acmqueue

Collecting Counters

Gathering statistics is important, but so is making them available to others.

Dear KV,

Over the past month I've been trying to figure out a problem that occurs on our systems when the network is under heavy load. After about two weeks I was able to narrow down the problem from "the network is broken" (a phrase that my coworkers use mostly to annoy me), to being something that is going wrong on the network interfaces in our systems.

You might think I'm going to ask you about networking problems, but I'm not. What I discovered in researching this problem was that the hardware is capable of recording a very large number of statistics and errors but that at least half of these are not accessible to anyone using the system, because they are not exposed to any layer above the driver. I was able to find this out because we're using an open source operating system, and I can read the driver source. There is definitely code to get these statistics from the device, but there is not code to make these available to anyone else. Why would anyone write code to gather statistics and not write the code to make them usable?

Driven by Drivers

Dear Driven,

The short answer to your question might simply be lack of time. The code you saw was the best intention of the driver writer at the time, but it was as far as he or she got before being forced to ship the code. Or, perhaps the writer is just a completely evil bastard who laughs himself to sleep at night knowing that thousands of people are unable to diagnose their network problems. I like to think it's the latter, because the former is far too pedestrian and uninteresting.

The real issue, though, has more to do with the interface between hardware and software engineers. Device drivers actually form an interesting sociological lens through which you can study the varied responses of two distinct social groups to their environments.

Hardware people, speaking quite broadly, are more constrained in the number of do-overs they get before their product fails. Creating a new version of a board, even something as simple as a network card, is an expensive and time-consuming process. The hardware folks worked out long ago that the best way to fix hardware, once they were no longer able to have the factories solder on green wires, was to leave it to the software people to deal with the problem. As a sort of back-door attempt at good will toward the software world they have added, in hardware, counters and statistics for every single thing that might go wrong with their hardware. Sometimes these counters are even documented, though the relations among them are rarely made clear enough to use without a deeper knowledge of the hardware you're working with.

The problem with software people is that given this plethora of counters, and a dearth of information as to which might or might not be useful in solving problems, they go in one of two directions: either they believe that they need to expose only the important counters, which means the ones that they understand; or they expose them all in a large block, which renders them difficult to use. Both of these approaches are the equivalent of the driver writer throwing up his or her hands

and screaming, “Enough already! Quit bugging me! Heal yourselves.” As you might have noticed, none of this is helpful to the systems integrator or the person who is trying to debug a problem.

I have a very basic rule for counter collection and display. If you can get at a counter, and getting at the counter doesn’t negatively impact the performance of the system, then you better be recording it somewhere. If you record something, you damn well better make it accessible to people who use your software, because not doing so is like giving someone an itch that they cannot scratch. Evil and fun at times, yes, but quite definitely bad software karma. When your code presents these counters to a user, they need to be grouped in some intelligent way (e.g., separating counters that are errors from counters that show non-errors). A concrete example in the case of a network driver would be to have all the counters for packet reception in one group, followed by all the counters for packet transmissions in another, and finally a set of counters for interrupts processed by the hardware.

One final thing to avoid is hiding counters within debug statements that require recompilation of the software in order to use them. It turns out the majority of people who use software are NOT software engineers, and do NOT want to have to rebuild a piece of software simply so they can find out what’s wrong with it. Segregating counters into one group that you want the user to be able to get at and another group that you believe they’ll never need is the proverbial road to hell. I can guarantee that each and every software developer who hides a counter in a debug block, not easily accessible to the user, will one day regret that choice. The counter that is in a debug block today will be the counter that you need the user to read back to you in a problem report, and if it’s not easy to get to, you will be totally screwed, and not in a nice way.

To sum up, if you can count it, do count it—and if you do count it, make sure that someone who is not you can get at it.

KV

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.

© 2010 ACM 1542-7730/10/0600 \$10.00