What did the Ignore button do in Windows 3.1 when an application encountered a general protection fault?

devblogs.microsoft.com/oldnewthing/20150717-00

July 17, 2015

Raymond Chen

In Windows 3.0, when an application encountered a general protection fault, you got an error message that looked like this:

Application error
CONTOSO caused a General Protection Fault in module CONTOSO.EXE at 0002:2403
Close

In Windows 3.1, under the right conditions, you would get a second option:

CONTOSO
An error has occurred in your application. If you choose Ignore, you should save your work in a new file. If you choose Close, your application will terminate.
Close Ignore

Okay, we know what Close does. But what does Ignore do? And under what conditions will it appear?

Roughly speaking, the Ignore option becomes available if

- The fault is a general protection fault,
- The faulting instruction is not in the kernel or the window manager,

- The faulting instruction is one of the following, possibly with one or more prefix bytes:
  - Memory operations: `op r, m`; `op m, r`; or `op m`.
  - String memory operations: `movs`, `stos`, etc.
  - Selector load: `lds`, `les`, `pop ds`, `pop es`.

If the conditions are met, then the Ignore option became available. If you chose to Ignore, then the kernel did the following:

- If the faulting instruction is a selector load instruction, the destination selector register is set to zero.
- If the faulting instruction is a pop instruction, the stack pointer is incremented by two.
- The instruction pointer is advanced over the faulting instruction.
- Execution is resumed.

In other words, the kernel did the assembly language equivalent of `ON ERROR RESUME NEXT`.

Now, your reaction to this might be, "How could this possibly work? You are just randomly ignoring instructions!" But the strange thing is, *this idea was so crazy it actually worked*, or at least worked a lot of the time. You might have to hit Ignore a dozen times, but there's a good chance that eventually the bad values in the registers will get overwritten by good values (and it probably won't take long because the 8086 has so few registers), and the program will continue seemingly-normally.

Totally crazy.

Exercise: Why didn't the code have to know how to ignore jump instructions and conditional jump instructions?

Bonus trivia: The developer who implemented this crazy feature was Don Corbitt, the same developer who wrote Dr. Watson.

Raymond Chen

Follow