Tip 38: Use Valgrind to check for errors

Ben Klemens

16 December 2011

level: once you're used to the debugger
purpose: find allegedly impossible-to-find bugs

Fail fast. If something goes wrong, then you want the program to start banging pots and pans the moment it happens.

Here's how I usually start my little lecture to people about debugging technique: *you need to find the first point in the program where something looks wrong.* Good code and a good system will find that point for you.

C gets mixed scores on this. The compiler runs loads of consistency checks, and thus finds many errors before they even occur. If you are dyslexic or otherwise commit frequent spelling errors, a language that requires declared variables is essential; an implicit-declaration language will take a typo like `conut=15` and generate a new variable that has nothing to do with the `count` you meant to set.

On the other hand, C will let you allocate to the tenth element of a nine-element array, and then trundle along for a long time before you find out that there's garbage in element ten. [Some scripting languages will just auto-reallocate the array to accommodate your error, which is an enthusiastic failure to fail fast.]

Those memory mismanagement issues are a hassle, and so there are tools to confront them. Within these, Valgrind is a big winner. Get a copy via your package manager.

Valgrind runs a sort of virtual machine that keeps better tabs of what is allocated where than the real machine does, so it knows when you hit the tenth element in an array of nine items.

Once you have a program compiled (with debugging symbols included via GCC's `-g` flag, of course), run

```
valgrind your_program
```

If you have an error, Valgrind will give you two backtraces that look a lot like the backtraces your debugger gives you. The first is where the misuse was first detected, and the second is Valgrind's best guess as to where the memory you meant to write to would have been allocated. The errors are often subtle, but having the exact line to focus on goes a long way toward finding the bug. Valgrind is under active development—programmers like nothing better than writing programming tools—so I'm amused at how much more informative the reports have gotten over time, and only expect better in the future.

You can also start the debugger at the first error, by running via

```
valgrind --db-attach=yes your_program
```

With this sort of startup, you'll get a line asking if you want to run the debugger on every detected error, and then you can check the value of the implicated variables as usual. At which point we're back to having a program that fails on the first line where a problem is detected.

Valgrind also does memory leaks, via

```
valgrind --leak-check=full your_program
```

This is slower, so you might not want to run it every time, but maybe it won't make any noticeable difference for your situation. At the end, you'll have a backtrace for every leak.

I take any memory leak under maybe 100KB as ignorable noise (unless I expect that the code could someday be re-run in the center of a loop). We're not working on computers with 64kb of memory, so don't stress about every line.