Your exception handler can encounter an exception

devblogs.microsoft.com/oldnewthing/20050317-00

March 17, 2005

Raymond Chen

Consider the following code, written in C# just for kicks; the problem is generic to any environment that supports exception handling.

```
void ObliterateDocument()
{
 try {
  try {
   document.DestroyAll();
  } finally {
   document.Close();
   document.DestroyExtensions();
   document.DestroyPlugins();
  }
 } finally {
  document.Destroy();
 }
}
```

Some time later, you find yourself facing an assertion failure from `document.Destroy()` claiming that you are destroying the document while there are still active plugins. But there is your call to `document.DestroyPlugins()`, and it's in a `finally` block, and the whole point of a `finally` block is that there is no way you can escape without executing it.

So why didn't `document.DestroyPlugins()` execute?

Because your exception handler itself encountered an exception.

The exception handler is not active during its own `finally` clause. As a result, if an exception is thrown during `document.Close()`, the exception handler search begins at the block **outside** the `finally` block.

(That the exception handler is not active during its own `finally` clause should be obvious. It would mean that if an exception were to occur during the `finally` clause, the program would go into an infinite loop. And it also wouldn't be possible to rethrow a caught exception; your `throw` would end up caught by yourself!)

In this case, the exception was caught by some outer caller, causing the remainder of the first `finally` block to be abandoned. The other `finally` blocks do run since they contain the one that died.

(This bug also exists in the proposed alternative to error-checking code posted by an anonymous commenter.)

Raymond Chen

Follow