Document Number: P1736R0
Date: 2019-01-21
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 2019/04/17-2019/06/12

Contents

Minutes for 2019/04/17 SG14 Conference Call

1.1 Roll call of participants
>>
> Guy Davidson, Jan Wilmans, Ben Craig, Niall , Andreas Fertig, Steven
Varga, Grafik Robot (Rene Rivera
), Michael Wong, Andreas Weis, Hubert Tong, Maged Michael, Paul McKenney,
Staffan TJ, Jens Maurer, John Macfarlane, Charles Bay

>
>> 1.2 Adopt agenda
>>
>>
>> Yes

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

> 1.4 Action items from previous meetings
>>
>> 2. Main issues (125 min)
>>
>> 2.1 General logistics
>>
>> Review last call discussions.
>>
>> CPPCON SG14 meeting is now up for attendance

>
>> 2.2 Paper reviews
>> 2.2.1 Embedded/freestanding vs hosted Summary of freestanding evening
>> session discussions
>> http://wiki.edg.com/bin/view/Wg21kona2019/SG14FreestandingImplementations
>>
>>
we did do polls on library stuff, added EWG, LWG, outside of SG14 SG1
1. add library feature to FS subset that dont use troublesome: 11/11/5/0/1
2. same as 1 but adds eh: 3/12/7/4/1
strongly against was chandler: concerned about being able to ship a
compiler that does not have a std library, and relies on an external vendor
for std library
have a lowest module that have the interestign bits that requires compiler

knowledge,
What part of proposal is inhibiting Chandler? he wants to ship FS impl with little as possible and have others layer the hosted part on top
if we add stringview and std sort then it adds a lot on implementation, its not libC++ that is providing it, he should provide a product that is partial or non conforming and let the user adds what ever is still needed

what about method on classes that throws, e.g. std array at, array makes a lot of sense in FS, but at throws if out of range

lot of neutrals on this poll
Make calls to potentially-throwing library functions ill-formed.
(7/6/10/1/2)dont provide std array at
Make the library turn throw statements into abort (for example, by preprocessor), as is common practice in no exception builds. (1/7/7/6/5)
today we terminate and going away from that is not realistic, Ben may have a way through this with some std weasel wording, nonrmative: in a program with no catch statement, it is untectable diff between callign std terminate and throwing an exception and your impl does not unwind the stack when an uncaught exception is thrown i.e. if you have no exception mode, then just call terminate because program has no catch statement
Make the library turn throw statements into terminate (for example, by preprocessor). (1/5/14/2/4)
Make the exception handling strategy implementation defined. (0/5/10/6/2)

future progress: on std library got go head to push it along for C++23
every meeting I had to update the paper, like a giant merge conflict
so split P0829 into multiple small papers

updating our editorial technique so other papers in flight can ride along
also touch on feature testing macros

core language side: have exception as the most important but most contentious
will discuss next month
teh size cost of eh
return value is smaller costs
by Belfast, plan for an exception runtime paper to show happy paths to start providing more paper
operator new and delete a paper on that through ewg, this seems one of the easier one to accept
this will say dont require allocating form of op new and delete, you can provide your own just like you can today
by default they are not required to be there

Herb and I talked at ACCU, he advises taht we we get what agreement we can in SG14 where SG14 is in full agreement, then go to EWG with SG14 blessing. but first makes sure we can get SG14 agreement first.

JM likes the split exception idea making a case indistinguishable if there is no catch clause
make code with throw statements compilable, but make it no different
AW: yes when std library throws eh, what is semantics, then semantics does not change,
upshot is you can compile these libraries
BC: must meet same requirement as on hosted implementations, so in those mode they would provide an array at

HT: that latter point of doing no unwinding unless you find a handler, if you have a binary distributed mode, of yoru library and it does throw, any stack unwinding that may happen in the library will be enough to kill this.
For those impl if they want to stay conforming, then they dont provide at, or dont unwind
so there is still burden for implementor imvestigation

I know GCC documents throwing with no catch does not unwind stack, clang might unwind

JW: this could force them to all do it the same way
could be whether they use libunwnd or libgcc_eh
or it could be the type info is not properly encoded when they have noexcept fn calling something that is noexcept code

