With what operations is LockWindowUpdate not meant to be used?

devblogs.microsoft.com/oldnewthing/20070222-01

February 22, 2007

Raymond Chen

Okay, now that we know what operations `LockWindowUpdate` is meant to be used with, we can look at various ways people misuse the function for things unrelated to dragging. People see the "the window you lock won't be able to redraw itself" behavior of `LockWindowUpdate` and use it as a sort of lazy version of the `WM_SETREDRAW` message. Though sending the `WM_SETREDRAW` message really isn't that much harder than calling `LockWindowUpdate`. It's twenty more characters of typing, half that if you use the `SetWindowRedraw` macro in `<windowsx.h>`.

Instead of	`LockWindowUpdate(hwnd)`
Use	`SendMessage(hwnd, WM_SETREDRAW, FALSE, 0)` or `SetWindowRedraw(hwnd, FALSE)`

Instead of	`LockWindowUpdate(NULL)`
Use	`SendMessage(hwnd, WM_SETREDRAW, TRUE, 0)` or `SetWindowRedraw(hwnd, TRUE)`

As we noted earlier, only one window in the system can be locked for update at a time. If your intention for calling `LockWindowUpdate` is merely to prevent a window from redrawing, say, because you're updating it and don't want the window continuously refreshing until your update is complete, then just disable redraw on that window. If you use `LockWindowUpdate`, you create a whole slew of subtle problems. First off, if some other program is misusing `LockWindowUpdate` in this same way, then one of you will lose. Whoever tries `LockWindowUpdate` first will get it, and the second program will fail. Now what do you do? Your window isn't locked any more. Second, if you have locked your window for update and the user switches to another program and tries to drag an item (or even just tries to move the window!), that attempt to `LockWindowUpdate` will fail, and the user is now in the position where drag/drop has stopped working for some mysterious reason. And then,

ten seconds later, it starts working again. “Stupid buggy Windows,” the user mutters. Conversely, if you decide to call `LockWindowUpdate` when a drag/drop or window-move operation is in progress, then your call will fail. This is just a specific example of the more general programming mistake of using global state to manage a local condition. When you want to disable redrawing in one of your windows, you don’t want this to affect other windows in the system; it’s a local condition. But you’re using a global state (the window locked for update) to keep track of it. I can already anticipate people saying, “Well, the window manager shouldn’t let somebody lock a window for update if they’re not doing a drag/drop operation.” But how does the window manager know? It knows what is happening, but it doesn’t know why. Is that program calling `LockWindowUpdate` because it’s too lazy to use the `WM_SETREDRAW` message? Or is it doing it in response to some user input that resulted in a drag/drop operation? Note that you can’t just say, “Well, the mouse button has to be down,” because the user might be performing a keyboard-based operation (such as resizing a window with the arrow keys) that has the moral equivalent of a drag/drop. Morality is hard enough to resolve as it is; expecting computers to be able to infer it is asking a bit much.

Next time, a final remark on `LockWindowUpdate`.

Raymond Chen

Follow