Document Number: P0794R0
Date: 2017-10-16
Authors: Michael Wong
Project: Programming Language C++, SG14 Games Dev/Low Latency/Financial Trading/Banking/Simulation/Embedded
Reply to: Michael Wong <michael@codeplay.com>

SG14: Low Latency Meeting Minutes 2017/08/09-2017/10/11

Contents

Minutes for 2017/08/09 SG14 Conference Call

Minutes by Michael

1.1 Roll call of participants

Billy Baker, Carl Cook, guy Davidson, John McFarlane, Michael Wong, Patrice Roy, Ronan Keryell, Samantha Luber, Steffan Tjernstrom, Vishal

1.2 Adopt agenda

Approved

1.3 Approve minutes from previous meeting, and approve publishing previously approved minutes to ISOCPP.org

Approved.

1.4 Action items from previous meetings

2. Main issues (125 min)

2.1 General logistics

CPPCON talks

https://groups.google.com/a/isocpp.org/forum/#!topic/sg14/IQU-Zp8iJ8Y

CPPCON proposals

-any other F2F

2.2 Paper reviews

2.2.1 A better alloc

The C proposal is here:

http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1085.htm

The C++ part is here:

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1953.html

http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1527.pdf

Can delete function specify an allocation size? Good for some of the allocators to supply the size.

will ask for Niall and Howard at CPPCON.

improvement plugs gaps in Std Library

2.2.2 any other proposal for reviews?

Arthur: fancy pointers

John: some interest

Arthur: may be a position paper

John: talk of putting allocators into language to preclude fancy pointers

John McFarlane: strategy for the numerics fixed point, and other numerics

Patrice: SG12: to use memcpy to start the lifetime of objects

co-author with JF cast on struct, also mark memory location on when life time gets started

Michael: working with Nat on TLS

Affinity,

channels, pipelines

Carl Cook to join heterogeneous C++ group

Vishal: Vulkan is interested in looking at this exception handlign in a concurrent environment (as isOpenCL/SYCL/HSA/ParallelSTL)

2.3 Domain-specific discussions

2.3.1 Embedded domain discussions

SG12 is now safety vulnerability study groups

2.3.3 Games Domain

2.3.4 Finance Domain

2.4 Other Papers and proposals

1 Additions: Intrusive smart pointer, Isabella Muerte

not online

2 inplace functions: Nicolas Flueery

Carl Cook: busy so far but will get on soon

3 Fixed point real numbers, John McFarlane

Toronto: was not talked about, has been forwarded to LEWG

not updated for Toronto

but wrote brief paper on library additions

might be able to adapt after seeing how meetings work

Ronan: interested in a bit-width internship at Xilinx, based on what is already opensource

present at CPPCON F2F

4 Ring span, Guy Davidson

Concurrent ring span deferred to concurrent queue,

well received. LEWG, reframed for next call,

present at CPPCON F2F

positive vote to come back for another iteration

currently working on 2D graphics API

5 Hazard Pointers + RCU Maged + Paul

HP moves to wording, RCU contineus work

HP is moving to LEWG, RCU to follow. Thanks to all who helped.

6 Thread constructor attributes, Patrice

SG1 recommends we study 2 approaches a) user control moment when threads are created

Make thread was discussed in Toronto, should we let user create threads with platform functionality, using nonexceptional way on when failure occurred

7 Intrusive containers, Guy Davidson

Paired with Hal Finkel, waiting for more time

8 plf colony/stack, plflib.org Matt Bentley

Guy Davidson presented this, need some rewrites, more use cases, motivations

Jeffrey Yaskin to help

colony is not the best name,is it a bag?

Carl Cook might work with him in New Zealand

9 Likely/unlikely

Clay,was it presented Toronto

10 Comparing virtual Functions , Scott Wardle

Scott is coming to CPPCON.

11. Heterogenous computing: Channels and Managed pointers

paper for context has affinity configuration, but not binding or affinity

https://github.com/kokkos/ISO-CPP-Papers/blob/master/P0737_ExecContext.rst

https://groups.google.com/forum/#!forum/hetero-c

call every other Monday at 10:30 ET

2.5 Future F2F meetings:

CPPCON Sept 27

2.6 future C++ Standard meetings:

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4633.pdf

2017-11 Albuquerque WG21 meeting information

Jacksonville : March 12

rapperswill: June 4-9

3. Any other business

Reflector

https://groups.google.com/a/isocpp.org/forum/?fromgroups=#!forum/sg14

As well as look through papers marked "SG14" in recent standards committee paper mailings:

http://open-std.org/jtc1/sc22/wg21/docs/papers/2015/

http://open-std.org/jtc1/sc22/wg21/docs/papers/2016/

Code and proposal Staging area

https://github.com/WG21-SG14/SG14

4. Review

4.1 Review and approve resolutions and issues [e.g., changes to SG's working draft]

4.2 Review action items (5 min)

5. Closing process

5.1 Establish next agenda

Sept 13 call

5.2 Future meeting

July 12 (cancelled as it is same day as Toronto C++ Standard meeting)

Aug 9

Sep 13 Just before CPPCON

Oct 11 (before DST switch)

Minutes for 2017/09/13 SG14 Conference Call

Meeting minutes by Michael

1.1 Roll call of participants

Andreas Fertig, Jeff Hammond, Mateusz Pusz, Philippe Groarke, John McFarlane, Ronan Keryell, Lin ya Yu, Michael Wong, Wouter Van Ooiijen

1.2 Adopt agenda

Approve

1.3 Approve minutes from previous meeting, and approve publishing previously approved minutes to ISOCPP.org

Approve.

