Document Number: P1071R0
Date: 2018-05-07
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 2018/04/11-2018/05/02

Contents

Minutes for 2018/04/11 SG14 Conference Call
Michael Wong, Ben Craig, Tony Tye, Barry Revzin, Herb Sutter, John Shaw , Billy Baker, Bran Sumner, Mateusz Pusz, Paul Bendixen, Guy Davidson, Vishal Ozer, Andreas Pokorny, The PhD.JeanHeyd Meneide, samantha Luber, Jan williams, Philip Johnston, Ben Saks
ACCU: Roger Orr, Phil Nash, Robin Joy, Odin Holmes, Felix Patricioni, Niall Douglas

1.2 Adopt agenda

Add polling at end of discussion "Does SG14 reviewed this paper and was in favor of moving it to EWG"
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

Review last call discussions

CPPCON SG14

2.2 Paper reviews

2.2.1 Exception and exceptionless EH by Herb Sutter

Herb presenting
Already reviewed by group of 20 people
but there are real concerns
Are we going to add another approach
make sure its worth solving,
not want to create yet another divergent thing
slide 4:

eh required but banned widely
survey of 3000 people asked which common error handling is used, 52% banned in all ...
they are not using std library, this fragments the language
need only 1 way of reporting errors
goal: 99% of code enables exception, reduce divergence (is this an OR statement?)
slide 6:
compare throwing n exception and catching it vs returning a std::error_code by value and checking it
in 2004, can eh be used for real time code, BS does not recommend it, but could it be solved in time
Slide 7:
not really zero overhead, not deterministic
what if we dont throw dynamic type, but throw known static type, then copy by value
propagation, its all local, can share the return channel
based on midori project by Joe duffy
error handling isomorphic to error codes, exception can be efficient
can get better then error code performance
Slide 8:
just throw a value
union{success; Error;} + bool using the same return channel
Slide 9:
Slide 10:
static exception spec throw->fn can throw std::error
an evolution of std::error_code +SG14 driven improvements already underway
Jan Williams: so bad_alloc will still allocate dynamically, and the stl doesn't change for that? that means I guess that this will still requirement dynamic allocation on embedded systems
get zero overhead + determinism,

Section 4.1.6
compare expected and this proposal
at call site, error handlign code is distinct from success code
most dont object to automatic propagation
all agree this is great
mow compare outcome with ths proposal
4.1.7: what we teach
4.1.8 how to migrate and toolability
back to slides 13
3 other proposed extensions
for those who want to throw other then std:error_code throw(E)
how to fix filesystem with dual error handling that use overload
can append operator++ throw(filesystem_error_
slide 14
exception control flow is invisible
need to reason about them
must use try with expression that can throw
Tony: How would slide 14 work in generic code that takes types which may or may not throw?

q&A has answer, some types may/not throw, query the type traits, std2 in this new model I would just decorate it,
conditional throw we can do it but not a fan of conditional noexcept

Slide 15
C++ EH adds a lot of boilerplates
catch{ is catch(error err){
Slide 16
cleanup
alternate success, use return
machine corruption, terminate
a programming bug, use contracts
after contracts: deprecate all error...and replace [[expects:..]
inherit future_error
Possible Polls:
Visahal: is throw part of the type system? Yes throw and throw(E), Duffy 2016, Midori
propagating a dynamic exception up leaks implementation details, now you know I am using bsave function
static exception
Additional queston ... please write that book
Timeframe: before cPPCON time, help in clang, and may be VS.

Barry Revzin:
Question: how does this interplay with std::expected? Is it intended to a complete replacement for it? Potential sugar for it? Is that union exposable as an expected?That is, is T f() throw(E) somehow equivalent to expected<T, E> f()

A: actively talking with other groups like expected and outcome in future scaling the back
Niall: outcome is throwing compielr to throw eh on yoru behalf, this proposal helps

Tony: In paper some mention that would be a compiler error if mismatch the try in a call expression with try on the function declaration being calling. So seems would imply have to write multiple versions of generic code to avoid getting the compile errors.

A: have thought about that: model in generic code on something that could throw,

hard compiler error could be done in generic code
constrainted generic code like concepts will have more knowledge

Vishal: abstract virtual functions may throw
A: if any overrider throws, must add throw to base, can be covariant

Jan Williams: from Jan Wilmans to everyone:
you mentioned in legacy code throws errror_code's are translated into dynamic exceptions and can be caught with catch (...), but should the error_code be translated to and std::exception dirivative so it would also be be caught with catch (std::exception&)?

