What is the process by which the cursor gets set?

devblogs.microsoft.com/oldnewthing/20061121-15

November 21, 2006

Raymond Chen

Commenter LittleHelper asked, “Why is the cursor associated with the class and not the window?” This question makes the implicit assumption that the cursor is associated with the class. While there is a cursor associated with each window class, it is the window that decides what cursor to use.

The cursor-setting process is described in the documentation of the `WM_SETCURSOR` message:

> The `DefWindowProc` function passes the `WM_SETCURSOR` message to a parent window before processing. If the parent window returns `TRUE`, further processing is halted. Passing the message to a window’s parent window gives the parent window control over the cursor’s setting in a child window. The `DefWindowProc` function also uses this message to set the cursor to an arrow if it is not in the client area, or to the registered class cursor if it is in the client area.

That paragraph pretty much captures the entire cursor-setting process. all I’m writing from here on out is just restating those few sentences.

The `WM_SETCURSOR` goes to the child window beneath the cursor. (Obviously it goes to the child window and not the parent, because the documentation says that `DefWindowProc` forward the message to its parent. if the message went to the parent originally, then there would be nobody to forward the message to!) At this point, your window procedure can trap the `WM_SETCURSOR` message, set the cursor, and return `TRUE`. Thus, the window gets the first priority on deciding what the cursor is.

If you don’t handle the `WM_SETCURSOR` message, then `DefWindowProc` forwards the message to the parent, who in turn gets to decide whether to handle the message or forward to its parent in turn. One possibility is that one of the ancestor windows will handle the message, set the cursor, and return `TRUE`. In that case, the `TRUE` return value tells `DefWindowProc` that the cursor has been set and no more work needs to be done.

The other, more likely, possibility is that none of the ancestor windows cared to set the cursor. At each return to `DefWindowProc`, the cursor will be set to the class cursor for the window that contains the cursor.

Here it is in pictures. Suppose we have three windows, A, B, and C, where A is the top-level window, B a child, and C a grandchild, and none of them do anything special in `WM_SETCURSOR`. Suppose further that the mouse is over window C:

```
SendMessage(hwndC, WM_SETCURSOR, ...)
 C's window procedure does nothing special
 DefWindowProc(hwndC, WM_SETCURSOR, ...)
  DefWindowProc forwards to parent:
   SendMessage(hwndB, WM_SETCURSOR, ...)
   B's window procedure does nothing special
   DefWindowProc(hwndB, WM_SETCURSOR, ...)
    DefWindowProc forwards to parent:
     SendMessage(hwndA, WM_SETCURSOR, ...)
     A's window procedure does nothing special
      DefWindowProc(hwndA) cannot forward to parent (no parent)
      DefWindowProc(hwndA) sets the cursor to C's class cursor
      DefWindowProc(hwndA) returns FALSE
     A's window procedure returns FALSE
    SendMessage(hwndA, WM_SETCURSOR, ...) returns FALSE
    DefWindowProc(hwndB) sets the cursor to C's class cursor
    DefWindowProc(hwndB) returns FALSE
   B's window procedure returns FALSE
  SendMessage(hwndB, WM_SETCURSOR, ...) returns FALSE
  DefWindowProc(hwndC) sets the cursor to C's class cursor
  DefWindowProc(hwndC) returns FALSE
 C's window procedure returns FALSE
SendMessage(hwndC, WM_SETCURSOR, ...) returns FALSE
```

Observe that the `WM_SETCURSOR` started at the bottom (window C), bubbled up to the top (window A), and then worked its way back down to window C. On the way up, it asks each window if it wants to set the cursor, and if it makes it all the way to the top with nobody expressing an opinion, then on the way down, each window sets the cursor to C's class cursor.

Now, of course, any of the windows along the way could have decided, "I'm setting the cursor!" and returned `TRUE`, in which case the message processing would have halted immediately.

So you see, the window really does decide what the cursor is. Yes, there is a cursor associated with the class, but it is used only if the window decides to use it. If you want to associate a cursor with the window, you can do it by handling the `WM_SETCURSOR` message explicitly instead of letting `DefWindowProc` default to the class cursor.

LittleHelper's second question: "Many programs call `SetCursor` on every `WM_MOUSEMOVE`. Is this not recommended?"

Although there is no rule forbidding you from using `WM_MOUSEMOVE` to set your cursor, it's going to lead to some problems. First, and much less serious, you won't be able to participate in the `WM_SETCURSOR` negotiations since you aren't doing your cursor setting there. But the real problem is that you're going to get cursor flicker. `WM_SETCURSOR` will get sent to your window to determine the cursor. Since you didn't do anything, it will probably turn into your class cursor. And then you get your `WM_MOUSEMOVE` and set the cursor again. Result: Each time the user moves the mouse, the cursor changes to the class cursor and then to the final cursor.

Let's watch this happen. Start with the scratch program and make these changes:

```
void
OnMouseMove(HWND hwnd, int x, int y, UINT keyFlags)
{
 Sleep(10); // just to make the flicker more noticeable
 SetCursor(LoadCursor(NULL, IDC_CROSS));
}
 // Add to WndProc
 HANDLE_MSG(hwnd, WM_MOUSEMOVE, OnMouseMove);
```

Run the program and move the mouse over the client area. Notice that it flickers between an arrow (the class cursor, set during `WM_SETCURSOR`) and the crosshairs (set during `WM_MOUSEMOVE`).

Raymond Chen

Follow