The seL4 Microkernel: Provable Security for the Real World

Gernot Heiser
UNSW and seL4 Foundation
gernot@unsw.edu.au
Cyberattacks Are Everywhere

Report Shows Cyber Attacks on Cloud Services Have Doubled

'Most serious cyberattack of the Ukraine war': Tens of thousands modems crippled

Increasingly used by
• organised crime
• state actors

Ransomware attack on Swissport causes delay at Zurich Airport
Core Problem: Complexity

Software-engineering rule of thumb:
• 1–5 bugs per 1,000 lines of quality code

Dependability, Security

System Complexity

Bluetooth protocol stack: 100s kSLOC

Linux/Windows kernel: 10s MSLOC
Standard Approach: Patch-and-Pray

A losing proposition!

ML accelerates!

10M SLOC
10k unknown bugs

10M SLOC
10k-1 unknown bugs

10M SLOC
10k-1 unknown bugs
1 known bug

Maintain

Hack

Patch
How Can We Do Better?
Step 1: Minimise Trusted Computing Base

Modularisation: Separate functions
- operating-system services
- applications

Microkernel enforces isolation
- kernel code reduced to minimum
- mediates hardware resources
- performance critical

Kernel Mode
User Mode

- File System
- Networking
- Device Driver
- Trusted Component
- Untrusted Component

Virtual Machine
- Linux App
- Linux

Hardware
Step 2: Mathematical Proof

- First OS with proof of implementation correctness
- Only verified OS with fine-grained protection (capabilities)
- Only protected-mode RTOS with sound and compete WCET analysis
- World’s fastest microkernel

Open Source!

Present limitations
- initialisation code not verified
- MMU, caches modelled abstractly
- Multicore not yet verified
How Do Refinement Proofs Work?

“Forward simulation”: Prove state correspondence of abstract and concrete levels.

Prove (interactive theorem proving)
Kinds of properties proved for functional correctness

- Behaviour is fully captured by abstract model
- Kernel never fails, behaviour is always well-defined
  - Assertions never fail
  - Will never dereference null pointer
  - Will never access array out of bounds
  - Cannot be subverted by mis-formed input
  - ...

Can prove further properties on abstract level!
Verification Assumptions

1. Hardware behaves as expected
   - Formalised hardware-software contract (ISA)
   - Hardware implementation free of bugs, Trojans, …

2. Spec matches expectations
   - Can only prove “security” if specify what “security” means
   - Spec may not be what we think it is

3. Proof checker is correct
   - Isabel/HOL checking core that validates proofs against logic

With binary verification do not need to trust the C compiler!
Minimise Trusted Computing Base

Modularisation: Separate components
- operating-system services
- applications

Verification makes isolation bullet-proof!

Microkernel enforces isolation
- kernel code reduced to minimum
- mediates hardware resources
- performance critical

Verification makes isolation bullet-proof!

Kernel Mode
User Mode

File System
Networking
Device Driver
Trusted Component
Untrusted Component

Hardware
Virtual Machine
Linux App
Linux

© 2023 Gernot Heiser – CC BY 4.0
Capabilities: Fine-Grained Protection

- Enforce *least privilege*
- No communication unless explicitly authorised!
Real-Time: Capabilities for Time

- **Critical: Control loop**
  - Sensor readings
  - Runs every 100 ms for ≤ 25 ms
  - Budget = 25,000 µs
  - Period = 100,000 µs
  - Utilisation = 25%

- **Untrusted: NW driver**
  - NW interrupts
  - Runs frequently for ≤ 2 µs
  - Must preempt control loop!
  - Budget = 2 µs
  - Period = 3 µs
  - Utilisation = 67%

-Time as first-class resource: capabilities provide bounded access to CPU
Worst-Case Execution-Time Analysis

- Only protected-mode real-time OS with complete, sound worst-case execution-time analysis
- High-assurance by connecting to correctness proofs

- Note: Armv6 only
  - insufficient timing info for modern processors
  - Open RISC-V implementations should enable it again!
The Benchmark for Performance

