

## Re: Memory leak when internal pointer passed out as parameter

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2007-04/msg00550.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2007-04/msg00550.html)

---

- *From:* Daniel Rudy <[spamthis@xxxxxxxxxxxxxx](mailto:spamthis@xxxxxxxxxxxxxx)>
  - *Date:* Thu, 05 Apr 2007 05:37:06 GMT
- 

At about the time of 4/4/2007 4:20 AM, santosh stated the following:

Daniel Rudy wrote:

At about the time of 4/3/2007 11:52 AM, Mike stated the following:

<snip>

Rational Purify checked the code, and reported memory leak on foo1 when we allocate memory. I assume that the compiler will allocate a new block of memory when foo1 returns. Then the memory allocated within foo1 will remain in the system heap forever. However I have no way to verify it.

The allocated memory will be returned to the system heap when the program exits, if there is a memory leak.

Nowhere does the C Standard guarantee this. Yes, modern memory protected operating systems do reclaim a program's allocated memory after the latter's termination, but C implementations exist on systems that're not as sophisticated or capable. A conforming C program must not make such assumptions.

I don't know what the C standard says about a lot of things, so thank you for pointing that out. I do know that on some machines, if you don't free the memory when the program exits, the host OS will die. Those are few and far between in this day and age.

Re: Memory leak when internal pointer passed out as parameter

You've also ignored the case of long-running processes like UNIX daemons. Memory leaks in such programs can cause the system memory to be slowly eaten up.

Excellent point. I had forgotten about that. The thing with Unix is that if the process keeps allocating memory until there is no more, then either the process will die by itself, or the host OS will forcibly kill it.

I have written a complete set of wrapper functions for malloc(3) and mmap(2) that solves the problem. The code is fully re-entrant since everything is a pointer. So when a thread terminates, everything that thread allocated also goes away. It keeps track of it using a open chained hash table of pointers returned by malloc(3) and mmap(2).

It's always better to explicitly free any memory when you're done with it.

I agree completely.

--

Daniel Rudy

Email address has been base64 encoded to reduce spam  
Decode email address using b64decode or uudecode -m

Why geeks like computers: look chat date touch grep make unzip  
strip view finger mount fcsk more fcsk yes spray umount sleep

.