Outcome from Dec call:
>> In the SG14 session, he mentioned 2 that he prefers
>> * Freestanding is signal / interrupt safe
>> * Freestanding requires no special dispensation from the operating
>> environment above what freestanding C99 requires
>>
>> But there are other possible directions
>> * Freestanding should be as small as possible
>> * Freestanding has all the same core language features as hosted
>>
>> Nov Evening Session:
>> http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1376r0.html
>>
>>
>> 2.2.2 Pointer Provedence:
>>
>> Pointer Provenance. I know this is a WG14 paper. But there is now
>> interest in following this with a WG21 paper.

>> http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2369.pdf
>> <https://www.google.com/url?q=http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2369.pdf&sa=D&source=hangouts&ust=1555590655253000&usg=AFQjCNEA6AjXuTUdhbL8PtTJyYfBGcrhzQ>
>>
>> http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1434r0.html
>>
>>
>> https://dedi5.nedprod.com/static/files/other/Dxxxx%20draft%202%20-%20Enhanced%20C%2B%2B%20memory%20and%20object%20model.pdf
>>
>> push from cambridge, peter Sewell's team trying to formalize C MM
mathematically
Chandler opposed
SG12 did more work work with C, Han did most of the liasion,
also was at Euro LLVM, this is Niall's interpretation
if object;'s lifetime, all conforming impl may have all pointers pointing
at that object invalid
if you new an item using placement new, then reuse that address, previous
pointers are invalid, new pointers to the new object is valid
these 2 ptrs can potentially equal, and this can cause problems
the rationale behind std launder which can point to an area to tell
lifetime is not what the compiler expects

C is now bringing this into their standard
hopefully get reconcilliation between the 2
compare equal and substitute one for the other are 2 different things

zap pointer life time
are they bitwise identical if pointed to the same object

N2369:
History on why we set it this way
What does C say,
best you can say is that if someone free a ptr, compiler is within its
right to cause indeterminate all copies of that pointer in the program no
matter where they are, cant load, compare or deref or touch the bits

difference in C++ lifetime (object lifetime) and storage duration (what
paul is talkign about)

What is the difference with C++?

If you free obj, all copies will be inderterminate, or at best have a trap
rep, cant load, store, or deref
this has been since C99, been there for a long time

for concurrent programming , this is too restrictive

Use after free bugs
have the free fn invalidate the pointer, std says you can do that, useful
for diagnotics
enables some optimizations,
if compiler can see ptr has it lifetime ending, then can kill one of the
branch, just use the else clause
Also future hardware can also traps on the load

ptr becoming indeterminate when obj lifetime ends in C (when you use
storage free and when stackframe dies),, can we have a motivating use case
for that

LIFO is old since 1973
classic algo push stuff on the stack
then atomically pop things off the stack and operate on it
set the next ptr, if we fail then try again
as soon as we load the value on the stack, this can become indeterminate

list_pop_all, get old value, just before excahnage, might get top of stack
and then we delay
we sequence through the stack, capture next ptr, calls foo on it
now copies of p and all ptrs to it become indeterminate
after p
sets p to next
as long as you dont mind having work done in reverse order, all you want to
compare the pointer

This should not work in C++, but some implementations might make it work

Niall says only trap derefernce then this algo will work fine

if you have additional pointer indirection, it could work, but will not
scale and invalidate decades of concurrent algo

if you coincidentally retrieve the same address, so spec use this death
trap to prevent impl from creating new memory, otherwise stack's won't work

Niall: I vaguely remember somebody on WG14 has a C compiler implementing
the trapping pointer thing. I thought it only traps dereference though, so
basically it's the same as setting all pointers to end of lifetimed objects
to null (though they don't do that)
In fact, the guy explained it that pointers get set to something like NaN,
so a non-bit-equal trap value

HT: if you compare ptrs, have to chaneg what compare means, between addr of
old and new object, for this to work to what people want, then have to
change spec to say what you want,
needto change definition of equality, if one or more ptr is to a lifetime
end of object, then it would have to be ok, to give a false positive
equality comparison
might not be a problem for C++

Niall:I thought the comparison of pointers of differing provenance is in
the proposed new C memory model? So, specifically, pointer to alive never
compares equal to pointer to dead?

