Saddle Up, Aspiring Code Jockeys

This month KV offers advice to readers considering careers as full-time coders. He suggests that new programmers learn everything they can by reading magazines, journals, and Web sites. And what better place to engage in such learning than this very column? It might be a while before we publish our deluxe, hardcover *Kode Vicious Compendium*, so until then we suggest you visit <http://www.acmqueue.com>, where we've archived all past KV columns for quick and easy reference.

Dear KV,

I am an IT consultant/contractor. I work mainly on networks (I'm a Cisco Certified Network Associate) and Microsoft operating systems (Microsoft Certified Systems Engineer). I have been doing this work for more than eight years. Unfortunately, it is starting to bore me.

My question is: How would I go about getting back into programming? I say "getting back into" because I have some experience. In high school I took two classes of programming in Applesoft BASIC (archaic, I know). I loved it, aced everything, and was the best programming student the teacher ever saw. This boosted my interest in computer science, which I pursued in college.

In college, I took classes in C++, Java, and Web development (HTML, XHTML, JavaScript). I did great and had fun. For various reasons, I ended up leaving college and becoming a network administrator, and for the past eight years have been doing this and that and everything else IT-related. But I haven't been programming. The extent of my programming work experience is with MS Excel macros (yuck!) and basic VB coding in Microsoft Access.

So how could I start becoming a programmer? Visual Studio 2005? Java? Eclipse? I enjoy self-learning and have found that achieving certifications gets my foot in the

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue coffee mug, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

door. Is there a particular suite that I could get certified in to get my newly desired career going?

Thanks in advance.

Jonesing for a Job

Dear Jonesing,

Wait, you've gotten so good at your job that you're bored? Why not take up golf, or painting? Perhaps download more "content" from the Internet! Why on earth would you want to give up a job that you know just to become a programmer?! Don't you know that programmers put in long hours to meet impossible deadlines, all to make a bunch of rich guys richer?

OK, if you've gotten this far, then I guess maybe I should actually answer your question. There are as many answers to your question, of course, as there are programmers. How you go about moving into a coding job depends on a few things. The first is, "What do you like to do?" There is no point in working hard to learn a language or a system if you don't enjoy it. You'll just wind up right back where you are now, wanting to do something else. Find the kinds of problems you like to solve, then see how they're solved today, and see if that's the kind of thing you like to do.

Since you say you already have a computer science background, I don't really see much reason to go for any special certifications. I have never seen the point of most certifications as they certify only that you can pass a test, not that you can think about problems, which is the more important skill. I also don't think you should worry about which set of tools you use just yet. There are other things to work through, like step 2.

The second step, and be thankful that there aren't 12 of them, is to pick a project. I can't learn anything unless I have a project around what it is I want to learn. Try to choose something you think you can actually do. "Write an operating system," though it's a fun goal and actually possible to do, is likely not the right place to start. Working on an open source project such as FreeBSD, Apache, or Open Office might be another way to get started. Find

something you need to use on a relatively frequent basis and try to extend or fix it. Most open source projects have long lists of open bugs. Pick a few of those and try to fix them and submit patches to the maintainers.

Finally, take every opportunity you can to learn about your new area. That doesn't mean spending lots of money—or conning your employer into spending lots of money—on classes and conferences. Of course, if you find a class in Hawaii and your employer is willing to send you, well, I can't argue with that but I can't call it learning either. Instead, regularly read the journals, magazines, and Web sites that cover your newfound area of expertise.

Once you think you have what it takes to start your new career, look at entry-level positions that will allow you to learn more. Be prepared to take a pay cut because moving from eight years of experience in network administration to entry-level programming isn't likely to be a big monetary win, but then you're doing this for the fun and challenge, right?

KV

Dear KV,

Because of performance needs, my team is reworking some old code to run multithreaded so that it can take advantage of the new multicore CPUs that are now shipping in high-end servers. We're estimating that it will take at least six months to break down our software such that it will be granular enough to run as multiple threads and to implement all the proper locking and critical sections. I came across an old presentation online, "Threads Considered Harmful," and was wondering what you thought of it. The paper was written long before we had multicore CPUs. At the time there were few commercial SMP (symmetric multiprocessing) machines, so perhaps it didn't make sense to go to all the bother of writing threaded code then, but now things are different. Have you heard of this paper? Do you think it is still valid?

Hanging by a...

Dear Hanging,

John Ousterhout's warning (Why Threads are a Bad Idea [for most purposes], 1996 Usenix Technical Conference) is as important today as it was when it was written, not because times and technology haven't changed, but because, alas, people haven't. Most people seem to decide to create multithreaded code for the reasons you state here—that is, to get a supposed performance boost from it. These same people never seem to bother to measure their code or see if it is even practical to run it in multiple threads. They just start slicing away at the code in the

vain hope that if they have enough threads, suddenly, as if by magic, their code will run faster.

Longtime readers of KV will know that I do not believe in magic bullets. Waving a wand labeled "threads" over your code is about as likely to make it run faster as is sacrificing a chicken. At least you can eat the chicken when you're done with it, which is more than I can say for your code. In actual fact, threading your code may make it run slower, because poorly written threaded code is often slower than poorly written nonthreaded code. The locking primitives required to get locking right are nontrivial and can, if improperly used, slow your code down, as it all blocks on the same lock or, even worse, can introduce subtle bugs.

Another problem with threaded code is that the tools used to debug it remain primitive. Although most debuggers now claim to handle threads properly, this is not always the case, and you really don't want to be debugging your debugger while debugging your code.

One last thing that people miss in their rush to thread their code is the support they get from libraries that they link against. If your program requires that the libraries it uses be multithreaded, then you may be in for a shock when you realize that some of them are not thread-safe. Using non-thread-safe libraries in your thread-safe program is going to cause you no end of trouble.

Given all of this, should you still continue to thread your code? Maybe. First you need to understand the trade-offs and see if the job the code does is amenable to being multithreaded. If the code has several components that can operate completely independently, then yes, multiple threads can be a boon. If, on the other hand, the components all need to access a small shared section of data all the time, then threads will get you nowhere. Your program will spend most of its time acquiring, freeing, and waiting on the locks that protect the shared data.

So, unless it's really a win, and you have thought about it a lot, I would try not to get hung up in threads.

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 has made San Francisco his home since 1990.

© 2006 ACM 1542-7730/06/1000 \$5.00