

## Re: The dangers of replacing malloc()

**Source:** <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/performance/2003-06/0103.html>

---

**From:** Terry Lambert ([tlambert2\\_at\\_mindspring.com](mailto:tlambert2_at_mindspring.com))

**Date:** 06/26/03

Date: Thu, 26 Jun 2003 00:11:31 -0700

To: "D. J. Bernstein" <[djb@cr.yp.to](mailto:djb@cr.yp.to)>

"D. J. Bernstein" wrote:

- > *Suppose that, taking the advice of inexperienced programmers who trumpet*
- > *weak linking, I use sbrk() to write my own malloc(), free(), etc. Here's*
- > *what can go horribly wrong.*

If you are referring to me, I have 24 years of professional software engineering experience and 20 years of UNIX kernel experience. 8-). But let's address your issues...

- > *Suppose the OS distributor doesn't go to the effort of dealing with*
- > *competition for sbrk(). This is the normal situation; it has been quite*
- > *explicitly tolerated by the sbrk() documentation for several centuries.*

The sbrk(2) system call is an implementation detail for specific systems. It has nothing to do with malloc(3), other than it is used on some systems to allocate private, anonymous memory. Other implementations use mmap(2) of /dev/zero to obtain pages of private, anonymous memory. In fact, mmap(2) *is* specified by IEEE 1003.1, while sbrk(2) is not. In other words, the implementation details of sbrk(2) are unimportant to the validity your argument.

- > *Suppose the OS distributor decides to write somedorkyosfunction() using*
- > *some funky new allocation function that I haven't replaced because I*
- > *haven't heard of it. Yesterday it was valloc(); tomorrow xyzalloc().*
- > *This happens all the time: look at the FreeBSD reallocf(), for example.*

The reallocf() function is a replacement-safe wrapper function; it is a red herring.

Memory allocation in the C library is *defined by standards*, all of which require the ability to replace memory allocation and freeing functions with equivalents, *as a unit*. The POSIX corrigenda is very clear on this matter.

For example, less of a red herring is "strdup()", as it's also defined by POSIX. But internally, it's required to use the

## freebsd-performance: Re: The dangers of replacing malloc()

externally visible allocation method "malloc").

- > *Suppose the OS distributor decides that valloc() or xyzalloc() should do*
- > *its own thing, rather than calling malloc().*

Then that OS distributor's OS no longer complies with standards.

Realize that programs are not written to interfaces, they are written to standards which are implemented by interfaces. The distinction is subtle, but very, very important: it means that if you write a program to a standard, and a system implements that same standard, then you are guaranteed that your program will compile and run on that system.

- > *This happens too: I tried the sample program shown below under*
- > *Linux, and somedorkyosfunction() ended up calling brk() rather*
- > *than my own malloc().*

You should complain on one of the Linux lists that Linux is in non-compliance with "IEEE POSIX 1003.1-2003 Issue 6", and will they please fix this function, since it is broken. The Linux people are generally reasonable and generally cognizant of standards; I'm sure if they are notified reasonably of any non-compliance, they'll very quickly hurry to fix it.

- > *Finally, suppose the OS distributor decides that some syscall I use*
- > *should be replaced by a library routine that uses somedorkyosfunction().*
- > *This happens too. Note for the reading-impaired: I'm not saying that the*
- > *name malloc() has ever been used for a syscall; I'm saying that poll(),*
- > *socket(), et al. have been used for allocating library routines.*

Link dynamic instead of static. Nothing that's currently a system call is guaranteed to remain a system call, so an OS has no contract which prohibits it. Technically, you are in non-compliance with the IABI ELF specification if you link your program statically.

If you are in non-compliance with the Intel Application Binary Interface specification, you should expect to \*minimally\* be required to relink, recompile, or have to modify your program source code, each time the OS major version number changes, for \*any\* IABI ELF compliant OS. The only contract it has with you is to not make symbols required by the standards with which it complies go away on you, without bumping the compliance level at the same time.

- > *Result: My program innocently calls that library routine, which calls*
- > *somedorkyosfunction(), which calls valloc() or xyzalloc(), which*
- > *incorrectly assumes that its sbrk() results are contiguous, destroying*
- > *the data allocated by my own malloc().*

Practically, only very bad programmers write libraries this promiscuous. Do you you really care if your software runs on OS's written by very bad

Re: The dangers of replacing malloc()

programmers?

If so, where do you draw the line? Am I permitted to, for example, add a parameter to all system calls to permit the optional passing of either a "0" or a mailbox address to deal with AST notification, thus making all system calls asynchronous?

At some point, you have to pick a standard to which you will code, and then expect that the functions that you call that are defined by that standard will not change out from underneath you at some point, until the platform on which you are developing willfully decides to no longer comply with the standard.

I know that you use "#include" in your programs; this works because there is a language standard call ANSI C which was defined by the ANSI X3J11 committee, and it includes standardization of things like preprocessor directives, and so on.

At some level, you have to pick a standard that you are going to trust to be present on the platform(s) on which you wish your code to run, and then code to that standard as if it were set in concrete.

*> As I said before, I encounter more than enough portability problems  
> without going out of my way to look for them. I wish OS distributors  
> would put a little more thought into the needs of people who \_don't\_  
> spend their entire lives working with a single platform.*

In general, we do: we attempt to comply with IEEE POSIX 1003.1; if you write to the 1998 version of that standard, we guarantee that your code will run on our platform, within the limits of the machine architecture (e.g. you can't map 400TB of disk file into your process address space on a 386 machine), or we will fix whatever it is that's preventing your program from running correctly.

This includes fixing all instances of "somedorkyosfunction()" that you are able to find on our platforms.

The Linux people are the same way; your example function, which you didn't name for us so we could get them to fix it, is something they would almost certainly immediately take care of, were you to bring it to their attention as a standards compliance issue.

-- Terry

---

freebsd-performance@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-performance>

To unsubscribe, send any mail to "freebsd-performance-unsubscribe@freebsd.org"