

## Re: xmalloc string functions

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2008-02/msg00223.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-02/msg00223.html)

---

- *From:* Jeffrey Stedfast <[fejj@xxxxxxxxxx](mailto:fejj@xxxxxxxxxx)>
  - *Date:* Sun, 03 Feb 2008 00:09:55 -0600
- 

On Sun, 27 Jan 2008 17:38:01 -0800, William Ahern wrote:

Yevgen Muntyan <[muntyan@xxxxxxxxxxxxxxxxxxxxxx](mailto:muntyan@xxxxxxxxxxxxxxxxxxxxxx)> wrote: <snip>

I'd think evolution crashed because it crashed. You can look at its list of bugs to see what I mean. If it was abort() inside g\_malloc(), then it probably leaked so much that it indeed has eaten everything and asked for more. In which case you can't really blame glib ;)

Actually, it was usually Galeon/Gecko which leaked, or rather whose leaks would begin to trigger out-of-memory errors. But that's beside the point. I'd much rather that Evolution fail to open a message than to exit entirely.

(Here I'd be actually grateful if such applications were killed immediately, because they first freeze X, and I've got to wait for ten minutes to switch to console and kill the application. They just don't die themselves.)

If these were the only choices (crashing applications or a frozen screen), I'd be more agreeable.

There were many bad spots in Evolution, but mostly the code was OK (I've combed through much of it). Evolution had to deal with bad UTF encodings, malformed MIME, networks coming up and going down. Handling memory allocation failures would hardly add much in the way of complexity or effort.

As someone who has looked over what I would guess was Camel code (you mentioned MIME, bad UTF encodings, etc) – you've likely seen that we (I'm one of the main original 2 developers who wrote that code) were pretty meticulous about checking error codes from syscalls everywhere.

We had some problems with g\_malloc() abort()ing on failure, but a lot (all?) of those problems were actually real programming errors (such as

## Re: xmalloc string functions

trying to malloc some ridiculously large amount of memory – e.g. in the neighborhood of 4GB). Most of these sorts of problems occurred in the IMAP provider of Camel (more than anywhere else in all of the mail code, likely, but certainly in relation to the rest of Camel).

For example, when requesting messages from the IMAP server, we would naively try to allocate a memory buffer using the size value provided in the LITERAL response from the IMAP server. There were also a few places that did some bad pointer arithmetic to arrive at negative values (which, when cast to size\_t are rather large ;–)

Before I moved on to more interesting projects, I started a complete rewrite of the IMAP provider which you can find in the evolution–data–server source tree under the `camel/providers/imap4' directory.

FWIW, glib also has g\_try\_malloc() and g\_try\_realloc() functions which retain the original libc behavior of malloc/realloc.

In my experience, glib–based applications "mostly" work.  
Not sure  
whether that has more to do w/ the xmalloc() wrapper, or  
other  
issues.

There is no xmalloc() wrapper in glib. Anyway, have you never seen "mostly working" application which do not use glib? Bugs are everywhere, on all platforms. Do you have a real base for saying that g\_malloc() is somehow responsible for crashes you have seen? Something other than "evolution crashed", that is, or "similar applications" (similar glib–based applications which open messages, huh?).

I know for a fact that Evolution and similar applications have exited because their malloc wrapper decided to exit. They've also crashed for numerous other reasons. Bugs are bugs, but defective by design is hardly excusable.

I wouldn't be surprised to learn that your experience was related to bad arithmetic calculating the size of memory to allocate as opposed to simply being out of memory. I'm fairly certain that's a far more likely scenario than being out of memory for most users.

My point is that while, yes, perhaps Evolution would not have crashed as a result of the malloc() call, it /probably/ would have crashed a few

Re: xmalloc string functions

## Re: xmalloc string functions

instructions later due to the logic errors that contributed to the original problem anyway.

Exiting on malloc failure makes sense for a utility like sort(1). It doesn't make sense for desktop applications, unless there's a separate strategy, like a multi-process configuration where a component exiting is part of the design of handling errors and making a best-effort recovery.

It makes sense for sort(1) because it categorically cannot perform its task on a failure.

Many times this is also the case for applications.

Sometimes it is not appropriate, but sometimes it is.

And its practical because sort is usually just a sub-component in a larger work. Sort can exit without killing the script or application which called it.

