be run concurrently. This is according to the C++ standard.

<code>std::this_thread::sleep_until(absTime)</code>	Puts thread <code>t</code> to sleep until the time point <code>absTime</code> . Needs a <i>time point</i> or a <i>time duration</i> as an argument.
<code>std::this_thread::sleep_for(relTime)</code>	Puts thread <code>t</code> to sleep for the time duration <code>relTime</code> . Needs a <i>time point</i> or a <i>time duration</i> as an argument.
<code>std::this_thread::yield()</code>	Enables the system to run another thread.
<code>t.swap(t2)</code> and <code>std::swap(t1, t2)</code>	Swaps the threads.

More on Swap

Also, note that threads cannot be copied, but they can be moved; the swap method performs a move when possible.

In the next lesson, we will demonstrate how a few of the methods are used in practice.

i Access to the system-specific implementation

The C++11 threading interface is a wrapper around the underlying implementation. You can use the method `native_handle` to get access to the system-specific implementation. This holds true for threads, mutexes, and condition variables.

