

Document number: P0940R1

Date: 20180507 (RAP)

Project: Programming Language C++, WG21, SG1, LEWG, LWG

Authors: Michael Wong, Olivier Giroux

Email: michael@codeplay.com, ogiroux@gmail.com

Reply to: michael@codeplay.com

Concurrency TS is growing: Concurrent Utilities and Data Structures

Change log	1
Introduction	1
Organization Proposal	2
Proposed Structure	2
Acknowledgement	4
References	4

Change log

- R0 - initial proposal
- R1 - removed Shared_ptr based on JAX discussion
 - Changed Contention to Synchronization
 - Import in thread library which has futures to replace the original futures section
 - Rebase this to latest Draft N4741

Introduction

This is a proposal for a draft new section to the C++ Standard to support SG1 Concurrency features. We foresee a number of upcoming features for inclusion. We also foresee some features in existing sections that are concurrency related that is worth moving into this new section.

There is no wording yet until we agree on the structure of the reorganization.

Organization Proposal

A large number of Concurrency features are coming for C++20. This is because Concurrency TS1 was not added to C++17. However some of the features in it has changed. There are also many new features aiming for Concurrency TS2. Lets us recap.

Concurrency TS1 was published in Jan 19, 2016[P0159] but still too late for C++17. It contains

- `atomic_shared_ptr` and `atomic_weak_ptr` class templates
- Latches and barriers
- Improvements to `std::future<T>` and Related APIs

Since its publication and through usage feedback, several of these facilities have been rethought. In a recent SG1 meeting in Toronto, `Atomic_shared_ptr` is now `atomic<shared<ptr>>`. Latches and barriers is undergoing a partial redesign to split the arrive/wait facilities. Even futures is being redesigned to serve the needs of executors, TLS, and other facilities better.

Concurrency TS2 is an ongoing WIP but should contain the following which has been making its way through WG21/SG1:

- Executors that links concurrency and parallelism constructs with different execution resources. There is a possibility that this may split off into its own TS.
- Constructs that deal with contention such as `apply()`, `latch`, `basic_barrier<F>`, `typedef barrier`, `binary_semaphore`, `counting_semaphore`, `synchronic`
- Data structures such as Concurrent queues, counters, `Synchronized<T>`, `Atomic_ref<T>`
- Several synchronization primitives for locked-free programming on concurrent data structures. These are `cell`, `hazard_ptr` and `RCU`. These extends the existing `shared_ptr` and the proposed `atomic_shared_ptr` which all have safe reclamation facilities. As such we also propose moving `shared_ptr` and `atomic<shared<ptr>>` to this new location. We suspect this part may be controversial, so would ask for discussion on this topic.

Given the proliferation of these and other facilities, as Concurrency TS editor, and before we move sections and inject new wordings, we propose the following new chapter to handle these concurrency utilities for Concurrency TS2 and TS1.

At this point, there is no plan to change or update Concurrency TS1. However, not all may agree with that. We would also invite a discussion on this in the upcoming meeting.

We rebase the discussion to the latest Draft N4741. From JAX review there was consensus to continue in this direction, but also

- Bring in thread support library. This effectively restructures Clause 33 Threads Support
- Keep `shared_ptr` where it currently is, but move `atomic<shared_ptr>` from Clause 23.

Proposed Structure

Clause 33: Concurrency Utilities Library

- **33.1 Concepts**
 - 33.1.1 Thread Support
 - 33.1.2 Executor Support
- **33.2 Thread Support**
 - 33.2.1 General
 - 33.2.2 Requirements
 - 33.2.3 Threads
 - 33.2.4 Mutual exclusion
 - 33.2.5 Condition variables
 - 33.2.6 Futures
- **33.3 Executor Support**
 - 33.3.1 Executors in-depth
- **33.4 Synchronization Contention**
 - 33.4.1 `apply()`
 - 33.4.2 `latch`
 - 33.4.3 `basic_barrier<F>`
 - 33.4.4 `typedef barrier`
 - 33.4.5 `binary_semaphore`
 - 33.4.6 `counting_semaphore`
 - 33.4.7 `synchronic`
- **33.5 Data structures**
 - 33.5.1 Concurrent queue
 - 33.5.2 Concurrent counters
 - 33.5.3 `Synchronized<T>`
 - 33.5.4 `Atomic_ref<T>`
- **33.6 Safe Reclamation**
 - 33.6.1 Atomic specialization for `shared_ptr`
 - 33.6.2 `Cell`
 - 33.6.3 `RCU`
 - 33.6.4 Hazard Pointers

The reason I am interested in moving `atomic<shared_ptr>` into the section on concurrency in some order with Safe Reclamation is that they are actually shared concurrency structures. `Shared_ptr` exists where it does (Clause 23.11 Smart Pointer) because at the time, it was delivered with the Boost Smart pointer as a package. At the JAX meeting, there was consensus to not include it here. In this paper [P0233], the authors illustrate in the table in Section 7 a comparison of the capabilities between the various facilities for Reclamation. Reference Counting is the implementation behind `shared_ptr` and Split reference Counting (or Reference Counting with DCAS) is the implementation behind `atomic_shared_ptr`. These have many capabilities similar to Hazard Pointers, Cell and RCU differing only in the performance and lock-free implications.

We would ask SG1 to give guidance on this structure reorganization at the next meeting.

Acknowledgement

The author wishes to thank Maged Michael and Paul Mckenney for the comparison and early feedback. We thank SG1 for feedback.

References

[P0159] Programming Languages — Technical Specification for C++ Extensions for Concurrency
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0159r0.html>

[P0233] Hazard Pointers: Safe Reclamation for Optimistic Concurrency
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0233r6.pdf>