P1675R1: `rethrow_exception` must be allowed to copy

Billy O'Neal

Audience: CWG, LWG

Changes since R0

- Retargeted at LWG after EWG and CWG review.
- Removed italics from exception object in the proposed resolution.
- Removed drafting note CWG resolved.
- Used 'throws' in wording instead of emits as requested by Daniel Krugler. (I really really really dislike statements that the library is throwing an exception when the library is in fact transparently.
- Applied LWG wording feedback.

Discussion – MSVC++'s current behavior

The `current_exception` wording was carefully written to allow both ABIs like MSVC++'s where the *exception object*s are generally constructed on the stack, and ABIs like the Itanium C++ ABI where the *exception object*s are generally constructed on the heap (and possibly reference counted). Implementations are given the freedom they need to (possibly) copy the *exception object* into the memory held by the `exception_ptr`, and similar. See http://eel.is/c++draft/propagation#8.

Unfortunately, such care was not taken for `rethrow_exception`. The language at http://eel.is/c++draft/propagation#10 suggests that the same *exception object* needs to be used for the rethrown exception. This is not possible on MSVC++'s ABI today, because the catch block handler calls the destructor for the *exception object* (if necessary). If we threw the same object stored in the exception_ptr, it would end up having its destructor called each time it was rethrown.

The existing implementation MSVC++ uses copies the stored exception into an `alloca`'d memory block in `rethrow_exception`, and changes the *currently active exception* TLS value to that `alloca`'d block, then rethrows that as if by `throw;`.

Previous WG21 examination of this area

In reflector discussion "[isocpp-core] [isocpp-lib] std::rethrow_exception might throw another exception type" Herb Sutter did some archaeology and found that the committee has considered this area in the past. Quoting his response on the reflector:

> Just to be crisp: It could be not just a different exception object of the same type, but a different exception object of an arbitrarily different type entirely. Here's a distilled example:
>
> ```
> struct X: std::exception {
>   // … data members whose copy could throw…
>   X() { }
>   X(X const&) { if(oh_no()) throw 42; /* else success */ }
> };
> ```

```
int main() {
   try {
      std::exception_ptr eptr;
      try { throw X(); }
      catch(X& x) { std::cout << "caught X with address " << (void*)&x;  eptr =
std::current_exception(); }
      std::rethrow_exception(eptr);
   }
   catch(X& x) { std::cout << " caught X with address " << (void*)&x; }
  catch(int) { std::cout << " caught int"; }
}
```

I think what Billy is saying is that all of the following results need to be possible:

a) "caught X with address XYZZY caught X with address XYZZY"

b) "caught X with address XYZZY caught X with address PLUGH"

c) "caught X with address XYZZY caught int"

... because on Windows it is impossible to do (a), an implementation can do only (b) or (c) (and can't guarantee (b) because the copy can throw, so also (c)). Correct?

It appears (b) was considered and rejected, and (c) was never considered.

Archaeology:

In Toronto (2007-07), N2179 was adopted. It says what the standard says now, to rethrow the same exception.

The Core wiki notes at that meeting mention only one issue (related to whether exception_ptr can be null), “Otherwise no issues found.” So this was not discussed as an issue for implementations.

In Oxford (2007-04), the LWG wiki notes says only that N2179 was discussed and “has strong support” (but no details).

The predecessor paper was N2107.

In Portland (2006-10), the EWG wiki notes say N2107 presented on Monday (no details, but presumably went against Option 1 (see next) because we never saw it mentioned again), and the Concurrency wiki notes say it was discussed on Friday (but no details).

In N2107, the only mention of a copy is in Option 1 where a copy could happen (but the wording for that option still said “reactivates the _currently handled_ exception” [emphasis added], and Option 2 is that it can’t copy at all (which was what Peter wrote up on N2179).

> Based on that history, changing this text to allow even outcome (b), a copy, would be a design change that was an option not approved (and appears to have been disapproved) by EWG in 2006. And I can't find any suggestion anywhere that you could end up with outcome (c), a totally different exception type. Allowing either (b) or (c) changes the design of the feature as approved by EWG (and Concurrency, LWG, Core) and affect how we teach users to use it.

Additionally Ville Voutilainen points out that there appears to have been an NB comment against C++11 arguing to force implementations to make a separate copy, tracked as LWG 1369 (discussion).

That previous consideration and rejection is different than proposed here. Then, the discussion was whether to force the Itanium-like ABIs to make a separate copy, not proposed here. This proposes only the minimum change necessary to make MSVC++'s ABI function (and presumably other ABIs where the rethrow *exception object*s must be distinct).

Unclear that the current wording reflects the original intent

Peter Dimov, the author of N2179 which appears to be the source of the current wording, also reported:

> > ... and Option 2 is that it can't copy at all (which was what Peter wrote
> > up on N2179).
>
> To clarify, this is not quite what I wrote, or at least not what I intended
> to write. N2179 is a bit light on specification, but that's because it
> wasn't clear, at the time it was written, whether a tightened version would
> be implementable. Still,
>
> Throws: the exception to which p refers.
>
> is intended to mean that `rethrow_exception` throws as-if
>
> throw *p;
>
> where `p` refers to the original exception or a copy of the original
> exception, as captured by `current_exception`.
>
> That is, a copy is allowed, and if the copy throws, the behavior is intended
> to match that of `throw x` when the copy constructor of `x`, if not elided,
> throws.
>
> Throwing the exact same object, down to the same address, was never intended
> to be required; you can call `rethrow_exception` more than once, so throwing
> a copy creates fewer problems, on net.

Moreover, even the current standards text seems to acknowledge that a copy might be made here. In http://eel.is/c++draft/propagation#7 the text says (emphasis mine):

> [Note: **If rethrow_exception rethrows the same exception object (rather than a copy),** concurrent access to that rethrown exception object may introduce a data race. Changes in the number of exception_ptr objects that refer to a particular exception do not introduce a data race. — end note]

Proposed Resolution

This wording is relative to N4810. Change [propagation]/9-10 as indicated:

```
[[noreturn]] void rethrow_exception(exception_ptr p);
```

-9- *Expects*: p is not a null pointer.

~~-10- *Throws*: The exception object to which p refers.~~

-?- *Effects*: Either throws the exception object to which p refers, or a copy of that exception object. It is unspecified whether a copy is made. If the implementation makes a copy:

- the memory for the copy is allocated in an unspecified way; if this is not possible, `rethrow_exception` throws an instance of `bad_alloc`

- if copying the exception to which p refers throws an exception, `rethrow_exception` throws that exception

- otherwise, rethrow_exception throws the copied exception.

Thanks

Special thanks to Herb Sutter, Ville Voutilainen, and Peter Dimov for finding these references, and to Jonathan Wakely and Jens Maurer for representing the Itanium ABI position in reflector discussions.