RTTI in std::current_exception

by T P K Healy

A C++ program can link with a shared library which has incomplete or inaccurate documentation for the exceptions it throws, or the program may link at runtime using '`LoadLibrary`' or '`dlopen`' with a library which it knows very little about. Code in the main program may handle an exception thrown from within the shared library, as follows:

```
#include <exception>  // exception, exception_ptr
#include <typeinfo>   // typeid, type_info
#include <new>        // bad_alloc

extern void SomeLibraryFunction(void) noexcept(false) {}

int main(void)
{
    try { SomeLibraryFunction(); }
    catch (std::bad_alloc const & ) { /* Do something */ }
    catch (std::exception const &e)
    {
        std::type_info const &ti = typeid(e);
        // As 'std::exception' has a virtual destructor, it is a polymorphic
        // type, and so 'typeid' will give us the RTTI of the derived class
    }
    catch (...)
    {
        std::exception_ptr const p = std::current_exception();
        // We have no idea the type of the object which was thrown ? ? ?
    }
}
```

I propose that the RTTI of the object which was thrown should be obtainable inside the body of the '`catch (...)`' as follows:

```
catch (...)
{
    std::type_info const &ti = std::current_exception_typeid();
}
```

This proposed feature would not be needed if we could only throw objects derived from '`std::exception`', however since we can throw any type (for example `int`, `std::complex<long>`), it is fitting that we should be able to get the RTTI inside '`catch (...)`', otherwise the C++26 standard should deprecate the throwing of an object which is not derived from '`std::exception`'.

I propose the addition of the following text to **Section 17.9.7** of the C++26 Standard under the heading of '*Exception propagation*':

```
type_info const &current_exception_typeid(void) noexcept;
        Returns: A reference to a 'type_info' object that refers to the
                 currently handled exception (14.4), or 'typeid(void)' (7.6.1.8)
                 if no exception is being handled. The return values of two
                 successive calls to 'current_exception_typeid' refer to the
                 same 'type_info' object.
```

Possible Implementations

GNU g++ version 12.2 - GodBolt: https://godbolt.org/z/17qja53K6

Compile with: `g++ -o prog.exe code.cpp -std=c++20`

```
#include <iostream>  // cout, endl
#include <typeinfo>  // type_info
#include <cxxabi.h>  // __cxa_current_exception_type

std::type_info const &current_exception_typeid(void) noexcept
{
    std::type_info const *const p = abi::__cxa_current_exception_type();
    if ( nullptr == p ) return typeid(void);
    return *p;
}

int main(void)
{
    try
    {
        throw 5.2L;
    }
    catch(...)
    {
        std::cout << current_exception_typeid().name() << std::endl;
    }
}
```

Microsoft MSVC version 19.33 - GodBolt: https://godbolt.org/z/hYasfEanx

Compile with: `cl /Fe:prog.exe code.cpp /std:c++20 /EHsc`

```
#include <iostream>  // cout, endl
#include <cstdint>   // uintptr_t
#include <typeinfo>  // type_info

#include <Windows.h> // AddVectoredExceptionHandler
#include <ehdata.h>  // ThrowInfo

namespace detail {
    thread_local std::type_info const *tl_ti = &typeid(void);
}

std::type_info const &current_exception_typeid(void) noexcept
{
    return *detail::tl_ti;
}

int main(void)
{
    try
    {
        throw 7.8L;
    }
    catch (...)
    {
        std::cout << current_exception_typeid().name() << std::endl;
    }
}

namespace detail {

using std::type_info;

type_info const *VectoredExceptionHandler_Proper(EXCEPTION_POINTERS const *const arg)
{
    using std::uintptr_t;

    if ( nullptr == arg ) return nullptr;

    EXCEPTION_RECORD const *const pexc = arg->ExceptionRecord;

    if ( nullptr == pexc ) return nullptr;
```

```
    switch ( pexc->ExceptionCode )
    {
    case 0x40010006 /*EXCEPTION_OUTPUT_DEBUG_STRING*/:
    case 0x406D1388 /*EXCEPTION_THREAD_NAME        */:
        return nullptr;
    }

    if ( 0x19930520 /* magic number */ != pexc->ExceptionInformation[0] )
        return nullptr;

    if ( pexc->NumberParameters < 3u ) return nullptr;

    uintptr_t const module =   (pexc->NumberParameters >= 4u)
                             ? pexc->ExceptionInformation[3u]
                             : 0u;

    ThrowInfo const *const pthri =
        static_cast<ThrowInfo const*>(
            reinterpret_cast<void const*>(
                static_cast<uintptr_t>(pexc->ExceptionInformation[2u])));

    if ( nullptr == pthri ) return nullptr;

    if ( 0 == pthri->pCatchableTypeArray ) return nullptr;

    if ( 0u == (0xFFFFFFFFu & pthri->pCatchableTypeArray) ) return nullptr;

    _CatchableTypeArray const *const pcarr =
      static_cast<_CatchableTypeArray const *>(
        reinterpret_cast<void const*>(
          static_cast<uintptr_t>(module + (0xFFFFFFFFu & pthri->pCatchableTypeArray))));

    if ( 0u == ( 0xFFFFFFFFu &
        reinterpret_cast<uintptr_t>(pcarr->arrayOfCatchableTypes[0u])  )  )
           return nullptr;

    CatchableType const *const pct =
      static_cast<CatchableType const*>(
        reinterpret_cast<void const*>(
          static_cast<uintptr_t>(module +
            (0xFFFFFFFFu & reinterpret_cast<uintptr_t>(
              pcarr->arrayOfCatchableTypes[0u])))));

    if ( 0u == (0xFFFFFFFFu & pct->pType) ) return nullptr;
```

```
    type_info const *const pti =
        static_cast<type_info const *>(
            reinterpret_cast<void const*>(
                static_cast<uintptr_t>(module + (0xFFFFFFFFu & pct->pType))));

    return pti;
}

long WINAPI VectoredExceptionHandler(EXCEPTION_POINTERS *const pointers)
{
    type_info const *const retval = VectoredExceptionHandler_Proper(pointers);

    if ( nullptr == retval ) tl_ti = &typeid(void);
                        else tl_ti = retval;

    return EXCEPTION_CONTINUE_SEARCH;
}

void *const dummy = ::AddVectoredExceptionHandler(1u, VectoredExceptionHandler);

} // close namespace 'detail'
```

Links to GodBolt

GNU g++ : https://godbolt.org/z/17qja53K6

Microsoft MSVC : https://godbolt.org/z/hYasfEanx

Related Proposals

Runtime type introspection with std::exception_ptr by Aaryaman Sagar:
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0933r1.pdf

How to catch an exception_ptr without even try-ing by Mathias Stearn:
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1066r1.html