<table>
<thead>
<tr>
<th></th>
<th>seL4</th>
<th>Fiasco.OC</th>
<th>L4Re</th>
<th>Zircon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (cycles)</td>
<td>986</td>
<td>2717</td>
<td>8157</td>
<td></td>
</tr>
<tr>
<td>Mandatory HW cost* (cycles)</td>
<td>790</td>
<td>790</td>
<td>790</td>
<td></td>
</tr>
<tr>
<td>Overhead absolute (cycles)</td>
<td>196</td>
<td>1972</td>
<td>7367</td>
<td></td>
</tr>
<tr>
<td>Overhead relative</td>
<td>25%</td>
<td>240%</td>
<td>930%</td>
<td></td>
</tr>
</tbody>
</table>

*: The Cost of SYCALL + 2 × SWAPGS + SYSRET = 395 cycles, times 2 for round-trip

Source:
Zeyu Mi, Dingji Li, Zihan Yang, Xinran Wang, Haibo Chen: “SkyBridge: Fast and Secure Inter-Process Communication for Microkernels”, EuroSys, April 2019

Smaller is better

World’s fastest microkernel!
Made For Real-World Use

- Autonomous vehicles
- Satellites
- Secure communication device
  - In use in multiple defence forces
- Laot: Critical infrastructure protection
“World’s Most Secure Drone”

We brought a hackable quadcopter with defenses built on our HACMS program to @defcon #AerospaceVillage. As program manager @raymondrichards reports, many attempts to breakthrough were made but none were successful. Formal methods FTW!
Using seL4 in Cyberphysical Systems
seL4 Principles

Proper microkernel:
- Minimal
- Provides policy-free mechanisms only
- Single access-control mechanism: Capabilities

Result: High barrier to uptake!

Security:
- Suitable base for security-critical systems
- Provably correct and secure

Performance:
- Security is no excuse for poor performance!
- Don’t pay for what you don’t use

Anti-Principles:
- Hardware abstraction
- Prevent foot guns
- Usability

User-level issue!

The microkernel is the assembly language of operating systems!
Taming seL4: The seL4 Core Platform

- Simple, single-threaded event-driven
- Minimal abstractions
- Thin wrapper of seL4
- Encourage “correct” use of seL4 primitives
- Aimed at IoT/cyberphysical

May be a virtual machine
seL4CP Verification

Conditions apply

CapDL spec

Proof-generating translation

System Spec

Compiler/Linker

system.elf

init.o

PD1.c PD2.c libsel4cp.c

sel4 spec

sel4CP spec

Push-button proof
seL4CP-based Highly Modular OS

- Hardware
  - seL4 Core Platform
  - Microkernel
  - Networking
    - IP Stack
    - Copier
    - Mux
    - NIC Driver
  - File System
Example: Networking

Strict separation of concerns: Large number of extremely simple components

Diagram:

- Client IP Stack
- Val
- Copy
- ARP
- Rx MUX
- Tx MUX
- Driver
- NIC
- IRQ
- Client
Comparison to Linux (i.MX8)

**Linux:**
- NW driver: 4k lines
- NW system total: 1M lines

**seL4 design:**
- NW driver: 700 lines
- MUX: 400 lines
- Copier: 200 lines
- IP stack: much simpler, client library
- shared NW system total: < 2,000 lines

Written by second-year student!

Performance?
Evaluation Setup

- External load generator
- Client echoes packets

2 context switches per packet

Client

IP Stack ↔ NIC Driver

NIC

10 context switches per packet

Client ↔ Driver

IP Stack

Tx MUX

Copy

ARP

Rx MUX

Copy

NIC

Client
Achieved Performance: i.MX8

- Gigabit Ethernet
- Single core

Simplicity wins!

• Smaller is better
• Bigger is better

Throughput (Mb/s)

CPU Utilization (%)

Load (Mb/s)

Linux Xput CPU
seL4 Xput CPU
Highly Modular OS: Timeline

- Q4’23: First release of OS
  - with point-of-sale reference system
- Q2’24: Release of matured, documented, OS & PoS system
  - including performance analysis
- Q4’24: Verification of key components of OS
Security is no excuse for bad performance!