How can I check whether a parameter is a pointer to a stack variable?

devblogs.microsoft.com/oldnewthing/20160728-00

July 28, 2016

Raymond Chen

A customer traced a bug in their program back to the fact that somebody passed a stack buffer to a function that operates asynchronously. Something like this:

```
// This function sends the buffer to the destination
// asynchronously. When the send completes, the completion
// handler is invoked.
HRESULT SendDataAsync(void *buffer, size_t bufferSize,
                      ITransferCompletion* completion);

class ColorSender : public ITransferCompletion
{
...
};

// Code in italics is wrong
HRESULT ColorSender::SendColor()
{
  COLORREF color = GetColorToSend();
  return SendDataAsync(&color, sizeof(color), this);
}
```

This bug led to the destination sometimes receiving corrupted data because it was a race between the stack variable being reused for something else and the data transport code copying the buffer into the transport channel. In this particular case, the customer fixed the problem by making the `color` local variable a member of the `ColorSender` class, because the `ColorSender` object itself must remain valid for the lifetime of the transfer seeing as it is the completion handler. (We're using the principle that code is mostly correct.)

The customer wanted to add some debugging code to the `SendDataAsync` function so that it can assert that the buffer is not on the stack. The customer understands that this may be overly restrictive, and would not be appropriate for a function with a broad audience, but this function is used only within their application, so being extra-restrictive is probably okay.[1]

On Windows 8 and higher, the customer can use the `GetCurrentThreadStackLimits` function to get the bounds of the current thread's stack and use that to verify that the buffer pointer is not in that range. (Since this is just for debugging purposes, they can skip the test if running on older versions of Windows, and hope that their testing on Windows 8 will catch the problem.)

[1] If it turns out that the check is too restrictive, they could create two functions. The first is called `SendPossiblyStackDataAsync` that does no validation, and the second is `SendDataAsync` which does the initial validation that the buffer is not on the stack, and then calls `SendPossiblyStackDataAsync`. Most callers use `SendDataAsync`, but if you're in a special case where you are sending stack data and you know that you won't return from the function until the completion occurs, then you can use `SendPossiblyStackDataAsync`. The team could have a rule that equires all uses of `SendPossiblyStackDataAsync` to undergo a special review.

Update: See follow-up discussion.

Raymond Chen

Follow