There's more to switching stacks than just loading a new stack pointer

devblogs.microsoft.com/oldnewthing/20080215-00

February 15, 2008

Raymond Chen

Sometimes people think they can switch stacks by just loading a new value into the ESP register. This may seem to work but in fact it doesn't, because there is more to switching stacks than just loading a new value into `ESP`. On the x86, the exception chain is threaded through the stack, and the exception dispatch code verifies that the exception chain is "sane" before dispatching an exception. If you summarily yank `ESP` into a location outside the stack the operating system assigned to the thread, then the exception chain will appear to be corrupted, and once the exception dispatch code notices this, it will declare your program to be unrecoverably corrupted. It can't even raise an exception to indicate that this has happened, even if it wanted to, because it doesn't even know where the exception handlers are! There are other parts of the system that rely on the stack pointer remaining inside the correct stack. For example, the code that expands the stack on demand needs to know where the stack is and how big it can get. (And the ia64 architecture has *two* stack pointers.) If a part of the system needs to do work with those values and it notices that the real stack pointer is "in la-la land", it will start taking drastic measures (typically by terminating the program).

If you want to switch stacks, use a fiber. Fibers provide a way to capture the state of a computation, which includes the instruction pointer and the stack.

Raymond Chen

Follow