1.4 Action items from previous meetings

2. Main issues (125 min)

2.1 General logistics

CPPCON talks
CPPCON proposals
-any other F2F

2.2 Paper reviews
2.2.1 A better alloc
The C proposal is here:
http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1085.htm
The C++ part is here:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1953.html

AI: ask the author to come to CPPCON SG14 presenation

2.2.2 any other proposal for reviews?

D0773 "Towards Meaningful Fancy Pointers". Arthur

2.3 Domain-specific discussions

SIG chairs/lead who can watch SG14 reflector, and gather proposals that are in these domains

2.3.1 Embedded domain discussions

Wooter, Odin

2.3.3 Games Domain

John, Guy

2.3.4 Finance Domain

Carl, Neal, Mateusz

2.4 Other Papers and proposals

1 Additions: Intrusive smart pointer, Isabella Muerte

Not here

2 inplace functions: Nicolas Fluery, Carl Cook

3 Fixed point real numbers, John McFarlane (CPPCON)

had a special call with Lawrence, FPGA Xilinx, and Taiwan university,
someone is working on an implement of John's proposal,
Lin Ya from Taiwan is working on this for FPGA
Is Jeff Hammond also working on this with Xinmin/Robert? Push them for C++
Also crosses into Khronos OpenCL/SYCL for DSP

4 Ring span, Guy Davidson (CPPCON)

Progressing

Concurrent ring span deferred to concurrent queue,

5 Hazard Pointers + RCU Maged + Paul (CPPCON)

HP moves to wording, RCU contineus work

Wording is being worked on

6 Thread constructor attributes, Patrice (CPPCON)

Will be represented

7 Intrusive containers, Guy Davidson/Hal Finkel (CPPCON)

Being worked on

8 plf colony/stack, plflib.org Matt Bentley

Matt sent and email, progressing

9 Likely/unlikely (CPPCON)

Clay sent an email, likely presented at CPPCON SG14

10 Comparing virtual Functions , Scott Wardle (CPPCON)

Unknown

11. Heterogenous computing: Channels and Managed pointers (CPPCON)

3 papers for next C++ STandard meeting (Sandia)
1. Context
2. Suevey
3. Exception design in a concurrent environment
4. Executors

SC: Heterogeneous/Distriuted C++
SYCL, Kokkos, Raja, HPX, UPC++, Boost.Compute, AMD/HCC
concern about combining distributed with heterogeneous, may be do it using networking TS

-other papers
integral constants proposal from John M. with UDL (CPPCON)
Jeff H may be interested, aligned with machine learning

2.5 Future F2F meetings:

CPPCON Sept 27 SG14

2.6 future C++ Standard meetings:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4633.pdf

2017-11 Albuquerque WG21 meeting information

SG14 meeting in Gdansk, Poland on September 18.
https://gdansk.4developers.org.pl

Oct 16 is the mailing deadline.

3. Any other business

Reflector
https://groups.google.com/a/isocpp.org/forum/?fromgroups=#!forum/sg14
As well as look through papers marked "SG14" in recent standards committee paper mailings:
http://open-std.org/jtc1/sc22/wg21/docs/papers/2015/
http://open-std.org/jtc1/sc22/wg21/docs/papers/2016/

Code and proposal Staging area
https://github.com/WG21-SG14/SG14
4. Review

4.1 Review and approve resolutions and issues [e.g., changes to SG's working draft]

4.2 Review action items (5 min)

5. Closing process

5.1 Establish next agenda
Sept 27 Courtyward Marriott

https://docs.google.com/document/d/1f6RrB1p8QvU_vkN3BTS2yjLhqz1y9PJbne7z8PNFRtU/edit?ts=59a5e060
Oct 11 is potentially next call

5.2 Future meeting

July 12 (cancelled as it is same day as Toronto C++ Standard meeting)
Aug 9 DONE
Sep 13 Just before CPPCON
Oct 11 (before DST switch)

Adjourn

Minutes for 2017/09/27 SG14 Conference Call

Minutes by Michael Wong
HI all, here are the notes. Thank you deeply to our scribe Patrice Roy who did it for all sessions including his own. Sorry for the delay as I have been travelling since CPPCON

SG14 meeting, Sept. 27 2017
Attendees:
Michael Wong: Chair, Codeplay, SCC, BSI,
Lee howes, Facebook
Maged Michael: Facebook
Paul McKenney, IBM
John McFarlane: A9.com BSI, co-Chair
Colden Cullen, Amazon Lumberyard
Justin Boswell, Amazon Lumberyard
Dan Kalowsky, Esri
IIIa Kirianovski, Jetbrains
Scott Wardle, EA
Paul Hampson, Wargaming
Arthur O'Dwyer, self
Greg Marr, Autodesk
Nr Friedman, Tower Research capitl
Peter Phillips, Riot Games
Oleg Zhylin, Salford Systems
Billy Baker, FlightSafety
Izakly Archangelsky,Wargaming
Hartmut Kaiser, LSU
Andrew Liesk
Vishal Ozo
Ada Yaxley, Havok
Dave Watson, Facebook
Frederic Vitzikam, Amazon
Matt Matheson, Uber
Jason Jurecka, Blizzard
Guy Somberg, Echtra Games
Carl Cook, Optiver
A. Joel Lamotte, Softbank Robotics EU
Clay Trychta, ITG
Kevin Alexandre Boissonneault, Behaviour Interactive
Patrice roy, U Sherbrooke, Scribe
Isabella Muerte, Mnmlstc
Jan wilmans, Promexx
Nicolas Guillemot, U Victoria co-chair
Gordon Brown, Codeplay

