When you call a function, your code doesn't resume execution until that function returns

devblogs.microsoft.com/oldnewthing/20101025-00

October 25, 2010

Raymond Chen

Consider this code fragment:

```
void foo()
{
  while (true) {
   bar();
   baz();
  }
}
```

When `foo` calls `bar()`, and `bar` has not yet returned, does `foo` continue executing? Does `baz` get called before `bar` returns?

No, it does not.

The basic structure of the C/C++ language imposes sequential execution. Control does not return to the `foo` function until `bar` returns control, either by reaching the end of the function or by an explicit `return`.

Commenter Norman Diamond asks a bunch of questions, but they're all mooted by the first:

> I can't find any of the answers in MSDN, and even an answer to one doesn't make answers to others obvious.
>
> Unless failures occur, the DialogBox function doesn't return until the new dialog's DialogProc calls EndDialog. It starts its own message loop. Dkring this time the hwndParent (i.e. owner not parent) window is disabled. However, disabling doesn't prevent delivery of some kinds of messages to the parent window's WindowProc or DialogProc, and doesn't prevent delivery of any messages to the application's main message loop, right? So aren't there two or more message loops running in parallel?

As long as the function `DialogBox` has not yet returned, control does not return to the application's main message loop, since it is the one which called `DialogBox` (most likely indirectly).

MSDN doesn't explain this because it is a fundamental property of the C and C++ languages and is not peculiar to Win32.

Disabling a window does not prevent it from receiving messages in general; it only disables mouse and keyboard input. This is called out in the opening sentence of the `EnableWindow` function documentation:

> The **EnableWindow** function enables or disables mouse and keyboard input to the specified window or control.

Messages unrelated to mouse and keyboard input are delivered normally. And they aren't dispatched by the application's main message loop because, as we saw above, the main message loop isn't executing!

I would recommend reviewing a book that covers the basics of Win32 GUI programming, since there appear to be some fundamental misunderstandings. Since I try to target an advanced audience, I generally assume that everybody understands the basics and is ready to move on to the intermediate and advanced topics. If you have trouble with the basics, you should work on that part first.

Raymond Chen

Follow