Doc No: P1685R0
Date: 2019-06-14
Audience: LEWG
Authors: Pablo Halpern <phalpern@halpernwightsoftware.com>

Make get/set_default_resource Replaceable

Contents

1 Abstract

This paper proposes that the functions `std::pmr::get_default_resource` and `std::pmr::set_default_resource` be modified to be replaceable functions (similar to `operator new` and `operator delete`) so that the application developer can replace them with more efficient or more flexible alternative implementations.

This proposal is targeted for the C++23 working paper.

2 Change History

2.1 R0 (Pre-Cologne, June 2019)

Initial revision.

3 Motivation and Proposal Overview

The default constructor for `std::pmr::polymorphic_allocator<T>`, obtains a pointer to the *default memory resource* by calling `std::pmr::get_default_resource()`. The default memory resource can be changed by calling `std::pmr::set_default_resource()`. This interface, while providing a simple way for the application owner to choose a default memory-allocation strategy for the program, has a few disadvantages:

1. The get/set interface must use atomic variables to avoid race conditions. The `acquire` barrier on each call to `get_default_resource` can have a negative performance impact on certain architectures.

2. There is no reliable way to set the default resource for objects defined in global scope or namespace scope.
3. A poorly written library can modify the default resource without the knowledge of the application owner, causing serious performance or even correctness problems.

In common usage, the default resource is set at most once at the start of a program, ideally before any use of `get_default_resource`, and is never changed throughout the course of the program.[1] This use case is best supported by defining `get_default_resource()` to simply return the application's preferred default memory resource and defining `set_default_resource()` either to be a no-op or to abort the program (as it could be considered an error to ever call `set_default_resource` in this case).

Another use case would be to have a thread-local default memory resource. In this case, `set_default_resource` would store a value in thread-local storage and `get_default_resource` would retrieve the value from thread-local storage, with no need for atomic operations.

Both of the above use cases, as well as just about any other scenario, can be supported in the standard by allowing the programmer to replace `get_default_resource` and `set_default_resource`. This paper proposes exactly that, borrowing language from other replaceable functions in the standard, namely `operator new` and `operator delete`.

4 Impact on the Standard

Although it changes the attributes of a pair of standard library functions, this proposal should have no effect on existing programs and minimal effect on existing standard library implementations.

5 Potential extensions

The committee might consider extending replaceability to all of the functions that install and retrieve handlers, specifically `set/get_new_handler` and `set/get_terminate`. It is not clear that there is sufficient motivation to do so, as these functions are used much less frequently than `pmr::get_default_resource`. However, if the LEWG decides that this direction is desirable, the author would be willing to update this paper or submit a new paper proposing such a change.

6 Formal Wording

6.1 Document Conventions

All section names and numbers are relative to the **March 2019 C++ Working Paper, N4810.**

[1] This assertion (that the common use case is to set the default resource once) comes from experience with Bloomberg's large codebase. Other common use patterns that may emerge are also supported by the change proposed in this paper.

Existing working paper text is indented and shown in dark blue. Edits to the working paper are shown with ~~red strikeouts for deleted text~~ and green underlining for inserted text within the indented blue original text.

Comments and rationale mixed in with the proposed wording appears as shaded text.

6.2 Changes to set/get_default_resource descriptions

In Section 20.12.4 [mem.res.global], amend the definitions of `set_default_resource` and `get_default_resource` as follows:

The *default memory resource* pointer is a pointer to a memory resource that is used by certain facilities when an explicit memory resource is not supplied through the interface. ~~Its initial value is the return value of `new_delete_resource()`.~~ The value of the default memory resource pointer is retrieved by calling `get_default_resource()` and can usually be set by calling `set_default_resource()`. The library provides default definitions for these functions, which are replaceable. A C++ program shall provide at most one definition of each of these functions. Any such function definition replaces the default version provided in the library (16.5.4.6 [replacement.functions]).

```
memory_resource* set_default_resource(memory_resource* r) noexcept;
```

~~*Effects*: If r is non-null, sets the value of the default memory resource pointer to r, otherwise sets the default memory resource pointer to `new_delete_resource()`.~~

Returns: The previous value of the default memory resource pointer.

~~Remarks: Calling the `set_default_resource` and `get_default_resource` functions shall not incur a data race. A call to the `set_default_resource` function shall synchronize with subsequent calls to the `set_default_resource` and `get_default_resource` functions.~~

Replaceable: A C++ program may define a function with this function signature, and thereby displace the default version defined by the C++ standard library.

Required behavior: Return the previous value of the default memory resource pointer or else terminate the program. [*Note:* There is no requirement that a replacement function modify the default memory resource pointer in any way – *end note*]

The required behavior does not require much – not even that the set/get functions be thread safe (not incur races). It is quite possible that `set_default_resource` would be called only before multiple threads were started. Thus, certain replacement versions of these functions could depend on the program to avoid races, rather than guaranteeing thread-safe behavior themselves.

Default behavior: If r is non-null, sets the value of the default memory resource pointer to r, otherwise sets the default memory resource pointer to `new_delete_resource()`. Calling the `set_default_resource` and `get_default_resource` functions shall not incur a data race. A call to the `set_default_resource` function shall synchronize with subsequent calls to the `set_default_resource` and `get_default_resource` functions.

```
memory_resource* get_default_resource() noexcept;
```

Returns: The current value of the default memory resource pointer.

Replaceable: A C++ program may define a function with this function signature, and thereby displace the default version defined by the C++ standard library.

Required behavior: Return a non-null pointer to a memory resource, whether or not `set_default_resource()` has previously been called.

Default behavior: Return the value set by the most recent call to `set_default_resource()`, if any; otherwise the value returned by `new_delete_resource()`.

6.3 Changes to Replacement Functions section

Amend section 16.5.4.6 [replacement.functions] as follows:

A C++ program may provide the definition for any of the following dynamic memory allocation function signatures declared in header <new> (6.6.5.4, 17.6):

```
operator new(std::size_t)
operator new(std::size_t, std::align_val_t)
…
operator delete[](void*, const std::nothrow_t&)
operator delete[](void*, std::align_val_t, const std::nothrow_t&)
```

or any of the following default-resource management function signatures declared in header <memory_resource> (20.12.4 [mem.res.global]):

```
std::pmr::memory_resource*
  std::pmr::set_default_resource(memory_resource* r) noexcept

std::pmr::memory_resource* std::pmr::get_default_resource() noexcept
```

The program's definitions are used instead of the default versions supplied by the implementation (17.6, 20.12.4). Such replacement occurs prior to program startup (6.2, 6.8.3). The program's declarations shall not be specified as inline. No diagnostic is required.