The poor man’s way of identifying memory leaks

devblogs.microsoft.com/oldnewthing/20050815-11

August 15, 2005

Raymond Chen

There is a variety of tools available for identifying resource leaks, but there’s one method that requires no tools or special compiler switches or support libraries: Just let the leak continue until the source becomes blatantly obvious.

Nightly automated stress testing is a regular part of any project. Some teams use screen savers as the trigger, others use a custom program, still others require manual launching of the stress test, but by whatever means, after you’ve gone home for the day, your computer connects to a central server and receives a set of tests that it runs all night.

One of the things that these overnight tests often turn up are memory leaks of one sort or another, identified by the stress team because your program’s resource usage has gone abnormally high. But how do you debug these failures? These machines aren’t running a special instrumented build with your leak detection tool, so you can’t use that.

Instead, you use the “target-rich environment” principle.

Suppose you’re leaking memory. After fifteen hours of continuous heavy usage, your program starts getting out-of-memory failures. You’re obviously leaking something, but what?

Think about it: If you are leaking something, then there are going to be a lot of them. Whereas things you aren’t leaking will be few in number. Therefore, if you grab something at random, it will most likely be a leaked object! In mathematical terms, suppose your program’s normal memory usage is 15 megabytes, but for some reason you’ve used up 1693 megabytes of dynamically-allocated memory. Since only 15 megabytes of that is normal memory usage, the other 1678 megabytes must be the leaked data. If you dump a random address from the heap, you have a greater-than-99% chance of dumping a leaked object.

So grab a dozen or so addresses at random and dump them. Odds are you’ll see the same data pattern over and over again. That’s your leak. If it’s a C++ object with virtual methods, dumping the vtable will quickly identify what type of object it is. If it’s a POD type, you can usually identify what it is by looking for string buffers or pointers to other data.

Your mileage may vary, but I've found it to be an enormously successful technique. Think of it as applied psychic powers.

Raymond Chen

Follow