

Re: xmalloc string functions

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-01/msg04109.html

- *From:* Yevgen Muntyan <muntyan@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 28 Jan 2008 04:17:54 GMT
-

William Ahern wrote:

Yevgen Muntyan <muntyan@xxxxxxxxxxxxxxxxxxxxxx> wrote:
<snip>

This is very very wrong. A typical GUI application does not do a switch like

```
switch (problem_to_handle)
{
...
}
```

to which you could add

case ALLOC_FAILED:

It's usually different, you got the main loop which got to spin, you got those controls you got to draw, and you got those callbacks which actually do the job. And the callbacks do one thing at a time, they do not handle dozens of exceptional conditions at once, they do not handle exceptional conditions at all in fact.

Is that why applications crash when, using a file dialog box, I attempt to save a file into a directory I don't have write permission to?

Re: xmalloc string functions

No idea, ask the developers of that buggy application. Failed `fopen()` is not an exceptional condition. Failed `malloc()` is. Or we are using different vocabularies.

To my mind, there's no difference in effort required to handle a NULL return from `fopen()`, than a NULL return from `malloc()`. Maybe more typing.

Then you are just really good. Because it's enormously more typing. And more than that, it's more design questions too: "what do I do in this situation, which I can't even possibly test?" All this apart from real problems you have to solve. Yes, *real*. No, `g_malloc()` aborting an application is not a real problem. Not for a regular desktop application.

This is just a resource acquisition issue, and even if you had infinite memory it's a pattern you still have to deal with.

Except you don't open files twenty times in a row in every function in your application. Memory is quite a different kind of resource. Different in how you use it, you know.

As to main loops, I'm very familiar with these. I write event based `async-io` network software, using an event dispatcher exactly like a GUI application might. I create and use more callback interfaces than I probably should. When I accept a connection, I might—though, try not to—do dozens of allocations. I try to write my code so any allocation failure is handled gracefully. I don't need a gigantic switch statement, or special language constructs. One designs the code to deal with such a circumstances as a matter of course. You minimize dependencies, isolate access to shared data, postpone committing to a particular state wrt to that context until you've acquired a minimal set of resources, etc. Any non-trivial application usually has multiple contexts within which such intermediate failures can be contained, with practical benefit.

Granted, I've not done much work with X11 applications, or GUI applications in general. But, I fail to understand how a caveat wrt to X11 justifies—absent other reasons—exiting when a string cannot be allocated.

So you click Save button then click Close. The application failed to

Re: xmalloc string functions

process Save click because it failed to allocate memory for the event structure to put into the event queue, but then it successfully handled Close because at the same time yet another document was closed and some memory returned to the malloc pool. You may not just lose events like that. *Everything* must be done in order, or the application is doomed, and the best it can do is to try to exit as nicely as it can (like save data or whatever). It can't just pretend nothing happened.

And yet out of all of them people will argue memory allocation alone can be completely ignored, simply because it's too burdensome.

<snip>

Of course I am talking about "small" allocations here, not about stuff like allocating memory to load an image file (for those `g_malloc()` is simply not used).

There's absolutely no qualitative difference between small and large allocations without reference to other circumstances (number of allocations, etc). If I have 4GB of memory, what does it matter that a 10MB allocation is checked but not a 12B allocation? When the application approaches the limit its not likely that one will be more susceptible to failure than the other. The choice is then arbitrary and almost absurd. Better, for consistency, to not bother at all.

All allocations are checked. It's what you do when they fail is different. If `malloc(12)` failed, then you are screwed because all your code wants memory. No memory => application isn't working. So you just don't try to handle (that is do something and not exit the application) possible `malloc()` failure when you are concatenating two strings to make up a string to display. Absurd, fine, I'll be delighted to see an application which handles `malloc()` failure when it draws a menu label (it *is* possible, it just doesn't make sense).

.