

Re: Can I Trust Pointer Arithmetic In Re-Allocated Memory?

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-08/msg03848.html

- *From:* "Bill Reid" <hormelfree@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sun, 20 Aug 2006 01:18:31 GMT
-

Herbert Rosenau <os2guy@xxxxxxxxxxxxxxxx> wrote in message
news:wmzsGguTDN6N-pn2-ixTCUZdIJlc7@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

On Fri, 18 Aug 2006 23:27:03 UTC, "Bill Reid"
<hormelfree@xxxxxxxxxxxxxxxx> wrote:

Herbert Rosenau <os2guy@xxxxxxxxxxxxxxxx> wrote in message
news:wmzsGguTDN6N-pn2-ysKU5ddsUS8X@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

On Thu, 17 Aug 2006 23:54:41 UTC, "Bill Reid"
<hormelfree@xxxxxxxxxxxxxxxx> wrote:

I guess I got the last two conditions
conflated in my mind; it just

seemed

logical to me that if realloc() failed it would
free the previous

block.

It SEEMS like it should.

But it does not so because you may need to continue your
work with the
old block.

Sure, but right off-hand I can't think of single case where I would
actually want to do anything with "half a loaf". When memory
allocation fails in whole or in part, I just want to get out of there
as quickly as possible, maybe just break out of that particular
module operation, quite often just to exit the whole program.

Re: Can I Trust Pointer Arithmetic In Re-Allocated Memory?

... and leaving the database in an undefined state.

Maybe YOUR database. I started this whole thread while looking at a specific operation in one specific module. Each operation in the module downloads several raw data files from the net, parses out the data items, and writes the data to a database. Unless I get ALL of the data, ALL of the files downloaded properly in the correct format and with the correct data, NOTHING, none of the about 100,000 data items in this specific operation gets written to the database.

I specifically DESIGNED these download modules to work this way, even though the database can never be "undefined" (if data is written to the database, attempts to "over-write" it can't occur and don't matter anyway), out of force of habit writing the routines that analyze the data. In those modules, it is a complete waste of time to in any way analyze partial data, so if I can't download it all from the database, I just give up (except that's actually never happened, probably because I run that stuff at night with no other processes running).

Maybe you needs a
lot of single steps on the incloplete data to do to unwind any action
you've already done with.

Well, maybe YOU do. It's not like I'm not aware of how I could corrupt my results or my database, it's just that I design the various parts of the program to avoid these types of situations.

So the data is needed by your program. Most
programs are more complex in data handling as you currently aware of.

Possibly...and maybe not. I've seen some REALLY "complex"
programs in my time, and on a LOC basis alone (or any other basis)
I can't really classify what I do as "simple"...but I'm definitely not
multi-threading or some other things that might dramatically increase
the "complexity"...

So the best realloc can do for you to flag the error that it is out of
memory and leave data it arrived unchanged. Sometimes you gets a
chance to continue when you free()s some other data area.
Sometimes you'll split the aready occupied block into one or more
smaller ones, write older, smaller blocks out and reuse them.

Re: Can I Trust Pointer Arithmetic In Re-Allocated Memory?

Hmmmm, yeah, I was afraid of this type of response...yeah, there's about a million things I COULD do, I just DON'T...

You may have missed the part where I said I only care that my stuff WORKS, works flawlessly, and blindingly-fast for the amount of data being processed. I'm not TRYING to generate bugs, cuz I don't get paid to fix them, I just LOSE money...

There are lots of possibilities to continue even without – or with – asking the user.

I thought the general drill was to ask the user (in this case, I'm the only user) to close out some other applications. But I guess another general strategy for the "commercial market" is to let the thing sit there and grind away pointlessly; at least it keeps the room warm on a cold day...

Exit is seldom the choice in real programs.

Yeah, I think I'm getting your point here, it's just that it doesn't really have anything to do with "complexity", but something else...

Often you will have a solution for "out of memory" on a higher level.

