Why does my thread get a broken string as its initial parameter?

devblogs.microsoft.com/oldnewthing/20240322-00

March 22, 2024

Raymond Chen

A customer passed a string to a newly-created `std::thread` but found that the `std::string` received by the thread was already invalid. How can that happen?

Here was how they passed the parameter to the thread:

```
void OnWidgetChanged(const char* reason)
{
    std::thread backgroundThread(ProcessWidgetChange,
        reason);
    backgroundThread.detach();
}

void ProcessWidgetChange(std::string reason)
{
    // the reason is corrupted!
}
```

The problem is that the raw `const char*` pointer is being converted to a `std::string` too late.

To avoid confusion, let me rename the two `reason` parameters:

```
void OnWidgetChanged(const char* rawReason)
{
    std::thread backgroundThread(ProcessWidgetChange,
        rawReason);
    backgroundThread.detach();
}

void ProcessWidgetChange(std::string stringReason)
{
    // the stringReason is corrupted!
}
```

The parameters to the `std::thread` constructor are captured by value, and the copies are then passed to the new thread, which executes the desired code by passing those copies to `std::invoke`. The specification calls out that the capturing happens in the calling thread as

part of the constructor.

Here's a flowchart of the operations that take place:

Original thread	
Capture parameters by value	
Start new thread	
⤳	New thread
	Convert parameters to match signature of target
	Call the target
	Destruct converted parameters

In the above example, it means that the raw `const char*` is captured and given to the thread. Only when the thread begins execution is the raw `const char*` converted to a `std::string`, and by that point, it's too late.

Original thread	
Copy the `rawReason` raw pointer	
Start new thread	
⤳	New thread
	Convert: Construct `stringReason` from `rawReason`
	Call the target with `stringReason`
	Destruct: Destruct `stringReason` and `rawReason`

The problem is clearer when the conversion is spelled out: The original code has already detached the thread and returned, at which point the caller of `OnWidgetChanged` is not under any obligation to keep the pointer valid any further. The conversion to a `std::string` therefore happens too late.

The author of the above code was under the incorrect belief that the operations occurred in a different order:

Original thread	
Convert: Construct `stringReason` from `rawReason`	
Capture converted parameters	
Start new thread	
⇝	New thread
	Call the target
	Destruct: Destruct `stringReason`

If you think about it, it's not possible to do things the second way. The `thread` constructor could try to infer the signature of the callable, but the callable might have multiple `operator()` overloads, so the tricks for deconstructing the function pointer don't work. And even if they did work, it's possible that the caller didn't pass the same number of parameters as the function expects, but expecting it to work because the missing parameters have default values. But there's no way to deduce the default values (because default values are not part of the function type signature), so the converter doesn't know what values to use for the missing parameters.

The solution is to convert the `rawReason` raw pointer to a `std::string` while the `rawReason` is still valid, pass that converted value to the `std::thread` constructor.

```
void OnWidgetChanged(const char* rawReason)
{
    std::thread backgroundThreadProcessWidgetChange,
        std::string(reason));
    backgroundThread.detach();
}

void ProcessWidgetChange(std::string stringReason)
{
    // use the stringReason
}
```

In the original customer version, the issue wasn't about an anticipated conversion from a raw pointer to a `std::string`, but rather an anticipated conversion of a raw COM pointer to a smart one:

```
void OnWidgetChanged(IWidget* widget)
{
    // Oops, captures raw pointer, not ComPtr.
    std::thread backgroundThread(ProcessWidgetChange,
        widget);
    backgroundThread.detach();
}

void ProcessWidgetChange(ComPtr<IWidget> widget)
{
    widget->Refresh();
}
```

The solution, as before, is to force the conversion early:

```
void OnWidgetChanged(IWidget* widget)
{
    std::thread backgroundThread(&ProcessWidgetChange,
        ComPtr<IWidget>(widget));
    backgroundThread.detach();
}
```

Bonus chatter: It would have been nice to use Class Template Argument Deduction (CTAD) to simplify this to

```
void OnWidgetChanged(IWidget* widget)
{
    std::thread backgroundThread(&ProcessWidgetChange,
        ComPtr(widget));
    backgroundThread.detach();
}
```

Unfortunately, as we saw earlier, WRL's `ComPtr` doesn't support CTAD.