

## Re: xmalloc string functions

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2008-02/msg00278.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-02/msg00278.html)

---

- *From:* Jeffrey Stedfast <[fejj@xxxxxxxxxx](mailto:fejj@xxxxxxxxxx)>
  - *Date:* Sun, 03 Feb 2008 13:35:44 -0600
- 

On Sun, 03 Feb 2008 15:54:25 +0000, Flash Gordon wrote:

Let's also not forget that the act of /showing/ the dialog may, in fact, require memory allocations depending on the way the system works.

So you have to look to see what workarounds there are for this. Once you've done that one, you can reuse the solution so even if it takes a week amortise that over all of the applications you will write!

but later you say there are no one-size-fits-all solutions? :)

For example, requesting that a dialog be shown may not actually show the dialog immediately... it might only queue the operation for the next rendering pass.

On at least some there are ways to get it rendered immediately. I know I've done that in the past. Once you have solved it for one application reuse the solution for others.

If the toolkit being used is not one of those, then it is irrelevant that some provide a means to do so, particularly if the "some" are not available for the platform being targeted.

Said rendering pass may require more allocations, but at this point it's too late to simply unwind the stack to the point where you requested the show(). Since no widget toolkit I know of has a way of notifying the application of said error, what is it to do?

## Re: xmalloc string functions

So you are saying all widget toolkits are badly designed. This is possible.

I never said "badly designed", though I would agree "sub optimal in an ideal world". There's a difference (to me, at least).

However, it has enough memory to do it.

How can you assert this?

If you try and fail  
you are no worse off, if you try and succeed you are better off.

I'll agree with that, and wherever I use malloc() directly (or g\_try\_malloc()), I do write error handling – which may or may not include attempting popping up an error dialog depending on the situation).

For Gtk+, you actually do have an option... GLib uses a vtable for malloc/ realloc/calloc/free that you can initialize with your own routines at init.

See, there *are* ways to deal with it!

Right, but as you mentioned was a problem for xmalloc(), we have the same problem here. Not enough context for most real-world applications to recover at this point.

You could potentially do your own NULL-check there so that you can be pre-warned about memory allocation errors coming up, but it'll lack context (who tried to allocate this memory? for what purpose?), but I suppose if you had everything pre-allocated, ready to go – you could call some global prepare\_for\_abort() function that could perhaps iterate thru all of your unsaved files and save them quick before the abort() call in the g\_malloc() wrapper. This wouldn't allow you to pop up any dialogs, however, because at this point its too late.

No, it's not too late, since as you save documents you can free up the

## Re: xmalloc string functions

memory they used giving you the memory to pop up dialogues! Or, as previously mentioned, have stuff pre-allocated.

Easier said than done, not that it /can't/ be done – but one could easily argue that this is more effort than it is worth, and unless you are able to test your failure cases thoroughly, not even reliable.

It is /more/ reliable to routinely auto-save the user's work (as you mentioned elsewhere, to a file other than the original) because it is much easier to warn users about problems (potential or no) and certainly easier to implement recovery should the application crash due to uncontrollable (kernel crash, power outage, etc) error conditions on the next application startup.

Depending on the document, one could write the application such that any button click (or whatever) would cause an auto-save in addition to some timeout, thus reducing the likelihood of there being any unsaved changes at any given point in time.

Since you obviously need this auto-save functionality in place if you are serious about protecting the user's data at all costs anyway, then it becomes no longer necessary to chain malloc() failures up your call stack in order to use static emergency buffers.

At this point g\_malloc() calling abort() becomes a moot point, particularly if your auto-save code is robust against memory allocation errors (keeping a small subsection of code bug free and robust against all possible error conditions is a lot easier and less costly in developer time than it is to do that for an application several million lines of code long).

