Performance tuning

Lecture “Performance profiling”
Profile buzz.c, improve its performance

Homework “Assembly language”
Make BigInt_add go faster.

Lecture “Dynamic memory management”
Make malloc/free go faster and use less space
(Problem: we don’t have the client!
Some clients benefit from coalescing, some don’t need it)
If we overtune for one client, we might cause problems in others.

The Ethics of Extreme Performance Tuning
Andrew W. Appel

Tune your violin (1600-2050)

Tune your radio (1910-2000)

Tune your car (1890-1990)

Tuning for horsepower might not coincide with tuning for economy or minimize pollution
Tune your program (1950-2050)

Name of the function | Name of the running program | Name of the binary executable | % of execution time spent in this function
--- | --- | --- | ---
70871 | 75.8807 | libc-2.17.so | buzz1
5732 | 20.8398 | buzz1 | buzz1
256 | 0.9345 | buzz1 | buzz1
46 | 0.3817 | buzz1 | buzz1
262 | 0.2727 | libc-2.17.so | buzz1
93 | 0.2654 | libc-2.17.so | buzz1
92 | 0.0364 | libc-2.17.so | buzz1
9 | 0.0327 | libc-2.17.so | buzz1
3 | 0.0291 | libc-2.17.so | buzz1
2 | 0.0109 | libc-2.17.so | buzz1
1 | 0.0036 | buzz1 | buzz1

Programming challenge

Implement a correct and fast integer cube-root function.

**Correct:** On any input (not just the "test harness"), it must have behavior indistinguishable from this reference implementation:

```c
#include <math.h>
#include "root.h"
int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

**Fast:** When connected to the "test harness" driver, the program should run as fast as possible.

This challenge was designed by Guy J. Jacobson '81 in 1995 when he was teaching COS 333 at Princeton University.

---

Fast integer cube roots

```c
#include <math.h>
#include "root.h"
int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

Performance measurement

(On a 1995 computer, much slower than today's)

```
testharness.o + slowroot.o:  20 seconds
```

```
testharness.o + noroot.o:       2 seconds
```

Note: noroot.c is really fast, but is not correct, that is, fails "on any input, it must have behavior indistinguishable from this reference implementation."

---

Challenge:

```c
#include "root.h"
int quickroot(int i) {
  return 0;
}
```

How to do it

```c
#include <math.h>
#include "root.h"
int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

---

Floating-point cube root from math.h

```
#include <math.h>
#include "root.h"
int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

How can ya beat the highly tuned cbrt function from the math library?

I dunno, use Newton's method?

But doesn't the cbrt function already use Newton's method?

Wait, I got it! cbrt calculates 64-bit precision, but we need only 32-bit precision, so Newton's method needs fewer iterations.
Newton’s method

Appel’s method

Am I being called from . . . ?

Am I being called from . . . ?

Performance measurement

General principle of extreme performance tuning

(On a 1995 computer, much slower than today’s)

testharness.o + slowroot.o: 20 seconds

testharness.o + noroot.o: 2 seconds

testharness.o + amazinglyfastroot.o: 0.0 seconds
Can I get away with this?

I didn’t turn in my program as a homework assignment
I didn’t sell my program to Boeing for use in passenger jets
All I did was publish a paper explaining how to do it . . .

Intensional Equality ;=) for Continuations, by Andrew W. Appel.

Sometime back in 2006 or so...

Let’s sell small diesel hatchbacks in the U.S.!
But boss, the pollution control equipment (selective catalytic reduction) is too expensive to fit into a small hatchback!
Well, go figure something out.

Sometime back in 2007 or so...

Hey boss, we’ve got it! We’ll use an NO$_x$ trap!
It uses a bit of extra fuel to burn off the pollutants.
Excellent! Ramp up production for the new model year!

Sometime back in 2008 or so...

Um, boss, we’ve got a problem. If we run the NO$_x$ trap all the time, it wears out faster, and it hurts fuel economy.
Be creative! Find an engineering solution! Quick, the cars will ship soon!

Emissions test harness

Let’s see... this is the USA’s measurement test harness. It must not pollute in the test harness. And on the road, it must get good gas mileage!

Hey Günter, I gotta idea!

General principle of extreme performance tuning

In the test harness
Run the \( \text{NO}_x \) trap
(uses more gas, wears out the \( \text{NO}_x \) trap)

Not in the test harness
Turn off the \( \text{NO}_x \) trap
(great gas mileage, but unfortunately, 40x more nitrous-oxide pollution)
Sometime back in 2008 or so...

Hey boss, problem solved!

But be sure to call it "cycle-tuning" in any e-mails about this stuff.

Excellent.

zyklusoptimiert = cycle-optimized

Driving around in cars with test equipment

Hey boss, our measurements show these Volkswagens are polluting a lot more than they're supposed to be!