A: this is 4.2 of paper part way down
if unhandled dynamic exception, if it is type errro we can just throw that, else translate it, thn badalloc (mostly what std lib throws),
if wrapped, just rethrow, else if throws exception throw bad_alloc, else throw e itself as dynamic exception,
only catch... would catch it?
A: yes
Paul B: if u forget the throw declaration and u proapage the static error, will it be converted to dynamic error
void f() {
the this is the second mapping, converted to dynamic exception,
would there be any type traits to see if there is any static or dynamic exception,
A: have not found need for such a trait, in Q&A section, all existing type traits just work
Do we need additional type traits to distinguish the 2 types? Have not found a need for that.
Use case is wheer currently cant use exceptions, ...
A; answer is in a different part, should be a mode to turn off exception, without turning off static exceptions
likely banning static exception anyway,

Guy Davidson: mixed mode in 4.1.2, in main with try calls g(), if try also calls one with dynamic exception specification, what happens
A: slide 10, any eh thrown from g will be caugh the catch (error), or just catch ...
might get potential confusing behavior

? thought we were using that mechanism of translating to std::error intrafunction
weaken conditional noexcept
Roger Orr: conditional noexcept, work with library implementers type composition

Guy: discussed this with engine lead of creative assembly: dont use it in games we release
cannot use it on PS4
usually just terminate

Odin: a month ago at embo conf, 100 kernel and lower microcontroller guys, Ben Craig freestanding is #1 issue, most did not know this
could propel us to use more C++

Ben Craig: millions of code in kernel side, so we use error code with constructor and destructors, want to use traditional throw and catch try,
has the potential,

Paul Bendixen: we really try not to unexpected in behaviour, just use constructors, ooperator overload

Andreas Pokorny Siemens: using C++ 14 just without eh, rtti, would like to use it, but runs out of memory when eh turned on

Q: can specify different type to be thrown as a static exception
does it need to be same size as std::error code
what kind of user defined type allowed?
A: in 4.2: default cibstructibe, no except movable
could be heap allocated using type erasure
this matches how exception is used today
Niall did measurement and feel it is in noise
need usability feedback
trivial, and noexcept movability
inclined to allow throw(e) and get experience with it

Ben: I wanted that but now less concern about it now, std::error_code can hold arbitrary exception
A: some want larger and smaller type

Barry rezvin: I don't have any interesting new opinions. I think it's an interesting idea.

Vishal: any contract implementation? attribute expected
A: no one knows
Odin: i have a comment on conditional throw and whether the compiler can static analysis on that
that should not be necessary, where would u get the info if not already in fn signature
arguing against the better programming ability

Jan williams: we use exception widely translte into a COM result, is that always a specifici eror code
A: have cstomization to enable that to write mapping their own error types
std::error code already has it.
A: have right inducement for me want to compile windows with translation

Phil Nash:
was in finance: can say that we disable exception, used ADT based exception, string conversion to double, would like this proposal
mentioned lot in paper, syntax is similar to swift
a: dynamic vs static
vISAL: void return type
A: Slide 8: as if returngn union success

1. Request unanimous consent: Yes

Assuming we can get performance numbers, if we can get better then dynamic EH
each is compared with status quo
2. void f() throw: unanimous consent:12 in favour and 2 neutral , none aganst,
are they mutually exclusive?
3. 4 in favour neutral: 2, against: 5

4. Slide 14: standalone error sugar to give you the code at bottom,
favour: 4+1(5) Neutral: 1 Against: 3+5 (8) for this specific form
Please email other sugars.

5. slide 16:
favour: 7+5 (12) neutral; 1 against: 0+1 (1) (breaks code)

Question from Paul on contracts: these are preconditions

- show quoted text -

5.1 Establish next agenda
May 2: Std; error by Niall

5.2 Future meeting

April 11: this meeting, Herb on Exceptionless vs Exception EH

May 2: outcome, expected, monad

June 13: after C++ Std meeting RAP may be cancelled

Minutes for 2018/05/02 SG14 Conference Call

Meeting minutes by Michael

Andreas Pokorny, Barry Revxin, Ben Craig, Charley Bay, Dalton Woodard, Herb Sutter, Niall Douglas, Paul Bendixon, Staffan Tjernstrom, Michael Wong, Jean Heyd Meneide, Jans Wilman

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

None.