, and then let you copy the  
previous twenty-digit result  
into  
another document for safe  
keeping.

ah, but that also requires a clipboard  
memory buffer be allocated...  
but you have no memory left ;-)

Have it pre-created, then if you need a larger buffer for the  
next  
step and you can't enlarge it you only loose that last step.

this one indeed is likely an easier and more reliable method for this

## Re: xmalloc string functions

particular instance, but not all desktop applications can go around using this type of approach.

For example, would it be a good idea for an email application to set aside this clipboard buffer? :)

I think we'd both agree the answer is no.

In that instance you would use a different solution, probably saving the email in a draft folder or something like that. People have already said there is not a one-size-fits-all solution!

Hey, guess what? Evolution did this using an auto-save approach and it used `g_malloc()` in much of the application code.

Different approaches, same end result. Oh, sure, maybe in your ideal case, the application exits from `main()` with a `'return 0;'` as opposed to an `exit()` call (or `abort()`), but that is irrelevant.

[snip]

Drop that background print-job or spell check that is consuming memory for your desktop app.

That means you'd have to have that print job context or spell-check context global somewhere, or have some way of getting it from a lot of different locations...

You probably need mechanisms to signal the spell checker and print process anyway to cope with the user choosing to abort them.

This is true, however you still need context information in order to do so. I never said that the application wouldn't have the ability to cancel the spell checker or printing, but in order to do so you need context. If you are in a function being called asynchronously from somewhere which might not even be your code which may not pass up your particular error condition, then you are pretty much screwed unless your contexts are all globally accessible.

While this may suggest the application (or the libs it depends on) is poorly designed (or at least not suitably designed), the argument does little to solve the actual problem at hand.

In the real world of end-user software development (e.g. not software written for space ships or other areas where human lives are on the line)

Re: xmalloc string functions

## Re: xmalloc string functions

where the application's design is based on incomplete specifications (as in they tend to change mid-development) in combination with insufficient allotted time, designing the perfect solution is downright impossible, and so it is, unfortunately, not all too uncommon for the application's design to be insufficient for every possible error condition.

If this is new to you, then you've never written real-world software and I would appreciate having your pity... because I, too, would love to live in Ideal World where I have sufficient time and specifications to use in order to come up with a proper design before I'm forced to begin implementation :)

[snip]

doable if you don't want to give any specifics... daemons are often not very user-friendly in their error reporting... depending on the daemon,

That it depends on the daemon shows that it is possible, or it would be a simple case that none do.

it might be as simple as an integer error code or as forthcoming as a string from `strerror()`, but rarely do they report something that the user is able to understand.

Normally they report something the system administrator is able to understand. At least, most that I use do.

Key word: most :)

Sure, "out of memory, cannot perform that operation" may work for simple applications where only 1 thing at a time is ever going on, but if the application happens to be doing many things at once the user will want to know /which/ operation could not be completed because memory was unavailable?

Yes, which is why things like `xmalloc` are a problem, because they do not have that context.

Agreed in so much as they are not an ideal solution to the failing `malloc` problem :)

## Re: xmalloc string functions

They are, however, /a/ solution to the problem and might, in some situations, be more than ample.

Trust me, this is the case... applications I've worked on have actually had these sorts of complaints filed against them. It's funny, because all the user testing I've seen indicates that users never read the dialogs anyway ;-)

Depends. I've had a report come back to me (via at least a couple of layers of intermediaries) that had exactly the information that the "dialogue" provided (it was not a GUI application).

As have I, in my gui applications even.

– wait until memory becomes available

With an appropriate pre-created dialogue you can do that on a GUI application as well.

see above.

Again, see above ;-)

On the other hand, say, a word processor application, if the user requests some sort of action and a malloc() fails for 12 bytes, what is it supposed to do?

Any of the above.

Easier said than done, I'm afraid...

Worth the effort though.

In an ideal world, perhaps. If you've already got an auto-save feature

Re: xmalloc string functions

Re: xmalloc string functions

then it is not necessarily worth the extra effort.

I would agree that it /is/ worth the effort in the case where the failing malloc() call is in the auto-save code, however :)

If the documents the user has open have already-opened file descriptors, the app might be able to save them before going down – but:

That is easy to arrange.