Andreas Weis, BMW
Rene Rivera, Disbelief
Mitsuhiro Kubota, Sony Interactive Entertainment Inc
Keita Abdoul-Kader, Facebook
Fuat Galeri, Facebook
Geoffrey Romer, Google
Sergey Ignatchenko, Three RISE

42 total

Bellevue, WA

-=-=-=-=-=-=-=-

Session starts at 08:41

Wong greets participants and sends an attendance sheet around

Patrice requests for people to write their names legibly

Wong explains the role and composition of SG14, and reminds participants that the normal rules of engagement and composition for ISO meetings apply since this is an official ISO meeting

Wong states the objectives of today's meeting, in particular reviewing papers

-=-=-=-=-=-=-=-

Triage

Wong ensures the agenda has the assent of all participants

Friedman : is this an appropriate forum to bring concerns with respect to low-latency in existing proposals? (mentions the outcome proposal)

Wong : yes

Wong mentions interest in putting together a group to look at lightweight error handling in C++. The outcome proposal comes to mind

SIG chairs / lead who can watch SG14 reflector, and gather proposals that are in these domains

Wong recommends naming per-domain leaders given special interest groups within SG14

Wong recommends:

* Wooter and Odin Holmes for Embedded
* John McFarlane, Guy Davidson and Paul Hampson for Games
* Carl Cooke, Neal Horlock, Mateusz and Clay for Finance

Hazard Pointers + RCU : Maged + Paul

Wong explains this a a group of lock-free technique such as those used in MongoDB and the Linux kernel, with a C++ interface

Wong : this proposal is mostly interface. It has passed SG1 and SG14 and is making its way through LWG

[P0233R5] [P0566R2]

Maged : presents sample code. There's one writer, multiple readers. Reading traversal meant to be fast and avoid hitting removed nodes

Maged : member function retire() signals intent to not use a nodes anymore; removal to be performed at a safe time

Maged : key classes are hazptr_domain, hazptr_obj_base<T,D = default_delete<T>> and hazptr_holder. hazptr_domain interacts with and optional pmr allocator, and there's a per-process default_hazptr_domain(). The cost for this is minimal (a few pointers), and would only be a price to pay if it was actually used

Q : how do hazard pointers work?

O'Dwyer : the problem with concurrent node access is racy deletion. There are two main ways to do this : RCU (the hazard is global, there is a quiescent state when nobody's reading, we use that to reclaim) and hazard pointers (someone said he was done, nobody says I want to use it, system calls reclaim)

Guillemot : why don't we use atomic_shared_ptr?

Maged : it's not just a reader, and doesn't scale well. With hazard pointers, readers only write to their own pointer, which is better for cache.

Vishal : is this a Linux-only thing?

Maged : it only relies on compare_exchange() but can be optimized through per-platform mechanisms

Patrice : there is a D parameter to hazptr_obj_base<T,D>. Can we change the D behavior on each retire?

O'Dwyer : yes, you could want to do different actions depending on context

Q : I suggest replacing operator bool() const noexcept by bool empty() const noexcept

Isabella : if we add empty() here, we should add it to all other smart pointers

Q : it should be explicit

Hampson : it's more of a guard than a smart pointer

Q : a named function might alleviate the confusion

Patrice : let's mark these questions in the proposal and let LEWG decide

Vishal : what happens when hazptr_holder::get_protected<
T>() fails?

Maged : it might return null, which is not necessarily failure

Maged provides some information on hazard pointer performance

McKenney then presents RCU

[D0461R2]

McKenney : the RCU API is adapted from the C interface, and is a bulk protection mechanism

McKenney : synchronize_rcu() waits for all preexisting readers. Old readers see the old nodes, new readers see the new nodes. There is an asynchronous companion function to that

McKenney : the RAII interface is std::rcu_reader. The ctor and dtor both perform the appropriate synchronization protocol. It's a moveable type

Vishal : can we use that for a long period of time, such as a whole game execution?

McKenney : maybe, but it might not be the right approach as it might keep allocated memory alive too long

McKenney : there is an updater type. std::rcu_obj_base<T,D> is similar to hazptr_obj_base<T,D>. There's a retire(D) member function and a std::rcu_retire(T,D) free function

Isabella : why hold the T object as a member instead of inheriting

from it, as it could provoke EBCO?

Geoffrey : it is an object.

