Timing and Profiling - Introduction*

Charles Severance
Kevin Dowd

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Perhaps getting your code to produce the right answers is enough. After all, if you only plan to use the program once in a while, or if it only takes a few minutes to run, execution time isn't going to matter that much. But it might not always be that way. Typically, people start taking interest in the runtime of their programs for two reasons:

- The workload has increased.
- They are considering a new machine.

It's clear why you might care about the performance of your program if the workload increases. Trying to cram 25 hours of computing time into a 24-hour day is an administrative nightmare. But why should people who are considering a new machine care about the runtime? After all, the new machine is presumably faster than the old one, so everything should take less time. The reason is that when people are evaluating new machines, they need a basis of comparison—a benchmark. People often use familiar programs as benchmarks. It makes sense: you want a benchmark to be representative of the kind of work you do, and nothing is more representative of the work you do than the work you do!

Benchmarking sounds easy enough, provided you have timing tools. And you already know the meaning of time.[1] You just want to be sure that what those tools are reporting is the same as what you think you're getting; especially if you have never used the tools before. To illustrate, imagine if someone took your watch and replaced it with another that expressed time in some funny units or three overlapping sets of hands. It would be very confusing; you might have a problem reading it at all. You would also be justifiably nervous about conducting your affairs by a watch you don't understand.

UNIX timing tools are like the six-handed watch, reporting three different kinds of time measurements. They aren't giving conflicting information — they just present more information than you can jam into a single number. Again, the trick is learning to read the watch. That's what the first part of this chapter is about. We'll investigate the different types of measurements that determine how a program is doing.

If you plan to tune a program, you need more than timing information. Where is time being spent — in a single loop, subroutine call overhead, or with memory problems? For tuners, the latter sections of this chapter discuss how to profile code at the procedural and statement levels. We also discuss what profiles mean and how they predict the approach you have to take when, and if, you decide to tweak the code for performance, and what your chances for success will be.

*Version 1.3: Aug 25, 2010 11:16 am -0500
†http://creativecommons.org/licenses/by/3.0/
[1] Time is money.