seL4: Formal verification of an OS kernel

Alejandro Gómez-Londoño

15 February 2017

Introduction

- Software systems are everywhere
- Every large enough code base is assumed to have bugs
- The OS is a fundamental component of most systems
- Operating systems tend to be huge and complex software projects

Hence:

- We are fucked

The seL4 microkernel

It is a third-generation microkernel that builds on the strengths of the L4 microkernel architecture, such as small size, high performance, and policy freedom, and extends it with a built-in capability model. [1]

[1] *SSRG @ Data61 - seL4 project page*

What is a microkernel

Core design principle:

> *A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations, would prevent the implementation of the system's required functionality.*[2]

[2] Liedtke, Jochen (December 1995). "On μ-Kernel Construction"

seL4

Features:

- Thread abstraction
- Inter process communication (IPC)
- Capabilities for authorization

"Missing" features:

- Drivers
- File systems
- Network stack
- Firewall
- More drivers!

Design choices

- An small kernel
- Limited preemption points
- External memory management
- Restrictions over the implementation language

Uses

- Embedded systems
- As a hypervisor
- As full fledged OS

What did we prove?

The binary code of the seL4 microkernel correctly implements the behaviour described in its abstract specification and nothing more. Furthermore, the specification and the seL4 binary satisfy the classic security properties called integrity and confidentiality.[3]

- Functional correctness
- Data cannot be changed without permission (Integrity)
- Data cannot be inferred without permission* (Confidentiality)

[3] SSRG @ Data61 - l4.verified project page

What we assume?

- Some assembly code
- Hardware
- Boot code
- Virtual memory
- DMA

What the proof implies

Absence of:

- Buffer overflows
- Memory leaks
- Pointer errors
- Undefined behaviours

Plus:

