NetTS, ASIO and Sender Library Design Comparison
Draft Proposal

Document #: P2471R0
Date: 2021-10-06
Project: Programming Language C++
Audience: LEWG Library Evolution
SG1 Concurrency and Parallelism
SG4 Networking
Reply-to: Kirk Shoop
<kirk.shoop@gmail.com>

Contents

1 Changes

2 Introduction

3 Notes

4 Tables

4.1 Asynchronous Operation design
4.2 Initiating function design
4.3 Algorithm design
4.4 Associated values design
4.5 Executor design
4.6 Execution Context design

5 References

1 Changes

— first revision

2 Introduction

I have never seen the library designs of these libraries compared. This paper started as an email to the LEWG reflector. I was asked to make it a paper.

I used the following papers to fill in these tables:

— [P1322R3] “Networking TS enhancement to enable custom I/O executors”
— [P0958R3] “Networking TS changes to support proposed Executors TS”
— [P1943R0] “Networking TS changes to improve completion token flexibility and performance”
— [P2444R0] “The Asio asynchronous model”
— [P2300R2] “std::execution”

I also searched in the ASIO repo
Corrections are welcome, especially for the ASIO and NetTS portions. I wish they had built something like this already.

I split these out into vertical tables because horizontal scrolling sucks.

The library designs compared in tables below are:

- Asynchronous Operation design
- Initiating function design
- Algorithm design
- Associated Values design
- Executor design
- Execution context design

Within each section are three tables:

- [N4771] as it currently stands.
- [ASIO] as it currently stands.
- [P2300R2] as it currently stands

## 3 Notes

The theme that revealed itself to me while compiling these is that ASIO & NetTS use traits and partial-specialization vs. Sender/Receiver use concepts and CPOs.

There are other disparities in specific places, but the approach to library design traits/concepts and specialization/CPOs are the repeated differentiators I saw.

Another thing that these tables revealed is all the changes in the ASIO design in the last 2-3 years (after sender/receiver was proposed).

## 4 Tables

### 4.1 Asynchronous Operation design

**Table 1: NetTS - (N4771 is missing P1322, P0958, P1943, P2444)**

13.2.7 Requirements on asynchronous operations

```cpp
concept completion_token:
    async_result<
        completion_token,
        signature>
    ::completion_handler_type;

async_result<
    completion_token,
    signature>
::return_type;

concept signature:
    (ErrorsAndValues...) -> void;

concept completion_handler_type:
    constructible<
        completion_handler_type,
        completion_token>;
    invocable<
        completion_handler_type,
        signature>;

concept result_type:
    constructible<
        result_type,
        completion_handler_type>;
```
Table 2: ASIO - (ASIO has P1322, P0958, P1943, P2444)

```plaintext
concept completion_token:
    async_result<
        completion_token,
        signature...>
::initiate(
    initiation,
    completion_token,
    Args...) -> Result;

concept signature:
    (ErrorsAndValues...) -> void;

concept initiation:
    (completion_handler, Args...) -> Result;

concept completion_handler:
    constructible<
        completion_handler,
        completion_token>;
    invocable<
        completion_handler,
        signature>...;
```

Table 3: Sender/Receiver - (P2300)

```plaintext
concept sender:
    connect(sender, receiver) -> operation_state;

concept operation_state:
    start(operation_state) -> void;

concept receiver:
    set_value(receiver, Values...) -> void;
    set_error(receiver, Error) -> void;
    set_done(receiver) -> void;
```

4.2 Initiating function design
13.2.7 Requirements on asynchronous operations

Any function that takes a completion_token as the last argument, and returns:

```
(..., completion_token)
  -> decltype(async_result<
      completion_token,
      signature>::result_type(
        async_result<
          completion_token,
          signature>::completion_handler_type(
            completion_token)))
```

Table 5: ASIO - (ASIO has P1322, P0958, P1943, P2444)

Any function that takes a completion_token as the last argument, and returns the result of:

```
(..., completion_token)
  -> decltype(async_result<
      completion_token,
      signature...>::initiate(
        initiation,
        completion_token,
        Args...))
```

Table 6: Sender/Receiver - (P2300)

Any function returning a sender

```
(...) -> sender;
```

4.3 Algorithm design
Unspecified, but without async_initiate the only option I know of is:

```
concept algorithm:
  (completion_token) -> completion_token;
```

Happy to be corrected.

---

Includes the above and:

Any specific completion_token can define a new composable_type and return that.

The deferred completion_token is an example of this.

One of the infinite possible shapes for that new composable_type could be:

```
concept algorithm
  (composable_type) -> composable_type;

concept composable_type:
  (completion_token) -> Result;
```

---

```
concept algorithm:
  (sender) -> sender
```

4.4 Associated values design
### 13.2.2 Executor requirements

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>associated_executor</td>
<td><code>associated_executor&lt;Source, Default&gt;::type;</code>&lt;br&gt;<code>associated_executor&lt;Source, Default&gt; ::get(source, default) -&gt; executor; // static</code></td>
</tr>
<tr>
<td>associated_allocator</td>
<td><code>associated_allocator&lt;Source, Default&gt;::type;</code>&lt;br&gt;<code>associated_allocator&lt;Source, Default&gt; ::get(source, default) -&gt; allocator; // static</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>associated_cancellation_slot</td>
<td><code>associated_cancellation_slot&lt;Source, Default&gt;::type;</code>&lt;br&gt;<code>associated_cancellation_slot&lt;Source, Default&gt; ::get(source, default) -&gt; cancellation_slot; // static</code></td>
</tr>
</tbody>
</table>

### 4.5 Executor design

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler_provider</td>
<td><code>get_scheduler(scheduler_provider) -&gt; scheduler;</code></td>
</tr>
<tr>
<td>allocator_provider</td>
<td><code>get_allocator(allocator_provider) -&gt; allocator;</code></td>
</tr>
<tr>
<td>stop_token_provider</td>
<td><code>get_stop_token(stop_token_provider) -&gt; stop_token;</code></td>
</tr>
</tbody>
</table>
13.2.2 Executor requirements

```cpp
concept executor:
    executor::context() -> execution_context;
    executor::on_work_started() -> void;
    executor::on_work_finished() -> void;
    executor::dispatch((), Allocator) -> void;
    executor::post((), Allocator) -> void;
    executor::defer((), Allocator) -> void;
```

13.2.3 Execution context requirements

```cpp
concept execution_context:
    execution_context::executor_type;
    execution_context::get_executor() -> execution_context::executor_type;
```

4.6 Execution Context design

```cpp
concept scheduler:
    schedule(scheduler) -> sender;
```

Table 13: NetTS - (N4771 is missing P1322, P0958, P1943, P2444)
5 References

[ASIO] Christopher Kohlhoff. ASIO github.
  https://github.com/chriskohlhoff/asio

  https://wg21.link/n4771

  https://wg21.link/p0443r14

[P0958R3] Christopher Kohlhoff. 2021-03-15. Networking TS changes to support proposed Executors TS.
  https://wg21.link/p0958r3

  https://wg21.link/p1322r3

[P1943R0] Christopher Kohlhoff. 2019-10-07. Networking TS changes to improve completion token flexibility
  and performance.
  https://wg21.link/p1943r0

  Adelstein Lelbach. std::execution.
  https://wiki.edg.com/pub/Wg21telecons2021/LibraryEvolutionWorkingGroup/P2300R2.html

  https://wg21.link/p2444r0