2022 T2 Week 10 Part 2
seL4 in the Real World & seL4 Research at TS@UNSW
@GernotHeiser
Copyright Notice

These slides are distributed under the Creative Commons Attribution 4.0 International (CC BY 4.0) License

• You are free:
  • to share—to copy, distribute and transmit the work
  • to remix—to adapt the work

• under the following conditions:
  • Attribution: You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:
    “Courtesy of Gernot Heiser, UNSW Sydney”

The complete license text can be found at http://creativecommons.org/licenses/by/4.0/legalcode
Today’s Lecture

• seL4 in the real world
  • HACMS & incremental cyber-retrofit
  • Usability: CAmkES & seL4 Core Platform

• seL4-related research at UNSW Trustworthy Systems
  • sDDF: High-performance driver framework
  • Pancake: Verifying device drivers
  • Verifying the seL4CP
  • Secure multi-server OS
  • Time protection: Verified timing-channel prevention
seL4 in the Real World
DARPA HACMS

Retrofit existing system!

Autonomous trucks

Unmanned Little Bird (ULB)

Develop technology

Off-the-shelf Drone airframe

GVR-Bot
ULB Architecture

- Mission Computer
  - Ground Station Link
  - Flight Computer
    - Sensors
    - Motors
  - Network
    - GPS
    - Camera
  - Motors
- seL4

Incremental Cyber Retrofit

**Original Mission Computer**

**Trusted**
- Mission Manager
- Crypto
- Local NW
- Ground Stn Link
- Linux

**Trusted**
- Mission Manager
- Crypto
- Local NW
- Ground Stn Link
- Linux
- Camera
- GPS
- Virt-Mach Monitor

**Trusted**
- GS Lk
- Miss Mgr
- Crypto
- GPS
- Linux
- VMM

**Camera**
**Local NW**
**Linux**
**VMM**
Incremental Cyber Retrofit
Incremental Cyber Retrofit

Original Mission Computer

[Trusted]
- Mission Manager
- Crypto
- Camera
- Local NW
- Ground Stn Link
- Linux

[Klein et al, CACM, Oct’18]

Cyber-secure Mission Computer

[Trusted]
- Crypto
- Mission Mngr
- Local NW
- Comms
- GPS
- Linux
- VMM

Camera
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!
HACMS Outcomes

• Demonstrated real-world suitability of seL4 and formal methods
  • Reversal of bad vibes from over-promiseing and under-delivering
  • Major re-think in US defence
• Dis-proved “security must be designed in from the start”
• Led to follow-on funding for seL4 and deployment in the field
Usability

CAmkES and the seL4 Core Platform
Issue: seL4 Objects are Low-Level

>50 kernel objects for trivial program!
Simple But Non-Trivial System
Recommended Framework: CAmkES

Higher-level abstractions of low-level seL4 constructs
CAmkES Framework

- Conditions apply
- CapDL: Low-level access rights
- Architecture specification

However:
- Forces use of kernel build system
- Fully static & hard to extend
- Significant overheads

- Good for assurance
- Bad for usability & functionality
New Framework: seL4 Core Platform

Small OS for IoT, cyber-physical and other embedded use

• Leverage seL4-enforced isolation for strong security/safety
• Retain seL4’s superior performance
• ”Correct” use of seL4 mechanisms by default
• Ease development and deployment
  • SDK, integrate with build system of your choice
• Retain near-minimal trusted computing base (TCB)
• Be amenable to formal verification of the TCB
seL4CP Abstractions

- Thin wrapper of seL4 abstractions
- Encourage “correct” use of seL4

Protected Procedure Call (PPC)

Communication Channel (CC)

Notification

Protection Domain (PD)

Memory Region (MR)
seL4CP Status

• Developed by Breakaway
• Used in products (Laot, AArch64-based)
• Virtualisation support in progress
• Platform and ISA ports in progress (x64, RV64)
• Dynamic features prototype:
  • fault handlers
  • start/stop protection domains
  • re-initialise protection domains
  • empty protection domains (for late app loading)
seL4-Related Research in TS

High-Performance I/O and I/O Virtualisation
I/O Architecture

- 1 syscall per I/O
- no fault containment

- many syscalls per I/O
- good fault containment

Can we get this to perform?
Device Sharing (aka I/O Virtualisation)

Can we get this to perform?

