What can influence how much stack is consumed when sending a message?

devblogs.microsoft.com/oldnewthing/20180621-00

June 21, 2018

Raymond Chen

Last time, we looked at a simple program that recursively sent itself a message, and we wondered, "Which will run out first? The user-mode stack or the kernel-mode stack?" Some of the calculations are under your control, but others can change at runtime.

If the destination window belongs to the same thread as the sender, and there are no applicable window hooks, then the user-mode portion of `SendMessage` says, "Well, I can handle this all by myself. I don't need to ask for any help from kernel mode at all!" In that case, the entire message processing happens in user mode, and no kernel mode stack is consumed at all.

On the other hand, if a window hook exists, then the user-mode portion of `SendMessage` needs to call into kernel mode to dispatch the hooks, and in that case, it means that there will be stack consumed in kernel mode.

The effect of window hooks means that the same program may crash in different ways depending on the configuration of the system it is running on. A program that installs a global message hook will change the failure mode.

If you ran the program from last time and encountered a stack overflow in user mode, then you can make these changes to install your own hook and watch the failure mode change:

```
HHOOK hhk;

LRESULT CALLBACK MessageHookProc(int nCode,
     WPARAM wParam, LPARAM lParam)
{
  return CallNextHookEx(hhk, nCode, wParam, lParam);
}

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
    ...
        hhk = SetWindowsHookEx(WH_CALLWNDPROC, MessageHookProc,
            hinst, GetCurrentThreadId());

        while (GetMessage(&msg, NULL, 0, 0)) {
            ...
        }

        UnhookWindowsHookEx(hhk);
        ...
}
```

Raymond Chen

Follow