Throw2goto: A step towards zero-overhead dynamic exceptions

**Date:** April 28, 2020

**Document Number:** idk

**Project:** Clang/LLVM

**Audience:** LLVM-dev

**Reply-To:** Jeff Linahan [jeff.linahan@gmail.com](mailto:jeff.linahan@gmail.com)

**Introduction**

We propose an optimization to exception handling that achieves zero overhead in both space and time compared to ordinary flow control in many cases. The technique can be used to transparently speed up existing C++ code without changing any language features.

**Motivation**

After decades of existing as a supported language feature, exceptions are still not universally accepted as an error handling mechanism in C++.  Many have cited that exception handling is a violation of the zero overhead principle.  Part (1) many developers use -fno-exceptions to prevent the compiler from generating RTTI data and part (2) the generated assembly code is often larger and slower than using other control flow mechanism.

The purpose of exception handling is to signal an error that cannot be handled locally, thus decoupling the code that detects it from the code that (attempts to) recover from it. However, much exception handling code is similar to the C-style of if-checks and a cleanup clause at the end of a function. Modern compilers are able to implement exceptions with no runtime overhead when everything is going fine at the expense of a severe slowdown during the stack unwinding process when a throw occurs. For this reason, many developers continue the C-style error checking of if’s and goto’s for errors that can be handled locally, often going so for as to disable exceptions entirely. This unfortunately means that if’s must be used everywhere, thus missing out on zero-overhead in the happy case.

Consider that the following program does nothing:

int main() try {throw 42;} catch (int e){}

Clang generates exception handling code with -O3 like this:

main:                                   # @main

        push    rax

        mov     edi, 4

        call    \_\_cxa\_allocate\_exception

        mov     dword ptr [rax], 42

        mov     esi, offset typeinfo for int

        mov     rdi, rax

        xor     edx, edx

        call    \_\_cxa\_throw

        mov     rdi, rax

        call    \_\_cxa\_begin\_catch

        call    \_\_cxa\_end\_catch

        xor     eax, eax

        pop     rcx

        ret

With proper optimization, this should get changed to:

main:                                   # @main

        xor     eax, eax

        Ret

If we std::cout the int to prevent the optimizer from eliding it, we have:

#include <iostream>

int main(){

    try{

        throw 42;

    } catch(int e){

        std::cout << e;

    }

}

And with -O3:

main:                                   # @main

        push    rbx

        mov     edi, 4

        call    \_\_cxa\_allocate\_exception

        mov     dword ptr [rax], 42

        mov     esi, offset typeinfo for int

        mov     rdi, rax

        xor     edx, edx

        call    \_\_cxa\_throw

        mov     rbx, rax

        cmp     edx, 1

        jne     .LBB0\_5

        mov     rdi, rbx

        call    \_\_cxa\_begin\_catch

        mov     esi, dword ptr [rax]

        mov     edi, offset std::\_\_1::cout

        call    std::\_\_1::basic\_ostream<char, std::\_\_1::char\_traits<char> >::operator<<(int)

        call    \_\_cxa\_end\_catch

        xor     eax, eax

        pop     rbx

        ret

        mov     rbx, rax

        call    \_\_cxa\_end\_catch

.LBB0\_5:

        mov     rdi, rbx

        call    \_Unwind\_Resume

The case of throwing int is not as niche as it might first appear, as throwing errno when mixing C and C++ is a common practice. Indeed, when throwing any built-in type we can simply gensym\* an identifier to hold the value and substitute it for the caught exception in the catch block. For the case of throwing nothing, if an exception is currently not being thrown (like in a function declared noexcept) the throw can simply be replaced with std::terminate(). If an exception is being thrown the \_\_cxa\_rethrow call remains.

We should note the output of int main(){throw;}

main:                                   # @main

        push    rax

        call    \_\_cxa\_rethrow

This \_\_cxa\_rethrow call can be changed to std::terminate() or the xor and return like above

If all [[1]](#endnote-1)

1. \*gensym is a function in Common Lisp that generates a symbol name guaranteed to be unique (i.e. not cause an ODR violation)

   **References**

   Sutter, Herb. Zero-overhead deterministic exceptions: Throwing values. <http://open-std.org/JTC1/SC22/WG21/docs/papers/2018/p0709r0.pdf>

   C++ exceptions and alternativeshttp://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1947r0.pdf [↑](#endnote-ref-1)