Subject: Re: memory problems
Posted by JD Smith on Tue, 12 Mar 2002 19:14:58 GMT
View Forum Message <> Reply to Message

Dan Larson wrote:
>
> My question is probably more of a windows question than an
> IDL question. When I free memory with IDL, the memory
> doesn't seem to return to the operating system(WIN 98).
> In other words, over the course of an IDL session, the
> system resources slowly dwindle to nothing. Is this
> problem fixable?
>

Ahh yes, the age old memory question. This surprising behavior, believe
it or not, is a *feature*, not of IDL, but of your operating system.
Here's an enlightening explanation and defense of the behavior I
received from one of RSI's top software engineers last year.

JD

 ===
==================
  If I had a dollar for every time someone has blamed me for
malloc()s behavior, I'd be a wealthy guy. There are
some things to say about this:

- IDL uses malloc/free, so the behavior of memory staying in
  the process versus being released to the OS is really not
  an IDL issue. Some do it, and others don't.
- Malloc is not perfect, but no general memory allocator can be
  perfect. This is an NP complete problem, so one should be
  reasonable about their expectations.
- In particular, the idea that IDL should do it's own memory
  management instead of relying on the system allocator is
  not well considered. People who think this have not really
  considered the theoretical intractability of the issue.
- Malloc is the best choice on most systems, because it is the
  system default, and is therefore highly optimized and
debugged.
  Writing your own memory allocator is a fools errand
  (unless it's an assignment for a CS course). We've
experimented
  with using alternative allocators, and my professional opinion
  is that it's a losing game. This problem is as old as computer
  science, and if the malloc on your system is still not
perfect,
  perhaps that should tell you something?

- Even if you write your own allocator, and avoid all the subtle memory corrupting bugs, and even if you do beat malloc's performance:
    - Your allocator will have terrible behavior under some circumstances that you have yet to understand.
    - If you think you've already handled the point above, you have yet to understand the true complexity of the problem.
    - You're still using malloc alot, because lots of things from the system that you are relying on use it.
    - Having more than one allocator in a process makes everything worse. You always have malloc, even if you don't want it, so anything else is going to raise the complexity level.
- The GNU/Linux malloc() tries to return memory to the OS, while other mallocs don't. Whether this is good or not depends on your point of view --- it's not always a pure win.
    - Shrinking the process requires the memory allocator to work hard. This makes it slower than it would be otherwise. Maybe you don't care about speed? The GNU malloc() does a good job of balancing this, but it's far from perfect.
    - As Craig pointed out, if you have sufficient swap defined, it's really a non-issue. As cheap as disks are, there's little reason to be upset about this. Just increase the swap and get on with life.

My main points then:

- General memory allocation is theoretically hard. It's unfair to expect IDL to be able to change this.
- In general, IDL/malloc do a good job.
- You have to have enough memory on the system to solve the problem you're trying to solve. With memory as cheap as it is, this is less painful than it used to be.
- My experience is that the Solaris malloc() is different from the GNU (Linux) malloc(), but if you configure your machine well, it's also excellent. Understand your machine...

==