Legacy support requires device sharing
Advanced I/O Architecture

**Challenge:**
- Performance

**Opportunities:**
- Re-think design
- Simplify driver model
- Simplify IP stack
- Reduce (avoid?) locking

**Enable verification?**
Driver Model

Driver model:
- Single-threaded
- Event-driven
- Simple!

Can we get this to perform?
Transport Layer

- Lock-free bounded queues
- Single producer, single consumer
- Similar to ring buffers used by NICs

- Mostly moves pointers between rings

Sole purpose: Hardware abstraction!
Transport Architecture Scales

- Components can be on separate cores
- Driver, MUX close to minimal critical sections
- Should scale well without locks!
Preliminary Evaluation: Setup

- Echo packets
- Extra copy to simulate Posix overhead

- Apply variable load
- Measure throughput

- Echo packets
- Extra copy to simulate Posix overhead

Load Generator

IP Stack

Driver

Client

Load Generator

IP Stack

Driver

Client

Echo packets

Extra copy to simulate Posix overhead

Apply variable load

Measure throughput

Load Generator

IP Stack

Driver

Client

Echo packets

Extra copy to simulate Posix overhead

Load Generator

IP Stack

Driver

Client

Extra copy to simulate Posix overhead

Load Generator

IP Stack

Driver

Client
Preliminary Evaluation: Performance
sDDF: Next Steps

Native web server
Build & evaluate
Safe re-use of legacy drivers
Optionally Linux driver in VM
Verify?

Extend for storage, USB

Client ↔ IP ↔ MUX ↔ Driver

Device

Safe re-use of legacy drivers
seL4-Related Research in TS

Verifying Device Drivers?
Remember: Verification Cost in Context

- **Revolution!**
  - L4 Pistachio: $100–150
  - Fast!
  - Slow!
  - Green Hills INTEGRITY: $1000

Cost ($/SLOC)

Assurance
Driver Dilemma

seL4 is one-off, justifies cost

Drivers are commodity, must be cheap!

Drivers are low-level, need C-like language

High seL4 verification costs partially due to C language

Better language would reduce cost

Idea:
1. Simplify drivers
2. Design verification-friendly systems language
3. Automate (part of) verification

sDDF driver model!

Verified compiler

• Well-defined semantics
• Memory-safe
CakeML: Verified Implementation of ML

- Mature functional language
- Large and active ecosystem of developers and users
- Code generation from abstract specs
- Managed ⇒ not suitable for systems code
- Used for verified application code

Re-use framework for new systems language: Pancake

https://cakeml.org
CakeML:
- functional language
- type & memory safe
- managed (garbage collector)
- high-level, abstract machine
- verified run time
- verified compiler
- mature system
- active ecosystem

Approach:
- re-use lower part of CakeML compiler stack
- pathway to verified Pancake compiler
- Retain mature framework/ecosystem

In progress
seL4-Related Research in TS
Verifying the seL4 Core Platform
seL4CP Verification

In progress: verified translation tool

Done already

seL4CP implementation

Exploration – using automated techniques (SMT solvers)
seL4-Related Research in TS

Secure Multi-Server OS
Recap: Secure Operating Systems

Secure OS: [Jaeger: OS Security]
Access enforcement satisfies the reference monitor concept

Enforces mandatory protection:
- non-bypassable
- tamperproof
- verifiable

Permission: relation over labels
**Secure, General-Purpose OS**

**Aim:** General-purpose OS that provably enforces a security policy

**Requires:**
- mandatory policy enforcement
- policy diversity
- minimal TCB
- low-overhead enforcement
seL4-Related Research in TS

Time Protection: Verified Prevention of Microarchitectural Timing Channels
Refresh: Microarchitectural Timing Channels

Contention for shared hardware resources affects execution speed, leading to timing channels.
OS Must Enforce *Time Protection*

 Preventing interference is core duty of the OS!

