Document number: P3428R2
Date: 2025-08-11
Project: Programming Language C++, LEWG
Authors: Maged M. Michael, Michael Wong, Paul McKenney
Email: maged.michael@acm.org, fraggamuffin@gmail.com, paulmck@kernel.org

Hazard Pointer Batches

Table of Contents

Introduction

This paper proposes extending the working draft (N5008) C++26 hazard pointer interface to support creation and destruction of batches of nonempty hazard pointers.

The R1 version of this paper was reviewed by SG1 in Wroclaw 2024 and forwarded toLEWG with feedback to relax preconditions and rename some functions and parameters. This revision P3428R2 revises R1 by following SG1 feedback.

Background: P2530R3 C++26 Hazard Pointers

Hazard pointer interface from P2530R3 merged into the working draft (N5008):

```
template <class T, class D = default_delete<T>>
class hazard_pointer_obj_base {
public:
  void retire(D d = D()) noexcept;
protected:
  hazard_pointer_obj_base() = default;
  hazard_pointer_obj_base(const hazard_pointer_obj_base&) = default;
  hazard_pointer_obj_base(hazard_pointer_obj_base&&) = default;
  hazard_pointer_obj_base& operator=(const hazard_pointer_obj_base&) = default;
  hazard_pointer_obj_base& operator=(hazard_pointer_obj_base&&) = default;
  ~hazard_pointer_obj_base() = default;
private:
  D deleter ; // exposition only
};

class hazard_pointer {
public:
  hazard_pointer() noexcept; // Constructs an empty hazard_pointer
  hazard_pointer(hazard_pointer&&) noexcept;
  hazard_pointer& operator=(hazard_pointer&&) noexcept;
  ~hazard_pointer();
  [[nodiscard]] bool empty() const noexcept;
  template <class T> T* protect(const atomic<T*>& src) noexcept;
  template <class T> bool try_protect(T*& ptr, const atomic<T*>& src) noexcept;
  template <class T> void reset_protection(const T* ptr) noexcept;
  void reset_protection(nullptr_t = nullptr) noexcept;
  void swap(hazard_pointer&) noexcept;
};

hazard_pointer make_hazard_pointer(); // Constructs a nonempty hazard_pointer
void swap(hazard_pointer&, hazard_pointer&) noexcept;
```

Brief notes (See P2530R3 for details):

- A **`hazard_pointer`** object is either *empty* or *nonempty*. It is nonempty if and only if it owns a hazard pointer. Only nonempty **`hazard_pointer`** objects can be used to protect protectable objects.
- The default constructor of **`hazard_pointer`** constructs an empty object, whereas the free function **`make_hazard_pointer`** constructs a nonempty object.

Motivation

The construction and destruction of multiple nonempty `hazard_pointer` objects in one batch has lower latency than their construction and destruction separately, e.g., 2 ns vs 6 ns for the construction/destruction of 3 nonempty hazard pointers.

Implementation and Use Experience

Batches of hazard pointers have been part of the Folly open-source library (under the name **`hazptr_array`** as a distinct class) and in heavy use in production since 2017.

Batches of Hazard Pointers

The construction and destruction of a nonempty `hazard_pointer` object typically involves access to thread-local storage and has low but non-negligible latency (low single digit nanoseconds). The P2530R3 C++26 hazard pointer interface supports only the construction and destruction of nonempty hazard pointers individually.

This paper proposes adding support for the construction and destruction of multiple nonempty hazard pointers in one batch to reduce the latency of such operations. This revision also adds a batch move function to make it easy for users to keep batches of hazard pointer objects empty or nonempty together.

