Thread affinity of user interface objects, part 1: Window handles

devblogs.microsoft.com/oldnewthing/20051010-09

October 10, 2005

Raymond Chen

Different objects have different thread affinity rules, but the underlying principles come from 16-bit Windows.

The most important user interface element is of course the window. Window objects have thread affinity. The thread that creates a window is the one with which the window has an inseparable relationship. Informally, one says that the thread "owns" the window. Messages are dispatched to a window procedure only on the thread that owns it, and generally speaking, modifications to a window should be made only from the thread that owns it. Although the window manager permits any thread to access such things as window properties, styles, and other attributes such as the window procedure, and such accesses are thread safe from the window manager's point of view, load-modify-write sequences should typically be restricted to the owner thread. Otherwise you run into race conditions such as the following:

```
wpOld = (WNDPROC)GetWindowLongPtr(hwnd, GWLP_WNDPROC);
SetWindowLongPtr(hwnd, GWLP_WNDPROC, (LONG_PTR)newWndProc);


LRESULT CALLBACK newWndProc(…)
{
 … CallWindowProc(wpOld, …); …
}
```

If modifications to the window procedure are made carelessly from any thread, then between the first two lines, a second thread may change the window procedure of the window, resulting in `newWndProc` passing the wrong "previous" window procedure to `CallWindowProc`.

Why, then, does Windows even allow a non-owner thread from changing the window procedure in the first place? Because, as we all know, 16-bit Windows was a co-operatively multi-tasked system, which means that one thread could do anything it wanted secure in the knowledge that no other thread would interrupt it until it explicitly relinquished control of

the CPU. Therefore, the above code sequence was safe in 16-bit Windows. And for compatibility reasons, the code continues to be legal, even though it isn't safe any more. (Note, however, that in an attempt to limit the scope of the damage, the window manager allows only threads in the process that owns the window to change the window procedure. This is a reasonable limitation since separate address spaces mean that function addresses in other processes are meaningless in the process that owns the window anyway.)

Next time, a look at device contexts.

Raymond Chen

Follow