

Re: xmalloc string functions

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-01/msg04343.html

- *From:* Flash Gordon <spam@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 29 Jan 2008 08:54:28 +0000
-

ymuntyan@xxxxxxxx wrote, On 28/01/08 23:30:

On Jan 28, 1:39 pm, Flash Gordon <s...@xxxxxxxxxxxxxxxxxxxx> wrote:

Yevgen Muntyan wrote, On 28/01/08 04:17:

William Ahern wrote:

Yevgen Muntyan
<munt...@xxxxxxxxxxxxxxxxxxxx> 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

Re: xmalloc string functions

file dialog box, I
attempt to
save a file into a directory I don't have write
permission to?

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.

The point is that they both require similar recovery strategies.

The point is that they don't. If `fopen()` fails, there is nothing
to recover from. Though if all your application does is `fopen()`,
then you can safely `abort()` when `fopen()` fails.

As others have pointed out, that is a good way to stop people using your apps.

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.

Not necessarily.

So, you work with a list, and you append an element to it.
Now you do `list = g_list_append(list, something);` with `malloc`
error handling you'll have to test whether `list_append()`
succeeded. Not much more typing, no. A little bit more, huh?
C++ exceptions would be appropriate here, but manual error
checking in C code **is** much much more typing.

Not always. I've done it using structured programming in assembler without implementing exceptions and at each point I checked the status and propagated the error until it could be handled. The handling consisted of processing what it had memory for giving degraded performance instead of giving up which would not have been acceptable. It was also easy to do because I new resources were limited and designed the SW assuming that they could run out.

And more than that, it's more design questions too:
"what do I do in this situation,

Re: xmalloc string functions

That is part of the normal design process.

which I can't even possibly
test?"

I can test it relatively easily. For example, I can use a malloc wrapper which allows me to force a return of a null pointer when I want during testing. Or I can use the debugger to put a breakpoint in and change the pointer value to null at the point I want to trigger the failure.

There are about five bazillion allocations, debugger won't do. Random malloc() failures will do as a nice stress test, yes. But you still won't be able to test it properly. At least not that piece of code where it will segfault when user runs it (here I assume that user will be able to see it, perhaps on windows). A better thing to do is to test malloc() failure in one place, and possibly do what you can do there, and abort.

You propagate the error upwards until it can be handled sensibly. So most of your malloc failure tests are simply returning a failure code after maybe some simple local recovery. Far better not to lose the users hard work for a little thought in design and implementation.

All this apart from real problems you have to solve.

Yes, this is called software design, a task that people writing software should perform.

Yes, *real*. No, g_malloc() aborting an application is not a real problem. Not for a regular desktop application.

You have just had pointed out to you a time when it has been a problem for a user.

Yeah, mozilla leaking too much. Or evolution leaking too much (?). It would be nice to see something more substantial than "I know for a fact" (debugged it, looked at the core file?). And would be nice to hear about gedit or gnumeric crashing because of malloc() failure.

Well, I have seen the Lotus Notes client report out-of-memory on attempting to open a window. The Notes client did not crash, it reported it and left the dialogue box there. When I could I closed down some other applications (not immediately) and avoided losing partly typed emails. I've also had VMWare report out-of-resource at times when the only resource that was tight was memory, and again it gave me the chance to recover the situation which saved me significant work because I had two VMs running and the state

Re: xmalloc string functions

between them was important and took time setting up.

I've previously pointed out times when applications have given me a chance to recover the situation and it has avoided me losing a lot of work.

Avoiding losing your work in an emergency condition is just a different story, say you can have your application lost its terminal or X connection, in those cases you can possibly do something to save user's work. And that's something you can (try to) do from inside `g_malloc()` when `malloc()` fails. It's not necessary to write a `g_list_append()` which can fail for that.

See above. I *use* applications that do not abort but instead allow me to keep all my data and continue.

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.

I'm sure that William does 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

Re: xmalloc string functions

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 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.

Do you have reason to believe that the X server is that brain dead?

Yes I actually do (try xorg sources). But it's not quite relevant, since I wasn't talking about X server. Though if X server dies then Xlib will kill the application too.

If the X-server itself is passing the event to the application then it is whether the X-server has the memory that is important. The application can generate events too, but the application do things to recover itself.

It is an easy problem to handle by allocating the space in advance so that when you get the event you *already* have the space to store it.

So, you got an event, you need to put it into the event queue. Either you allocate memory for that (and it fails), or you preallocate memory, it is not enough, and you try to allocate again (and it fails). What can you do apart from some emergency

Re: xmalloc string functions

action (saving important data or something) and exit? How do you "recover"?

You increase the space allocated *before* you are out so that if it fails you still have the resources to pop up a dialogue telling the user and giving them a chance to do something about it.

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.

William has suggested *not* pretending nothing happened.

Right, he didn't suggest anything.

He did not suggest the specifics because the specifics vary depending on the situation. However he did say that you don't throw away the users data.

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).

Re: xmalloc string functions

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.

Not necessarily. For example, if you have done your job correctly the *recovery* code already has the memory it needs allocated, so that can run successfully.

Recovery code will be able to run successfully, so what?
The rest of the code still wants memory.

The recovery may be as simple as telling the user there is no memory to open a new window and let the user continue with the existing ones (I've *seen* this behaviour). Or it may give the user the option of retrying or aborting. Or...

No memory => application isn't working.

Only if you don't design a suitable recovery process.

Again, no recovery process will be able to make your main loop spin happily again. The recovery process can't get memory from nowhere (unless by recovery you mean killing parts of application, in which case it again doesn't make sense to proceed in normal way).

Well, I've seen GUI applications do sensible things thus avoiding me from loosing the data. See above.

Re: xmalloc string functions

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).

I've had windows programs pop up message boxes telling me that they did not have enough memory to open the window I asked them to open. It is possibly, it just requires designing the software to handle the problem.

Perhaps. Except it's not "just". Again, I'll be delighted to see an application which handles malloc() failure when it draws a menu label.

I've seen it done on attempting to open a window. Either it was not enough memory to open the window or it was some component of the window, either case is not hard to handle.

Preferably its code, to learn from. Oh, and see the code which works with list allocated on heap, which handles every possible failure of list_append() (no exceptions and alike please, don't we agree that all we need is 'if (failed()) recover()?).

The precise strategy varies. Fortunately the applications I have to use **do** handle the problem in ways that avoid me loosing data.

—

Flash Gordon

.