O'Dwyer : let's let library implementers choose

McKenney : we have a non-intrusive rcu_retire() with the free function. In that case, it's up to the library to make it efficiently, and it can be done even though the illustrative example in the paper is not clever at all

McKenney : the updater is not stuck waiting when the readers are going through a list. When you do want to wait synchronously, you can use synchronize_rcu() and rcu_barrier(), the latter which waits for all the "retires" to complete

McKenney : we don't really have a rcu_holder. In the Linux kernel, you can use just any pointer

McKenney : for consume, we have mechanisms to avoid the compiler guessing the value of a pointer without checking

Q : does this only support default constructible types? (small discussion ensues)

Wong : this will probably go to SG1 for another round

-=-=-=-=-=-=-=-

At 10:00, we break for plenary / lunch

At 13:08, session resumes

-=-=-=-=-=-=-=-

inplace_function

Cooke : inplace_function is meant to be a non-allocating drop-in replacement for std::function

Cooke : it's used in finance and in gaming already

Q : does that mean on-stack?

Cooke : depends on user context. It has automatic storage

Q : I implemented something like that. What makes it hard to implement is the choice to fulfill a wide interface (copyable, moveable, etc.). I find inplace_function to be easier to grasp that function. You can constraint the erased types to move-only, noexcept-only, etc.

Q : the reason to allocate is that contained objects can be arbitrarily sized. How do you deal with size?

Cooke : size is part of the template's signature. You can assign a smaller inplace_function to a larger one

Q : how about Vittorio Romeo's function_ref?

Dionne : it solves a different problem; function_ref is non-owning

Q : it seems very useful. Can you replace its contained function object?

Cooke : yes, as long as size constraints are respected. It's a common use-case

Q : shouldn't this be a specialization of std::function?

Dionne : it would break ABI. I doubt LEWG would let it pass

Q : can we parameterize SSO for function?

Cooke : no

Dionne : I think the functionality is great, but lacks generality. It's possible to use all kinds of storage, not just in-place. I would encourage exploring basic_inplace_function and some aliases to customize the more common use-cases

Cooke : should we allow a custom allocator?

Q : I don't think we should confuse storage policy and allocation policy

Q : can we specify alignment?

Cooke : indeed (it's on another slide :))

Q : could we automatically infer the required size at compile-time?

Dionne : with deduction guides

(there's a mention on boost::callable_traits to deduce some additional properties of the functions; Cooke asks for a note of this in today's proceedings, so there)

Q : passing std::function to a function without allocation is good, but we don't want to fail. Are there guarantees at compile-time?

Cooke : I expect a compiler error

Vishal : is the implementation available?

Cooke : github

Dionne : there are many kinds of *_function. Where does it stop?

Cooke : inplace_, unique_, ...

Dionne : please generalize. We'll talk

Patrice : why no make_inplace_function?

Cooke : sticking as close to std::function as possible

Wong : we encourage this to move forward

P0484R1 (make_thread()) : Patrice

Patrice : describes the P0484 papers and the P0320 thread::attributes papers

Questions are for clarification only

There is no opposition to pursuing the make_thread() that creates a std::thread. This approach is deemed preferable by SG14 to one that would use platform-specific thread adoption

Likely/unlikely: Clay Trycha

[P0479]

Clay : this was presented to EWG. They requested that [[likely]] and [[unlikely]] could be added at many locations.

Clay : with the current proposal, someone could write a [[likely]] if

then break then make it unlikely. Should we write standardese to prevent such cases?

MacFarlane : these are just hints. The optimizer can confidently disagree

Q : the question is whether diagnostics should be emitte

Q : what is the scope of the attribute?

Wong : attributes appertain to the thing just beside

Clay : in if(f()) [[likely]] { ... } they apply to the braces

Q : applying these potentially to every statement seems like a very bad interface

Cooke : it's easy to make things very slow if we don't understand what the compiler does; on the other hand, everyone does it. Can we express degrees of likeliness?

Clay : I was encouraged not to go down that road

Hampson : how about collapsing the attribute into statement productions?

Clay : I'll look into that

Vishal : is there guidance as to when to use [[likely]]/[[unlikely]] or not?

Clay : measure

Baker : EWG wants evidence of whether this spreading of attributes is a good idea or not. That's what we expect from the expert users in this room

Wong : let's vote.

Vote : apply likely / unlikely on "all statements" listed in P0479R2

SF F N A SA
1 3 9 11 1

Baker : EWG voted strongly for this. The voters are those writing compilers. I think we need justification to back this position

Vishal : the only thing I'm against is the jump statements

Q : if constexpr, what's the point of likely / unlikely?

Clay : none. It's just part of the grammar

Q : does this tie into "expect"

Clay : same idea

Hampson : more info on jump statements, what du they apply to?

Wong : attributes are to the right of what they apply to

Patrice : I need to know what the rationale for rejection is

R : I don't see what the meaning would be to put this on a statement block

R : reading C++ as a beginner is hard. This makes reading code more complex

R : for some jump statements, it makes little sense

Q : it's useless, but harmless

MacFarlane : anything can make C++ code unreadable if you abuse it

Patrice : I think some people have misread the grammar changes, as they apply to more than [[likely]] / [[unlikely]]

Hampson : because of the wide number of places where you can put it, it's easy to place it in the wrong location for the effect you want

Q : you can put anything in [[]] with C++17. At worst, compilers will just glance at it

Q : if we put this at the statement level, we have to clarify semantics. It seems tied to __builtin_expect

Q : there's a risk of inviting users to put nonportable stuff wherever they want to avoid annoyance

Q : we need semantics first

Hampson : "what does it apply to" seems more important to me

Q : if we don't know what it means in different places, we will end up with implementation divergence. Restricting to well-defined points seems better to me

Q : shouldn't we restrict it to places where it applies already with proprietary extensions, due to implementation experience?

