The Role of Language Technology in Trustworthy Operating Systems

Gernot Heiser
NICTA and University of New South Wales
Sydney, Australia
LANGUAGES AS TOOLS
seL4 Aim: Formal Verification

- Design & Specify
- Formal Model
- Proof
- Proof
- High-Performance C implementation
- Safety Theorem
Clash of Mentalities

Formal Methods Practitioner

Kernel Hacker
Standard Kernel Design

Kernel Hacker View

Design & Specify

High-Performance C implementation

Prototype on Real Hardware

White-board

Formal Model

Proof

Safety Theorem

Step 2
Formal Design

Formal Methods View

Design in Theorem Prover

Design & Specify

Formal Model

Proof

High-Performance C implementation

Step 2

Safety Theorem

Proof
Haskell as Lingua Franca

Design & Specify

Formal Model

Haskell Model

Proof

High-Performance C implementation

Safety Theorem

Iterative Design

Inspired by existing code

Productivity
Haskell Model as Executable Prototype

- Haskell Model
  - Built with Standard Toolchain
- QEMU
  - Execute
  - Syscall
  - Return
- Test App Binary
- Custom Interface

Built with Standard Toolchain
Haskell Model as Intermediate Refinement

Nice

Ugly

Abstract Model

Executable Model

C Implementation

Formalization

Proof

Safety Theorem

Haskell Spec

Haskell Model

Impl.

Low-level Data-struct. & Algorithms

Proof

Translation
LANGUAGES FOR TRUSTWORTHINESS
Trustworthiness

These are full-system properties!
Real-World Trustworthiness

Prerequisites: Isolation, communication and legacy support!
Two Approaches to Isolation

MMU-enforced protection

- Kernel:
  - controls HW
  - IPC for communication
  - Address spaces for isolation

Type Safety

- Language runtime
  - controls HW
  - manages memory
  - …?
Representative Systems

**MMU-protected: L4**

- State-of-the-art microkernels
  - for 18 years
- IPC performance still unbeaten
  - lots of published data
- Widely deployed:
  - OKL4 on 1.2 billion devices

**Type-safe: Singularity**

- Most complete recent system
- Some published performance
  - Surprisingly no L4 comparison!

---

Hardware

Kernel

- IPC
- Addr-Spaces
- H/W control

Trusted Userland

App

App

Language runtime support

- Stuff
- Garbage Collector
- H/W control

TCB

Hardware
Cost of Isolation

**MMU-enforced protection**
- Context switching
  - thread context
  - protection context
  - IPC semantics
- Other execution at full speed

*Large per-switch overhead*

**Type Safety**
- Run-time bounds checks
- Garbage collection
- Switching is just function call

*Small continuous overhead*
Performance: Intra-Domain Execution

L4: Direct overhead: **zero**
- Any code runs at native speed

Indirect overheads: TLB reloads
- Dependent on program size
- Mostly low, can be 10s of %

Singularity
Bounds-check overhead: 4.7%

Type-safe system incurs same O/H when supporting legacy software!

Does not include:
- Cost of garbage collector
- Optimization opportunity

---

Hardware

Kernel
- IPC
- Addr-Spaces
- H/W control

Trusted Userland

App

App

Language runtime support
- Stuff
- Garbage Collector
- H/W control

Hardware

TCB
Performance: Cross-Domain IPC

**L4 WINNER**

AMD-64 @ 1.6 GHz:
- 230 cycles for 0–24 bytes
  
  [http://www.l4ka.org/126.php]

**Singularity**

AMD-64 @ 2.0 GHz
- 803 cycles for 1 byte
- 933 cycles for 4 bytes
  
  [Hunt & al, EuroSys’07]
**TCB Size**

**L4**
- Kernel code: \(\sim 10\) kLOC
- seL4 on ARM: 9 kLOC

Userland: depends
- seL4: as small as 1.5 kLOC for real-world systems

**Singularity**
- No published data, but
  - "probably bigger than L4"
- Functionality, e.g. no legacy support

Formally verified \(\Rightarrow\) size no longer matters!

---

**Hardware**
- IPC
- Addr-Spaces
- H/W control

**Kernel**
- Trusted Userland

**Language runtime support**
- Stuff
- Garbage Collector
- H/W control

**App**

---

©2011 Gernot Heiser NICTA
Summary

MMU-enforced protection

• Faster
• Probably smaller TCB
• Functionally-correct

Type Safety

• ???

Hardware

Kernel

IPC
Addr-Spaces
H/W control

Trusted Userland

App
App

Language runtime support

Stuff
Garbage Collector
H/W control

HW control

Hardware

App
App

TCB

©2011 Gernot Heiser NICTA

PLOS’11 Keynote
Does Memory-Safety Help Safety or Security?

- It’s better than nothing
  - … but on its own it doesn’t help much in proving safety
- Type safety doesn’t stop:
  - your garbage collector being buggy
    - possibly destroying type safety
  - your scheduler being buggy
    - leading to unsafe thread execution order
    - leaks information through scheduling decisions
  - your IPC primitive having unsafe side effects
    - affecting or leaking data to third threads
The seL4 Experience

Confidentiality
0 py
By construction

Availability

Integrity
1 py
4 months

Abstract Model

Executable Model

C Implementation

Proof

Confidentiality

WCET Analysis

2 py, 1 year
Mostly for tools

Mostly for tools

Confidentiality

Proof

Confidentiality

Proof

Proof

Confidentiality

4 months

2 py, 1 year

Proof

Proof

Proof
seL4: Next 12 Months

Confidentiality

Availability

Integrity

Abstract Model

Proof

Non-Interference

Executable Model

Proof

C Implementation

Proof

Timing-Channel Mitigation

WCET Analysis

©2011 Gernot Heiser NICTA
seL4 for Safety and Security

Safety
- Timeliness
- Termination
- Integrity

Security
- Availability
- Functional Correctness
- Memory Safety

Confident. / Info Flow
seL4 Summary

• First (and still only) general-purpose OS kernel with
  – Functional correctness proof
  – Integrity proof
  – Complete, sound WCET analysis

• Yet, performance at par with any comparable system!
  – 200 cycle IPC on ARM11

• Likely to be the first kernel with
  – Confidentiality proof
  – Non-interference proof
  – Sound covert-channel mitigation
Let’s Stop Kidding Ourselves

… and the people who trust our expertise!

- By implying that type-safe = safe
  - Type-safe ≪ safe; type-safe ≪ secure
  - … and there’s no easy way to get there
- By implying that a system where all code is managed is practicable
  - Nothing will be used if it can’t provide legacy support
  - Test: Can it run Linux?

Trustworthiness is best achieved through functional correctness!
- Excellent basis for showing integrity and confidentiality
Our View of Implementation Languages

Your choice! (… but managed is clearly better)

DSL/C
C + asm
Formal Verification

managed
C + asm
Your choice!

Managed runtime
GC
Other Stuff
Linux
App

Trusted Userland

Kernel
IPC
Addr-Spaces
H/W control

Hardware

mailto:gernot@nicta.com.au

Google: “ertos”