WinMain is just the conventional name for the Win32 process entry point

devblogs.microsoft.com/oldnewthing/20110525-00

May 25, 2011

Raymond Chen

WinMain is the conventional name for the user-provided entry point in a Win32 program. Just like in 16-bit Windows, where the complicated entry point requirements were converted by language-provided startup code into a call to the the user's WinMain function, the language startup code for 32-bit programs also does the work of converting the raw entry point into something that calls WinMain (or wWinMain or main or _wmain).

The raw entry point for 32-bit Windows applications has a much simpler interface than the crazy 16-bit entry point:

```
DWORD CALLBACK RawEntryPoint(void);
```

The operating system calls the function with no parameters, and the return value (if the function ever returns) is passed to the the ExitThread function. In other words, the operating system calls your entry point like this:

```
...
  ExitThread(RawEntryPoint());
  /*NOTREACHED*/
```

Where do the parameters to WinMain come from, if they aren't passed to the raw entry point?

The language startup code gets them by asking the operating system. The instance handle for the executable comes from GetModuleHandle(NULL) , the command line comes from GetCommandLine , and the nCmdShow comes from GetStartupInfo . (As we saw before, the hPrevInstance is always NULL.)

If you want to be hard-core, you can program to the raw entry point. Mind you, other parts of your program may rely upon the work that the language startup code did before calling your WinMain . For example, the C++ language startup code will run global constructors before calling into WinMain , and both C and C++ will initialze the so-called *security cookie* used as part of stack buffer overrun detection. Bypass the language startup code at your peril.

Bonus chatter: Notice that if you choose to return from your entry point function, the operating system passes the return value to `ExitThread` and not `ExitProcess`. For this reason, you typically don't want to return from your raw entry point but instead want to call `ExitProcess` directly. Otherwise, if there are background threads hanging around, they will prevent your process from exiting.

Raymond Chen

Follow