Abstract

This paper proposes the removal of system_executor and system_context from the Networking TS.

Background

The Networking TS [1] provides two related classes: system_executor and system_context. system_executor satisfies the named requirements DefaultConstructible and Executor. system_context satisfies the named requirement ExecutionContext.

The type alias system_context::executor_type names system_executor. The member function system_context::get_executor returns a default constructed system_executor. The member function system_executor::context returns a reference to a system_executor with static storage duration. Note that system_executor::context need not return a system_context which is a “Meyer’s singleton” [2].

The only direct use of system_context in the Networking TS is in the specification of system_executor.

system_executor is directly used by the Networking TS in the specification of associated_executor and associated_executor_t where it is used as a “fallback” executor. This means that the unary versions of dispatch, defer, and post may make use of this type.

During review of the design of the asynchronous model of the Networking TS [3] concerns were raised about the implication and usage of system_context::stop and system_context::join. This review and those concerns were discussed in SG4 in Prague 2020.

Motivation

Global variables have been considered an antipattern for some time [4]. Hiding an object with static storage duration (the singleton instance of system_context) behind a member function of
a stateless object (system_executor::context) does not change the fact that it is morally a
global variable. This has been an established issue with the singleton pattern for some time.

system_context is not just a global variable. The reference to the singleton instance obtained
by way of system_executor::context is not const. system_context has non-const member
functions. The interface of system_executor allows work items (which must be persisted et
cetera) to be submitted to the singleton instance of system_context. Therefore the singleton
instance of system_context is mutable global state.

system_context is more than just state. It may or may not manage running threads, and may
or may not have work. Establishing an answer to the question of whether or not it has threads,
and whether or not it has work requires a user to interact with the singleton system_context.
This elegantly illustrates the problem with global state: The user has to know that they have to
interact with system_context. The requirement to interact with system_context to have
well-defined behavior during program shutdown is imposed depending on whether or not any
component has interacted with the system_context. Due to the fact any component can obtain
a reference to the singleton instance there’s no dependency injection channel forcing such
components to “document” the fact that they interact with this object. Due to the fact the
singleton instance of system_context is not a “Meyer’s singleton” its construction and
destruction order is not necessarily well-specified with respect to other objects which may
interact with it.

Since system_executor is the default second template parameter to associated_executor,
and since the second parameter of associated_executor::get is a default constructed
system_executor it is straightforward for users to accidentally use this type. All that is required
is that they not associate a completion handler with an executor and in the absence of an I/O
executor (§13.2.7.8 [async.reqmts.async.io.exec]) system_executor will be used to run the
completion handler.

Associated Executor

Without a major overhaul of the associated_executor facility (beyond the scope of this paper)
something must replace system_executor.

This paper proposes inline_executor as a replacement. Instances of this type invoke function
objects submitted via dispatch and defer in a blocking manner and throw on any attempt to
submit work via post.

Since the Executor named requirement from the Networking TS requires that x1.context()
(where x1 is a possibly const Executor) be well-formed and return E& where E satisfies the
ExecutionContext named requirement this paper also proposes the class inline_context.
Implementations

The author has implemented this paper against “standalone” Asio [5].

Acknowledgements

The author would like to thank Chris Kohlhoff for his support in exploring this design space.

References

[5] https://github.com/RobertLeahy/asio/tree/c63a4d899735f9af1be179851a1f108bafe5a39