Please post to Wg14 and C++ parallel mailing list

No one objects to moving forward with some intermediate solution that is
not the status quo for C.

Move forward for C++: because invalid ptr value is the same case for C++,
C++ says its implementation defined
to have it reliably work, then we need to do this, so we can do more with
invalid ptr values

Possibly the same wording, equality compare offer false positive, introduce
the idea of address space and say invalid ptr value just represents an
address
its ok to say they are not equal when they are

make this work for distinct source code, there is push back
existing source code using these algo needs to continue to work, because
our civilization depends on it
if impl change in a place where behaviour is undefined, then change would
only affect use cases in that category

we can check the behaviour of implementations

Jens M: I've been told multiple times that (equality) comparisons are not
in scope of the "provenance" papers.

2 ptrs that compare equal mght be same type, but in C++, can placement new
replace object in same storage,
ptr to old object, ptr to new object, same type, they will compare bitwise
identical
but those pointers are not interchangeable, because the object is not there
any more,
this is why storage duration and lifetime, is more distinct in C++ then C

so const member can have different values on these things, old ptr can
retain old value, but might see the new value as well

stack auto variable can return null

if you are happy to split difference between stack vs heap, this is the
solution based on storage duration
this might invaldate some algorithms but we dont know what those are

2.2.3 Linear Algebra update from April 3rd
>> http://lists.isocpp.org/sg14/2019/04/0076.php
>>
>> Next call: May 1 3 PM ET
>>
>> 2.2.4: Any serious study on cost of Exception vs cost of Error Codes
>>
>>
>> 2.2.5 any other proposal for reviews?
>>
>>
>> 2.3 Domain-specific discussions
>>
>> 2.3.1 Embedded domain discussions: Ben Craig, Wooter and Odin Holmes
>> 2.3.3 Games Domain: John McFarlane, Guy Davidson and Paul Hampson
>> 2.3.4 Finance Domain: Carl Cooke, Neal Horlock, Mateusz Pusz and Clay
>> Trychta
>>
>> 2.4 Other Papers and proposals
>>
>>
>> 2.5 Future F2F meetings:
>>
>> 2.6 future C++ Standard meetings:
>> https://isocpp.org/std/meetings-and-participation/upcoming-meetings
>>
>> - *2019-07-15 to 20: Cologne, Germany; *Nicolai Josuttis
>> - *2019-11-04 to 09: Belfast, Northern Ireland;* Archer Yates
>> -
>> - 2020-02-10 to 15: Prague, Czech Republic
>>
>>
>> - 2020-06-01 to 06: Bulgaria
>> - 2020-11: (New York, tentative)
>> - 2021-02-22 to 27: Kona, HI, USA
>>
>> 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
>>
>> May 8
>>
>>
>> 5.2 Future meeting
>> Apr 17: todays call
>> May 8
>> June 12: June 17 mailing deadline
>> July 10: likely cancelled due to Cologne Meeting July 15

Minutes for 2019/06/12 SG14 Conference Call

Meeting minutes by Staffan Tjernstrom

1.1 Roll call of participants

Staffan Tjernstrom, Michael Wong, Ben Craig, Billy Baker,Brett, Charles Bay, David Stone, Jan Wilmans, John McFarlane, Matthew Butler, Rene Rivera(Partial), Ronen Friedman, Guy Davidson (Partial)

1.2 Adopt agenda

Adopted

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

Review last call discussions.

2.2 Paper reviews
2.2.1 Embedded/freestanding vs hosted

Ben Craig:

The drafts of D1641R0.0 "Freestanding Library: Rewording the Status Quo"
and D1642R0.0 "Freestanding Library: Easy [utilities]". These papers can
also be reached at the following URLs:

https://raw.githack.com/ben-craig/freestanding_proposal/master/library/status_quo.html

https://raw.githack.com/ben-craig/freestanding_proposal/master/library/easy_utilities.html

These are follow-on papers from P0829 "Freestanding Proposal". I'm still
going in that direction, just with lots of little papers now, instead of
one big paper.

