Document number: P1202R4
Date: 2022-02-14
Reply-to: David Goldblatt <davidtgoldblatt@gmail.com>
Audience: LEWG, LWG

P1202R4: Asymmetric Fences

Background

Some types of concurrent algorithms can be split into a common path and an uncommon path, both of which require fences (or other operations with non-relaxed memory orders) for correctness. On many platforms, it's possible to speed up the common path by adding an even stronger fence type (stronger than `memory_order::seq_cst`) down the uncommon path. These facilities are being used in an increasing number of concurrency libraries. We propose standardizing these asymmetric fences, and incorporating them into the memory model.

The proposed ship vehicle is Concurrency TS 2.

In Prague, LEWG voted (unanimous consent) to forward P1202R2 to LWG. December 2021 electronic polling, LEWG voted (also unanimous consent) for P2396, which applies some minor editorial changes to address issues caught in LWG pre-review (the header name and feature test macro). The LEWG chairs asked for a "clean" paper with all changes from P2396 applied for LEWG to vote on again.

This paper also rolls-in the changes asked for in LWG pre-review (which appeared in P1202R3), which don't affect the interface or functionality, but apply wording fixes and simplifications and tighten up the standardese into a list of diffs for the TS editor.

In the interest of brevity, this omits much of the context that already has directional approval; see P1202R0 for an in-depth description of the technique and its uses, and P1202R1 for an argument that this is the "right" memory-model-ese for this technique.

Wording

As a diff for the TS to apply to the IS:

31.4 Order and consistency [atomics.order]
In subclause 31.4 [atomics.order], strike the word "four" in the phrase "the following four conditions are required to be satisfied by S:" and add the following two bullets to the list:

- if a `memory_order::seq_cst` lightweight-fence X happens before A and B happens before a `memory_order::seq_cst` heavyweight-fence Y, then X precedes Y in S; and

- if a `memory_order::seq_cst` heavyweight-fence X happens before A and B happens before a `memory_order::seq_cst` lightweight-fence Y, then X precedes Y in S.

And, as a pure insertion, with a section number to be filled in by the editor:

X.Y Header <experimental/asymmetric_fence> synopsis
Add the following declarations to the synopsis of the header `<experimental/asymmetric_fence>`:

```
namespace std::experimental::inline concurrency_v2 {
  // ?.2.1 asymmetric_thread_fence_heavy
  void asymmetric_thread_fence_heavy(memory_order order) noexcept;
  // ?.2.2 asymmetric_thread_fence_light
  void asymmetric_thread_fence_light(memory_order order) noexcept;
}
```

X.Z Asymmetric fences [atomics.fences.asym]

This section introduces synchronization primitives called *heavyweight-fences* and *lightweight-fences*. Like fences, heavyweight-fences and lightweight-fences can have acquire semantics, release semantics, or both, and can be sequentially consistent (in which case they are included in the total order S on `memory_order::seq_cst` operations). A heavyweight-fence has all the synchronization effects of a fence as specified in 31.11 [atomic.fences]. [Note: Heavyweight-fences and lightweight-fences are distinct from fences. -- end note]

If there are evaluations A and B, and atomic operations X and Y, both operating on some atomic object M, such that A is sequenced before X, X modifies M, Y is sequenced before B, and Y reads the value written by X or a value written by any side effect in the hypothetical release sequence X would head if it were a release operation, and one of the following hold:
- A is a release lightweight-fence and B is an acquire heavyweight-fence; or
- A is a release heavyweight-fence and B is an acquire lightweight-fence

then any evaluation sequenced before A strongly happens before any evaluation that B is sequenced before.

```
void asymmetric_thread_fence_heavy(memory_order order) noexcept;
```

1. Effects: Depending on the value of `order`, this operation:
- has no effects, if `order == memory_order::relaxed`;
- is an acquire heavyweight-fence, if `order == memory_order::acquire` or `order == memory_order::consume`;
- is a release heavyweight-fence, if `order == memory_order::release`;
- is both an acquire heavyweight-fence and a release heavyweight-fence, if `order == memory_order::acq_rel`;

- is a sequentially consistent acquire and release heavyweight-fence, if `order == memory_order::seq_cst`.

`void asymmetric_thread_fence_light(memory_order order) noexcept;`

1. Effects: Depending on the value of `order`, this operation:
   - has no effects, if `order == memory_order::relaxed`;
   - is an acquire lightweight-fence, if `order == memory_order::acquire` or `order == memory_order::consume`;
   - is a release lightweight-fence, if `order == memory_order::release`;
   - is both an acquire lightweight-fence and a release lightweight-fence, if `order == memory_order::acq_rel`;
   - is a sequentially consistent acquire and release lightweight-fence, if `order == memory_order::seq_cst`.

[Note: Delegating both heavy and light fence functions to an `atomic_thread_fence(order)` call is a valid implementation.]

Add a feature test macro in `<experimental/asymmetric_thread_fence>`:
#define __cpp_lib_experimental_asymmetric_fence 202XYZ