D.7. <thread> header

The `<thread>` header provides facilities for managing and identifying threads and provides functions for making the current thread sleep.

Header contents

```
namespace std
{
    class thread;

    namespace this_thread
    {
        thread::id get_id() noexcept;

        void yield() noexcept;

        template<typename Rep,typename Period>
        void sleep_for(
            std::chrono::duration<Rep,Period> sleep_duration);

        template<typename Clock,typename Duration>
        void sleep_until(
            std::chrono::time_point<Clock,Duration> wake_time);
    }
}
```

D.7.1. std::thread class

The `std::thread` class is used to manage a thread of execution. It provides a means of starting a new thread of execution and waiting for the completion of a thread of execution. It also provides a means for identifying and provides other functions for managing threads of execution.

Class definition

```
class thread
{
public:
    // Types
    class id;
    typedef implementation-defined native_handle_type; // optional

    // Construction and Destruction
    thread() noexcept;
    ~thread();

    template<typename Callable,typename Args...>
    explicit thread(Callable&& func,Args&&... args);
```

```
    // Copying and Moving
    thread(thread const& other) = delete;
    thread(thread&& other) noexcept;

    thread& operator=(thread const& other) = delete;
    thread& operator=(thread&& other) noexcept;

    void swap(thread& other) noexcept;

    void join();
    void detach();
    bool joinable() const noexcept;

    id get_id() const noexcept;

    native_handle_type native_handle();

    static unsigned hardware_concurrency() noexcept;
};

void swap(thread& lhs,thread& rhs);
```

Std::Thread::Id Class

An instance of `std::thread::id` identifies a particular thread of execution.

Class definition

```
class thread::id
{
public:
    id() noexcept;
};

bool operator==(thread::id x, thread::id y) noexcept;
bool operator!=(thread::id x, thread::id y) noexcept;
bool operator<(thread::id x, thread::id y) noexcept;
bool operator<=(thread::id x, thread::id y) noexcept;
bool operator>(thread::id x, thread::id y) noexcept;
bool operator>=(thread::id x, thread::id y) noexcept;

template<typename charT, typename traits>
basic_ostream<charT, traits>&
operator<< (basic_ostream<charT, traits>&& out, thread::id id);
```

Notes

The `std::thread::id` value that identifies a particular thread of execution shall be distinct from the value of a default-constructed `std::thread::id` instance and from any value that represents another thread of execution.

The `std::thread::id` values for particular threads aren't predictable and may vary between

executions of the same program.

`std::thread::id` is `CopyConstructible` and `CopyAssignable`, so instances of `std::thread::id` may be freely copied and assigned.

Std::Thread::Id Default Constructor

Constructs a `std::thread::id` object that doesn't represent any thread of execution.

Declaration

```
id() noexcept;
```

Effects

Constructs a `std::thread::id` instance that has the singular *not any thread* value.

Throws

Nothing.

Note

All default-constructed `std::thread::id` instances store the same value.

Std::Thread::Id Equality Comparison Operator

Compares two instances of `std::thread::id` to see if they represent the same thread of execution.

Declaration

```
bool operator==(std::thread::id lhs,std::thread::id rhs) noexcept;
```

Returns

`true` if both `lhs` and `rhs` represent the same thread of execution or both have the singular *not any thread* value. `false` if `lhs` and `rhs` represent different threads of execution or one represents a thread of execution and the other has the singular *not any thread* value.

Throws

Nothing.

Std::Thread::Id Inequality Comparison Operator

Compares two instances of `std::thread::id` to see if they represent different threads of execution.

Declaration

```
bool operator!=(std::thread::id lhs,std::thread::id rhs) noexcept;
```

Returns

```
!(lhs==rhs)
```

Throws

Nothing.

Std::Thread::Id Less-Than Comparison Operator

Compares two instances of `std::thread::id` to see if one lies before the other in the total ordering of thread ID values.

Declaration

```
bool operator<(std::thread::id lhs,std::thread::id rhs) noexcept;
```

Returns

`true` if the value of `lhs` occurs before the value of `rhs` in the total ordering of thread ID values. If `lhs!=rhs`, exactly one of `lhs<rhs` or `rhs<lhs` returns `true` and the other returns `false`. If `lhs==rhs`, `lhs<rhs` and `rhs<lhs` both return `false`.

Throws

Nothing.

Note

The singular *not any thread* value held by a default-constructed `std::thread::id` instance compares less than any `std::thread::id` instance that represents a thread of execution. If two instances of `std::thread::id` are equal, neither is less than the other. Any set of distinct `std::thread::id` values forms a total order, which is consistent throughout an execution of a program. This order may vary between executions of the same program.

Std::Thread::Id Less-Than or Equal Comparison Operator

Compares two instances of `std::thread::id` to see if one lies before the other in the total ordering of thread ID values or is equal to it.

