Not All Agents Have TLS

This paper: P1367R0.

Author: ogiroux@nvidia.com

Date: 11/11/2018.

The reality of `thread_local` is much more complicated than wording in the Standard allow us to even explain. This paper proposes to standardize existing practice, not to materially improve it.

Revision history

R0: This is the first version. It already includes some feedback from the reflector.

The current nature of reality

The Standard guarantees that object definitions with `thread_local` storage are instantiated with every thread. It's a little vague whether the word 'thread' here refers to each thread of execution, or only instances of `std::thread` and the *implementation-defined* thread that runs `main`. Other than that, it's unambiguous that `thread_local` is required in C++11, and later.

Implementations don't always meet the requirements of the Standard in this area; most implementations either don't implement, or implement different semantics for `thread_local` objects in at least some of the execution agents that they comprise. This situation is not going to change because the cost/benefit calculus that buttresses this situation is not likely to change.

Proposed direction

Here is my 5-point proposal to reframe TLS...

1. `thread_local` always refers to the thread of execution.

It is not referring to an assumed `std::thread`. It is the one and only keyword to declare an object with X-local storage. If the execution agent you are executing on has TLS, and I'll explain how to figure this out below, then `thread_local` objects are local to that agent.

Similarly, we should strive for the `std::this_thread` namespace to also refer to the thread of execution. There, we may want to provide functions to query execution agent properties. For instance, we could expose `bool std::this_thread::has_thread_local();`.

2. `std::thread`, agents of `std::async`, support `thread_local`.

That this is unconditionally true is required for compatibility with C++11. It's *implementation-defined* whether these exist in Freestanding C++, however. An implementation that claims it is Freestanding C++ can provide alternate forms of threads with different TLS semantics so long as they are not spelled `std::thread`.

3. The thread that runs `main` has *implementation-defined* TLS.

However, this is bounded by two conditions:

- If `std::thread` is implemented, then `main` supports `thread_local`.
- If `main` doesn't support `thread_local`, then `thread_local` objects have static storage instead.

4. Executors can expose the `has_thread_local` property.

If `execution::can_query_v<Ex,execution::has_thread_local_t>` is `false`, then execution agents provided by this executor support `thread_local`. Otherwise, they support `thread_local` if `has_thread_local` is `true`. If `thread_local` is not supported then uses of such objects could be ill-formed-no-diagnostic-required or undefined (enabling the `this_thread` query, for one).

Parallel algorithms have *implementation-defined* support, as if there existed a default executor.

5. Attributes can reenable selective support for execution agents.

Implementations can define attributes so that, e.g. `[[xyz_local]] thread_local int x;` introduces a `thread_local` object that is not ill-formed for the execution agents of feature `xyz`. However, since it is not valid for executor to set `has_thread_local` to `true` without supporting all `thread_local` objects, implementations should define new executor properties, e.g. `has_xyz_local` specific to the attributes they also define.

Execution agents where `has_thread_local` is `true` ignore these attributes because they support all `thread_local` objects unconditionally.

See also

http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2012/n3487.pdf

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3556.pdf

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4439.pdf

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0097r0.html

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0108r1.html

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0772r1.pdf