

## Re: LuxAsm questions

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-06/0746.html>

---

**From:** f0dder (*f0dder\_spicedham\_at\_flork.dk*)

**Date:** 06/24/04

Date: Thu, 24 Jun 2004 05:55:29 +0200

Beth, when timing stuff, you should discard all but the fastest results, as this will show the best possible you can achieve by the code, and will not be affected by the (sometimes wild) fluctuations you can get under a protected mode OS. (Yes, the timing method can be discussed to death and is a religions matter, but consider running averages – if one run, for some reason, has a disk access, it can skew the results massively.)

Also, consider `_what_` you're timing. A call like `time` is bound to have `_very_` little code, and thus call overhead can chew up a measurable amount of time. But how often do you call the kernel for such simple tasks? You should consider measuring the overhead from syscalls like disk I/O, process creation, IPC, and other things that are complex, and get called a lot more than `time`.

My plea is that you optimize where it matters. A "call write" or "call time" will be a lot more readable than loading up a constant in `eax` and `int 80h`, unless you comment stuff vigorously. Not to mention that it will be a heck lot easier to port (oops, I said a no-no word ;) your sources to other `*u*x'es`. You don't necessarily have to link with `libc`, you could write your own tiny wrapper library for LuxAsm. Doesn't mean you have to put parameters on the stack, you can keep them in registers – just "call write" rather than "int 80h".

Also, a size notice... I assume syscall number is stored in `eax` – "mov `eax, BLAH`" followed by "int 80h" is 7 bytes, `eip`-relative call is 5 bytes. Solve the equation " $5x + 7 < 7x$ ", and you get something like (rounded up), " $x > 4$ " (sorry if my math sucks, it's late.) So, if you call a given syscall more than 4 times, doing a "call write" that sets up `eax` and does `int80` will be shorter. Of course things change if only `AL` has to be loaded ;)

Dunno how bad it is to link in `libc` shared, anyway. I do have the impression that linux is retarded, but there must be limits to HOW retarded it can be... at least it sports copy-on-write (and hopefully demand-loading, too.) This means that, apart from pagetables and COW'ed regions, the `libc` runtime should be shared with other processes running on the system. Considering linux's nature, there will be a lot of

## alt.lang.asm: Re: LuxAsm questions

these, so you can be pretty much guaranteed libc will already be in memory.

Localization through plugins... is a bad idea. Localized software becomes pretty huge this way. If you really want localization (which I personally think is a horrible idea, especially for something as 'advanced' as an assembler), it's better to put all your strings in an external (UTF-8) file, and load strings runtime. You can write your own code for this, or use one of the existing projects. I've done this for commercial projects, and it's \*so\* much easier to manage than hunting through the source for strings.

Btw, your windows DirectX fallback example is a bit silly :). If you need DirectX 9 capability, you `_need_` it, and no other version will do. If you need something earlier, that's what you're going to get (from your COM request). You can say what you want about COM and it's implementation, but it's one of the better systems for providing nearly-100% backwards support... the idea of "falling back" is good, but consider `_where_` you can apply it...

*>And, yes, I'm being sarcastic about the 90+% stuff to a  
>degree...after all, for all this fuss about "CPU portability",  
>how many `_ACTUAL_` target processors and OSes are there in  
>reality?*

If you care about desktop machines at home, it's like "x86 + windows". If you're doing commercial stuff, you shouldn't really care about anything else, unless you're writing niche software – since "x86 + windows" is almost the only market. If you start writing more "funky" stuff, or target servers or embedded devices, there's suddenly so many OS'es and CPU's that you might understand why portable code is a good idea ;). And of course there's the `_stupid_` push for "64bit PC's for everybody!"...

PS: OpenGL isn't all that portable :). To do anything interesting (well, by these days' standards), you need vendor-specific extensions. To use vendor-specific extensions, you need OS-specific "GetProcAddress"-like calls. This leads down a wormy path of a lot of `#ifdef`'s and platform+vendor specific code.

PPS: "C is portable" :). Apart from the petty little things that a lot of people write, a lot of code (at least in the term of "what carries the workload") will be happening in algorithmic stuff, that isn't tied to CPU or OS. It's no big task writing this code so that it will compile out-of-the-box on little/big-endian 32-bit machines or 64bit machines (16-bit compatibility requires a lot, but I hope I'll never have to code for such a limited platform.) Of course things get messy once you need to use advanced features like multithreading or GUIs, but if you plan ahead and don't write "API code", this isn't too bad either (though you may have to live with a crappy-looking UI and sloppy code, wxWindows or whatever.)

alt.lang.asm: Re: LuxAsm questions

PPPS: "old man with a pipe" wouldn't happen to look something like  
<http://www.virtualbrum.co.uk/tolkien/tolkien-jrr.jpg> ?

PPPPS: you write long posts ;)