How can I turn a structured exception into a C++ exception without having to use /EHa, if I can constrain exactly where the structured exception is coming from?

devblogs.microsoft.com/oldnewthing/20200117-00

January 17, 2020

Raymond Chen

Last time, we looked at how you can handle both structured exceptions and C++ exceptions coming out of a plug-in. But the `_set_se_translator` function requires that your code be compiled with the `/EHa`. The customer was wondering if there was something that avoided the need for `/EHa`.

One possibility is compiling parts of the program with `/EHa` and parts without. This is explicitly called out as not recommended. The reason is that code compiled with `/EHs` (the opposite of `/EHa`) assumes that exceptions can be raised only by `throw` statements, and `_set_se_translator` violates that rule because it allows C++ exceptions to be thrown by any Win32 exception.

But all is not lost. In this case, we need the asynchronous-to-synchronous conversion only for a specific block of code. If we can make sure no asynchronous exceptions escape into code compiled with `/EHs`, we won't violate the assumptions of `/EHs`.

```
HRESULT InvokeWithCustomExceptionTranslation(
    CALLBACK_FUNCTION fn, /* other arguments */)
{
  __try {
    return fn(/* other arguments */);
  } __except (GetExceptionCode() == MSVC_EXCEPTION ?
              EXCEPTION_CONTINUE_SEARCH : EXCEPTION_EXECUTE_HANDLER) {
    throw win32_exception(GetExceptionCode(),
                          get_stack_trace(GetExceptionInformation()));
  }
}

// InvokeCallback same as before
```

If an asynchronous exception occurs during execution of the lambda, we check if it was a C++ exception. If so, then we let it go through so that the runtime can deal with it. Otherwise, we convert it to a C++ exception on the spot.

It is important that there not be anything with a destructor in the `__try` block, because the asynchronous exception will bypass all destructors. Fortunately, the compiler will yell at you if you make this mistake.

It is also important that this function handle all asynchronous exceptions (aside from C++ exceptions themselves), so that no asynchronous exception escapes into `/EHs` code.

On the other hand, we hard-coded some secret knowledge of the compiler's implementation, namely, that `MSVC_EXCEPTION` is the exception code used for C++ exceptions. If you are running in an environment where the plug-in could be written in managed code, then you will turn managed exceptions into `win32_exception`.

Which brings us back to what we had before: Using the `_set_se_translator` function to switch from `/EHs` mode to `/EHa` mode temporarily.

This is one of those cases where advice needs to come with a rationale: If you understand why a rule exists, you can understand when you are in a case where the rule doesn't apply.

And we are in one of those cases. The rason for the guidance against mixing `/EHs` and `/EHa` in the same program is that you don't want asynchronous exceptions to be converted to synchronous exceptions when the calling code isn't expecting it. By scoping the use of the `_set_se_translator` function to a single block of code, we can verify by inspection that asynchronous exceptions never escape into code that doesn't expect it.

This code needs to be carefully commented with a note that it needs to be compiled in a specific way, and giving an explanation not only why that is necessary, but also why it is safe.

Raymond Chen

Follow