- *Memory protection* is well established
- *Time protection* is completely absent
Time Protection: No Sharing of HW State

What are the OS mechanisms?
Spatial Partitioning: Cache Colouring

System permanently coloured

Partitions restricted to coloured memory

Initial process

RAM
Spatial Partitioning: Cache Colouring

- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory ⇒ colouring userland colours kernel memory

Shared kernel image
Channel Through Kernel Code

*Raw channel*

Channel matrix: Conditional probability of observing output signal (time) given input signal (system-call number)
Colouring the Kernel

Remaining shared kernel data:
- Scheduler queue array & bitmap
- Few pointers to current thread state

Each partition has own kernel image

Kernel clone!
Spatial Partitioning: Cache Colouring

- Partitions get frame pools of disjoint colours
- seL4: userland supplies kernel memory ⇒ colouring userland colours kernel memory
- Per-partition kernel image to colour kernel

Must ensure deterministic access to remaining shared kernel state!
Channel Through Kernel Code

- **Raw channel**
- **Channel with cloned kernel**

The diagrams show the number of LLC misses for different seL4 system call counts. The x-axis represents the seL4 system call count, and the y-axis represents the LLC misses. The color gradient from purple to yellow indicates the frequency of LLC misses.
Temporal Partitioning: Flush on Switch

Must remove any history dependence!

2. Switch user context
3. Flush on-core state

6. Reprogram timer
7. return
**seL4**

D-Cache Channel

- Raw channel
- Channel with flushing

![Graph showing D-Cache Channel performance](image)

- Output (cycles)
- Input (sets)

<table>
<thead>
<tr>
<th>Raw channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output (cycles)</td>
</tr>
<tr>
<td>6500</td>
</tr>
<tr>
<td>6000</td>
</tr>
<tr>
<td>5500</td>
</tr>
<tr>
<td>5000</td>
</tr>
<tr>
<td>4500</td>
</tr>
<tr>
<td>4000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Channel with flushing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output (cycles)</td>
</tr>
<tr>
<td>7900</td>
</tr>
<tr>
<td>7850</td>
</tr>
<tr>
<td>7800</td>
</tr>
<tr>
<td>7750</td>
</tr>
<tr>
<td>7700</td>
</tr>
<tr>
<td>7650</td>
</tr>
</tbody>
</table>
Flush-Time Channel

Raw channel
Temporal Partitioning: Flush on Switch

1. $T_0 = \text{current\_time}()$
2. Switch user context
3. Flush on-core state
4. Touch all shared data needed for return
5. while ($T_0 + \text{WCET} < \text{current\_time}())$
6. Reprogram timer
7. return

Must remove any history dependence!

Latency depends on prior execution!

Time padding to remove dependency

Ensure deterministic execution

Latency depends on prior execution!
Flush-Time Channel

- Raw channel
- Channel with deterministic flushing
Performance Impact of Colouring

Splash-2 benchmarks on Arm A9

- Overhead mostly low
- Not evaluated is cost of not using super pages [Ge et al., EuroSys’19]

Architecture | x86  | Arm  |
---|---|---|
Mean slowdown | 3.4%  | 1.1%  |

<table>
<thead>
<tr>
<th>Arch</th>
<th>seL4 clone</th>
<th>Linux fork+exec</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>79 µs</td>
<td>257 µs</td>
</tr>
<tr>
<td>Arm</td>
<td>608 µs</td>
<td>4,300 µs</td>
</tr>
</tbody>
</table>
A New HW/SW Contract

For all shared microarchitectural resources:
1. Resource must be spatially partitionable or flushable
2. Concurrently shared resources must be spatially partitioned
3. Resource accessed solely by virtual address must be flushed and not concurrently accessed
4. Mechanisms must be sufficiently specified for OS to partition or reset
5. Mechanisms must be constant time, or of specified, bounded latency
6. Desirable: OS should know if resettable state is derived from data, instructions, data addresses or instruction addresses

aISA: augmented ISA

Cannot share HW threads across security domains!

[Ge et al., APSys’18]
Time Protection: On-Going Work

Assumes sane (non-existent) hardware

Include TP mechanisms in RISC-V ISA

Prove: no leakage

Verify efficacy

Fix hardware

Develop usable system model

Make usable

Integrate with temporal integrity (MCS)

Make complete

Validated on ETH Zurich RISC-V processor (Ariane) [Wistoff, DAC’21]
Real-World Use
Courtesy Boeing, DARPA
Thank you!

To the dedicated AOS students for their interest and dedication
To the world-class Trustworthy Systems team for making all possible

Please remember to do the myExperience survey
There’ll also be a more detailed one we’ll invite you to fill in