1. it certainly doesn't have the option of displaying an error dialog.

Yes it does if it pre-creates it during application start-up.

see above, although I suppose if you really wanted to, you could make an exception for the "out of memory" dialog case as opposed to other error dialogs your application might use.

You have to take extra care with any out-of-resource error to ensure you can report it without the resource in question.

First... I wonder if there are any widget toolkits that don't already abort() (or similar) when they run out of memory or in any other conditions without giving my calling code a chance to handle it?

Lotus Notes has given me an out-of-memory dialogue. I'll leave you to draw your own conclusions from this.

I would conclude, that, like some parts of Evolution, if it is unable to allocate resources for some non-critical data structure(s), that it is able to report the "out of memory" issue to the user.

I seem to recall you claiming VMWare reported "out of memory" conditions to the user as well, but as Ben Pfaff noted, VMWare uses xmalloc-like wrappers as well.

## Re: xmalloc string functions

As someone else mentioned, X already has this limitation... so right there, that means there's no Unix toolkits that you can use.

Maybe you cannot trap and deal with all of them if the underlying system does not let you, but that does not mean you should ignore those you can deal with!

Never said otherwise!

Guess we'll  
all just have to write applications for ... does Windows or MacOSX  
handle this? I somehow doubt it.

It may depend on exactly where you hit it. Of course, any time when your application or library calls malloc it has the opportunity to do it!

Sure, the same goes for any application written on top of glib!

(Not the case if you use g\_malloc() of course, but you are hardly forced to use only g\_malloc() just because you link with glib).

2. if any of the files are unnamed or  
otherwise would require any of:

Don't allow them to be unnamed. You can create a name at  
the same time  
as creating the otherwise unnamed document.

What if the act of creating a name is what finds the out-of-memory  
condition?

Then that is before the user has had a chance to enter any data in the  
unnamed document, so they won't be as upset of it pops up a dialogue  
saying "Out of memory, cannot create new document".

Not necessarily, but I will agree that this is /likely/ the case.

## Re: xmalloc string functions

A single logging file to allow recovery on application restart is possible. It requires some work on synchronisation, but if designed in from the start is possible.

I've used this approach for some simpler applications.

Auto-save is actually not that much different to this.

I maintain that a better way is simply to auto-save user's work... far far simpler to implement and far more reliable a solution unless you are able to provide 100% test coverage for your application.

Yes, regular auto-save is another way to protect users data, as long as you are not saving over the original.

Agreed.

Remember, we are talking about free software desktop applications written by volunteers, here, not programmers paid 150k/yr to write branches for each memory allocation they ever make in their application or library.

You may only be talking about free software written by volunteers,

I am talking about software written by anyone, but especially volunteers.

I am talking about all software whether written by volunteers or not. The open source community (some of it at least) wants to be taken as a serious alternative to closed source, so it should take the same effort to produce robust applications and libraries.

Amusing to me is that none of these developers are writing GUI apps or libs afaict ;-)

It's not hard to find command-line programs and/or general purpose libs that /are/ robust, like Ben Pfaff's AVL tree library for example, but

Re: xmalloc string functions

## Re: xmalloc string functions

none of the ones I know of for writing GUI applications are of this quality.

If I wanted to write an application that would meet your ideal criteria, I'd have to write my application from the ground up, including the widget toolkit. This is not only impractical from the development standpoint, but also from the user's perspective where the application does not look like any of his other applications. It would also not be able to share much with the other applications running on the user's desktop and so would use a lot more resources than a Good Enough solution.

Especially provided that these programmers can at best assume that the authors of the libraries they are building on top of have also done their work of error checking every malloc and properly handling it and/or chaining it up to the caller.

Yes, you do need the libraries you are building on to pass up the errors, hence the comments about glib.

Anyone using glib stand-alone should probably reconsider, especially if they are writing "mission critical" applications.

