`reference_wrapper` Associations

Document Number: P2380R1
Date: 2021-06-03
Reply-to: Robert Leahy <rleahy@rleahy.ca>
Audience: LEWG

Abstract

This paper proposes that `associated_allocator` and `associated_executor` have specializations for `reference_wrapper<T>`.

Background

The Networking TS [1] provides “associators” (§13.2.6 [async.reqmts.associator]) (`associated_allocator` and `associated_executor`) which allow types and instances of certain named typed requirements (`ProtoAllocator` and `Executor` respectively) to be retrieved through a completion handler. The asynchronous model proposed by the Networking TS uses these associators to obtain executors and allocators for use in its operations.

`reference_wrapper` has been a part of standard C++ since C++11 and allows references to be transported inside a wrapper which behaves as one would expect a C++ class to: Assignable, et cetera. Moreover if the target of a `reference_wrapper` models the named type requirement `Callable` then `reference_wrapper` itself models this named type requirement.

Motivation

The standard provides `reference_wrapper` to enable the use of reference semantics with `Callable` objects where the algorithms, types, et cetera in question are written with value semantics (for example the standard algorithms accept their predicates and operations by value). The fact that the Networking TS doesn’t provide specializations of `associated_allocator` and `associated_executor` for `reference_wrapper<T>` means that `reference_wrapper` can’t fill this niche out of the box when interacting with Networking TS operations with completion handlers which customize the associated `ProtoAllocator` and/or `Executor`.

Being unable to use `reference_wrapper` in these situations is the best case scenario. More problematic is the possibility that users (accustomed to reaching for `reference_wrapper` when they need to pass a `Callable` by reference) will be unaware of the fact that the Networking TS does not provide the requisite specializations and will use `reference_wrapper` in such situations notwithstanding. Particularly where the `Executor` association has been customized this would likely lead to the user unknowingly writing incorrect code: Their synchronization and/or execution requirements would not be honored which could be the difference between their program being data race free and containing undefined behavior.

Proposed Changes

§13.1 [async.synop]:

```
template<class T, class ProtoAllocator = allocator<void>>
  struct associated_allocator;

template<class T, class ProtoAllocator>
  struct associated_allocator<reference_wrapper<T>, ProtoAllocator>;
```

[...]

```
template<class T, class Executor = system_executor>
  struct associated_executor;

template<class T, class Executor>
  struct associated_executor<reference_wrapper<T>, Executor>;
```

§13.5 [async.assoc.alloc]

```
namespace std {
namespace experimental {
namespace net {
inline namespace v1 {

  template<class T, class ProtoAllocator = allocator<void>>
  struct associated_allocator
  {
    using type = see below ;

    static type get(const T& t, const ProtoAllocator& a = ProtoAllocator())
      noexcept;
  };

  template<class T, class ProtoAllocator>
  struct associated_allocator<reference_wrapper<T>, ProtoAllocator>
  {
    using type = associated_allocator_t<T, ProtoAllocator>;
    static type get(reference_wrapper<T> t, const ProtoAllocator& a =
      ProtoAllocator()) noexcept;
  };

} // inline namespace v1
} // namespace net
} // namespace experimental
} // namespace std
```

§13.5.2 [async.assoc.alloc.refwrap]

```
type get(reference_wrapper<T> t, const ProtoAllocator& a = ProtoAllocator())
  noexcept;
```

Returns: `associated_allocator<T, ProtoAllocator>::get(t.get(), a)`.

§13.12 [async.assoc.exec]

```
namespace std {
namespace experimental {
namespace net {
inline namespace v1 {

  template<class T, class Executor = system_executor>
  struct associated_executor
  {

    using type = see below ;

    static type get(const T& t, const Executor& e = Executor()) noexcept;
  };

  template<class T, class Executor>
  struct associated_executor<reference_wrapper<T>, Executor>
  {

    using type = associated_executor_t<T, Executor>;

    static type get(reference_wrapper<T> t, const Executor& e = Executor())
      noexcept;
  };

} // inline namespace v1
} // namespace net
} // namespace experimental
} // namespace std
```

§13.12.2 [async.assoc.exec.refwrap]

`type get(reference_wrapper<T> t, const Executor& e = Executor()) noexcept;`

Returns: `associated_executor<T, Executor>::get(t.get(), e)`.

References

[1] J. Wakely. Working Draft, C++ Extensions for Networking N4771

Revision History

Revision 1

- Corrected mistakes in specifying the specializations of `associated_allocator` and `associated_executor`
- The `type` member of the specializations of `associated_allocator` and `associated_executor` are now specified in terms of `associated_allocator_t` and `associated_executor_t`
- Corrected minor editorial issue within a comment

Review History

SG4 Teleconference June 3, 2021

Draft of revision 1 was presented to SG4. The following poll was taken and passed with unanimous consent:

Forward P2380R1 to LEWG to merge into the SG4 networking draft?