2. Main issues (125 min)

2.1 General logistics

Review last call discussions

2.2 Paper reviews

2.2.1 papers by Niall Douglas

1. D1028 draft 2 *SG14 status_code and standard error object for P0709 Zero-overhead deterministic exceptions*

Charley Bay: Ambiguity on error code is another issue
Niall presenting on whiteboard
status_code <domain type>, returns void * payload
-success
-failure
-arbitrary payload (code domain)

status_code<void> : has no type, and is type erased, so no copy/move

3rd type is where everyone gets stuck
status_code<place_holder type erased <U>>
-u: any type bigh enough for domain type
-not copyable, to allow legal reinterpretcast
need info to go from known type to erased type
string ref

issues:
1. use of std::string
arthur saids const char *
Charley saids there is locality issues

Barry: I don't think you can just reinterpret_cast between arbitrary trivially copyable types.
That's still UB?
<I missed the answer>
Barry: yes that answers my question

2. Not freestanding compatible
are reference library allowed in freestanding:
there is none right now, but I like to allow it

3. proliferation of 2 api libraries
Herb will talk about that later

4. Singletons
error code relies on singleton to be single
e.g. asio,
each domain has a unique 64bit id
no state
and constexpr
that is how we got around the singleton problem
Paul: only one to pull out this random number is error code domain
Yes, so how large is it and how random?
they also may forget to update the number
it is 128 bit but that is excessive
Paul: yes that answers

next existing error coded lacks constexpr,
can factor out implementations so it is constexpr, Boost has it
this design is constexpr from ground up
codegen is denser, more optimized
different shippign implementations mileage will vary

next issue is fixed payload
the int is limited, prefer a void *
I use it for stack match tracing
as long as it is trivially copyable

next issue is comparison
error code and error condition
comparison are literal
this generates confusion
if not in same domain, then they should not compare

Dalton: comment on collision probability
Paul: std:hash domain name as a possible implementation
I can't enforce that other then documentation

Can throw?
pass to herb

what does if (ec) actually mean?
Beman and Chris K knows
this is not askign if error
its asking if the code is all bifs zero, independent of category

we replaced status_code with success or failure so it is not ambiguous
failure is implemented with virtual function call

various form of status code by Lawrence, but none gained traction
Ben: we should send to Chris K to get feedback
know his viewpoint, we should fix it along the way but he probably does not agree with that,
Beman said the same too
I agree its not a must have

Ben: you mention a way to embed an exception_ptr , requires registrar, heavier weigh status code, is there a way for someone who accepts nontrivial copies, and not deal with local store
Yes that is actually the next paper, on relocating moves

Ben: I am supportive of that, is there a way to store an exception_ptr in a status code with a registrar
Herb: just liek explicit new and delete with raw ptr except doing it by hand
you can just embed the type directly and call the destructor
if you dont know if the type is trivially relocatable, then store a pointer to it
good if we can do it for broader types or do automation in the language
Right Niall? Yes I sent the 2nd paper to Richard implementing it move relocate in clang
Herb: even the reference impl with global handle table, it is a distraction, does not need it
Any other feedback on the paper?

Is this paper SG14 would like to move forward to SG14
SF/Wf/N/WA/SA
8/2/1/0/0

Paul, does this work in freestanding?
Ben: it would work, but currently use std::string

2. D1029 draft 5 *SG14 [[move_relocates]]*

Arthur, they both get you to your goal, Richard may be incorrect doing move relocate as same as trivial abi, these are not the same
Feedback from sg14 is needed
This is a very small paper
Do people thing this will work
Ben: yes
Is there any other way of doing it?
Ben: for lightweigh exception handling case, passing thing through an out parameter, don't get to use the return channel
may be not ideal codegen, but decent,

Poll: Do you feel this approach moving exception pointer and other nearly trivial type around, non destructive move, is this the absolute minimal subset (and other things that involve a move-only)
Is there any relation to free standing library
No this is a language feature, can help library, sort of orthogonal, generate nicer code to embedded systems
Paul: can it be ignored as a QOI, other compilers can just take it as an optimization
In clang u opt in with attribute, so we have to enhance the language
Arthur: this is an ABi issue
trivial ABI attribute is recognized by itanium ABI
move operations have conservative lifetime
shows how the unique ptr optimization is eliminated in the paper,
Arthur: yes that is something you can do, but I think we will continue to disagree
when it passes the unique ptr across abi boundary, yes it must be indirect
within one function, not crossing abi boundary, (return or parameter) then that is a register

No have note talked to Chandler for many years, nor Pablo, just waiting for SG14, then will send.
Ben: dont thnk this attribute breaks ABi, which is good, but trouble when you start applying this attribute to existing types, that could break abi, it changes calling convention
be aware of that and it will make things difficult without extra wrapper types.
yes that makes sense
compiler have interesting exception_ptr impl, none use shared_ptr
SF/F/N/WA/A

1/10/2/0/0
Even I would ld prefer destructive move in the language

3.