Most people, however, use glib via Gtk+ – and being that is the /only/ practical widget toolkit available to C software developers for Unix, you can't easily write off glib altogether.

I honestly would not be surprised if the other major contender in the widget toolkit space (being Qt) had similar problems wrt memory allocation failure conditions, but even if it did, you wouldn't be able to write the application in C afaik (you'd have to switch to c++).

This is, in fact, where your whole argument falls apart. As a developer of GNOME desktop applications (hell, scratch that – of X11 based applications), you already KNOW that I cannot rely on glib or gtk+ (or Xlib) to gracefully handle all memory allocation errors... so I have no choice but to resort to my auto-save approach.

Or a form of logging as the user goes that you can use to recover (I've used non-gui applications that do this). Of course, you have to make sure your auto-save and/or logging handle resources very carefully so that they do not lose the last good state if they run out of memory.

## Re: xmalloc string functions

Yes, this is what I've been saying.

It's too easy to forget that large applications are generally built on top of code that other people wrote that you may or may not be able to read the source code for to verify that they properly handle all errors.

It's very easy to remember I find since I \*am\* building my SW on top of 3rd party libraries.

You could even make this argument for daemon authors – how many of you have actually read through all of the source code for the libc you build your applications on top of? None? Remember that and you might want to rethink not using the auto-save approach (in addition to error checking).

These days no one has time to check all the code they rely on (and often the source code is not available for everything). So yes, you rely to a degree on others doing the job right.

Glad we agree so far.

As part of that you point out when it is done wrong!

Well, discussing it here isn't going to get the problem solved. If you truly feel that strongly about it, then you should either fix the problem (free software, afterall) or at the very least submit a bug report! ;-)

Oh wait, I forgot that this whole thread is actually a pissing contest more than anything else, so that people who don't actually write desktop applications can feel superior to those who do.

I think it is more annoyance at application we might otherwise

## Re: xmalloc string functions

consider using that would just throw away our hard work in situations some of us do hit.

I've hit it as well, and yes, I'm also annoyed when it happens – but it is a problem that cannot be easily remedied by the application developers if the software stack somewhere underneath the code they've written is faulty (and I have personally run into Linux kernel and GNU libc bugs that have cause problems in applications I've written).

You can't deal with *\*everything\** but we were talking about dealing with something where the libc *\*does\** report a failure.

You /assume/ that all code paths properly handle OOM conditions internally and propagate them back up the call stack. But libc is still only implemented by humans last I checked, so there is a possibility of bugs.

That's a pretty hefty assumption that you CANNOT rely on for mission critical user data (since that's what your whole argument revolves around in the `g_malloc()–is–evil` argument).

Because of this possibility, you MUST implement a safety net – aka auto–save. Once you have auto–save in place and properly written to handle every conceivable error condition that /it/ may encounter (OOM being one), then the value gained by using `malloc()` over `g_malloc()` in the remaining areas of the code begins to rapidly lose their practical value (if the goal is simply to make sure the user's data is saved before exiting).

Wouldn't you agree?

Since you cannot rely on error checking to catch all errors, it's best to have a fallback plan – which is auto–save. It's a lot less likely to fail and a lot more tolerant of buggy code (either in your application or below your stack).

I've no problem with autosave being part of the recovery strategy. As you say, it can help when there is nothing that can be done because the kernel has crashed.

Right.

## Re: xmalloc string functions

In the real world, developers do not have the luxury (or desire, most of the time) to write applications from the ground up. They build on top of software that already exists.

The do have the luxury of choosing which libraries to build on and of reporting things which are a problem. You also have the luxury of not using malloc wrappers that don't allow you to do suitable recovery.

Not always.

For bonus reading, you might check out Richard Gabriel's paper on **Worse Is Better**.

GLib's `g_malloc()` must be "good enough" because more and more Gtk+ applications keep popping up like wildfire just as C overtook LISP due to the **Worse Is Better** rule.

Jeff

.