If the cursor clip rectangle is a global resource, how come I can't change it with ClipCursor?

devblogs.microsoft.com/oldnewthing/20140102-00

January 2, 2014

Raymond Chen

A customer reported that the `ClipCursor` function was not working. “The cursor clip rectangle is a shared global resource, so I should be able to change it any time I like. My app installs a global mouse hook and sets the clip cursor inside the hook function, but the change doesn’t have any effect. Why can’t I change the clip cursor inside a mouse hook?” Sure, you can change the clip cursor inside a mouse hook. But remember, a shared global resource cuts both ways. Since anybody can change it, your app can change it any time it likes. But since anybody can change it, *another app can also change it any time they like*. In this case, what’s happening is that your hook comes in and sets the clip rectangle. And the application gets the mouse message and passes it to `DefWindowProc` , and the default behavior for focus changes is to clear the clip rectangle so that any clip rectangle set by the previous window doesn’t spill over into the new focus window. The convention for the clip rectangle is that the focus window (perhaps after some negotiation with one of its ancestors) controls the clip rectangle. This convention is not enforced for a few reasons. First of all, you can’t programmatically determine whether code is executing on behalf of any particular window. Even if you say “can be called only during the handling of a message”, that doesn’t prove that the code is associated with the window. The message handler might call into some other component, and that other component might decide to clip the cursor just because. Another reason the rule isn’t enforced is that the clip cursor was invented back in the day when programmers were trusted to do the right thing. The theory was that preventing people from doing sneaky things would also prevent them from doing clever things.

(Nowadays, the API design philosophy prefers to prevent people from doing sneaky things, even though it also prevents them from doing clever things, because the bad guys are sneakier than the good guys are clever.)

Raymond Chen

Follow