Signs that the symbols in your stack trace are wrong

devblogs.microsoft.com/oldnewthing/20091106-00

November 6, 2009

Raymond Chen

One of the things programmers send to each other when they are trying to collaborate on a debugging problem is stack traces. Usually something along the lines of "My program does X, then Y, then Z, and then it crashes. Here is a stack trace. Can you tell me what's wrong?"

It helps if you at least glance at the stack trace before you send it, because there are often signs that the stack trace you're about to send is completely useless because the symbols are wrong. Here's an example:

> We are testing our program and it gradually grinds to a halt. When we connect a debugger, we find that all of our threads, no matter what they are doing, eventually wind up hung in `kernel32!EnumResourceLanguagesA`. Can someone explain why that function is hanging, and why it seems all roads lead to it?
>
> ```
>    0  Id: 12a4.1468 Suspend: 1 Teb: 000006fb`fffdc000 Unfrozen
> kernel32!EnumResourceLanguagesA+0xbea00
> kernel32!EnumResourceLanguagesA+0x2b480
> bogosoft!CObjMarker::RequestBlockForFetch+0xf0
> ...
>    1  Id: 12a4.1370 Suspend: 1 Teb: 000006fb`fffda000 Unfrozen
> kernel32!EnumResourceLanguagesA+0xbea00
> kernel32!EnumResourceLanguagesA+0x2b480
> bsnetlib!CSubsystem::CancelMain+0x90
>    2  Id: 12a4.1230 Suspend: 1 Teb: 000006fb`fffd8000 Unfrozen
> NETAPI32!I_NetGetDCList+0x117e0
> kernel32!EnumResourceLanguagesA+0x393a0
> ntdll!LdrResFindResource+0x58b20
> ...
>    3  Id: 12a4.cc0 Suspend: 1 Teb: 000006fb`fffd6000 Unfrozen
> kernel32!EnumResourceLanguagesA+0xa80
> bsnetlib!BSFAsyncWait+0x190
> ...
>   4  Id: 12a4.1208 Suspend: 1 Teb: 000006fb`fffd4000 Unfrozen
> kernel32!EnumResourceLanguagesA+0xbea00
> kernel32!EnumResourceLanguagesA+0x2b480
> bogosoft!TObjList<DistObj>::Get+0xb0
>   5  Id: 12a4.1538 Suspend: 1 Teb: 000006fb`fffae000 Unfrozen
> kernel32!EnumResourceLanguagesA+0xbf3d0
> kernel32!EnumResourceLanguagesA+0x2c800
> bsnetlib!Tcp::ReadSync+0x340
> ...
>    6  Id: 12a4.16e0 Suspend: 1 Teb: 000006fb`fffac000 Unfrozen
> ntdll!LdrResFindResource+0x61808
> ntdll!LdrResFindResource+0x1822a0
> kernel32!EnumResourceLanguagesA+0x393a0
> ntdll!LdrResFindResource+0x58b20
> ...
> ```

This stack trace looks suspicious for a variety of reasons.

First of all, look at that offset `EnumResourceLanguagesA+0xbea00`. It's unlikely that the `EnumResourceLanguagesA` function (or any other function) is over 750KB in size, as this offset suggests.

Second, it's unlikely that the `EnumResourceLanguagesA` function (or any other function, aside from obvious cases like tree walking) is recursive. And it's certainly unlikely that a huge function will also be recursive.

Third, it seems unlikely that the `EnumResourceLanguagesA` function would call, `NETAPI32!I_NetGetDCList`. What does enumerating resource languages have to do with getting a DC list?

Fourth, look at those functions that are allegedly callers of `EnumResourceLanguagesA`: `bogosoft!CObjMarker::RequestBlockForFetch`, `bsnetlib!CSubsystem::CancelMain`, `bsnetlib!Tcp::ReadSync`. Why would any of these functions want to enumerate resource languages?

These symbols are obvious wrong. The huge offsets are present because the debugger has access only to exported functions, and it's merely showing you the name of the nearest symbol, even though it has nothing to do with the actual function. It's just using the nearest signpost it can come up with. It's like if somebody gave you directions to the movie theater like this: "Go to city hall downtown and then go north for 35 miles." This doesn't mean that the movie theater is in the downtown district or that the downtown district is 35 miles long. It's just that the person who's giving you directions can't come up with a better landmark than city hall.

This is just another case of the principle that you have to know what's right before you can see what's wrong. If you have no experience with good stack traces, you don't know how to recognize a bad one.

Oh, and even though the functions in question are in `kernel32`, you can still get symbols for that DLL with the help of the Microsoft Symbol Server.

Raymond Chen

Follow