

Re: Why no C++ in Windows itself?

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2005-02/0651.html>

From: Severian (severian_at_chlamydia-is-not-a-flower.com)

Date: 02/10/05

Date: Thu, 10 Feb 2005 17:11:18 -0500

On Thu, 10 Feb 2005 18:07:35 +0000, Tom Widmer
<tom_usenet@hotmail.com> wrote:

>Severian wrote:

>> On Wed, 9 Feb 2005 14:07:44 +0330, "Herr Lucifer"

>> <"\n"HerrLucifer\n@microsoft.com> wrote:

>>

>>

>>>It is a long time now that I am using "Dependency Walker for Win32" and
>>>navigating through different Dlls in "System32" folder to find my target API
>>>functions. The thing I have noticed is that, 'almost' all system .dll and
>>>.exe

>>>files are written in C and not C++. If C++ is a powerful OO language and
>>>it's much greater than C for large application development, then why haven't
>>>Microsoft used it in its operating system? Is there a compatibility issue
>>>here that I am not aware of? Or something else?

>>>---

>>

>>

>> **IMHO:**

>>

>> For time-critical issues,

>

>Do you mean that C is faster executing? Or that its performance is more
>consistent (for a realtime system)? I don't think either statement is
>particularly true, though some C and C++ features should be avoided in
>code that should run in a bounded amount of time (e.g. no dynamic
>allocation, no disk IO, etc.).

Sorry, I wasn't trying to start a language war. I program in both C and C++, and have seen plenty of good and bad examples of both. I was comparing well-written C with well-written C++ (not simply C-like code compiled by a C++ compiler).

But generally, when using the constructs that make C++ a great language, it is very much simpler to estimate the overhead of C statements vs. C++ statements. (I'm not talking about C code compiled by a C++ compiler!)

microsoft.public.vc.language: Re: Why no C++ in Windows itself?

> *smaller code size, better memory*
>> *consumption,*
>
>*C does generally produce slightly smaller code, due to the overhead EH*
>*and RTTI in C++. However, both of those may be avoided if desired.*

In my experience, you have to know a lot more about C++ (and how it is implemented on a particular platform) to make it as efficient (memory and speed-wise) as well-written C.

> *improved control over efficiency,*
>
>*How so?*

Because, as examples, overloaded operators and virtual functions introduce non-obvious overhead that is more explicit in C.

> *improved code*
>> *generation,*
>
>*The code generation is generally worse in C, since I believe that type*
>*based alias analysis isn't as good in C. Is there any other respect in*
>*which you think C can generate better code?*

You're probably right, in the general case of compiling simple C++ code; and in some cases (i.e., templates) even when generating complex code. But once you start using the things that make C++ such a wonderful language, the generated code tends, in my experience, to become bloated and wasteful — and much more difficult to debug.

>*simpler interface to assembler,*
>
>*What do you mean?*

I am speaking of calling assembly from C++ or vice versa, not inline assembly, which is quite platform-specific.

One example: since assembly language has no concept of C++ objects, you have to "deobjectify" any data you want to access from assembler (or C for that matter!), or know a lot about how a particular C++ compiler represents objects in memory and write extra assembly language to work with them.

> *and easier debugging*
>> *(especially the kernel and low-level code)*
>
>*But macros are far harder to debug than inline functions!*

Most modern C compilers include inline functions, though by looking at code generated by MS and GNU C++, macros are often much more efficient, and thus often more appropriate for OS code.

Re: Why no C++ in Windows itself?

The difficulty I was referring to was the semi-invisible things that C++ does, that require you to know the implementation of classes you use. Overloaded operators (-> and []) come immediately to mind) make debugging a loathsome task unless you have access to the source code for the original class. Even in those cases, it's often difficult to figure out what exactly is going on behind the scenes.

I've seen well-written C++ code that is a joy to work with, but a lot more of it is obfuscated (unintentionally?) to the point of hair-pulling. Most of Microsoft's C++ falls into the latter category!

When writing C++, I love STL and wish that MS had not tried to make their own horrid messes first.

> *the guts of Windows are*
>> *mostly written in C.*
>
>*That I know nothing about, but it certainly sounds true. But I believe*
>*much of the Linux kernel is now written in C++ (as was BeOS). C++ is*
>*primarily a systems programming language – it was specifically designed*
>*for writing OSes (among other things), hence the principle of "You*
>*don't pay for what you don't use."*

I agree that well-written C++ code (in a `_specific_` rather than generic way) is very appropriate for operating systems, as long as provisions are made for using the OS API from other languages.

>*I'd definitely prefer to write an OS in C++ rather than C, and the*
>*resulting OS would probably be faster and better for it (though possibly*
>*not smaller).*

I think it boils down to whether the code is well-written or not; the simpler basis of C makes it easier, in my opinion, to write efficient and understandable code. That being said, I have worked with and (hope) I have written efficient and easily maintainable C++ code as well.

--
Sev