Wong : attributes in general can be applied to anything for future-proofing. We could claim that SG14 recommends being conservative

Hampson : likely break likely worries me

Patrice : I don't think we can contradict EWG for the general attribute position thing like we're doing

Wong : let's move this forward

Numerics: John McFarlane

* http://wg21.link/p0554
* http://wg21.link/p0675
* https://gist.githubusercontent.com/johnmcfarlane/c7dc07e179de56653272585adce70cb5/raw/e5a5edeceac6ff5b39a7797b39f6b93e33ef388d/constant.md

Wong explains the reasons for the fixed_point proposal and the evolution of that work

MacFarlane : [P0554] Composition of Arithmetic Types

MacFarlane : this paper describes the general approach I'm taking for fixed_point now. It's different from my initial approach to this problem

MacFarlane : I'm trying to make types out of other types from composition of templates. By default, we use fixed_point<int,int>

MacFarlane compares his approach from Crowl's approach in P106

MacFarlane : what to do on overflow is an orthogonal issue for fixed_point in my opinion

MacFarlane : I'm making sure the type detects and adapts (in possible) to size or precision changes or failures, and call this elasticity

MacFarlane : I have design questions. For example, numeric traits have been put in another paper, broken in parts like Walter Brown's doing for the numeric_limits, and following his naming guidance

Patrice : the set_num_digits name seems weird to me...

MacFarlane : it gives you back a type with sufficient digits for your needs

Q : does set_num_digits resemble the _least_t types in terms of functionality?

MacFarlane : indeed. I could follow the <cstdint> naming style

Q : define_num_digits or something similar?

MacFarlane : to think about

Q : can we get exceptions if overflow occurs?

MacFarlane : yes. I use static_assert in my proposal for illustrative purposes. Note that some compilers support integral types with more than 64 bits, so it's nice to have a way to use them

Q : how does that mesh with common_type?

MacFarlane : that's where we complete older designs and break the 32 bits boundary

Patrice : is this facility a customization point? I'm not sure numeric_limits is, and they seem similar

MacFarlane : HPC people sometimes have exotic solutions to problems, so yes. I'm going for as much customization as possible

Q : is there a fixed_point for non-integrals?

MacFarlane : elasticity can apply to floats, and fixed_point<float> is possible

Q : I remember using _least in GPU programming to reduce transfer size. Do you support that too?

MacFarlane : elastic integrals go for the minimum. Vincent Riverdy's working on low-level bit manipulation functions for some manoeuvers

Vishal : what about endianness?

MacFarlane : it's orthogonal (discusses narrowing after that)

Q : can a number of bits of a type be different from int?

MacFarlane : it has to be signed

Patrice : it will end up being Integral

Q : how is the type resulting from a multiplication computed?

MacFarlane explains.

Q : what rounding guarantees do we provide with a + b + c?

MacFarlane : since it's integrals under the hood, it will be exact

Hampson : numeric_limits is customizable

-=-=-=-=-=-=-=-

At 15:00, break

At 15:30, session resumes

-=-=-=-=-=-=-=-

MacFarlane explains how to address the division problem, and why it is counterproductive to do this through fixed_point itself. He explains how he does this with integral division

MacFarlane explains that there are some ugly cases. There is a nice but slow solution. He then explains how elasticity provides and interesting way of solving the conundrum, and how he goes to fixed_point<elastic_integer<N>,...>

MacFarlane seeks guidance on the approach to prefer

Q : you have two options, both are available, and want to know which one should be the default? My vote goes to correct by default

MacFarlane : there's a usability issue with fixed_point (explains the corner cases)

Q : for the novice user, choosing between super-fast and correct is a no-brainer. I highly advise to make user-friendly the default

MacFarlane : like allocators, fixed_point is not an entry-level feature

Q : how about policies?

MacFarlane : I think it's too "micro-managey"

Q : default needs to be sane an clear. Also, using long in portable code seems inappropriate

Guillemot : two important use-cases. One : mobile devices without floating point (pick your sizes). Another : using fixed_point for highly precise code, e.g. a rasterizer. Do an analysis of the problem domain to make your choice

Q : is there an option of a fixed fixed_point?

MacFarlane : it's feasible. Some do that. You need to double the width when multiplying

Vote : should option A (codenamed "The Ugly" in the document) be the default?

SF F N A SA
2 4 8 4 2

Wong : we need to switch to another presenter for now. We'll come back to the other fixed_point paper if we have time

slot_map

Wong : this was presented in Toronto, where they encouraged more work

Alan : see this as a pool allocator with a map-like interface

Alan : I have specific questions to answer :

* Should a supplementary key type be provide equal to default but throwing on generation overflow?
* Should clear() reset generation counters and the free list?
* Should raw access to the underlying container be provided?

Q : what are the other names for this?

Alan : pool allocator? Lookup is O(1) and there's no hashing

MacFarlane : what are the key advantages of generatiom overflow behavior?

Alan : not getting key / value mismatches

Patrice : I don't think direct access to the underlying container is required given the container's interface

Q : what is the default for overflow?

Alan : silent failure. It's unusual

Q : could the pair be changed to a different helper type?

Alan : yes

Patrice : has the possibility of making this a non-owning container been discussed?

