On Programming: Reflections of a Gray Haired Computer Scientist

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

September 10, 2012

Part I

Context

Context

- In Fall 2006 I attended a panel at CCS on cybercrime
- I was blown away by the effectiveness of cyber criminals
- And the enormous variety of techniques they use to attack systems
- I had an epiphany—*a sudden perception of essential nature*
- Existing software is dead (unfixably broken)
- New software is needed
- And thus the Ethos project was born
- ...to make applications **robust**—*resistent to attack*

What is wrong with existing software?

- Bloated: millions and millions of lines to do trivial things
- Feature heavy (swiss army knife)
- Weak (fragile) abstractions (gratuitous complexity)
- Does not compose well (gratuitous complexity)
- Flexible over semantics
- Backwards compatible (e.g., passwords)
- Ineffective/missing security mechanisms (e.g., naked networking)
- Poor layering

The power of layering

- Consider a security hole such as buffer overflow
- In C, need to do code reviews to find such problems
- Automated tools can help, but the problem is **undecidable**
- Can't even find them all, never mind automatically fix them
- In higher level languages, buffer overflow cannot occur
- (if the language implementation is correct)
- Thus (semantic) layering
  - Can ensure, by construction, that certain properties hold
  - Which are undecidable without layering

OS is the most fundamental layer

- Has an enormous impact on security
- Also the layer with the least well developed semantics
- Ethos goals
  - Create better (higher level) abstractions
  - Bake in semantics which are much harder to exploit
  - Compositional
- Semantics is everything

Ignorance and Confidence

To succeed in life, you need two things: ignorance and confidence.
–Mark Twain

- Ethos is a ridiculously large project
- It involves writing a substantial and complex piece of code (the OS kernel)
- When I started this project, I had never written a line of OS kernel code
- Ignorance in spades
- But I had a lot of confidence due to the many areas of CS I had worked in

What I knew

- Architecture
- Concurrency
- Parallel memory models
- The high level design of kernels
- File and storage systems
- Networking
- Authorization
- Authentication
- C
- Tool chain
- Development tools

What I didn't know

- Detailed kernel design
- Kernel coding experience
- x86 architecture
- *these are serious deficiencies*

What's hard about writing a kernel?

- Primitive environment: need to build everything from scratch
- Limited resources: need to keep running even when it bumps up against resource limits
- Deconstruct tools: e.g. linker
- Low level mechanisms: break them, its difficult to understand what is happening

De-risking kernel development

- Virtual Machine based: minimized need for device drivers
- Based on existing low-level kernel (Mini-OS)
  - Did not support processes
  - Extracted from Linux
  - Small 6,881 LoC
  - Working Xen interface
  - Contained most of the needed low-level design
- Simple monitor-based kernel
- Single processor kernel
- 32-bit (64-bit VM likely to have bugs at the time we started)
- Paravirtualized (avoid bugs)
- C-based kernel

Part II

How things worked out

Xen (the virtual machine)

- Poorly documented
- Look at the source code
- But now, 300 KLoC + Linux Kernel 10 MLoC
- Chisnal book helped
- Paravirtualization (vs. hardware virtual machine) a good idea since it was supported on every development platform
- 32-bit was a good idea
- but we are just now getting 64-bit to work
- porting exposes bugs in common code
- Biggest surprises (both positive):
  - Profiling and debugging of OS built in
  - Mini-OS

Mini-OS

- Stripping out code is harder than putting it in
- We needed to strip out a bunch of code for locking
- We seemed to have damaged Mini-OS's memory allocator
- And a few other things
- But these things seem to be fixed now
- We also rewrote parts of Mini-OS (e.g. storage allocators)
- Getting rid of ugly code

C

- C is machine-independent assembly language
- No packages
- No automatic garbage collection
- Weak type system
- No runtime
- Simple concurrency semantics
- C doesn't do much for you, but it doesn't get in the way either

Go

- Go is a programming language from Google
- Written by Rob Pike, Russ Cox, and Ken Thompson
- All operating system guys
- So it is properly layered
- (vs. programming language such as Scala built on top of Java)
- And supports systems programming
- It replaced Python as our user space target

Programming (the early days)

- Satya Popuri (master's student) worked on virtual memory (needed to support processes)
- Andrei Wartekin (undergrad) worked on the low-level parts to bring up processes
- After Andrei left, we found problems with something he had built, a customized file driver
- It was implemented in the Linux kernel and Ethos.
- But there were bugs in the early Xen version we were using
- so we wanted to use a later version of Xen
- This unfortunately broke the Linux kernel driver Andrei had built
- And also some low-level Ethos code
- We spent 9 months not making any progress due to this issue

