Clarifying atomic<thread::id>::compare_exchange_*

Document Number: **P1801 R0** Date: 2019-07-17
Reply-to: Herb Sutter (hsutter@microsoft.com)
Audience: SG1, LEWG, LWG

1 Overview

`atomic<thread::id>` is intended to work and is used in practice, including in Boost.Thread, MongoDB, Firefox, Chromium, and protobuf. (See: http://lists.isocpp.org/parallel/2019/06/2688.php.)

`thread::id` meets all of the requirements on `T` of `atomic<T>`, including that it is trivially copyable (see [thread.thread.id]/2). It also provides user-defined comparison operators. However, the standard doesn't explicitly state the intent that `thread::id` is `cmpxchg`-friendly.

`atomic<thread::id>::compare_exchange_*` can be easily implemented for any conforming `thread::id` in the same way as general `atomic<T>` where `T` can be an arbitrarily large trivially copyable struct, by masking any thread-id-irrelevant bits (such as padding or status bits) in one parameter each in `.store` and `.compare_exchange_*`. EWG already decided on this approach in general for all `T`, not just `thread::id`, in Albuquerque 2017.

Thanks to Anthony Williams for pointing out this issue, to JF Bastien for P0528 and reminder of this issue's history, and to the following for their additional comments: Olivier Giroux, Daniel Krügler, Jens Maurer, Billy O'Neal, Detlef Vollmann, Ville Voutilainen, Jonathan Wakely, Anthony Williams.

2 Discussion

2.1 Key history: P0528

P0528 has covered much related ground in previous EWG and SG1 discussions, including that EWG already decided on this approach for all `atomic<T>` including `atomic<big_struct>` (Albuquerque 2017 EWG wiki notes):

> `Straw polls: SF | F | N | A | SA`
>
> `Make the padding bits of atomic and the incoming value of T have a consistent value for the purposes of read/modify/write atomic operations? 3 | 14 | 3 | 0 | 0`

This technique works with all types that are not unions having members of different sizes, so the only thing that remains is to ensure `thread::id` is not a union whose members could have different padding, which is fine because no known implementation of `thread::id` is a union or uses one (see also `pthread_t` discussion later on).

2.2 Status quo in the standard

`thread::id` meets all of the requirements on `T` of `atomic<T>` in [atomics.types.generic]/1: "The template argument for T shall meet the Cpp17CopyConstructible and Cpp17CopyAssignable requirements. The program is ill-formed if any of `is_trivially_copyable_v<T>`, `is_copy_constructible_v<T>`, `is_move_constructible_v<T>`, `is_copy_assignable_v<T>`, or `is_move_assignable_v<T>` is false." And it is already intended to store unique values, per [thread.thread.id]/1-2: "An object of type `thread::id` provides a unique identifier … The library may reuse the value …".

In [atomics.types.operations]/18, `atomic<T>::compare_exchange_*` "atomically compares the value representation" for equality. It does not use the object representation or overloaded comparison operators. (Note: per previous discussion of P0528, padding bits and other related bitwise requirements are not a concern.)

EWG Albuquerque 2017 decided that the intent is that `atomic<T>` work for types `T` that may contain padding bits as long as they otherwise meet the `atomic<T>` requirements (which `thread::id` does), as long as those padding bits are known at compile time, which effectively means "not a union with members of different sizes."[1]

So we just need to make it clearer that `thread::id` is `cmpxchg`-friendly.

2.3 Status quo in implementations, and `pthread_t`

`atomic<thread::id>` works on the Microsoft compiler and Windows.

libstdc++ and libc++ implement `thread::id` as `pthread_t` or direct wrapper thereof, which appears to work in practice on major platforms even though there are several issues, most of which are about meeting the requirements of `thread::id` itself:

1. `pthread_t` is not guaranteed to support a "not a valid thread" value, which is required by `thread::id`'s default constructor.
2. `pthread_t` is not supposed to be directly copied, whereas `thread::id` must be trivially copyable and copy/move constructible and assignable.
3. `pthread_t` is only guaranteed to support equality comparison via `pthread_equal`, but not ordered comparisons as required by `thread::id`. Further, the behavior of `pthread_equal` is undefined for "not a valid thread" values (see #1 above).
4. `pthread_t` is not guaranteed to be bitwise comparable — this is the part that affects `compare_exchange_*`, and it is just a special case of the more general #3 above.

Some notes about `pthread_t`:

- In practice, `pthread_t` must support many operations it does not formally support, such as copying. See for example this lively pragmatic-vs-pedantic 2007 discussion.
- On several common platforms, `pthread_t` is a pointer or an integer with all bits used, and `pthread_t` equality comparison is implemented as pointer/integer equality in libstdc++ and using `pthread_equal` in libc++. On those platforms, `pthread_t` can satisfy the `thread::id` and `compare_exchange_*` requirements directly, even though POSIX also permits implementations that do not satisfy these requirements. For example, see this glibc pthread.h.
- On the platforms and implementations where `pthread_t` cannot directly satisfy the `compare_exchange_*` requirements, it does not satisfy the other `thread::id` requirements either. For example, see Facebook's Folly library pthread.h where `pthread_t` is an alias for a `std::shared_ptr<detail::pthread_t>` which is not trivially copyable.
- I don't know of any implementation of `pthread_t` that is a union. Searching Google for "union pthread_t" returns one hit, which when followed does not lead to a definition using a union (it leads to a definition that is an integer). Searching codesearch.isocpp.org for union pthread_t returns no hits.

[1] The `atomic<thread::id>` specialization can mask off non-id bits on the argument to `.store` and on the expected value to `.compare_exchange_*`. Such an implementation can still be lock-free if an integer of the same size would be lock-free.

3 Proposed resolution

3.1 Alternative 1: Repeat the [atomics.types.operations] text

Note: This option was unanimously approved by SG1 in Cologne.

In [thread.thread.id]/2, after "`thread::id` is a trivially copyable class" add:

> and has no padding bits that participate in the object's value representation

That is sufficient to make it clearly meet the existing requirements in [atomics.types.operations], by repeating the text from there.

3.2 Alternative 2: Say it's not a union

As already noted, the only types that won't `cmpxchg` properly all the time are unions with members of different size, and no known implementation of `thread::id` is in terms of an underlying union type.

In [thread.thread.id]/2, after "`thread::id` is a trivially copyable class" add:

> and is not a union and does not have a base class or data member of union type

4 Summary and implementation impact

`atomic<thread::id>` is used in practice, and the standard should clarify that it works. Either alternative proposed resolution appears to be sufficient, and as far as I know it does not create any problems for any conforming implementation of `thread::id`.

5 Additional note

Separately from this paper's issue:

- The wording of [atomics.types.operations] could be improved to replace "has no padding bits" with "ignores any padding bits." (And possibly add "not a union whose members have different padding bits" to `atomic<T>`'s requirements on `T`.)