Alan : no

Vote : should a supplementary key type be provide equal to default but throwing on generation overflow?

SF F N A SA
3 9 7 0 0

Consensus

Q : for reset(), it's equivalent to all counters overflowing, but at least I get to choose when

Guillemot : isn't there the risk of the equivalent of a dangling reference?

Alan : yes. A dangling index.

Alan : in this case, the question is : should this behavior be equivalent to erase(begin(), end()) or to default construction followed by reserve?

Q : I'm worried be erase(begin(),end()) having different meaning from clear()

Q : does the standard have to specify this behavior?

Alan : I think so, as it impacts your key

Q : if clear() doesn't free the list, what happens to the objects?

Alan : they are being deconstructed

MacFarlane : when your key overflow, you will forget objects?

Alan : the promise this container makes is that keys are unique

Vote : should clear() reset generation counters and the free list?

SF F N A SA

1 14 1 4 1

Consensus

Q : is this read-only access?

Alan : yes

Vishal : apart from with a debugger, why use this?

Q : this could be useful for fast serialization

Q : there seems to be three containers in there. Is it safe to given access to only one?

Alan : yes because it's const

Q : this is a container adapter. Providing raw access would make it the only such container

Q : here, the approach proposer resembles that of flat_map

Q : there is a way to access the underlying container if it's a protected member and you derive from it. It seems more coherent to me

Q : are there ordering guarantees?

Alan : if you insert the same elements in the same order

Vote : should raw access to the underlying container be provided?

SF F N A SA
0 4 3 9 5

Alan : I have two other questions

Q : exposing this leads to potential abuse

MacFarlane : this seems more like an optimization to me

Vote : should we offer insert_at() and emplace_at()? If yes, are we Ok with using N/8 bytes to make them O(1)?

SF F N A SA
0 0 10 8 1

No consensus

Alan : what about the name and the interface?

Odin : wouldn't the variadic signature for the underlying container cause problem with non-type template parameter?

Alan : the one that comes to mind is std::array and it wouldn't work in the case anyway. It needs reserve() and some other functions

Q : there does not need to be an exact match between the underlying container's value_type and the slot_map's value_type

Alan explains the underlying data structures

Patrice : we need the variadic template template to propagate allocators

Odin : ... or use polymorphic allocators

Baker : I recommend writing a paper with examples for SG14

Hampson : I would favor three template template parameters

Someone suggests that it would be interesting to group the three containers together in a single one. Alan thinks it does not work

Vote : are we happy with the interface?

SF F N A SA
1 11 5 2 0

Consensus

Heterogeneous and Distributed computing

Wong makes a presentation of future directions in heterogeneous and distributed computing

Wong : we need to make the parallel STL algorithms asynchronous

Wong : we need to make them run on multiple devices

Wong : we need executors. They are enablers for everything else

Wong : affinity is more urgent than data movement

Patrice : what's the tension that's slowing down asychronous parallel algorithms?

Wong : competing approaches

Guillemot : why not wrap them in a continuation through std::async()?

Wong : it has to be built in the algorithms themselves to get the best out of the feature

Gordon presents a paper on memory affinity in C++

Gordon : affinity is defined with respect to some execution context, some resource partition

Gordon : moving a computation from a CPU to some other execution unit has to be possible

Q : concrete example?

Gordon : on a NUMA system, there are many nodes and regions or memory. You can want to define affinity to a particular CPU and a particular region of memory

Patrice : partition?

Gordon : is describes an execution resource and is exposer by an executor

Patrice : note to SG1 : we nee executors soon as they are a bottleneck for progress right now

Gordon : at least at first, SG1 guidance is to use memory in a single location, but make it scalable

Gordon displays D0737R0

Odin : do you have a ballpark of how heavyweight this is going to be? For embedded systems, it has to be _small_.

Patrice : I suggest fusing the functionality of thread execution context and P0320's thread::attributes as there will be too much overlap between

them for two distinct types to coexist in the standard

Q : what is the meaning on affinity here?

Wong : identifiying the resource, and then binding to it. We're still working on it, in different stages. It ties into gather / scatter. I was involved in the design of affinity for OpenMP, so this inspires me

Exception handling in a concurrent environment

Wong : the fundamental problem is that we can have saturation (too many exceptions) and dynamic memory allocation (each exception raised allocates a little something)

Wong : we plan for an error_buffer that would preallocate statically resources up to a limit

Wong : this would be attached to the execution policy, not the algorithms

Audio

Guy : audio is half of the game (sort of). It makes your game better

Guy : the tools at our disposal are incredible

Guy : there are many middlewares, great tools per engine... but we have figured out how to write interfaces to communicate with audio devices. There's only one way to really make it right

Guy : I was talking with an Unreal lead developer today and he's implementing this exactly the right way. All old ways are bad

Guy : I think we're ready for std::audio. There's no research here; everyone is doing this the same way

Guy presents a preview of his talk at CppCon 2017 on the topic

Guy presents the interface. A null object (the design pattern) models the absence of an actual audio output

Patrice : I recommend using an istream instead of an ifstream to stream from a more varied set of sources

Q : where is the audio graph for submixes?

Guy explains that part of his interface

Guillemot : effects in 3D games?

Guy : you would have to do some effects myself

Guillemot : then, I'd use a tool that does it for me

Guy : we could provide sane defaults

Q : how do you assemble samples?

