On installing a custom unhandled exception filter and intentionally raising an exception to get its attention

devblogs.microsoft.com/oldnewthing/20160817-00

August 17, 2016

Raymond Chen

A customer reported that they were seeing inconsistent behavior when they intentionally raised a Win32 exception and tried to catch it in a custom unhandled exception filter.

```
#define CUSTOM_EXCEPTION 0xABCDEF01

LONG WINAPI CustomFilter(EXCEPTION_POINTERS* exceptionPointers)
{
 if (exceptionPointers->ExceptionRecord
                      ->ExceptionCode == CUSTOM_EXCEPTION) {
  return EXCEPTION_CONTINUE_EXECUTION;
 }
 return EXCEPTION_EXECUTE_HANDLER;
}

void Test()
{
 auto previousFilter = SetUnhandledExceptionFilter(CustomFilter);
 RaiseException(CUSTOM_EXCEPTION, 0, 0, nullptr);

 try {
  RaiseException(CUSTOM_EXCEPTION, 0, 0, nullptr);
  throw (int)0;
  Log("Returned from throw");
 } catch (int) {
  Log("Caught");
 }
 SetUnhandledExceptionFilter(previousFilter);
}
```

The customer observed a few things.

First, if the `Test` function was called from inside a window procedure, then the behavior changed depending on the execution environment, as documented here.

Second, the custom filter was not called at all if the program was running under the debugger. The documentation for `SetUnhandledExceptionFilter` says that if the program is not being debugged, then the custom unhandled exception filter is called, but it doesn't say what happens if the program *is* being debugged.

The customer's question was "What is the expected behavior if the program is being debugged?"

First, let's answer the question: The expected behavior if the program is being debugged is that the custom unhandled exception filter is ignored.[1]

But let's step back and look at the bigger picture here.

This program is violating one of the cardinal rules of Win32 exceptions: Exceptions must not cross foreign stack frames. If you are 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 your little game. After all, if they aren't aware of your game, you don't know what they will do when they see your custom exception!

The unhandled exception filter runs as the very last exception filter, which means that before control reaches the unhandled exception filter, it must go through every single active stack on your thread, including the stack frames outside your control, like the ones that set up the call to the window procedure. So you've already left the world of predictable behavior.

[1]The intended purpose of the custom unhandled exception filter is to capture additional program state for post-mortem debugging purposes. If a debugger is connected to the process, the thinking is that you connected the debugger because you want it to be informed of the exception and freeze the program so you can, y'know, debug it.

Raymond Chen

Follow