Why do rescued stack traces often have instability at the start?

devblogs.microsoft.com/oldnewthing/20220527-00

May 27, 2022

Raymond Chen

When you try to recover a lost stack trace, and you make a guess as to where things should resume, sometimes you get a few stack frames of garbage before things finally settle down into a proper stack trace. Why does that happen?

Recall that the stack frame chain is a linked list of stack frames. If you draw the stack left-to-right and growing to the right, it might look like this:

Each stack frame consists of the return address (R) and a pointer to the previous stack frame. The current stack frame is marked with a star. The space between stack frames contains other stack things, like saved registers, stack-based function parameters, and local variables.

When a function is called, a new stack frame is added to the linked list, and when a function returns, its frame is popped off the linked list.

But even though the frame has been popped off the linked list, it still remains in memory until overwritten.

Sometimes, the memory for an expired frame is reused right away, but sometimes it lingers. For example, the next function might decide to use that memory for a variable which hasn't yet been written to. And maybe through the course of the function, it *never* gets written to: Maybe the variable is used in a branch that was not taken, or the variable is a large buffer, and only part of the buffer was needed.

If you dump the stack, you may see these ghosts of long-ago stack frames still haunting the stack memory. What happens if you ask the debugger to dump the stack starting at one of these ghost stack frames?

The debugger doesn't know it's a ghost. The debugger just follows the linked list. If the list is badly corrupted, the resulting stack trace is complete nonsense. But if the list is only slightly corrupted, you could see a few frames of garbage, and then a good stack trace emerges.

Why does a good stack trace emerge?

The thing about the ghost frames is that they are not total fabrications. They were at one time valid frames, and that means that their pointer did point to the previous function on the stack.

Here's a time-lapse of what happens to the stack frame chain as functions are called and return. At the start of our story, we have a call stack with three frames.

Now, suppose that the current function calls a new function. This pushes a new frame onto the stack:

That function now returns, popping it off the stack but leaving a ghost behind, which I will draw in gray.

And the we call another function:

Eventually that function returns:

And say the returned-to function also returns.

And then the first function in our diagram calls some new function.

Now suppose you didn't know that the stack frame chain began with the starred element F. What happens if you try each of the different possible starting points?

For example, if we started at the element labeled C, then we would get C, then follow the arrow to B (which is a dotted arrow but we don't know that), and then follow the arrow again to A, yielding a stack trace of C-B-A.

Here's what you get for all of the possible starting points.

Starting point	Result
A	A–…
B	B–A–…
C	C–B–A–…
D	D–C–B–A–…
E	E–C–B–A–…
F	F–B–A–…

Notice that each one gives you a stack trace for what was the stack at the point the frame was last alive. And once you get to a live frame (A or B), the stack backtrace is all good.

Some of the labels or arrows for ghost stack frames become corrupted because the memory was reused for some other purpose. Depending on how severe the corruption is, the stack trace could turn into complete nonsense, or it could still produce a trace but with a little bit of corruption. The corruption is possible up until the ghost stack trace reaches far back enough to hit a live stack frame, at which point everything from that point onward is still good.

In practice, most ghost frames get overwritten relatively quickly, so if you find a ghost frame, there's a good chance that it's only a few frames away from a live frame, and a reasonable chance that none of the ghost frames leading to the live frame have been corrupted too badly.

That's why, if you ask for a stack trace starting from the wrong place, and you are sufficiently lucky, you get a few frames of garbage followed by a good stack trace.

Raymond Chen

Follow