Document number: P1074R0
Date: 20180507 (RAP)
Project: Programming Language C++, WG21, SG1, Core
Authors: Maged Michael, David Goldblatt, Michael Wong, Paul McKenney
Reply-to: maged.michael@acm.org

CWG defect Defined Behavior of Invalid Pointers

Core Wording defect
CWG xxxx Defined Behavior of Invalid Pointers
Section: 6.6.4 [basic.stc] **Status:** open **Submitter: Maged Michael** **Date:** 2018-05-07

We want to make the following case of implementation-defined behavior have defined behavior. This was discovered during hazard pointer review, but it is not specific to the hazard pointer method [P0566] proposal.

The following is a possible implementation of member function `try_protect` of class template `hazptr_holder`, with function parameters `ptr` and `src` [P0566]. It demonstrates an occurance of the above-mentioned implementation-defined behavior in a practical use case.

```
// Possible conditions:
// - ptr points to object of type T (or is null) and src.load() == ptr
// - ptr points to object of type T (or is null) and src.load() != ptr
// - ptr points to free memory (it must be src.load() != ptr)

hazptr_holder::try_protect(T*& ptr, const std::atomic<T*>& src) {
  /* If ptr points to free memory at Step 1 or Step 2, then such step
     has implementation-defined behavior per N4727. */
  T* p = ptr; // Step 1
  reset_protected(ptr); // Step 2: store ptr to a private atomic pointer
  /* Fence */
  ptr = src.load(std::memory_order_acquire);
  if (p == ptr) return true; // success
  reset_protected(); // failure: store nullptr to the private atomic pointer
  return false; // failure
}
```

According to current wording in [basic.stc] (6.6.4 in N4727, the 2018-02-18 working draft), this is implementation-defined behavior.

> **6.6.4 Storage duration [basic.stc]**
>
> 1 The storage duration is the property of an object that defines the minimum potential lifetime of the storage containing the object. The storage duration is determined by the construct used to create the object and is one of the following:
> (1.1) — static storage duration
> (1.2) — thread storage duration
> (1.3) — automatic storage duration
> (1.4) — dynamic storage duration
>
> 2 Static, thread, and automatic storage durations are associated with objects introduced by declarations (6.1) and implicitly created by the implementation (15.2). The dynamic storage duration is associated with objects created by a new-expression (8.5.2.4).
>
> 3 The storage duration categories apply to references as well.
>
> 4 When the end of the duration of a region of storage is reached, the values of all pointers representing the address of any part of that region of storage become invalid pointer values (6.7.2). Indirection through an invalid pointer value and passing an invalid pointer value to a deallocation function have undefined behavior. Any other use of an invalid pointer value has implementation-defined behavior.[35]
>
> [35]) Some implementations might define that copying an invalid pointer value causes a system-generated runtime fault.

Suggested fix:
One possible fix is to make the invalid pointer only affect the innermost execution agent that actually did the delete. This would allow the standard debugging technique of nulling the pointer past the delete without introducing data races by nulling every other pointers past the execution agent.
Are there other fix proposals?

Reference
[N4727] 2018-02-18 working draft.
[P0566R5] Proposed Wording for Concurrent Data Structures: Hazard Pointer and Read-Copy-Update (RCU).

Proposed change:

> 4 When the end of the duration of a region of storage is reached, the values of all pointers **within that same execution agent** representing the address of any part of that region of storage become invalid pointer values (6.7.2) **unless an object of the same type is allocated in the region after being deleted by a different thread from the thread operating on the pointer**. Indirection through an invalid pointer value and passing an invalid pointer value to a deallocation function have undefined behavior. ~~Any other use of an invalid pointer value has implementation-defined behavior.35~~
>
> ~~35) Some implementations might define that copying an invalid pointer value causes a system-generated runtime fault.~~