Sucking the exception pointers out of a stack trace

devblogs.microsoft.com/oldnewthing/20060821-17

August 21, 2006

Raymond Chen

Often, you find yourself staring at a stack trace for a caught exception and want to see the original exception.

```
ChildEBP RetAddr  Args to Child
030c21d0 76df3448 00000000 030c6138 76db6b0d ntdll!DbgBreakPoint
030c21dc 76db6b0d 030c2204 77b8d585 030c220c ole32!PeekMessageExceptionFilter+0x42
030c21e4 77b8d585 030c220c 00000000 030c220c ole32!CCliModalLoop::MyPeekMessage+0x41
030c220c 77f36992 030c25d0 030c6128 030c22e8 msvcrt!_except_handler3+0x61
030c2230 77f36964 030c25d0 030c6128 030c22e8 ntdll!ExecuteHandler2+0x26
030c22d8 77f36884 030c1000 030c22e8 00010007 ntdll!ExecuteHandler+0x24
030c25b8 77f6e0dd 030c25d0 00000000 00000000 ntdll!RtlRaiseException+0x3d
030c262c 77d3c239 77d4a4b6 77d3e2c5 030c3767
ntdll!RtlDeactivateActivationContextUnsafeFast+0x233
030c2630 77d4a4b6 77d3e2c5 030c3767 030c26a0 USER32!UserCallWinProcCheckWow+0x167
030c2634 77d3e2c5 030c3767 030c26a0 77d4a46f USER32!_NLG_Return2
030c265c 77d3e288 030c57b4 ffffffff 030c2688 USER32!__local_unwind2+0x70
030c2688 77f36992 030c26f8 030c57b4 030c27a4 USER32!_except_handler3+0xd5
030c26ac 77f36964 030c26f8 030c57b4 030c27a4 ntdll!ExecuteHandler2+0x26
030c2a74 77b8d36d 030c6128 77b8d36d 00000000 ntdll!ExecuteHandler+0x24
030c2a9c 77b8d59d 030c6128 030c2ac0 00000000 msvcrt!__global_unwind2+0x18
030c2ac0 77f36992 030c2ba4 030c6128 030c2bc0 msvcrt!_except_handler3+0x75
030c2ae4 77f36964 030c2ba4 030c6128 030c2bc0 ntdll!ExecuteHandler2+0x26
030c2b8c 77f36796 030c1000 030c2bc0 030c2ba4 ntdll!ExecuteHandler+0x24
030c2b8c 77b7aa54 030c1000 030c2bc0 030c2ba4 ntdll!KiUserExceptionDispatcher+0xe
030c3300 77b7b4dc 030c3324 7715b1b4 00000000 msvcrt!_woutput_l+0x18
```

(You too can get symbols for operating system binaries, either by using the symbol server to get the symbols on-demand or, if you have a gigabyte of disk space, you can download symbol packages to get them all at one go. Even if you go for the symbol package, you still need the symbol server, since it gets updated with symbols for binaries that have been updated since the most recent service pack.)

Here, we caught an exception in the `PeekMessageExceptionFilter`. What was the exception? Well, an exception filter receives a pointer to an `EXCEPTION_POINTERS` structure as its argument.

```
typedef struct _EXCEPTION_POINTERS {
    PEXCEPTION_RECORD ExceptionRecord;
    PCONTEXT ContextRecord;
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

Here, we see that the parameter to `PeekMessageExceptionFilter` is

```
030c21dc 76db6b0d 030c2204 77b8d585 030c220c ole32!PeekMessageExceptionFilter+0x42
0:0000> dd 030c2204 l2
030c2204 030c25d0 030c22e8
         -------- --------
         .exr     .cxr
```

The first value points to the exception record and the second points to the context record. You can view the exception by typing `.exr 030c25d0` and view the context for the exception (i.e., what code was executing when the exception occurred) by typing `.cxr 030c22e8`. Those two values also appear as the first and (go figure) third parameters to `ExecuteHandler2`.

It so happens that doing the `.exr` on this particular exception record reports that the exception was `c015000f` which happens to be `STATUS_SXS_EARLY_DEACTIVATION`, and after setting the context to the exception context record, the stack goes

```
ChildEBP RetAddr
030c262c 77d3c239 77d4a4b6 77d3e2c5 030c3767
ntdll!RtlDeactivateActivationContextUnsafeFast+0x233
030c2630 77d4a4b6 77d3e2c5 030c3767 030c26a0 USER32!UserCallWinProcCheckWow+0x167
030c2634 77d3e2c5 030c3767 030c26a0 77d4a46f USER32!_NLG_Return2
030c265c 77d3e288 030c57b4 ffffffff 030c2688 USER32!__local_unwind2+0x70
030c2688 77f36992 030c26f8 030c57b4 030c27a4 USER32!_except_handler3+0xd5
030c26ac 77f36964 030c26f8 030c57b4 030c27a4 ntdll!ExecuteHandler2+0x26
030c2a74 77b8d36d 030c6128 77b8d36d 00000000 ntdll!ExecuteHandler+0x24
030c2a9c 77b8d59d 030c6128 030c2ac0 00000000 msvcrt!__global_unwind2+0x18
030c2ac0 77f36992 030c2ba4 030c6128 030c2bc0 msvcrt!_except_handler3+0x75
030c2ae4 77f36964 030c2ba4 030c6128 030c2bc0 ntdll!ExecuteHandler2+0x26
030c2b8c 77f36796 030c1000 030c2bc0 030c2ba4 ntdll!ExecuteHandler+0x24
030c2b8c 77b7aa54 030c1000 030c2bc0 030c2ba4 ntdll!KiUserExceptionDispatcher+0xe
030c3300 77b7b4dc 030c3324 7715b1b4 00000000 msvcrt!_woutput_l+0x18
```

Wow, we took an exception while trying to handle another exception! (It so happens this was easy to spot in the original stack trace, but in the general case, the next outer exception may require digging.)

Repeat the exercise with this next exception:

```
0:000> .exr 030c2ba4
ExceptionAddress: 77b7aa54 (msvcrt!_woutput_l+0x00000018)
   ExceptionCode: c00000fd (Stack overflow)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 00000001
   Parameter[1]: 030c2e88
0:000> .cxr 030c2bc0
eax=030c33b0 ebx=00000000 ecx=0000005c edx=00000000 esi=030c33c4 edi=030c33c4
eip=77b7aa54 esp=030c2e8c ebp=030c3300 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
msvcrt!_woutput_l+0x18:
001b:77b7aa54 53          push    ebx
```

Aha, so the SXS exception was triggered by a stack overflow. At this new context, you can use the “k” command to see how we got into this state.

It so happens that this particular bug was, as predicted, a stack overflow caused by unintended recursion when a call to an off-thread COM object forced the calling thread to wait for the reply, during which time a new request came in. The precise details of the problem aren’t important; the goal today was to show how you can suck the exception pointers out of the stack to see what Win32 exception was the source of the problem.

Raymond Chen

Follow