acmqueue

Avoiding Obsolescence

Overspecialization can be the kiss of death for sysadmins.

Dear KV,

What is the biggest threat to systems administrators? Not the technical threat (security, outages, etc.), but the biggest threat to systems administrators as a profession?

A Budding Sysadmin

Dear Budding,

Career questions are quite a bit more difficult than technical questions because they require me to look into the future, and much as I might enjoy doing that, well, my doctor keeps telling me to lay off the hard drugs, at least during working hours.

I think the question you're really asking is, "What might make me obsolete?" and that is a question that anyone in any field, but particularly in a fast-moving technical field, should ask. The biggest risks to a systems administrator, then, are overspecialization and allowing others to define your job too narrowly, and proving your worth.

When most people think of overspecialization they think of factory workers, who were, of course, made to specialize so that they could be better cogs in whatever means of production they were working on. The assembly-line worker who did one job for 10 years would have to be retrained when the machine that he or she worked with was changed, or, as was more likely, he or she was laid off and a cheaper, younger worker was brought in as a replacement. Believing oneself immune to these types of problems because of current income or current perceived social class could be a career-ending mistake.

Overspecialization is a risk to anyone in a fast-moving field, in which some highly valued skill might be automated next week. I could even argue that the more valuable the skill, the more likely it is to be automated, because your corporate masters are interested in reducing their overhead so they can make points with their boss and get a bigger bonus. I have always felt that it's a good idea to have a broad set of interests in your area and then to have more than one area in which you can specialize. That way, if your particular specialty is suddenly made obsolete, then you have something else, which is interesting to you, with which to pay your bills.

How can you tell if you're overspecialized? The best indication is if your job is to repeat, over and over, the same task, and that task is designed and dictated by someone else. If your job is to configure systems, but does not include deciding how they're configured, then you're definitely at risk. At some point the configuration part, the repetitive process, will be automated, and if you haven't graduated to configuration architect, then you're likely to find yourself looking for new work. The issue isn't necessarily related to level, but it is related to scope. If you don't have sufficient scope to be making decisions, then you are simply a tool that is used by others—and tools get replaced.

Avoiding overspecialization is not difficult, but it takes work on your part. Taking a broad interest in your entire discipline helps, as does reading books and attending conferences and tutorials. The

key is to choose your venues carefully so that you get as much exposure to areas with which you are unfamiliar. Looking at a list of books, conference sessions, or courses and picking the one that I know the least about is my favorite tactic. If you find yourself saying, “I have no use for X,” then you better make damned sure you know that subject well and do not dismiss it out of hand.

The flip side of overspecialization is when someone else is defining your role for you. All businesses, and in particular large businesses, want to place their workers into well-defined boxes so that they can more easily calculate wages and benefits. The people who draw these boxes rarely understand what systems administrators are or what they do.

The usual way in which these boxes are drawn is that the person doing the drawing does a Web search for some terms, many of which are already woefully out of date, and then draws a box and puts your name in it. If you complain about this kind of treatment and you’re lucky, they may even ask you to define your role, thereby doing their work for them. Let me recommend against defining your role as, “The god who makes it possible for you to get your work done.” No matter how true this might be, no one likes it when you say that sort of thing. At this point you need to think about what it is you do that is creative, thought based, and relevant to the company. It’s all too easy to box yourself in by defining your role as something repetitive, overspecialized, and easily replaced (see above).

A brief aside here about architects. Over the past 10 years it has become popular to give senior individual contributors the title of *architect*. I am sorry, but architects design buildings—not software, not systems, and not networks. I actually worked with a group in which receiving this title was a source of great humor, rather than pride, and that’s the kind of group that KV likes to work with. Usually I find it’s easy enough to co-opt the language of the management ladder. You’re a junior X, or a senior X, or a director of X, or a VP of X. If you want to point out that you’re not managing any people, then put in *technical*, as in senior technical network specialist. *Specialist* is another good generic word that says you have a defined role but one that’s not too tightly defined.

The final area I want to talk about is proving your worth to the organization that you work in. Any field—and systems administration falls into this category—that is responsible for the smooth, day-to-day running of an operation suffers from two significant handicaps right from the start.

