Tip 13: Use a debugger

Ben Klemens

27 October 2011

level: absolutely basic
purpose: interact with your allegedly non-interactive program

Next time, I'll run a tip about debugging technique using GDB. But today's tip is short and brief:

Use a debugger, always.

I think some of you will find this to be not much of a tip, because who possibly wouldn't use a debugger? As a person who promotes C to people who typically are using Java or Python, I can tell you the number of people who try to write C without a debugger are myriad.

If there's a fault in Java or Python code, the machine throws up a backtrace immediately. Our coder tries C, commits a similar error, gets the entirely unhelpful `segmentation fault. Core dumped.` error, and gives up. There are more than enough C textbooks that relegate the debugger to the *other topics* segment, somewhere around Chapter 15, so it's understandable that so many people don't have the reflex of pulling up the debugger at the first sign of trouble.

[Why *Segmentation fault*, exactly? Because the computer allocates a segment of memory for your program, and you are touching memory outside that segment.]

About that *always* clause: there is virtually no cost to running a program under the debugger. I have never been able to perceive a difference in speed between running with the debugger and running without a net (and this is the sort of thing I write tests for when I can't focus on real work).

The debugger isn't just something to pull out when you really need the backtrace and variable states. It's great being able to pause anywhere, increase the verbosity level with a quick `print verbose++`, force out of a `for (int i=0; i< 10; i++)` loop via `print i = 100` and `continue`, or test a function by throwing a series of test inputs at it. The fans of interactive languages are right that interacting with your code improves the development process all the way along; they just never got to the debugging chapter in the C textbook, and so never realized that all of those interactive habits apply to C as well.

To do:

Get to know a debugger. I am a luddite, so I use GDB, but your IDE might have one built in. There are graphical front-ends to GDB with animal mascots; I never liked them well enough to switch, but they may fit your tastes perfectly.

Here are some simple things to try on any program you may have with at least one function beyond `main`. They are the absolute basics in debugging technique, so if you find that your debugging system somehow makes one of these steps difficult, then dump it and find another.

- pause your program at a certain point,
- get the current value of any variables that exist in that function;
- jump to a parent function and check variable values there;
- step past the point where you paused, one line of code at a time.