Huh! Let's report it to the California emissions control board.

http://articles.sae.org/12610/

VW Is Said to Cheat on Diesel Emissions; U.S. to Order Big Recall

WASHINGTON — The Obama administration on Friday directed Volkswagen to recall nearly a half-million cars, saying the automaker illegally installed software in its diesel-powered cars to evade standards for reducing soot.
Aside: State DMV emissions testing

Traditional (since 1980s) DMV emissions testing

Real-life NJ DMV test harness

New style (in many states) DMV emissions testing for cars made since 1996

How the test harness works

Programming challenge

Write a program that cheats on this test:

Are you polluting?

Nope.

OK, cool.

Nope.

OK, cool.

Solution:

printf(“Nope.”);

Obviously trivial! Therefore we rely on law and ethics to prevent this cheating.
And now for something completely different

What if you didn’t cheat on purpose?

The Internet of Things

Manufacturer A sells a “thing” (wifi router, toaster, thermostat, baby monitor, coffee maker, fitbit, football helmet, ...) for $50,

. . . full of security vulnerabilities (buffer overruns, SQL injection, etc ... )

Manufacturer B pays their engineers to spend a few more days, be a bit more careful, sells the “thing” for $51.

The Internet of Things

Hack a million devices, gain a million DDOS nodes

Consumer can’t tell the difference, might as well buy the cheaper one

Server
Does carelessness pay?

Fixing the “IoT security problem” is an open problem, from a regulatory point of view.

From a software engineering ethics point of view:
Your bug may harm the entire Internet.

Don’t make and sell stupidly insecure devices.

Cat-and-mouse regarding the buffer overrun problem

Niklaus Wirth designs Pascal language, with supposedly ironclad array-bounds checking.

Robin Milner designs ML programming language, with provably secure type-checking.

Everything is still written in C . . .

Robert T. Morris, graduate student at Cornell, exploits buffer overruns in Internet hosts (sendmail, finger, rsh) to bring down the entire Internet.

. . . became the first person convicted under the then-new Computer Fraud and Abuse Act.

(400 hours community service. Now an MIT prof.)
1990s

Everything is still written in C . . .

Buffer overrun attacks proliferate like crazy

“Solution:”

Every time the OS “execvp”s a new process, randomize the address of the base of the stack.

That way, code-injection attacks can’t predict what address to jump to!

Buffer overrun with random stack-start

The nop-sled attack

“Solution:” Every time the OS “execvp”s a new process, randomize the address of the base of the stack.

That way, code-injection attacks can’t predict what address to jump to!

“Solution:” hardware permissions

“Solution:” In the virtual memory system, mark the stack region “no-execute.” (required inventing new hardware mechanism?)

This DOES protect against the “A” version of homework 5

(1) doesn’t protect against return-to-libc attacks (such as the “B” version of homework 5

(2) doesn’t protect against code injection into the heap (such as the “A” version of homework 5)

“Solution:” more hardware permissions

“Solution:” In the virtual memory system, mark the BSS region “no-execute.”

This DOES protect against the “A” version of homework 5

(1) There are still ways to defeat it

(2) Costs overhead, never much caught on

“Solution:” canary values

“Solution:” Check whether the canary has been overwritten, just before returning from the function.

This DOES protect against the “A” version of homework 5

This DOES protect against return-to-libc attacks

Stackguard detected an attack, execution terminated

BUT:

(1) Does not protect against return-to-libc attacks (such as the “B” version of homework 5

(2) Costs overhead, never much caught on
Heartbeat
Component of OpenSSL
Used across the Internet

http://xkcd.com/1354/

Bug in OpenSSL
If strlen() doesn’t match given length . . .
buffer overrun

HeartBleed
First Internet bug report with:
• catchy name,
• logo
• web site

Consequence:
Read up to 64 kilobytes from your OS address space, send it to attacker.
If those happen to contain crypto keys or other secret info, you’re hacked!

http://xkcd.com/1354/

Those protections don’t work against HeartBleed
Stack randomization: doesn’t protect.
Stack no-execute: doesn’t protect
BSS no-execute: doesn’t protect
Canary: doesn’t protect

Heartbleed is a buffer-overrun vulnerability, but it’s a “read-only” attack!
It’s not code-injection, it’s not return-to-libc.

“Solution:” adjust C with array-bounds checks
There have been a dozen or more language designs like this. None have ever caught on. The problem is, then it’s really not C any more.
(And what to do about malloc/free insecurities?)

“Solution:” Java, C#, etc.
Type-safe languages with array-bounds checking and garbage collection . . .

Actually, that is the solution.
Language choice as an ethical issue?

From a software engineering ethics point of view:

If you deliberately choose an unsafe programming language, there had better be a justified reason.

If you carelessly choose an unsafe programming language, then you’re being unethical.

MISC. EXTRA SLIDES


http://autoweek.com/article/vw-diesel-scandal/more-defeat-devices-audi-vehicles#ixzz4RyW47YNd

REPORT: CARB DISCOVERS MORE TECH DESIGNED TO DETECT EMISSIONS TESTING...