The first handicap is that people expect things to “just work” without understanding what it takes to keep a set of systems running such that they appear to be always available. The only time people notice you or your group is when something breaks. Then suddenly they’re all up in arms and screaming about how they can’t get to the Web (where they were probably wasting time instead of working anyway), or their particular application is broken, and so forth. I am quite sure you’ve experienced this problem already, even as a budding systems administrator. Of course, randomly unplugging network cables, waiting for the phone to ring, and then plugging them back in might be an amusing way to make sure that people understand your worth, but even I can’t really recommend this course of action.

The second handicap suffered in the systems administration field is that most people in the business do not correctly perceive the worth of your work. The programmers and engineers often get kudos for making their code work and getting the project, whatever it is, out the door, but the role that is played by systems administrators in making sure that all those programmers are productive is rarely recognized, even by programmers themselves, who often think, “Who the hell are they?” and look down upon “supporting” groups such as the sysadmins. This kind of dynamic is akin to drivers of expensive cars complaining about the people who build and fix the roads. It takes a road to drive

a car, and you should be thankful for good roads. People who use your systems ought to be thankful when they receive good service, but usually they aren't.

Both of these handicaps need to be addressed in roughly the same way: through communication. While it's vitally important to communicate problems and outages, these should not be the only things that users learn about from the systems administration group. Whenever a new system comes on line or a new service is successfully rolled out, that fact should also be noted, and not in that horrifically saccharine-sweet way so often favored by the HR department. You're not celebrating little Annie's birthday, after all; you're informing your users that their work just got easier. A simple one-page e-mail, stating clearly what was changed and why it's better, is all that's necessary.

If you can remain interested and knowledgeable in a broad set of topics, help to define your own role, and communicate to your users just what it is you do and why it's important to their day-to-day lives, you will definitely lower your risk of becoming obsolete. And all this advice goes for just about everyone in a technical field. Now... what was my password again?

KV

Dear KV,

When I was in school I read a paper on how threads were considered dangerous, but that was before most CPUs were multicore. Now it seems that threads are required to get improved performance. I haven't seen anything that indicates that threaded programming is any less dangerous than it used to be, so would you still consider threaded programming to be dangerous?

Hanging by a Thread

Dear Threaded,

You might just as well have asked me if guns are still dangerous, because the answer is closely related: only if the gun is loaded, and definitely if the business end is pointed at you.

Threads and threaded programming are dangerous for the same reasons they always were: because most people do not properly comprehend asynchronous behavior, nor do they do a good job of thinking about systems in which two or more processes work independently.

The most dangerous people are those who think that simply by taking a single-threaded program and making it multithreaded, the program will somehow, as if by magic, get faster. Like all charlatans, these people should be put in a sack and hit with a stick—an idea I got from the comedian Darragh O'Brien, who wants to use that method for psychics, astrologers, and priests. I'm just adding one more group to his list.

Probably my favorite example of not thinking clearly about threaded programming was a group that wanted to speed up a system they had developed that included a client and a server component. The system was already deployed, but when it was scaled up to handle more clients, the server, which could handle only one request at a time, couldn't serve as many clients as was called for. The solution, of course, was to multithread the server, which the team dutifully did. A thread pool was created, and each thread handled a single request and sent back an answer to a client. The new server was deployed and more clients could now be served.

Just one thing was left out when the new server was multithreaded: the concept of a transaction identifier. In the original deployment, all of the requests were handled in a single-threaded manner, which meant that a reply to request N could not be processed before request N-1. Once the system

was multithreaded, however, it was possible for a single client to issue multiple requests and for the replies to return out of order. A transaction ID would have allowed the client to match its requests to the replies, but this was not considered; and when the server was not under peak load, no problems occurred. The testing of the system did not expose the server to a peak load, so the problem was not noticed until the system had been completely deployed.

Unhappily, the system in question was serving banking information, which meant that a small but nonzero number of users wound up seeing not their own account information but that of other customers, resulting in not just the embarrassment of the development team, but the shutting down of their project, and in several cases, firings. Alas, the firings were not out of cannons, which I always felt was a pity.

What you ought to notice about this story is that it has nothing to do with inter-thread locking, which is what most people think of when they're told that a piece of code is multithreaded. There is no magic method to make a large and complex system work, threaded or not. The system must be understood in total, and the side effects of possible error states must be well understood. Threaded programs and multicore processors don't make things more dangerous per se; they just increase the damage when you get it wrong.

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 1542-7730/10/0400 \$10.00