Executors Naming

Document #:	P2213R0
Date:	August 16, 2020
Project:	Programming Language C++
Audience:	LEWG
Authors:	Amir Kirsh <kirshamir@gmail.com>
	Inbal Levi <sinbal2l@gmail.com>
	Dan Raviv <dan.raviv@gmail.com>
	Ran Regev <regev.ran@gmail.com>
	Dvir Yitzchaki <dvirtz@gmail.com>
	Andrei Zissu <andrziss@gmail.com>
Contributors:	Yehezkel Bernart <yehezkelshb@gmail.com>
	Michael Peeri <michael.peeri@protonmail.com>
Reply-to:	Amir Kirsh <kirshamir@gmail.com>

1 Motivation

We have received indications from the authors of P0443 that name suggestions are welcome. We believe that some of the names in the current proposal are confusing and would like to propose alternative names that, as we believe, better express the essence of the operations or entities. The paper collects comments made by different members of the Israeli NB, on names which appear in executors proposal [P0443R13] and in algorithms for executors [P1897R3].

Educational purposes will also be best served by names which make the most sense - this being especially important since executors clearly present a steep learning curve.

2 Name Suggestions

We believe that names should be clear to the users of a library, the same as they are for the initial creators of the library, and therefore propose reconsidering some of the names related to executors library, as describe in the alternatives below.

2.1 ::connect

2.1.1 Proposed alternatives:

::pipe

2.1.2 Reasoning:

The action is to prepare a ‘pipe‘ of a sender and a receiver, whereas the term ‘connect‘ expresses the technical operation but not the semantics of the operation. Moreover, connect can be bidirectional whereas pipe is unidirectional, like in Linux and in ranges, same as the actual operation performed by connecting a sender to a receiver.

2.2 ::submit

2.2.1 Proposed alternatives:

::start

2.2.2 Reasoning:

Creating different customization point for ‘::connect‘ and ‘::start‘ is advocated in [P2006R1]. However, some of us believe that for the purpose of readability, this should be reconsidered, since there’s a value to using start operation for a chain of tasks with or without prior preparation. The verb ’start’ seems to fit, as the entity to which we submit the task is hidden, whether the action performed is to start a chain of tasks.

Note: start would need overloaded operator() for two use cases: getting sender and receiver OR getting a prepared chain of tasks via ::connect / ::pipe.

2.3 operation_state

2.3.1 Proposed alternatives:

operation

Other names that were considered here are: ‘packaged_operation‘, ‘startable‘.

2.3.2 Reasoning:

The concept ‘operation_state‘ manages the chain of tasks and not only their “state”. Moreover, it does not manage a state but rather only propagates it.

2.4 sender, receiver

2.4.1 Proposed alternatives:

producer, consumer

2.4.2 Reasoning:

Executors jointly provide two different functionalities/capabilities: - running operations in an execution context - being able to pipe operations.

We think that it would be better to use a name related to the piping operation in order to emphasize the second functionality. We also think that ‘producer‘ is better at capturing cases in which the sender doesn’t send anything (‘set_value‘ sets ‘void‘ value) and only has side-effects. We think "sender" may indicate that something is being sent, while other names may suite better by being more neutral.

2.5 ::set_done

2.5.1 Proposed alternatives:

::set_canceled.

Another name that was considered is: ::set_stopped

2.5.2 Reasoning:

Some of us believe the name ‘set_done‘ is confusing as it is usually called when the operation was not actually done but rather stopped / canceled. Our understanding is that the ‘set_done‘ name was left over from a previous version where it was allowed to be called even after a successful ‘set_value‘ call. As in the current P0443 version, it is used only to convey cancellation, it should be renamed accordingly. We realize that ‘set_done‘ can be called in case the operation was fulfilled in another context, but we believe ::set_stopped or ::set_canceled would handle this scenario well, while also handling actual cancellation / stop of the pipe (when the operation was not fulfilled). The addition of the prefix ‘set_‘ as opposed to just ‘execution::stop‘ or ‘execution::cancel‘ is in order to make it conform with ‘set_value‘ and ‘set_error‘.

2.6 scheduler (concept)

2.6.1 Proposed alternatives:

execution_context

2.6.2 Reasoning:

The ‘scheduler’ concept does not deal with scheduling but rather with an execution context being passed around. As stated in P0443 - a scheduler is a factory for creating senders from a known execution context.

2.7 scheduler::schedule

2.7.1 Proposed alternatives:

::create_sender / ::create_producer

(based on the name that would be chosen for the ‘sender‘ concept)

2.7.2 Reasoning:

The operation create a sender from a given execution context. Similarly to the previous section, We believe the operation name ‘schedule‘ is confusing, as it does not schedule something but rather creates a sender.

2.8 thread_pool::scheduler

2.8.1 Proposed alternatives:

thread_pool::execution_context

2.8.2 Reasoning:

The function doesn’t preform scheduling, but rather returns the execution context of the thread pool to be passed on for later execution of tasks in that context.

2.9 thread_pool::attach

2.9.1 Proposed alternatives:

thread_pool::attach (e.g. **keep as is**)

Another name that was considered here: thread_pool::attach_this_thread

2.9.2 Reasoning:

An alternate name suggested in P0443 for this function is ‘join()‘. We believe that the name ‘attach‘ better describes the purpose, as the method attaches the current thread to the pool, while ‘join‘ usually means the opposite: waiting for the object being called upon to join the current thread. The other proposed alternative ‘thread_pool::attach_this_thread‘ is a more verbose option.

2.10 just (from P1897R3)

2.10.1 Proposed alternatives:

send / produce

(based on the name that would be chosen for the ‘sender‘ concept)

2.10.2 Reasoning:

The operation is to create a sender for forwarding arguments, the proposed name tries to capture the meaning of this operation. We feel that ‘just‘ is just not saying much. We are aware that this call actually creates a sender that forwards its parameters and does not perform the send by itself, however the semantics is for sending / producing parameters down the pipe.

Note: same decision should apply also to ‘just_on‘.

2.11 let_value, let_error (from P1897R3)

2.11.1 Proposed alternatives:

upon_value, upon_error

Other names considered here: value_handler, error_handler OR if_value, if_error

2.11.2 Reasoning:

This customization point is a sender adapter for assigning value / error case, we believe the proposed names would be more fluent in the piping syntax. The alternative 'on_' already has different meaning in the paper, thus we propose the prefix 'upon_' or the other alternatives. The semantics of 'let_' seem less expressive in our opinion.

3 References

[1] P0443R13: A Unified Executors Proposal for C++
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p0443r13.html

[2] P1897R3: Towards C++23 executors: A proposal for an initial set of algorithms
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1897r3.html

[3] P2006R1: Eliminating heap-allocations in sender/receiver with connect()/start() as basis operations
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2006r1.pdf