Declaration

```
bool operator<=(std::thread::id lhs,std::thread::id rhs) noexcept;
```

Returns

```
!(rhs<lhs)
```

Throws

Nothing.

Std::Thread::Id Greater-Than Comparison Operator

Compares two instances of `std::thread::id` to see if one lies after the other in the total ordering of thread ID values.

Declaration

```
bool operator>(std::thread::id lhs,std::thread::id rhs) noexcept;
```

Returns

```
rhs<lhs
```

Throws

Nothing.

Std::Thread::Id Greater-Than Or Equal Comparison Operator

Compares two instances of `std::thread::id` to see if one lies after the other in the total ordering of thread ID values or is equal to it.

Declaration

```
bool operator>=(std::thread::id lhs,std::thread::id rhs) noexcept;
```

Returns

```
!(lhs<rhs)
```

Throws

Nothing.

Std::Thread::Id Stream Insertion Operator

Writes a string representation of the `std::thread::id` value into the specified stream.

Declaration

```
template<typename charT, typename traits>
basic_ostream<charT, traits>&
operator<< (basic_ostream<charT, traits>&& out, thread::id id);
```

Effects

Inserts a string representation of the `std::thread::id` value into the specified stream.

Returns

```
out
```

Throws

Nothing.

Note

The format of the string representation isn't specified. Instances of `std::thread::id` that compare equal have the same representation, and instances that aren't equal have distinct representations.

Std::Thread::Native_Handle_Type Typedef

`native_handle_type` is a typedef to a type that can be used with platform-specific APIs.

Declaration

`typedef` *implementation-defined* `native_handle_type;`

Note

This typedef is *optional*. If present, the implementation should provide a type that's suitable for use with native platform-specific APIs.

Std::Thread::Native_Handle Member Function

Returns a value of type `native_handle_type` that represents the thread of execution associated with `*this`.

Declaration

```
native_handle_type native_handle();
```

Note

This function is *optional*. If present, the value returned should be suitable for use with the native platform-specific APIs.

Std::Thread Default Constructor

Constructs a `std::thread` object without an associated thread of execution.

Declaration

```
thread() noexcept;
```

Effects

Constructs a `std::thread` instance that has no associated thread of execution.

Postconditions

For a newly constructed `std::thread` object `x, x.get_id()==id()`.

Throws

Nothing.

Std::Thread Constructor

Constructs a `std::thread` object associated with a new thread of execution.

Declaration

```
template<typename Callable,typename Args...>
explicit thread(Callable&& func,Args&&... args);
```

Preconditions

`func` and each element of `args` must be `MoveConstructible`.

Effects

Constructs a `std::thread` instance and associates it with a newly created thread of execution. Copies or moves `func` and each element of `args` into internal storage that persists for the lifetime of the new thread of execution. Performs *INVOKE (copy-of-*`func`*,copy-of-*`args`*)* on the new thread of execution.

Postconditions

For a newly constructed `std::thread` object `x`, `x.get_id()!=id()`.

Throws

An exception of type `std::system_error` if unable to start the new thread. Any exception thrown by copying `func` or `args` into internal storage.

Synchronization

The invocation of the constructor happens-before the execution of the supplied function on the newly created thread of execution.

Std::Thread Move-Constructor

Transfers ownership of a thread of execution from one `std::thread` object to a newly created `std::thread` object.

Declaration

```
thread(thread&& other) noexcept;
```

Effects

Constructs a `std::thread` instance. If `other` has an associated thread of execution prior to the constructor invocation, that thread of execution is now associated with the newly created `std::thread` object. Otherwise, the newly created `std::thread` object has no associated thread of execution.

Postconditions

For a newly constructed `std::thread` object `x`, `x.get_id()` is equal to the value of `other.get_id()` prior to the constructor invocation. `other.get_id()==id()`.

Throws

Nothing.

Note

`std::thread` objects are *not* `CopyConstructible`, so there's no copy constructor, only this move constructor.

Std::Thread Destructor

Destroys a `std::thread` object.

Declaration

```
~thread();
```

Effects

Destroys `*this`. If `*this` has an associated thread of execution (`this->joinable()` would return `true`), calls `std::terminate()` to abort the program.

Throws

Nothing.

Std::Thread Move-Assignment Operator

Transfers ownership of a thread of execution from one `std::thread` object to another `std::thread` object.

Declaration

```
thread& operator=(thread&& other) noexcept;
```

Effects

If `this->joinable()` returns `true` prior to the call, calls `std::terminate()` to abort the program. If `other` has an associated thread of execution prior to the assignment, that thread of execution is now associated with `*this`. Otherwise `*this` has no associated thread of execution.

Postconditions

`this->get_id()` is equal to the value of `other.get_id()` prior to the call. `other.get_id()==id()`.

Throws

Nothing.