You know, I have heard things like you are saying only from people who talk about xmalloc() and related things. Never from users. Why is that? Perhaps because buggy applications are buggy applications, not some poor creatures crashing because glib memory handling is broken?

Users are, sadly, inured to this issue. Nor can they discern the reason for failure, so they aren't capable of judging the cost/benefit of any particular behavior.

You sort of missed part of my point regarding Evolution. What I was experiencing was a denial of service caused by processing untrusted data.

true, but it could also be argued that prevention could (and should) have been done prior to malloc'ing :)

And by using glib you have no recourse.

As mentioned above, there are `g_try_malloc()` and `g_try_realloc()` functions.

While, granted, you don't have that option if you are trying to create a

## Re: xmalloc string functions

widget and Gtk+'s use of `g_malloc()` fails, but you can certainly minimize the problem by using `g_try_malloc()/g_try_realloc()` ... or god forbid, `malloc()` and `realloc()` from `libc` instead.

Most likely if creating a widget fails, though, you're probably SOL anyway – it would be a lot of work to handle this situation by shutting down other parts of the application in order to make enough room to do something useful, particularly if you don't have access to parts of the application that would be ok to shut down.

You should also note that in order to allow application programmers to handle all of these memory allocation failures properly would require a more cumbersome widget toolkit API which would likely deter most non-paid free software developers from developing applications using said toolkit.

If you don't believe me, just look at how few people actually error check all of their syscalls in any real-world software you've ever read the code for :)

If I got a penny for every unchecked `malloc()`, `open()`, `read()`, `write()`, `close()`, or slew of other calls I ever saw in free software projects I've looked over the code for, I'd be a very rich man.

Hell, I'd be a rich man just for the little amount of time I spend reading code in this newsgroup :)

(come on, laugh, you know it's true ;-)

Indeed, much of `glib`'s functionality is simply a rehash of, by now, widely supported library interfaces, but with the feature of exiting when memory becomes tight.

There's definitely a lot wrapper API around `libc` functionality... like `g_mkdir()` and so forth (I guess this is needed for Win32 porting? \*shrug\*).

I also really hate the `g<type>` types, I always use `size_t` when I want `size_t`, for example.

<snip>

One of my rules of thumb is that if a network daemon uses `glib`, I automatically exclude it from consideration. I can deal when an application crashes and destroys my work. I don't want to be responsible for installing an application which crashes and

## Re: xmalloc string functions

destroys  
or interrupts `_other_` people's work.

It's fine, nobody promised glib will work for any program. It certainly won't; but nevertheless it doesn't make abort-on-failing-malloc less sensible strategy for a whole class of applications.

The problem is that such a strategy is usually the wrong one for the class of applications which glib serves: desktop applications,

I would tend to disagree depending on the situation in which the malloc failed.

Perhaps that means we agree, but to me it sounds like you are saying it is never ok to abort on malloc failures which I happen to disagree with.

and  
increasingly network daemon services.

Well, network daemon services I would agree very much with – but network daemons are usually a lot less complex and a whole different breed of program than desktop applications.

Both of those usually involve monolithic applications doing complex tasks for which memory allocation failure is only one of dozens or hundreds of exceptional conditions. And yet out of all of them people will argue memory allocation alone can be completely ignored, simply because it's too burdensome.

Besides, a glib application can set up some sort of emergency memory pool or something, so that failed malloc doesn't necessarily lead to immediate abort(). Same sort of science fiction as "graceful exit with saving data on *\*any\** failed malloc() call in any possible application in any possible situation" which seems to be so popular here ;)

It's not science fiction. It's just difficult. And sometimes the answer involves not using C, rather than using C and choosing not to address the problem.

This is definitely true.

## Re: xmalloc string functions

Being able to throw an exception, for example, in a language like Java, c++, or heaven forbid, C# definitely gives the programmer better options for achieving proper error handling for memory allocation failures.

Unfortunately, it's not always trivial to do so in large applications written using C.

A strategy you might find many application developers take is that of periodically saving user-data as a precautionary measure to avoid data loss should the application unexpectedly exit (due to a `g_malloc()` failure or any number of error conditions that may not be accounted for).

I would personally love to see a large desktop application written (in C) that handles all `malloc()` failures properly as some (presumably non-application developers) in this thread have been dictating. It would be a learning experience for me and I would greatly appreciate learning new techniques :)

Jeff

.