

Re: A solution for the allocation failures problem

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2008-02/msg00641.html

- *From:* Kelsey Bjarnason <kbjarnason@xxxxxxxxxx>
 - *Date:* Wed, 6 Feb 2008 02:02:24 +0000
-

On Wed, 06 Feb 2008 09:19:54 +0000, Herbert Rosenau wrote:

On Tue, 29 Jan 2008 11:47:27 UTC, jacob navia <jacob@xxxxxxxxxx> wrote:

1:
It is not possible to check EVERY malloc result within complex software.

This is a lie!

No kidding. One wonders if they feel the same way about, oh, opening files, or establishing network connections – is it impossible to check every call to fopen, every call to connect? If not, why is memory allocation so magically difficult to check?

2:
The reasonable solution (use a garbage collector) is not possible for whatever reasons.

Where in the standard is a garbage collector mentioned?

Nowhere, but the whole issue is really nothing more than one of simple laziness, and whatever issues GC may have, good or bad, it tends to reduce the effort involved in managing memory. (Until it doesn't, then, well, have fun.)

1) At program start, allocate a big buffer that is not used elsewhere in the program. This big buffer will be freed when a memory exhaustion situation arises, to give enough memory to the error reporting routines to close files, or otherwise do housekeeping chores.

Re: A solution for the allocation failures problem

Releasing such buffer may not effect the ability of malloc() to return or not memory at all.

In theory, at least, it should – as the definition of free explicitly says it releases the memory for subsequent allocation, which means (presumably) that a conforming implementation cannot hand it back to the OS – it wouldn't be available for allocation if it's being used elsewhere.

```
fprintf(stderr,  
"Allocation failure of %u bytes\n",  
nbytes);  
fprintf(stderr,"Program exit\n");
```

Who says that the application is able to print to stdout? Any GUI app on my OS has neither stdin, stdout nor stderr pointing to something known as (virtual) device.

Granted, but if you're gonna dump diagnostic messages, it's hard to beat this as a more or less standard approach.

Crappy because a simple return NULL to the caller of the function calling malloc() is the only manageable solution to recover from that failure as only the caller or its caller will be able to win enough memory to continue successfully.

Dealing with the failure "at point of failure" isn't inherently evil. For example, if the calling code is trying to, oh, send a packet across the network and the connection has failed, it might be nice if it tried to re-establish the connection before giving up. Depends on the app, but it's not a wholly unreasonable concept.

I *think* this is the guiding logic here; try to allocate, if it doesn't work, try to recover, if you can't, bail.

In terms of dealing with things "at point of failure", where you probably have the best bet of actually coping with the problem, it's not so bad a concept.

Where it becomes bad, for me, is that the failure strategy is to simply throw in the towel, without giving the caller any chance to do anything about the situation.

In an example I used elsewhere, if my word processor can't allocate

Re: A solution for the allocation failures problem

Re: A solution for the allocation failures problem

enough memory to load a fourth document, I have at least two options: crash and burn, or simply accept I can only work with three documents right now.

Crashing and burning is not so good if I have three documents open with edits I'd like to keep; I'd much prefer simply getting a failure indicator – such as NULL instead of a valid pointer – and coping with it in the calling code. If nothing else, I can save the existing documents before bailing.

4:

Using the above solution the application can abort if needed, or make a long jump to a recovery point, where the program can continue.

No, it can not because loss of human life, loss of other goods may be the consequence of simply `exit()`. The only who knows to handle the inability of `malloc()` is the caller or its parent of `malloc()`.

....

`exit()` is forbidden because it will left much devices in an unmaintend state, like switches open, signals set, pumps pumping, rockets firing, gates opened, filters passing, firing the 3. WWW up,

"Yes, the nuclear plant melted down and spewed radiation across three counties, all because we couldn't allocate enough memory for another week's worth of status logs. We could have simply returned NULL and let the caller free the previous week's logs, freeing up the space for the new data, but it was easier to just abort, leaving valves opened, temperatures and pressures unmonitored, that sort of thing. So for your own safety, would you mind not breathing until, oh, 2150? Thank you." :)

And all that because the programmer was to braindead to write a single `if`.

You really have to wonder. Do these people actually just blindly assume every file open works? Every file read or write works? Every network connection establishes, every packet transmits?

I suspect not; I suspect they do, in fact, check these things as a matter of habit. Memory allocation, though, invokes some weird sort of black

Re: A solution for the allocation failures problem

magic which cannot, for some reason, be dealt with in the same way.

I don't get it, and it sounds like you don't, either. Maybe one of 'em will explain it in terms that involve some sort of consistency: if you check one sort of resource acquisition as a matter of habit or design, why is another sort of resource acquisition treated so much differently?

.