Why does my __FILE__ macro produce an invalid address, which mysteriously becomes valid a few moments later?

devblogs.microsoft.com/oldnewthing/20170417-00

April 17, 2017

Raymond Chen

A customer couldn't understand what the debugger was showing them. They had a function that used the `__FILE__` macro, but the address of the resulting string was reported by the debugger as invalid:

```
// source code:
char *fileName = __FILE__;

// In the debugger
0: kd> ?? fileName
char * 0x00796d60
 "--- memory read error at address 0x00796d60 ---"
0: kd> db 0x00796d60
00796d60  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796d70  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796d80  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796d90  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796da0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796db0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796dc0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00796dd0  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

The `__FILE__` macro expanded to an invalid address. How can this be?

But wait, it gets even stranger: After executing a few lines of code, the pointer suddenly becomes good!

```
0: kd> ?? fileName
char * 0x00796d60
"C:\contoso\file.cpp"
```

What's going on here?

The thing to note is the prompt. The customer is using kd, the kernel debugger, rather than a user-mode debugger.[1] The kernel debugger shows what is in memory right now. If something is paged out, the kernel debugger won't try to page it in, because that would require the kernel to run, and that would kind of interfere with kernel debugging. (Besides, the kernel debugger doesn't know whether it's safe to page in memory right now. The current IRQL may be one that disallows page faults.)

Executing a few lines of code did something that caused the memory be paged in. The most likely thing that happened is that the file name was printed to a log file, and that means reading the file name, which means paging it in.

But that's not the only thing that could've caused the memory to be paged in. Another possibility is that the program accessed a variable that happens to reside on the same page. The unit of paging is the (um) page, so any variables that share the same page will be paged in and out together.

But the root cause for the confusion is that the customer is using the kernel debugger to debug user-mode code. It is possible to debug user mode with the kernel debugger, but it's quite cumbersome because you are debugging at a very, very low level, and lots of things you take for granted are no longer available. If you're going to be debugging user-mode code, you probably want to use a user-mode debugger.

[1] The customer disputed this assertion that they were using the kernel debugger. "We are using WinDbg, not kd." Explain.

Raymond Chen

Follow