Summary of freestanding Kona evening session discussions
http://wiki.edg.com/bin/view/Wg21kona2019/SG14FreestandingImplementations

Outcome from Dec call:
In the SG14 session, he mentioned 2 that he prefers
* Freestanding is signal / interrupt safe
* Freestanding requires no special dispensation from the operating
environment above what freestanding C99 requires

But there are other possible directions
* Freestanding should be as small as possible
* Freestanding has all the same core language features as hosted

Nov Evening Session:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1376r0.html

Papers now have P Numbers

Trying to keep a small number of papers in flight at any point in time just in time LWG changes their editorial policy.

Discussion/Clarification that weâ€™re removing the requirement for freestanding not to provide feature test macros for non-freestanding features.

Discussion of what to do in those cases where we want to not include the whole overload set. Std::abs() with the floating point overloads is a case in point. Certain of the swap() cases may also be troublesome. For these papers we think weâ€™re ok, but the issue will very likely crop up in the later papers that start dealing with more string like things.

Calling the type operator new/delete is ok, but calling the global operator new/delete is not. Hence unique_ptr<> makes it in.

Both papers need champions for Cologne.

2.2.2 Concurrent Queue from David Stone

In response to P0260r3, an overview of the current art. Separate out concurrent vs non-concurrent is a viable strategy. Also papers by Guy Davidson (D????), P1470r0, and others.

Additional queues from Tony Van Eerd, and the Folly Team.

2.2.3 Error Size Benchmarking by Ben Craig
P1640R0: Error size benchmarking
https://raw.githack.com/ben-craig/error_bench/master/error_size_benchmarking.htmlvbfTSwvF3b5ym3XCQIh0_iFRNJbNk-FCc&m=_OFSroXnnYHKfBQqw8TVSac0et4fEQ80IMeaj-lWcD4&s=LGjT-

TVB94ptHzUmdPNh4LJr1eMpKuAcmL7pQSWzxxA&e=>
 Initial error neutral size cost is relative to the stripped_abort case.
Question about whether unreachable would be an even lower bound. No measurements as of yet.
Visual Studio has different implementation strategies between 32-bit and 64-bit compilations.
The light bars show the need imposed by the platform ABI to store unwind information on 64-bit. That restriction is not necessarily there on WinX32, or indeed on some embedded platforms.
The first time that you call a function (say a C function) that is not marked noexcept, with exceptions turned on, you pay the exception size penalty. This happens for Visual Studio in /EHc mode when calling abort().
As ever with microbenchmarks there was some necessary fighting the optimizer.
DG will be very interesting in these results.

2.2.4 Linear Algebra update from April 3rd
http://lists.isocpp.org/sg14/2019/04/0076.php

Michael gave a quick update on the current status. The idea of row and column vector is still a topic of discussion. There is a contrasting paper preferring a more eager evaluation technique from National Labs, avoiding expression templates.

Next call: July 10 2 PM EDT â€“ will very likely be cancelled due to Cologne.

Backup date is August 14 2 PM EDT.

2.2.5: Any serious study on cost of Exception vs cost of Error Codes

2.2.6 any other proposal for reviews?

2.3 Domain-specific discussions

2.3.1 Embedded domain discussions: Ben Craig, Wooter and Odin Holmes
2.3.3 Games Domain: John McFarlane, Guy Davidson and Paul Hampson
2.3.4 Finance Domain: Carl Cooke, Neal Horlock, Mateusz Pusz and Clay Trychta

2.4 Other Papers and proposals

2.5 Future F2F meetings:

2.6 future C++ Standard meetings:
https://isocpp.org/std/meetings-and-participation/upcoming-meetings

- *2019-07-15 to 20: Cologne, Germany; *Nicolai Josuttis
- *2019-11-04 to 09: Belfast, Northern Ireland;* Archer Yates
-
- 2020-02-10 to 15: Prague, Czech Republic

- 2020-06-01 to 06: Bulgaria
- 2020-11: (New York, tentative)
- 2021-02-22 to 27: Kona, HI, USA

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

TBD

5.2 Future meeting
Apr 17: todays call
May 8
June 12: June 17 mailing deadline
July 10: likely cancelled due to Cologne Meeting July 15