Note

`std::thread` objects are *not* `CopyAssignable`, so there's no copy-assignment operator, only this move-assignment operator.

Std::Thread::Swap Member Function

Exchanges ownership of their associated threads of execution between two `std::thread` objects.

Declaration

```
void swap(thread& other) noexcept;
```

Effects

If `other` has an associated thread of execution prior to the call, that thread of execution is now

associated with `*this`. Otherwise `*this` has no associated thread of execution. If `*this` has an associated thread of execution prior to the call, that thread of execution is now associated with `other`. Otherwise `other` has no associated thread of execution.

Postconditions

`this->get_id()` is equal to the value of `other.get_id()` prior to the call. `other.get_id()` is equal to the value of `this->get_id()` prior to the call.

Throws

Nothing.

Swap Nonmember Function For Std::Threads

Exchanges ownership of their associated threads of execution between two `std::thread` objects.

Declaration

```
void swap(thread& lhs,thread& rhs) noexcept;
```

Effects

```
lhs.swap(rhs)
```

Throws

Nothing.

Std::Thread::Joinable Member Function

Queries whether or not `*this` has an associated thread of execution.

Declaration

```
bool joinable() const noexcept;
```

Returns

`true` if `*this` has an associated thread of execution, `false` otherwise.

Throws

Nothing.

Std::Thread::Join Member Function

Waits for the thread of execution associated with `*this` to finish.

Declaration

```
void join();
```

Preconditions

`this->joinable()` would return `true`.

Effects

Blocks the current thread until the thread of execution associated with `*this` has finished.

Postconditions

`this->get_id()==id()`. The thread of execution associated with `*this` prior to the call has finished.

Synchronization

The completion of the thread of execution associated with `*this` prior to the call happens-before the call to `join()` returns.

Throws

`std::system_error` if the effects can't be achieved or `this->joinable()` returns `false`.

Std::Thread::Detach Member Function

Detaches the thread of execution associated with `*this` to finish.

Declaration

```
void detach();
```

Preconditions

`this->joinable()` returns `true`.

Effects

Detaches the thread of execution associated with `*this`.

Postconditions

```
this->get_id()==id(), this->joinable()==false
```

The thread of execution associated with `*this` prior to the call is detached and no longer has an associated `std::thread` object.

Throws

`std::system_error` if the effects can't be achieved or `this->joinable()` returns `false` on invocation.

Std::Thread::Get_Id Member Function

Returns a value of type `std::thread::id` that identifies the thread of execution associated with `*this`.

Declaration

```
thread::id get_id() const noexcept;
```

Returns

If `*this` has an associated thread of execution, returns an instance of `std::thread::id` that identifies that thread. Otherwise returns a default-constructed `std::thread::id`.

Throws

Nothing.

Std::Thread::Hardware_Concurrency Static Member Function

Returns a hint as to the number of threads that can run concurrently on the current hardware.

Declaration

```
unsigned hardware_concurrency() noexcept;
```

Returns

The number of threads that can run concurrently on the current hardware. This may be the number of processors in the system, for example. Where this information is not available or well defined, this function returns `0`.

Throws

Nothing.

D.7.2. Namespace this_thread

The functions in the `std::this_thread` namespace operate on the calling thread.

Std::This_Thread::Get_Id Nonmember Function

Returns a value of type `std::thread::id` that identifies the current thread of execution.

Declaration

```
thread::id get_id() noexcept;
```

Returns

An instance of `std::thread::id` that identifies the current thread.

Throws

Nothing.

Std::This_Thread::Yield Nonmember Function

Used to inform the library that the thread that invoked the function doesn't need to run at the point of the call. Commonly used in tight loops to avoid consuming excessive CPU time.

Declaration

```
void yield() noexcept;
```

Effects

Provides the library an opportunity to schedule something else in place of the current thread.

Throws

Nothing.

Std::This_Thread::Sleep_For Nonmember Function

Suspends execution of the current thread for the specified duration.

Declaration

```
template<typename Rep,typename Period>
void sleep_for(std::chrono::duration<Rep,Period> const& relative_time);
```

Effects

Blocks the current thread until the specified `relative_time` has elapsed.

Note

The thread may be blocked for longer than the specified duration. Where possible, the elapsed time is determined by a steady clock.

Throws

Nothing.

Std::This_Thread::Sleep_Until Nonmember Function

Suspends execution of the current thread until the specified time point has been reached.

Declaration

```
template<typename Clock,typename Duration>
void sleep_until(
    std::chrono::time_point<Clock,Duration> const& absolute_time);
```

Effects

Blocks the current thread until the specified `absolute_time` has been reached for the specified `Clock`.

Note

There's no guarantee as to how long the calling thread will be blocked for, only that `Clock::now()` returned a time equal to or later than `absolute_time` at the point at which the thread became unblocked.

Throws

Nothing.