acmqueue

Bugs and Bragging Rights

It's not always size that matters

Dear KV,

I've been dealing with a large program written in Java that seems to spend most of its time asking me to restart it because it has run out of memory. I'm not sure if this is an issue in the JVM (Java Virtual Machine) I'm using or in the program itself, but during these frequent restarts, I keep wondering why this program is so incredibly bloated. I would have thought Java's garbage collector would prevent programs from running out of memory, especially when my desktop has quite a lot of it. It seems that eight gigabytes just isn't enough to handle a modern IDE anymore.

Lack of RAM

Dear Lack,

Eight gigabytes?! Is that all you have? Are you writing me from the desert wasteland where PCs go to die? No one in his or her right mind runs a machine with less than 48 GB in our modern era, at least no one who wants to run certain, very special, pieces of Java code.

While I would love to spend several hundred words bashing Java—for, like all languages, it has many sins—the problem you're seeing is probably not related to a bug in the garbage collector. It has to do with bugs in the code you're running, and with a certain, fundamental bug in the human mind. I'll address both of these in turn.

The bug in the code is easy enough to describe. Any computer language that takes the management of memory out of the hands of the programmer and puts it into an automatic garbage-collection system has one fatal flaw: the programmer can easily prevent the garbage collector from doing its work. Any object that continues to have a reference cannot be garbage collected, and therefore freed back into the system's memory.

Sloppy programmers who do not free their references cause memory leaks. In systems with many objects (and almost everything in a Java program is an object) a few small leaks can lead to out-of-memory errors quite quickly. These memory leaks are hard to find. Sometimes they reside in the code you, yourself, are working on, but often they reside in libraries that your code depends on. Without access to the library code, the bugs are impossible to fix, and even with access to the source, who wants to spend their lives fixing memory leaks in other people's code? I certainly don't. Moore's law often protects fools and little children from these problems, because while frequency scaling has stopped, memory density continues to increase. Why bother trying to find that small leak in your code when your boss is screaming to ship the next version of whatever it is you're working on? "The system stayed up for a whole day, ship it!"

The second bug is far more pernicious. One thing you didn't ask was, "Why do we have a garbage collector in our system?" The reason we have a garbage collector is because some time in the past, someone—well, really, a group of someones—wanted to remedy another problem: programmers who couldn't manage their own memory. C++, another object-oriented language, also has lots of objects floating around when its programs execute. In C++, as we all know, objects must be created

or destroyed using new and delete. If they're not destroyed, then we have a memory leak. Not only must the programmer manage objects, but in C++, the programmer can also get direct access to the memory that underlies the object, which leads naughty programmers to touch things they ought not to. The C++ runtime doesn't really say, "Bad touch, call an adult," but that is what a segmentation fault really means. Depending on your point of view, garbage collection was promulgated either to free programmers from the tedium of managing memory by hand or to prevent them from doing naughty things.

The problem is that we traded one set of problems for another. Before garbage collection, we would forget to delete an object, or double delete it by mistake; and after garbage collection, we had to manage our references to objects, which, in all honesty, is the exact same problem as forgetting to delete an object. We traded pointers for references and are none the wiser for it.

Longtime readers of KV know that silver bullets never work, and that one has to be very careful about protecting programmers from themselves. A side effect of creating a garbage-collected language was that the overhead of having the virtual machine manage memory was too high for many workloads. The performance penalty has led to people building huge Java libraries that do not use garbage collection and in which the objects have to be managed manually, just as they did with languages such as C++. When one of your key features has such high overhead that your own users create huge frameworks that avoid that feature, something has gone terribly wrong.

The situation as it stands is this: with a C++ (or C) program, you're more likely to see segmentation faults and memory-smashing bugs than you are to see out-of-memory errors on a modern system with a lot of RAM. If you're running something written in Java, then you had better pony up the cash for all the memory sticks you can manage because you're going to need them.

KV

Dear KV,

I cannot help but notice that a lot of large systems call themselves "Operating Systems" when they really don't bear much resemblance to one. Has the definition of *operating system* changed to the point where any large piece of software can call itself one?

OS or not OS

Dear OS,

Certainly my definition of operating system has not changed to the point where any large piece of software can call itself one, but I have also spotted the trend. An old joke is that every program grows in size until it can be used to read e-mail, which, if you can believe Wikipedia, is attributed to Jamie Zawinski, based on an earlier joke by Greg Kuperberg, "Every program in development at MIT expands until it can read mail." Now, it seems, mail is not enough. Every large program expands until it gets "OS" appended to its name.

An operating system is a program that is used to give efficient access to an underlying piece of hardware, hopefully in a portable manner, though that is not a strict requirement. The purpose of the software is to provide a consistent set of APIs to programmers such that they do not need to rewrite low-level code every time they want to run their programs on a new computer model. That may not be what Oxford defines as an OS, but as it recently added *selfie* to its dictionary, I'm starting to think a bit less of the quality of their output, anyway.

I think the propensity for programmers to label their larger creations as operating systems comes from the need to secure bragging rights. Programmers never stop comparing their code with the code of their peers. The same can be seen even within actual operating-system projects. Everyone seems to want to (re)write the scheduler. Why? Because to many programmers, it's the most important piece of code in the system, and if they do a great job, and the scheduler runs really well, they'll give their peers a good dose of coder envy. Never mind that the scheduler really ought to be incredibly small, and very, very simple, but that's not the point. The point is the bragging rights one gets from having rewritten it, often for the umpteenth time.

None of this is meant to belittle those programmers or teams of programmers who have slaved long and hard to produce elegant pieces of complex code that make our lives better. If you look closely, though, you'll find that those pieces of code are appropriately named, and they don't need to tack on an OS to make them look bigger.

KV

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who currently lives in New York City.

© 2013 ACM 1542-7730/13/0900 \$10.00