is_nothrow_connectable_in

Document Number: P4191R0
Date: 2026-04-17
Reply-to: Robert Leahy <rleahy@rleahy.ca>
Audience: SG1, LEWG

Abstract

This paper proposes adding two type traits to the `std::execution` namespace which allow the determination of whether `std::execution::connect` throws for a certain sender in a certain environment.

Background

Under the original design of `std::execution` [1] one could check whether or not `std::execution::connect` threw an exception via:

```
noexcept(
  std::execution::connect(
    std::declval<Sndr>(),
    std::declval<Rcvr>()))
```

Which is verbose, difficult to read, and requires the concrete type of the sender and receiver to be known. Notably the concrete type of the receiver may not be known when the sender is asked to determine its completion signatures despite the fact the throwingness of some nested operation might influence the completion signatures reported.

The above motivated a change [2] which allowed the throwingness of `std::execution::connect` to be determined given only a sender and an environment via, for example:

```
template<typename Env>
struct receiver_archetype {
  using receiver_concept = std::execution::receiver_tag;
  // set_value, set_error, and set_stopped
  Env get_env() const noexcept;
};

template<typename Sndr, typename Env>
constexpr bool is_nothrow_connectable_in_v = noexcept(
  std::execution::connect(
```

```
    std::declval<Sndr>(),
    std::declval<receiver_archetype<Env>>()));
```

However no utility to do this was proposed leaving users to roll their own [3].

Discussion

The standard provides a type alias to determine the operation state type obtained by `std::execution::connect`: `std::execution::connect_result_t`. It seems only natural to provide type traits to support other routine interrogations of said customization point object.

Since users may find themselves either with an environment or a receiver both forms should be supported:

- `std::execution::is_nothrow_connectable_in`: Accepts a sender and an environment
- `std::execution::is_nothrow_connectible_to`: Accepts a sender and a receiver

Wording

33.4 [execution.syn]

[...]

```
  // [exec.connect], the connect sender algorithm

  struct connect_t;
  inline constexpr connect_t connect{};

  template<class Sndr, class Rcvr>
    using connect_result_t =
      decltype(connect(declval<Sndr>(), declval<Rcvr>()));

  template<class Sndr, class Rcvr>
    constexpr bool is_nothrow_connectible_to_v =
      is_nothrow_invocable_v<connect_t, Sndr, Rcvr>;

  template<class Sndr, class Rcvr>
    struct is_nothrow_connectible_to
      : is_nothrow_invocable<connect_t, Sndr, Rcvr> {};

  template<class Env>
    struct receiver-archetype {
      using receiver_concept = receiver_tag;
```

```
      template<typename... Args>
      void set_value(Args&&...) && noexcept;
      template<typename E>
      void set_error(E&&) && noexcept;
      void set_stopped() && noexcept;
      Env get_env() const noexcept;
    };

  template<class Sndr, class Env>
    constexpr bool is_nothrow_connectable_in_v =
      is_nothrow_connectible_to_v<Sndr, receiver-archetype<Env>>;

  template<class Sndr, class Env>
    struct is_nothrow_connectable_in
      : is_nothrow_connectible_to<Sndr, receiver-archetype<Env>> {};
```

[...]

Implementation Experience

nVidia's stdexec provides a `__nothrow_connectable` concept [3] and combines it with an archetype receiver [4] to achieve the effect of the traits proposed by this paper.

Acknowledgements

The author would like to thank Ian Petersen and Dietmar Kühl for suggesting that "connectable" be spelt inconsistently between the two traits to see if anyone in LEWG notices.

References

[1] M. Dominiak et al. std::execution P2300R10
[2] R. Leahy. When Do You Know `connect` Doesn't Throw? P3388R3
[3] https://github.com/NVIDIA/stdexec/blob/91782e6cbfad5df74237bac139b5d61f6cf40313/include/stdexec/__detail/__connect.hpp#L269-L271
[4] https://github.com/NVIDIA/stdexec/blob/91782e6cbfad5df74237bac139b5d61f6cf40313/include/stdexec/__detail/__receivers.hpp#L253-L283