Guy explains how he sees things

Q : how about input?

Guy : it's orthogonal. One thing at a time

Alex : I have many counterexamples for this example. It doesn't "skate where the puck is", it skates where it was. How will this scale to a movie theater with 64 output speakers? Working on a Playstation, I'm convinced this will not take advantage of the hardware

Guy disagrees

MacFarlane : this doesn't have to be applicable to AAA games. Like the 2D graphics proposal, this can be useful for educational purposes

Patrice : make a demo to show that this scales during the experimental:: stage

Q : think about relevant defaults

5.1 Establish next agenda

Oct 11

5.2 Future meeting

* July 12 (cancelled as it is same day as Toronto C++ Standard meeting)
* Aug 9 DONE
* Sep 13 Just before CPPCON
* SEPT 27 CPPCON
* Oct 11 (before DST switch)

-=-=-=-=-=-=-=-

At 17:55, session concludes

Minutes for 2017/10/11 SG14 Conference Call

Minutes by Michael

1.1 Roll call of participants

Michael wong, Billy Baker, Charley Bay, Guy Davidson, Jan Wilmans, Hana Dusikova, Linyay, Mateusz Pusz, Matthew Bentley, Odin Holmes, Paul Bendixen, Tony Tye, Wooter Van Ooijen, Arthur O Dwyer, Voshal Oza,

1.2 Adopt agenda

probably no time to look at Monad paper

1.3 Approve minutes from previous meeting, and approve publishing previously approved minutes to ISOCPP.org

Approve

1.4 Action items from previous meetings

2. Main issues (125 min)

2.1 General logistics

CPPCON F2F

SIG leaders.
Mathew Bentley.

2.2 Paper reviews

2.2.1 System Errors

https://groups.google.com/a/isocpp.org/forum/#!topic/sg14/0lNLQ41V0HI

C++11 header error, based on Boost and asio
the error mechanism using type erasure, derive from error category,
domain with enums, + associated strings
a reference to that error category
mostly unused, could get adoption,
need for a cross module error transport

possible future improvement
not constexpr
got some confusing design choices
std. expected and oucome library wrappings
there is an exception payload wrapped around an error payload
coudl wrap with monad

Arthur: ongoing expected outcomes, how do you wrap these with
what does the default constructed error code means,
error category are singleton, hard to use with dlls and shared library
use of std::string in error category is great for customizable but needs heap allocation

filesystem TS uses system error, prefer expected and outcome
networking TS also uses System error, pass by value semantics

Design issues in system error
1. 2 apis, throwing and non-throwing, both have to have different names or function signature, if you add out parameter, which wnats to nothrow version, if there is no out parameter then its throwing
better may be to put the nothrow version in their own names

want to break it up into 2 issues:
in boost outcome review: people hate it when there are 2 interfaces, and they diverge
outcome try to take that with T or an error
1. idomatic use is first topic of problem

2. Can we change the error code in the short term, but it now pulls in memory subsystem to change it

use of string on small chip is a show stopper, accessing heap is bad here
singleton is not a problem in embedded domain
virtual call may be a problem, devirtualization may fix this to be ok

error category has a virtual destructor,
most of the time comparing by pointer, this can just go into rom
error code is a trivial class

std string is when there is an error code and to get the message out
should not switching on the message, should be const char *
can we use string view instead of char *, they have the same lifetime issue, then dont have to worry who has a reference to it
where did foo.txt come from? who remembered it?
erro category is where you would store everything
message may enable locale based translation
context life is still in the allocation

managing the lifetime of these strings,

Niall outcome desgns
he derives from error code, payload in derived class, works, solid, aims for low latency, its a safe slicing
use an expected or outcome thing, rather then error code by value

Charles most important to least important design criteria
(1) Desire for Semantic comparison of 'bool operator==(std::error_code, std::error_code);' (currently is exact-match)
there is error code and error condition
error code is non-lossy is passed in through api, passed up to the caller, who is going to do some semantic mapping on it
there is also error condition class, comparing a code to a condition, then there is a mapping in low system resources,
always semantic map and never use error condition
another view is that never the twain shall meet, these are separate ideas, error code enum or error condition enum, if it is generic condition, then it is error condition, STL does not follow that intuition either,
some feel the twain did meet and people are throwing error condition
2 chars is not using any of these
best practice if people follow arthurs methodology, but its not being followed, or cases liek asio networking layer with network controller, I am giving people low level platform specifics, then it is lossy,
(2) Need for 'constexpr std::error_code' instances (perhaps need 'std::null_category' for default 'std::error_code' instances)
without constexpr then there is a lot of domain that cannot use this
embedded case: one error domain, if you want add code from someone with a different error domain,
not look at the message in 2 domains, then that should be constexpr instead of a singleton
its about being pure functional, no side effects, generic friendly
make it go to rom
another case is use it as an error path in constexpr functions, and you turn off eh, it gripes at you when throwing eh when eh is off then you are bad
best practice for transporting errors in a constexpr function
idiom is return expected, consistency between constexpr, and not constexpr,
now constexpr functions look just like runtime functions
that seems to makes sense, others agree
signallign threading context, then we use constexpr to tell if someone is cheating, use that idiom to transport errors
what if its marked constexpr returning an error time instead of throwing, then you would check the results at compile time then do it with if constexpr, and static assert the result,
any use to having string with error code?
Use case in other domains like filesystems is useful, but embedded domain, dont recover from errors, just give user a deterministic behaviour,
small end of chip this is not that useful,