Proposed Interface

```
/* Takes a span of hazard_pointer objects and makes them nonempty. */
void make_hazard_pointer_batch(std::span<hazard_pointer> span);
```

Effects: Constructs *N* nonempty `hazard_pointer` objects, where *N* is the number of empty elements of `span`.
Postconditions: All elements of `span` are nonempty.
Throws: May throw `bad_alloc` if memory for the hazard pointers could not be allocated.

```
/* Takes a span of hazard_pointer objects and makes them empty. */
void clear_hazard_pointer_batch(std::span<hazard_pointer> span) noexcept;
```

Effects: Moves from the elements of `span` and destroys the moved objects.
Postconditions: All elements of `span` are empty.

```
/* Takes two spans of hazard_pointer objects, and moves one to the other. */
void move_hazard_pointer_batch(std::span<hazard_pointer> span_from,
                               std::span<hazard_pointer> span_to) noexcept;
```

Preconditions: `span_from.size() == span_to.size()`.
Effects: Moves the elements of `span_from` to the corresponding elements of `span_to`.

Usage Example

The following table shows two functionally-equivalent code snippets using the P2530R3 C++26 hazard pointer interface and using hazard pointer batches.

P2530R3 C++26	Hazard Pointer Batches
`{`	`{`
`  hazard_pointer hp[3];`	`  hazard_pointer hp[3];`
`  /* Three hazard pointers are made`	`  /* Three hazard pointers are made`
`     nonempty separately. */`	`     nonempty together. */`
`  hp[0] = make_hazard_pointer();`	`  make_hazard_pointer_batch(hp);`
`  hp[1] = make_hazard_pointer();`	
`  hp[2] = make_hazard_pointer();`	
`  assert(!hp[0].empty());`	`  assert(!hp[0].empty());`
`  assert(!hp[1].empty());`	`  assert(!hp[1].empty());`
`  assert(!hp[2].empty());`	`  assert(!hp[2].empty());`
`  // Use the hazard pointers as usual`	`  // Use the hazard pointers as usual`
`  // src is atomic<T*>`	`  // src is atomic<T*>`
`  T* ptr = hp[0].protect(src);`	`  T* ptr = hp[0].protect(src);`
`  /* etc */`	`  /* etc */`
	`  clear_hazard_pointer_batch(hp);`
	`  /* Three nonempty hazard pointers are`
	`     emptied together. */`
`} /* Three nonempty hazard pointers are`	`} /* The three emptied hazard pointers are`
`     destroyed separately. */`	`     destroyed separately. */`

History

The Varna 2023 plenary voted in favor of including hazard pointers in the C++26 standard library ([2023-06 LWG Motion 7] P2530R3 Hazard Pointers for C++26).

P3135R1 was presented to SG1 in Tokyo 2024, reviewing potential extensions of the P2530R3 C++26 interface, and proposing two of those for inclusion in the standard library. The proposal for extending the P2530R3 C++26 interface to support batch creation and destruction of nonempty hazard pointers was voted on by the SG1 in Tokyo 2024 with unanimous consent:

```
we want to continue work on hazard pointer batches for C++26, except as free-functions
that take and return an existing type of collection (e.g. std::array, or a range,
or...)
```

R0

P3428R0 was a follow up on P3135R1, focusing on extending the C++26 hazard pointer interface to support batch creation and destruction of nonempty hazard pointers, revised to take into account SG1 feedback:

- Use free-functions that take and return an existing type of collection (e.g. std::array, or a range, or...) instead of a new class.

R1

R1 added draft wording to R0. D3428R1 was reviewed by SG1 in Wroclaw 2024 and SG1 voted with unanimous consent to forward (D)P3427R1 to LEWG for C++26 with feedback:

```
Forward (D)P3428R1 to LEWG for C++26 with notes:
* The preconditions don't have to be this strict
* The names reset and move can be changed by LEWG
```

There was also feedback to change the parameter names `span1` and `span2` to `span_from` and `span_to`, and to replace N with `span.size()`.

R2

R2 revises R1 to follow the feedback from SG1 and for review by LEWG:

- Relax the preconditions and update the wording accordingly.
- Change `reset_hazard_pointer_batch` to `clear_hazard_pointer_batch`.
- Change `span1` and `span2` to `span_from` and `span_to`.
- Use `span.size()` instead of *N* where applicable.

References

- P2530R3: Hazard Pointers for C++26 (2023-03-02).
- P3135R1: Hazard Pointer Extensions (2024-04-12).
- Folly: Facebook Open-source Library.