The Formally Verified seL4 Microkernel
High-Assurance Foundation for MCS

Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser
• RTCSA Keynote, Aug’20
https://trustworthy.systems
What Is Needed For Mixed-Criticality?

During a review process, ca Aug’17:

- [Gernot:] Temporal isolation is *necessary* for mixed criticality systems.
- [Reviewer:] Wrong, temporal isolation is *sufficient*.
What Is a Mixed-Criticality System?

“A mixed-critical system [...] supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform.” [Barhorst’09]

Criticality of a component is defined by the impact of failure:
• loss of life
• injury
• inconvenience

Certification of critical component must not depend on behaviour of less critical components
⇒ must prevent any interference by less critical components!
Preventing Interference – The OS’s Job

Operating System

High criticality

 Modify data

Affect timing

Low criticality

We need an OS that can guarantee the absence of interference!
seL4: Provable Isolation
What is seL4?

The world’s *first* operating-system kernel with *provable* security enforcement

The world’s *only* protected-mode OS with complete, sound timeliness analysis

World’s most advanced mixed-criticality OS

The world’s *fastest* general-purpose microkernel, designed for *real-world* use

Open Source
A Microkernel is not an OS

Device drivers, file systems, crypto, power management, virtual-machine monitor are all usermode processes

Microkernel = context-switching engine
Any system call is invoking a capability:
\[
\text{err} = \text{method}(\text{cap}, \text{args});
\]

**Capability = Access Token:**
Prima-facie evidence of privilege

**Capabilities provide:**
- Fine-grained access control
- Reasoning about information flow

**Eg. thread, address space**

**Eg. read, write, send, execute…**
Proved Spatial Isolation

Confidentiality

Integrity

Availability

Abstract Model

Functional correctness: C code only behaves as specified

Model enforces isolation

Translation validation: Binary retains C-code semantics

Limitations (work in progress):

- Kernel initialisation not yet verified
- MMU & caches modelled abstractly
- Multicore version not yet verified
- Timing channels not ruled out

Translation validation:

Binary retains C-code semantics

C Implementation

Binary code

RTCSA Keynote, Aug'20
Binary Code Verification

C Source \rightarrow\text{Formalised C}

\text{Formal C Semantics} \rightarrow\text{Rewrite Rules}

Compiler \rightarrow\text{Graph Language}

Proof? \rightarrow\text{Proof}

Graph Language \rightarrow\text{SMT Solver}

Symbol Tables \rightarrow\text{De-compiler}

Binary Code \rightarrow\text{Formalised Binary}

Target of functional correctness proof
Isolation by Architecture
Issue: Capabilities are Low-Level

>50 capabilities for trivial program!
Simple But Non-Trivial System
Component Middleware: CAmkES

Higher-level abstractions of low-level seL4 constructs

Component

Comp A

RPC

Connector

Comp B

Comp C

Shared memory

Semaphore

Interface
Trivial System in CAmkES
HACMS UAV Architecture

Security enforcement: Linux only sees encrypted data

Uncritical/untrusted, contained

Radio Driver

CAN Driver

Data Link

Crypto

Wifi

Camera

Linux
Enforcing the Architecture

Architecture specification language

Low-level access rights

Radio Driver
Data Link
Crypto
Uncritical/untrusted, contained
Linux
WiFi
Camera

Thread Object
CSpace
CNode
VSpace

Send
Receive

EP

Low-level access rights

Compiler/Linker

Conditions apply

glue.c
driver.c
VMM.c

binary
Military-Strength Security

DARPA HACMS: Retrofit existing system!

Unmanned Little Bird (ULB)

Secure Comms Dongle

Autonomous trucks

Cross-Domain Desktop Compositor

DARPA HACMS: Retrofit existing system!
Temporal Isolation: WCET Analysis
High-Assurance WCET Analysis

Program binary → Control Flow Graph → Micro-architecture model → Analysis tool

Loop bounds → Integer linear equations → ILP solver → WCET

Proved at C level, transferred to binary through translation-validation toolchain.
Temporal Isolation: Controlling Time
Mixed Criticality: Critical + Untrusted

NW driver must preempt control loop
- … to avoid packet loss
- Driver must run at high prio
- Driver must be trusted not to monopolise CPU

Runs every 100 ms for few milliseconds
Sensor readings

Critical: Control loop

Untrusted: NW driver

NW interrupts

Runs frequently but for short time (order of µs)
MCS Challenge: Sharing

Critical

Vehicle control must see consistent state

Vehicle Control → Shared Data → Navigation

Updates

Less critical
Sharing: Delegation to Resource Server

Control $P_1$

Navig. $P_2$

Communication endpoint (port)

Server $P_S$

Single-threaded, guarantees atomicity

Who pays for server time?

Implements immediate priority ceiling protocol (IPCP) if $P_S \geq \max(P_1, P_2)$
Solution: Time Capabilities

Classical thread attributes
- Priority
- Time slice

New thread attributes
- Priority
- Scheduling context capability

Scheduling context object
- T: period
- C: budget (≤ T)

Limits CPU access – sporadic server

Not runnable if null

Capability for time

Enables reasoning about time and temporal isolation for mixed-criticality systems

C = 2
T = 3
MCS with Scheduling Contexts

- **Control loop**: P = low
- **NW driver**: P = high

Sensor readings

- Runs every 100 ms for few milliseconds
- Runs frequently but for short time (order of µs)

- **NW interrupts**

<table>
<thead>
<tr>
<th>Context</th>
<th>Period C</th>
<th>Total T</th>
<th>Utilisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NW driver</td>
<td>25,000</td>
<td>100,000</td>
<td>25%</td>
</tr>
<tr>
<td>Control loop</td>
<td>2</td>
<td>3</td>
<td>67%</td>
</tr>
</tbody>
</table>
Shared Server Time Charged to Client

Client is charged for server’s time

Running

Client

Running

Server runs on client’s scheduling context

Timeout exception to deal with budget exhaustion
seL4 MCS Support

• **Time as a first-class resource:**
  - Enforcement of delegatable time budgets
  - Suitable for formal reasoning
  - Verification to be completed this year

• **Status:**
  - Functional correctness of MCS extensions presently being verified for Arm and RISC-V

• **To Do:**
  - Proving scheduler properties
  - Formal framework for reasoning about timeliness of applications
Thank You!