The history of calling conventions, part 3

devblogs.microsoft.com/oldnewthing/20040108-00

January 8, 2004

Raymond Chen

Okay, here we go: The 32-bit x86 calling conventions. (By the way, in case people didn't get it: I'm only talking in the context of calling conventions you're likely to encounter when doing Windows programming or which are used by Microsoft compilers. I do not intend to cover calling conventions for other operating systems or that are specific to a particular language or compiler vendor.) Remember: If a calling convention is used for a C++ member function, then there is a hidden "this" parameter that is the implicit first parameter to the function.

All

The 32-bit x86 calling conventions all preserve the EDI, ESI, EBP, and EBX registers, using the EDX:EAX pair for return values.

C (__cdecl)

The same constraints apply to the 32-bit world as in the 16-bit world. The parameters are pushed from right to left (so that the first parameter is nearest to top-of-stack), and the caller cleans the parameters. Function names are decorated by a leading underscore.

__stdcall

This is the calling convention used for Win32, with exceptions for variadic functions (which necessarily use __cdecl) and a very few functions that use __fastcall. Parameters are pushed from right to left [*corrected 10:18am*] and the callee cleans the stack. Function names are decorated by a leading underscore and a trailing @-sign followed by the number of bytes of parameters taken by the function.

__fastcall

The first two parameters are passed in ECX and EDX, with the remainder passed on the stack as in __stdcall. Again, the callee cleans the stack. Function names are decorated by a leading @-sign and a trailing @-sign followed by the number of bytes of parameters taken by the function (including the register parameters).

thiscall

The first parameter (which is the "this" parameter) is passed in ECX, with the remainder passed on the stack as in __stdcall. Once again, the callee cleans the stack. Function names are decorated by the C++ compiler in an extraordinarily complicated mechanism that encodes the types of each of the parameters, among other things. This is necessary because C++ permits function overloading, so a complex decoration scheme must be used so that the various overloads have different decorated names.

There are some nice diagrams on MSDN illustrating some of these calling conventions.

Remember that a calling convention is a contract between the caller and the callee. For those of you crazy enough to write in assembly language, this means that your callback functions need to preserve the registers mandated by the calling convention because the caller (the operating system) is relying on it. If you corrupt, say, the EBX register across a call, don’t be surprised when things fall apart on you. More on this in a future entry.

Raymond Chen

Follow