Yeah, but I would suspect these would all slow you down to a crawl...for my purposes, I'll pass, but again, for the "commercial" market it might make sense...and since most software is written for the "commercial" market, maybe it DOES make "statistical" sense for realloc() to work that way...

exit() is good for ingenious programs but the real world is more complex.

Well, I don't generally break all the way out to exit(), except in the case of the database initialization stuff whenever I start up, but I probably could since each function tends to clean up after itself on error and return success or failure in some way...

So realloc gives you the chance for a real cleanup of any kind you may need. It is simply a bug to overwrite a data area you

Re: Can I Trust Pointer Arithmetic In Re-Allocated Memory?

have already allocated with a null pointer. You have to cleanup.

SOME people have to clean up in that fashion. I'm already clean,
I just have to close some open files, release some related memory,
I'm good...

I suspect that is true of the vast majority of programs out there.
So not automatically freeing the block seems to be a case of "the
needs of the few outweighing the needs of the many".

You not written a reals program using realloc. You will then quickly
revise your standpoint.

If I'm lucky I never WILL write a "real" program. I would hate
to be "corrupted"...

When you have no need for the old block you have to free()
that
yourself, else you should free() it. In any case you needs its
address.

In this imperfect world, I guess so...what a friggin' hassle...

As an orthogonal point, what horrible things
happen if you try
to free() a NULL pointer?

Nothing. free(NULL); works like a noop. Nothing occurs.
That is
guaranteed.

So I've kind of been wasting a little time with these types of
generalized pre-exit memory cleanup routines:

```
void free_time_series_mem(void) {  
    unsigned ts_idx;  
  
    for(ts_idx=0;ts_idx<TS_MAX;ts_idx++) {  
        if(time_series[ts_idx]!=NULL) {  
            free(time_series[ts_idx]);  
            time_series[ts_idx]=NULL;  
        }  
    }  
}
```

Re: Can I Trust Pointer Arithmetic In Re-Allocated Memory?

```
}  
  
num_series=0;  
}
```

Not only don't I need:
if(time_series[ts_idx]!=NULL)

What the hell is the point of:
time_series[ts_idx]=NULL;

That is needed to save yourself from accessing data you've given back to the system.

Again, you are correct sir! Well, sort of...

I mean, that's a good reason TO do it; it's just that I incorrectly described that particular function as a "pre-exit" memory clean-up routine when I quickly looked at it. If was truly "pre-exit", then there would be no need worry about an actual address in the pointer, the next call is to exit() (or return to main(), exit())...

The only point where you would savely NOT set the free()'d pointer to NULL is when the pointer and the data it points to is only alive only inside the function you calls free() AND one of the next statements is return, In any other case setting it to NULL will give you a clean "access of pointer to 0 instead of undefined behavior.

....but I realized after I posted that I also call it when "the user" leaves one particular module for another. Then I could POSSIBLY come back later and have the old pointer laying around to screw me up; it would specifically mess up the purpose of the function I use to populate the array on an "as-needed" basis as any module requires those particular structures:

```
TIME_SERIES *create_time_series(void) {  
    unsigned ts_idx;  
  
    for(ts_idx=0;ts_idx<TS_MAX;ts_idx++) {  
        if(time_series[ts_idx]==NULL) {  
            if((time_series[ts_idx]=  
                (TIME_SERIES *)malloc(sizeof(TIME_SERIES)))==NULL) {  
                #ifdef __CONSOLE__  
                    printf("\nNot enough memory to create time series");  
                #endif  
                #ifdef __WINGUI__
```

Re: Can I Trust Pointer Arithmetic In Re-Allocated Memory?

```
if(MessageDlg("Not enough memory to create time series",
mtConfirmation,TMsgDlgButtons()<<mbOK,0)==mrOk) ;
#endif
break;
}

time_series[ts_idx]->ts_vars=init_tsv;
time_series[ts_idx]->ds_vars=init_dsv;
num_series++;
break;
}
}

return time_series[ts_idx];
}
```

Of course, that's all screwed up in the first place because of the ridiculous cast of malloc()...

William Ernest Reid

.