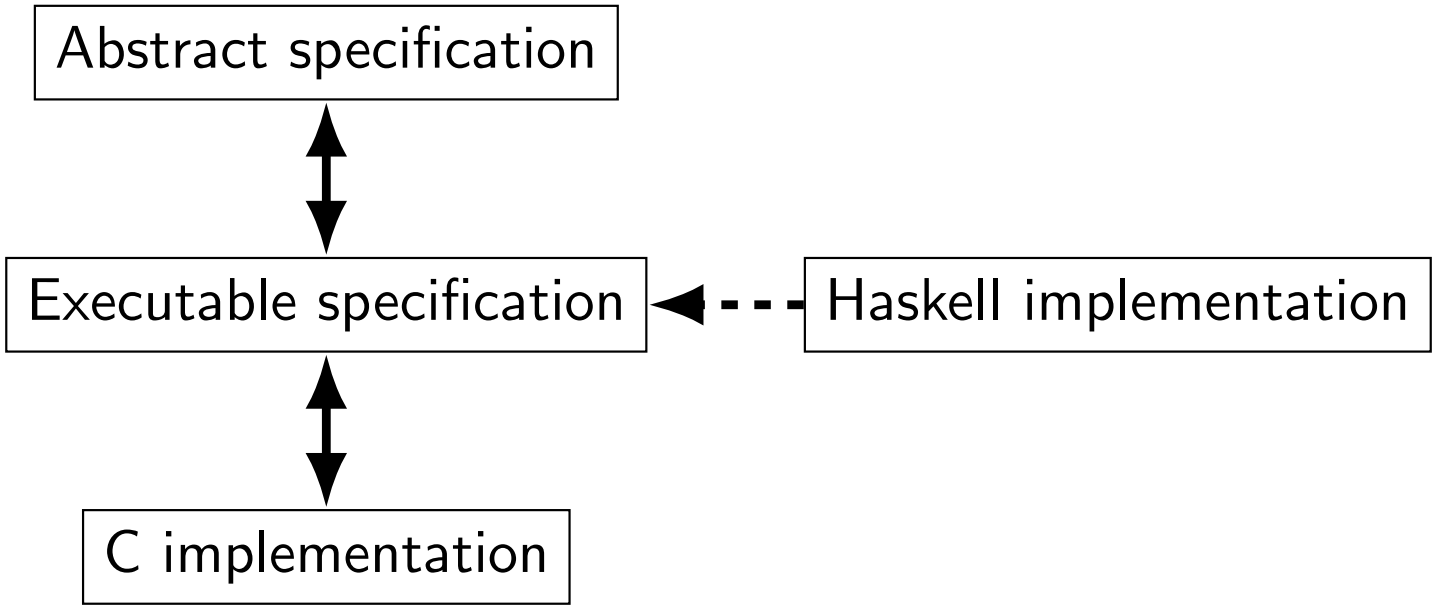
- Termination
- Non-failure

The proof

Key techniques:

- Formal refinement
- Forward simulation
- Hoare logic

Specifications



Refinement

A refinement proof establishes a correspondence between a high-level (abstract) and a low-level (concrete, or refined) representation of a system.

The correspondence established by the refinement proof ensures that all Hoare logic properties of the abstract model also hold for the refined model. [4]

[4] seL4: Formal verification of an OS kernel. Gerwin Klein et al., ACM Symposium on Operating Systems Principles 2009

Hoare logic


```
{P} f {Q}
```

Where:

```
P :: s → Bool
f :: State s r
Q :: r → s → Bool
```

Example:

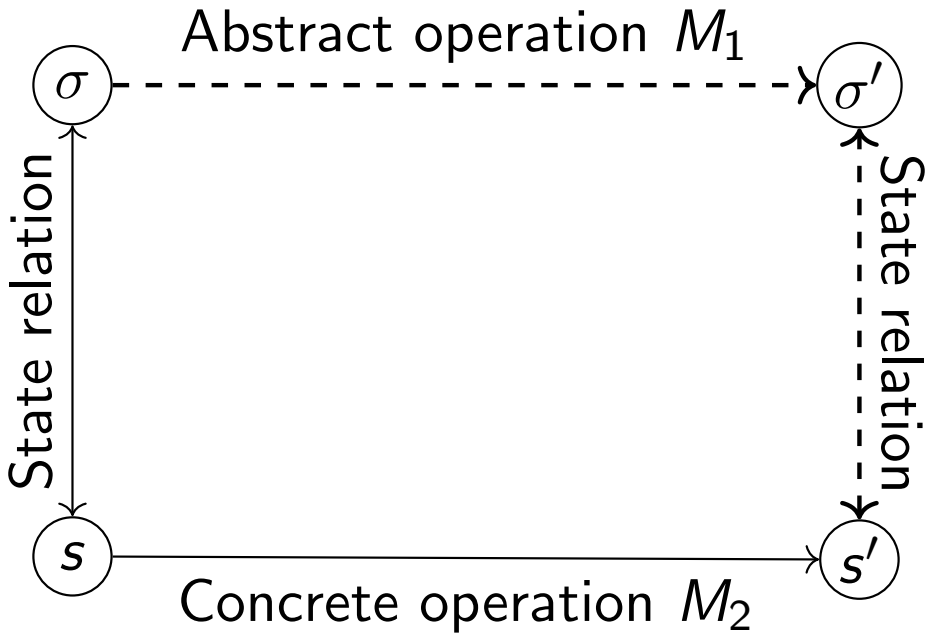
```
{ ⊤ } a = 5 { a ≡ 5 }

{ b ≡ 1 } a = b + 1 { a ≡ 2 }
```

Forward simulation

To show that a concrete state machine M_2 refines an abstract one M_1, it is sufficient to show that for each transition in M_2 that may lead from an initial state s to a set of states s' , there exists a corresponding transition on the abstract side from an abstract state σ to a set σ'.[5]

[5] seL4: Formal verification of an OS kernel. Gerwin Klein et al., ACM Symposium on Operating Systems Principles 2009

Forward simulation



Demo