I hate bugs

- The solution was simple, suggested by Andrew Trumbo
- Use Mini-OS networking, instead of the specialized driver
- It removed all the Linux kernel code and Xen version dependencies
- Reduced overall project complexity
- I got so mad
- That we wasted so much time over something that should have cost us no more than 2 weeks
- The reason was that I didn't have enough knowledge to reason about the issue
- I spent the next summer coding in the kernel, incrementally making things better.
- I determined never to depend on others for moving forward

Part III

What works

What works

- KISS (Keep It Simple)
- Project moto: *Simple enough for a professor to understand*
- Automatic testing
- Many simple tests
- When a test fails, easy to understand what is broken
- Make small changes, run the test script, scan the test results
- Add test for new features, newly discovered bugs

What works (cont'd)

RCS	Revision control system (track changes, make branches for development)
TeXnotes	project notes before use
Refactoring	we've moved much code out of the kernel for dual use
ASSERTS	simple consistency statements useful for development
Bug tracking	puts pressure to make progress on issues, ensure bugs not forgotten about.
Coding Style	Makes it look like everything coded by me
Coding Rules	Rules to ensure OS properly built
Code Reviews	making code better, understanding what people have done

Part IV

Attitude

Attitude

- Tools are important
- But the most important thing is people
- The most important thing about people is attitude
- A good systems person always wants to know more
- And spends considerable time acquiring knowledge
- Ethos depends on such people
- Their contribution far exceeds those without this attitude

Programming (the middle days)

- Andrei Wartekin learned all about OSs before I started working with him
- He had worked at Microsoft as an intern in the OS group after sophomore year
- Pat Gavlin was another undergraduate
- He ported Go, worked on 64-bit Ethos, optimized Ethos, and RPC
- Mike Petullo is a Ph.D. student
- Good news: finally, a student who is not going to graduate in a year
- Bad news: He's here for under 3 years (less than 1 year left)

Programming (the middle days)

- Mike started work on Ethos the summer before joining UIC
- From Afganistan
- He worked on the paging system
- To support PAE, and enabled us to move to modern Unix distributions
- (every time we changed software bases, I was either forced to do so or it was done by others)
- He has since worked on Networking, Authentication, Authorization, Types, Go, and
- Too many other projects to count

Other major contributors

- Ameet Kotian: Network Stack
- Francesco Costa: Graphics system
- Xu Zhang: kernel stability (networking, interrupts, processes) and porting improvements
- Also contributing:
  - Wenyuan Fei: distributed authentication
  - Siming Chen: memory allocator
  - Yaohua Li: port of network code

10,000 hours

- Kernel hacking is knowledge intensive
- Expertise take 10,000 hours
- Think about that ...
- Many heavy contributors started in high school
- Or well before they started on the project
- The more you know
- the less time you waste
- the easier it is to fill in what's missing and
- then you can look up details

Lifetime of a programmer

- Programmer lifetime (NSF study)

  > *Six years after finishing college, 57 percent of computer science graduates are working as programmers; at 15 years the figure drops to 34 percent, and at 20 years—when most are still only in their early 40s—it is down to 19 percent.*

- Programming is about
  1. How well you think
  2. What you know (and how well you apply)
  3. What procedures you follow
  4. What wisdom you achieve
- If most programs are bad, we aren't harvesting 2–4.

The earlier in the software lifecycle, the better the returns

- You should do everything you can to avoid time spent debugging.
- If you are spending a lot of time debugging, you don't know what you are doing.
- Design, design, design
- Make it simple
- Make sure it is absolutely clear in your mind

Hone your skills

- Understand how to do things
- Write programs to test
- Think about how to do it better
- Then implement for real
- Good programs are the result from a thorough understanding
- If you realize the program is not good, re-factor
- *One of my most productive days was throwing away 1000 lines of code.* Ken Thompson

Program defensively

- Humility is essential to good programming
- Your going to make mistakes
- Try to be conservative
- Whenever possible, take small steps
- Simplify, simplify, simplify

Part V

Project future

Current state

- We are finishing up Ethos V 1.0
- Porting to 64-bit
- And thinking about user space
- We started working on graphics
- Something minimal that can be used by 90% of programs
- And EI, a scripting language
- which supports Ethos types

Applications

- Creating a set of tools and libraries for doing programming is our key challenge
- This is a far larger job than creating an OS kernel
- The things which we build must of course conform to Ethos principles

Future Musings

- We'll re-implement Ethos once we learn enough from the current implementation
- (make the mistakes in the prototype).
- It will use hardware virtualization
- It might be based on a Microkernel such as L4
- It might be implemented in a higher level language such as Go
- It will definitely run on ARM and x86

Part VI

Conclusions

Conclusions

- I didn't pick Ethos, it picked me
- I just had to do it, once I realized what has been needed
- Its been a tremendous journey, an experience I would not give up for anything
- And I've gotten to know some amazing people
- My advice to you: reach for the stars
- Even if you fail, its better to have done so