Non-psychic debugging: Why you're leaking timers

devblogs.microsoft.com/oldnewthing/20101001-00

October 1, 2010

Raymond Chen

I was not involved in this debugging puzzle, but I was informed of its conclusions, and I think it illustrates both the process of debugging as well as uncovering a common type of defect. I've written it up in the style of a post-mortem.

> A user reported that if they press and hold the F2 key for about a minute, our program eventually stops working. According to Task Manager, our User object count has reached the 10,000 object limit, and closer inspection revealed that we had created over 9000 timer objects.
>
> We ran the debugger and set breakpoints on `SetTimer` and `KillTimer` to print to the debugger each timer ID as it was created and destroyed. Visual inspection of the output revealed that all but one of the IDs being created was matched with an appropriate destruction. We re-ran the scenario with a conditional breakpoint on `SetTimer` set to fire when that bad ID was set. It didn't take long for that breakpoint to fire, and we discovered that we were setting the timer against a `NULL` window handle.
>
> A different developer on the team arrived at the same conclusion by a different route. Instead of watching timers being created and destroyed, the developer dumped each timer message before it was dispatched and observed that most of the entries were associated with `NULL` window handles.
>
> Two independent analyses came to the same conclusion: We were creating a bunch of thread timers and not destroying them.
>
> A closer inspection of the code revealed that thread timers were not intended in the first place. Each time the user presses F2, the code calls `SetTimer` and passes a window handle it believes to be non-`NULL`. The timer is destroyed in the window procedure's `WM_TIMER` handler, but since the timer was registered against the wrong window handle, the `WM_TIMER` is never received by the intended target's window procedure, and the timer is never destroyed.
>
> The window handle is `NULL` due to a defect in the code which handles the F2 keypress: The handle that the code wanted to use for the timer had not yet been set. (It was set by a later step of F2 processing.) The timer was being set by a helper function which is called both before and after the code that sets the handle, but it obviously was written on the assumption that it would only be called after.
>
> To reduce the likelihood of this type of defect being introduced in the future, we're going to introduce a wrapper function around `SetTimer` which asserts that the window handle is non-`NULL` before calling `SetTimer`. (In the rare case that we actually want a thread timer, we'll have a second wrapper function called `SetThreadTimer`.)

I haven't seen the wrapper function, but I suspect it goes something like this:

```
inline UINT_PTR SetWindowTimer(
    __in HWND hWnd, // NB - not optional
    __in UINT_PTR nIDEvent,
    __in UINT uElapse,
    __in_opt TIMERPROC lpTimerFunc)
{
    assert(hWnd != NULL);
    return SetTimer(hWnd, nIDEvent, uElapse, lpTimerFunc);
}
inline UINT_PTR SetThreadTimer(
    __in UINT uElapse,
    __in_opt TIMERPROC lpTimerFunc)
{
    return SetTimer(NULL, 0, uElapse, lpTimerFunc);
}
__declspec(deprecated)
WINUSERAPI
UINT_PTR
WINAPI
SetTimer(
    __in_opt HWND hWnd,
    __in UINT_PTR nIDEvent,
    __in UINT uElapse,
    __in_opt TIMERPROC lpTimerFunc);
```

There are few interesting things here.

First, observe that the annotation for the first parameter to `SetWindowTimer` is `__in` rather than `__in_opt`. This indicates that the parameter cannot be `NULL`. Code analysis tools can use this information to attempt to identify potential defects.

Second, observe that the `SetThreadTimer` wrapper function omits the first two parameters. For thread timers, the `hWnd` passed to `SetTimer` is always `NULL` and the `nIDEvent` is ignored.

Third, after the two wrapper functions, we redeclare the `SetTimer`, but mark it as `deprecated` so the compiler will complain if somebody tries to call the original function instead of one of the two wrappers. (The `__declspec(deprecated)` extended attribute is a nonstandard Microsoft extension.)

Exercise: Why did I use `__declspec(deprecated)` instead of `#pragma deprecated(SetTimer)`?

Raymond Chen

Follow