seL4 Present and Future
@GernotHeiser & Team
NICTA and UNSW Australia
What is seL4?

seL4: The latest (and most advanced) member of the L4 microkernel family – 20 years of history and experience
What is seL4?

seL4: The world’s most (only?) secure OS kernel – provably!

GPLed
2014-07-29
seL4: Mathematical Proof of Security

Confidentiality
No buffer overflow, un-init vars, NUL-deref, stack smashing, code injection, ROP…

Integrity
Proof

Availability
Isolation properties World First!

Abstract Model
Functional correctness World First!

C Implementation
Translation correctness World First!

Binary code
Timeliness World First!

Exclusions (at present):
• Initialisation
• Privileged state & caches
• Multicore
• Covert timing channels

No need to trust C compiler!
Real upper bounds for IRQ latencies

©2015 Gernot Heiser, NICTA
What seL4 is NOT: An Operating System

All device drivers, OS services are usermode processes
What’s Different to Other L4 Microkernels?

Design for isolation: No memory allocation in the kernel

- Resources fully delegated, allows autonomous operation
- Strong isolation, No shared kernel resources
High-Assurance System on seL4

DARPA HACMS Program:
• Provable vehicle safety
• “Red Team” must not be able to divert vehicle

Boeing Unmanned Little Bird (AH-6) Deployment Vehicle

SMACCMcopter Research Vehicle
SMACCM Research Vehicle Architecture

CONTROL BOARD
- Software:
  - Control
  - Mission Plan
  - Sensor Filtering
  - Monitor
  - CAN bus
  - eChronos
- Hardware:
  - Sensors
  - Radio Modem
  - Microcontroller
  - Speed Controller
  - Radio Rxer

MISSION BOARD
- Software:
  - Command & Control Task
  - Image Processing (Payload)
  - Ethernet Driver
  - Unverified Linux Kernel
  - seL4
- Hardware:
  - ARM A15 processor
  - Unverified C&C Radio
  - COTS Network Camera

CAN Bus
Current NICTA Work on seL4

• High-performance multicore support
  – Release ETA: few months (ARM, x86)

• Full support for virtualisation extensions
  – Release ETA: few months (ARM, x86)

• 64-bit support
  – Release ETA: few month (x86), ??? (ARM64)

• Mechanisms for eliminating timing channels
  – ETA: 2015 (ARM and x86)

• Temporal isolation and mixed-criticality scheduling
  – ETA: 2015 (ARM and x86)

• Hardware failure resilience (DMR/TMR on multicore)
  – ETA: 2015 (ARM and x86)
What Else Is Cooking?

• **Aim:** Cost reduction by automation and abstraction
  – Present seL4 cost: $400/SLOC, high-assurance, high-performance
  – Other “high” assurance: $1,000/SLOC, no proof, poor performance
  – Low assurance (Pistachio): $200/SLOC, no proof, high performance

• **Device driver synthesis**
  – Synthesise driver code from hardware and OS interface specs
  – works already for simple devices

• **Code and proof co-generation**
  – High-level spec in DSL describes logic, generate C code and proofs
  – File systems as case study

• **Type- and memory-safe high-level languages**
  – Do verification cheaper in HLL semantics
  – Requires verified HLL run-time and compilers
seL4 Ecosystem: Kernel Development

Private branch

Commitment to verify! (Eventually…)

release

seL4 experimental

Proofs

seL4 master

Public, Github, GPLv2
How Can YOU Contribute?

• Libraries presently extremely rudimentary
  – POSIX! …
• Platform ports
  – Especially popular ARM boards: Tegra, RK3188, Beaglebone, …
• Drivers!!!!!!
  – Very few available ATM
• Network stacks and file systems
  – Presently have lwIP, incomplete functionality
• Tools
  – Have component system (CAmkES), glue generators
• Languages
  – Core C++ support just released, lacks std template lib
  – Haskell presently in progress (with Galois) – stay tuned
  – Python would be awesome!
Why NOT Use seL4?

• Very rudimentary programming environment!
  – Fair enough
  – You can help to fix this!
• I like unsafe/insecure systems!
  – Ok, go shoot yourself
• I like the thrill of danger!
  – Like getting sued for building a critical system on outdated technology
• Actually, I want to use seL4!
  – Right answer ;-)