What happens to custom unhandled exception filters if a debugger is not running?

devblogs.microsoft.com/oldnewthing/20180726-00

July 26, 2018

Raymond Chen

A customer wanted to know what happens to custom exception filters if a debugger is not running. They included this sample code fragment:

```
#define CUSTOM_CODE 0x87654321

LONG CALLBACK CustomFilter(EXCEPTION_POINTERS* pointers)
{
 Log("Inside exception filter");
 if (pointers->ExceptionRecord->ExceptionCode == CUSTOM_CODE) {
  Log("Handling custom exception");
  return EXCEPTION_CONTINUE_EXECUTION;
 } else {
  Log("Allowing handler to execute");
  return EXCEPTION_EXECUTE_HANDLER;
 }
}

void Test()
{
 SetUnhandledExceptionFilter(CustomFilter);
 Log("Raising custom exception");
 RaiseException(CUSTOM_CODE, 0, 0, nullptr);
 Log("Raised custom exception");

 try {
  Log("Raising custom exception inside try");
  RaiseException(CUSTOM_CODE, 0, 0, nullptr);
  Log("Raised custom exception inside try");

  Log("Throwing runtime_error inside try");
  throw std::runtime_error("oh dear");
  Log("Threw runtime_error inside try");
 } catch (std::runtime_error&) {
  Log("Caught runtime_error");
 }
}
```

The customer reported that the results of the `Test` function are highly context-dependent. If the `Test` function is called from `main`, then it works, provided that a debugger is not attached. But if they call the `Test` function from inside their window procedure, then the behavior is different based on whether the program is 32-bit or 64-bit, and whether the operating system is 32-bit or 64-bit, as discussed on MSDN.

The customer also noted that the documentation for the `SetUnhandledExceptionFilter` function talks about what happens if no debugger is attached, but what about the case where there *is* a debugger?

Okay, let's answer the last question first. If a debugger is attached, then the custom unhandled exception filter is ignored.

What about the complicated behavior if you raise an exception from inside a window procedure and try to handle it in the unhandled exception filter?

Yeah, well, that's complicated.

But you shouldn't be doing that in the first place.

Because you are violating one of the cardinal rules of Win32 exceptions: Exceptions must not cross foreign stack frames. If you're going to raise an exception in one place and handle it in another, then every stack frame that the exception travels through must be aware of the funny game that you're playing.

The unhandled exception filter runs as the very last exception filter. (And only if there is no debugger attached.) This means that before control even reaches the unhandled exception filter, it must go through every active stack frame on your thread, including stack frames outside your control (such as the code that manages the window procedure callback). Which means you have already broken the rules.

So don't do that.

Raymond Chen

Follow