

# Re: xmalloc string functions

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2008-02/msg00754.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-02/msg00754.html)

---

- *From:* [ymuntyan@xxxxxxxxxx](mailto:ymuntyan@xxxxxxxxxx)
  - *Date:* Wed, 6 Feb 2008 20:11:19 -0800 (PST)
- 

On Feb 6, 6:22 pm, "Herbert Rosenau" <os2...@xxxxxxxxxxxxxxxx> wrote:

On Thu, 31 Jan 2008 22:47:37 UTC, ymunt...@xxxxxxxxxx wrote:

On Jan 31, 2:26 pm, "Herbert Rosenau" <os2...@xxxxxxxxxxxxxxxx> wrote:

On Tue, 29 Jan 2008 08:12:51 UTC, ymunt...@xxxxxxxxxx wrote:

There is nothing to recover because failure of fopen() is a normal situation. Absolutely different from a failure of malloc() when you are trying to allocate a structure to push into the event queue to scroll text a bit later.

There is no difference at all. When I can't open a file I will tell my caller that fact like I do when malloc() fails to deliver me the amount of memory I ask it for.

The caller will check the error condition it receives and will react properly on that and then tell its caller the the error when it is unable himself to recover from that error to continue as it should when possible. Its caller does the same until there is a point that is either

- action is complete undone – clean state normal work goes on
- action is complete undone – no chance to to continue normal run

## Re: xmalloc string functions

shut down program cleanly – a restart of the program may be more successful

There is nothing that requires an abort() or exit() abnormally.

What exactly are you talking about here? If you do shut down the program then you do shut it down, that's it. Whether you do it up the stack in the main() or in the memory allocator may or may not be different. If it's not different, then it

- 1) doesn't make much sense to unwind the stack;
- 2) actually is more expensive to unwind the stack: the more code the more bugs you got.

Where it's not true, it's not true.

exit() does NOT write buffers not already given into the stream, save data local to the functions in the call chain, lots of other work a clean shutdown has to do.

Depends, doesn't it? You don't have to flush buffers if you are not writing anything. You don't have to flush buffers if you are writing a temporary file and your writer is not going to succeed. Not sure what it means to "save data local to the functions", but that may or may not be needed. You certainly don't need to free() your allocated blocks if your process is going to die now (it's not a general situation, remember?)

Lots and lots of work is done on clean shutdown, certainly. Much of that work needs memory too. You can do it? Great, do it. You can't? Well, you insist on trying anyway, I am saying it's pointless.

However catching any error (including malloc() fails) is a must.

Yup.

Sure, it costs a bit more to get a program failsave – but it saves lots of money, lots of time and holds customer because no

## Re: xmalloc string functions

customer  
will ever complain because the program aborts, loses or  
destroy work  
or data only because the programmer was crazy enough to  
kill its  
program only because there was a moment where an open()  
failed, there  
was for a period of time not enough memory to allocate a  
small big or  
very big amount of memory to get a single action done.

It makes always sense to check malloc() for success –  
because this  
saves a lot of money for useless maintenance, earning  
annoyed  
customers, ending up in losing trust.

Right, malloc() result must always be checked. Except it doesn't  
imply your code must be cluttered with if() for every call to  
whatever\_func\_you\_use\_to\_allocate\_memory().

Ah, you means really that you should never check for error but  
shorthand exit() when an fopen() fails, a fread() or fwrite() can't do  
its job its asked for? Because your code gets cluttered on if on that?

No, I didn't mean that.

By that my code is full of functions designed to use in that program  
only once – because these functions are designed to hide details of  
work from the process through the action.

This is completely false. Even a GUI application can handle  
any lack  
of memory well without the need to crash the whole app.

Right, it can. Except it can't draw if it doesn't have  
memory to draw. I don't care if the application is still  
up if it doesn't display the stuff it's supposed to display.  
Nor do I care how it exits on OOM, using abort() or exit(0)  
or exit(EXIT\_FAILURE); does it exit from main() after everything

Re: xmalloc string functions

properly returned an error or right in the draw\_thingie() call (AFTER it properly cleaned up its crap, how about that?).

There is no need to exit() or abort() a failsave app.

Failsafe application? It won't fail, that's true. I don't think I am talking about failsafe applications though.

You can't draw?  
Defer the action until you can do it again.

Good one.

It doesn't matter why you can't. In some time later you'll be able to do the the job.

Yeah. Hope and you'll get what you need.

You can't show an error to the user? Hi, your log will show that fact.

What is that log again?

However only the caller or one of its parents in the chain will know how to handle the failure "no memory" in a usefull way.

BS. Why is it everybody obsessed with "callers" and "parents"? Sounds like you have no idea about how gui applications work.

There is a need to unwind the stack until the error can be handled useful. Hey, only that may give you the needed resources to interact with the user or other parter your program interacts with.

Yep, main() will get more memory to the toolkit\_main\_loop() function.

## Re: xmalloc string functions

However when your program comes in a situation where it can't continue the active action because lack of resource needed unwind until it is in the state "nothing done yet" again. Then redo the same thing may end successful or fail again. Depending on the current environment there can be a chance to retry, defer to later or simply shut down clean. But in any way there is no need to exit() or abort() only because a resource needed to continue is unavailable yet.

Try harder and you'll succeed! There are no situations where you should fail! Did I get it right?

At least it doesn't matter if a job can't success because lack of resource (memory, handles, atoms, or whatever else), important is that the app gets back to a state it is able to save anything worth to get saved before it gets lost.

Application shouldn't lose data, absolutely. And? Banana and apple are fruits. Yep.

Anyway, let me make things easier for you and others, so you don't have to read what I didn't write.

=====  
= Wisdom starts here

The only thing an application can do when malloc() fails is abort(). Any application, in any situation. In fact, applications should use the following API to be robust and for user data to be safe:

```
void *alloc_or_die(size_t n)
{
void *ptr = malloc(n);
if (!ptr)
abort();
return ptr;
}
```

```
FILE *fopen_or_die(const char *filename, const char *mode)
{
FILE *file = fopen(filename, mode);
if (!file)
abort();
return file;
}
```

I got more, but I think you get the idea.

Re: xmalloc string functions

Re: xmalloc string functions