2.2.2 Herb's discussion

Should heap exhaustion be treated specially?
https://groups.google.com/a/isocpp.org/forum/#!topic/sg14/9JMDxLvlBMc
Table at the top summarize 5 conditions
2 of these are not recoverable these you want to report to, use contract programming bugs, need to move those out of recoverable errors
because calling code, not the human can do something about it
the bottom of the table: alternate success, alternate post condition, then just return success, not an error please
is out of memory different from reportable not a programing bug
its requested from somewhere
heap exhaustion not due to abstract machine corruption
the program asked for memory, so different from programmigng bug
qualitatively different from cat 4: recoverable error
testing is different, code cannot test for it, no way to test for all those error paths
note to make it more explicit
recovery is different, write code in different way
push back for a vector, likely encounter failure even during recovery process
3rd on linux systems that overcommit memory,
cant implement bad alloc on that
alternative in virtual memory system, u may still not encounter itdue to thrashing
the reporting reference is also different
Andy saids want to fail fast
memory exhaustion is different and just terminate
in std in system_error, say dont represent it as system_error
if we take std library that now many fns can be noexcept
this paves the way to make it a std default and put noexcept on large number of std library fns

I want code understanding, what are all the things that could throw, 20 things, how do you recent about it, may be RAII
polling SG14 on this feeling, also within/outside microsoft
many are already doing these things
change from throwing bad alloc for memory exhaustion, such that they terminate
for every std function, try to allocate big buffer, image, but support that code with differet
P0132 proposes this kind of function, LEWG was warm to the idea
yes that is why this proposal start a discussion
I think most code wont notice, because they wont reach allocation failure in practice,
terminate by default and install new handler, major projects are already doing that such as google nothrow stl
excel does this for try, often will get out of memory on large spreadsheet, they tell people go use 64 bit excel
by default terminate on failure, new handler shows teh message dialog and gets same behaviour
carrot is tat we get rid of this pervasive class of errors,
Jan: do does terminate by default make it easier to test?
Herb: it would separate the way its reported from other errors
handling them differenty may lead to better tools
Jan: this cleans up some error paths
Herb: yes, lets treat it in a targetted way
Jan: it is not in error path as the others, clean up is needed
Herb: now opting in there is clean up
Jan: is that your goal? would cleanup of error handling path?
Herb: main benefit is there is a different class of error, so removing it allows the other to be noexcept,
noexcept gets lesser code paths, better optimization,
Niall: STL has non determinate, can we start with new set of containers
Herb: lots of thngs aimed for std 2, but in JAX we decide to not go that way, WG21 wants to incrementally improve what we have
Niall: I like it with bad_alloc, my work uses it as control flow path, that kind of code would break
what would it mean for that kind of code, we need to discuss, opt in may be
try the call tree example
Dalton's comment:
we see code that thinks it handles bad_alloc, hard without data to guess what the ratio is,90% that thinks they handle the code...
Guy Davidson:
discussed Herb's proposal: what to do with bad_alloc
Herb: unique situation, sound like you are saying it should be treated differently
Jan: dont see how mapping bad alloc to std terminate would make it more recoverable
Guy: more specific thing in the client code to make it make space
Herb: 2 ways: terminate makes it more recoverable
after test time: by havign code that believes it can handle exhaustion, then its got to try everywhere
2. use of fuzzers, but std now makes my bad alloc terminate, so now I just wrap them all, but I opted into it. so I can call the try function

Herb:
1. pursue bad alloc in principle to investigate separating handling then the other proposals
Niall: does this cover alisdair's allocator method?
I dont know
SF/WF/N/WA/SA
9/0/3/0/0
2. specifically along the proposal for bad alloc to teminate, plus new no throw and try_functions
SF/WF/N/WA/SA
2/6/3/2/0

Guy:?
anything that emits bad_alloc today would terminate, gets a handler, does not throw,

Jan: why change the default to terminate if we have handler today
Herb: cant make new handler to throw an exception, this makes the new handler to use noexcept, now tell people to use 64bit and bail

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

2.5 Future F2F meetings:

2.6 future C++ Standard meetings:

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4738.pdf

2018-06 RAP WG21 meeting information

Find a hotel towards Zurich and near a train station,

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 after June 13

5.2 Future meeting

April 11: this meeting, Herb on Exceptionless vs Exception EH

May 9: status code outcome, expected, monad

June 13: after C++ Std meeting RAP may be cancelled