make it a const char* should go into rom, and that would be cheap enough,
returning light weigh error code, there is not a usecase to have a dynamic string
are there any use cases where people would take state of error code before passing it on, and would that be affected by havign a const char *: yes that is possible, if you have resouce then just throw an exception then you can have a dynamic string
light wright version that does not pass dynamic string
what about concurrent environment ? Error codes are passed back up through the caller, would not put it into shared memory,
in parallel heterogeneous environment, then we would prefer simple return values,
we have that in error code, higher level wrapping will be using outcome
its a T or an error code or an exception
Niall feels he does not like expected or outcome that pulls in exception,
expected models std::optional, has operator * (if you dont use exceptions), and dot value (will throw if there is an error)

(3) Confusion regarding 'std::error_condition' (is it needed?)
(4) Reliance upon 'singleton std::error_category' instances (a possible barrier to header-only libraries)
(5) Wide-contract (throw) vs. Narrow-contract (no-throw) propagation of 'std::error' instances (e.g., std::expected<T>, outcome<T>, etc.)
we cover that above as well
(6) (pipe-dream), 'instance-specific' message state (e.g., support for "structured logging")
we will never do instance specific, and remove allocation machinery, or we will do instance specfic state
e.g. is in std fileystem where they throw a filesystem error and 2 additional strings that you fill in
author writes 2 functions, throwing ones get more info in those 2 extra strings
nonthrowing version, might know those strings, throwing version enables propagating
you just give me the error, I should know the context,
actor model forwards everythign through callbacks, instance specific state
there is a way to wrap context specific state in expected outcome then error code, the other way does not work, if it is in error code, people cannot take it away, in embedded may not be able to allocate in that context
Not a problem that the interface is diverging, because we dont the information. Filesystem

Error code had operator bool, error or not, there is confusion about that, it now defaults to 0, people use this trap value to something, may be ambiguous and non-helpful
Arthur may be able to do this.

2.2.2 Monads:

https://wg21.tartanllama.xyz/monadic-optional

2.2.3 any other proposal for reviews?

2.3 Domain-specific discussions

2.3.1 Embedded domain discussions
2.3.3 Games Domain
2.3.4 Finance Domain

2.4 Other Papers and proposals

Ptf:colony.
1st question: lots of memory blocks, when they become empty and are erased and freed to OS, someone feel we shoudl preserve and reuse, makes sense larger memory blocks as they have a growth vector, this means programmer has to get rid of all memory blocks at that point, should be automatic, programmer control,
shrink to fit on a vector grows to some high water mark and want some of that memory back, except there is no ptr to be invalidated and ops are not as expensive, no deallocate to old block, plus its segmented
std:deque also has shrink to fit ? Can copy, no semantics are different, shrink to fit is expensive operation, which reorders all the pieces, so it is copying expensive shrink,
coalescing sparsely populated std::map is implemented as copy construction, could do that
what is memory .. recycling cost in allocator?yes, might be best left up to allocator, rather then container

means reserve function working properly, no allowance empty unsed segment in colony, segment can only be certain length, so reserve can only reserve a certain nnumber 65000 max, then using this way, can get a proper reserve function.
compaction for cache locality and deallocation of unused capacity, stealth copy and shrink to fit (a copy construction in vectors where growing is an expensive operation while deque does not have reserve as it is not as expensive)
do we need reserve in colony?
when is deallocation happening? when block is empty it is automatically deallocated, if 2% perf differenece, then should consider opt-in, making deadline for 60fps game, this could be the difference
will that freeing cause any jitter, even if going through allocator
can see it controversial, is 2% an average or is there a more worst case
this was on haswell,
2% depends on where it is, in our last console game, it was a lot, but there are places where we have bandwdth to spare

std::list question: perf advantage over traditional link lists: std list needs partial splicing, full splicing is fine i.e.e deprecate splice signature with 2 arguments
putting a proposal to make partial splicing to get more perf version
will have breakage issues
use std2 namespace and see how people will use it from there.
this reserved by committee
is insertion constant O(1)? yes same amortized

vector push back is amortized, list is constant.

2.5 Future F2F meetings:

2.6 future C++ Standard meetings:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4633.pdf

2017-11 Albuquerque WG21 meeting information

Meeting C++ SG14 led by Guy Davidson.

3. Any other business
Reflector
https://groups.google.com/a/isocpp.org/forum/?fromgroups=#!forum/sg14
As well as look through papers marked "SG14" in recent standards committee paper mailings:
http://open-std.org/jtc1/sc22/wg21/docs/papers/2015/
http://open-std.org/jtc1/sc22/wg21/docs/papers/2016/

Code and proposal Staging area
https://github.com/WG21-SG14/SG14
4. Review

4.1 Review and approve resolutions and issues [e.g., changes to SG's working draft]

4.2 Review action items (5 min)

5. Closing process

5.1 Establish next agenda
Dec 13 (Nov 8 is C++ Std Meeting)

5.2 Future meeting

July 12 (cancelled as it is same day as Toronto C++ Standard meeting)
Aug 9 DONE
Sep 13 Just before CPPCON
SEPT 27 CPPCON DONE

Oct 11 (before DST switch)