


Embedded Systems

- Increasing functionality
- Increasing software complexity
 - Millions of lines of code
 - Mutually untrusted SW vendors
- Consolidate functionality

Connectivity

- Attacks from outside

- No longer close systems
 - Download SW




IIES08/se/L4

1

Embedded Systems

- Diverse applications
- Real-time Vs. best effort
- Tight resource budgets
- Mission/life- critical applications
- Sensitive information

Reliability is paramount



The collage consists of five distinct images arranged in a grid-like fashion. In the top left is a silver PDA with a stylus. To its right is a laptop displaying a colorful abstract image. Below the PDA is a silver flip phone showing a woman's face on its screen. To the right of the phone is a humanoid robot with a metallic, segmented body and a helmet. At the bottom of the collage is a silver hatchback car, viewed from a front-three-quarter angle.

IES08/seL4 2

Small Kernel Approach

- Smaller, more trustworthy foundation
 - Hypervisor, **microkernel**, isolation kernel,
- Facilitate controlled integration and isolation
 - Isolate: fault isolation, diversity
 - Integrate: performance

The diagram illustrates the Small Kernel Approach architecture. It is divided into two main sections by a vertical line: 'Untrusted' on the left and 'Trusted' on the right. In the Untrusted domain, there are three components: 'Legacy App.' (represented by a stack of three orange rectangles), 'Linux Server' (an orange rectangle), and 'Device Driver' (an orange rectangle). In the Trusted domain, there are three components: 'Sensitive App.' (a stack of three light green rectangles), 'Trusted Service' (a stack of three light blue rectangles), and 'Device Driver' (a light blue rectangle). Below these components, there are three horizontal layers representing the system stack: 'Supervisor OS' (a light blue rectangle), 'Small kernel (e.g. Microkernel)' (a light blue rectangle), and 'Hardware' (a green rectangle). The 'Supervisor OS' and 'Small kernel' layers span across both the Untrusted and Trusted domains, while the 'Hardware' layer is at the base.

Small Kernel Approach

- Smaller, more trustworthy foundation
 - Hypervisor, **microkernel**, isolation kernel,
- Facilitate controlled integration and isolation
 - Isolate: fault isolation, diversity
 - Integrate: performance
- Microkernel should:
 - Provide sufficient API
 - Correct realisation of API
 - Adhere to isolation/integration requirements of the system**

The diagram illustrates the Small Kernel Approach architecture. A vertical line separates the system into two main regions: 'Untrusted' on the left and 'Trusted' on the right. Above the line, 'Legacy App.' (represented by three stacked orange rectangles) is in the Untrusted region, while 'Sensitive App.' (represented by three stacked light green rectangles) is in the Trusted region. Below the line, 'Linux Server' and 'Device Driver' (represented by two orange rectangles) are in the Untrusted region, while 'Trusted Service' and 'Device Driver' (represented by two light blue rectangles) are in the Trusted region. All these components sit on top of a 'Supervisor OS' layer (represented by a light blue rectangle). Below the Supervisor OS is the 'Small kernel (e.g. Microkernel)' layer (represented by a light blue rectangle). At the bottom is the 'Hardware' layer (represented by a green rectangle). The vertical line also passes through the Supervisor OS and Small kernel layers, indicating that these layers are responsible for maintaining the isolation between the Untrusted and Trusted regions.

Issue

- Kernel consumes resources
 - Machine cycles
 - Physical memory (kernel metadata)

Example:

- threads – thread control block,
- address space – page-tables
- bookkeeping to reclaim memory

IIES08/seL4 4

Possible Approaches

How do we manage kernel metadata?

- Cache like behaviour [EROS, Cache kernel, HiStart..]
 - No predictability, limited RT applicability
- Static allocations
 - Works for static systems
 - Dynamic systems: overcommit or fail under heavy load
- Domain specific kernel modifications?

IIES08/seL4 5

Modified \neq Verified

- L4.Verified project:**
 - Formally verify the implementation correctness of the kernel
- Properties:
 - Isolation, information flow ...
- Formal refinement
 - Formally connect the properties with the kernel implementation

IIES08/seL4 6A

Modified \neq Verified

- L4.Verified project:**
 - Formally verify the implementation correctness of the kernel
- Properties:
 - Isolation, information flow ...
- Formal refinement
 - Formally connect the properties with the kernel implementation

IIES08/seL4 6B

Modified \neq Verified

- L4.Verified project:**
 - Formally verify the implementation correctness of the kernel
- Properties:
 - Isolation, information flow ...
- Formal refinement
 - Formally connect the properties with the kernel implementation
 - Modifications invalidate refinement
 - Verification is labour intensive
 - 10K C-lines = 100K proof lines (1st refinement)
 - Memory management is core functionality

IIES08/seL4 6C

Approach in a nutshell

- No implicit allocations within the kernel
 - No heap, no slab allocation etc..
- All abstractions are provided by first-class kernel objects
 - Threads – TCB object
 - Address space – Page table objects
- All objects are created upon explicit user request

IIES08/seL4 7

Memory Management Model

NICTA

- No implicit allocations within the kernel
- Physical memory is divided into **untyped objects**
 - Authority conferred via capabilities
 - Untyped capability is **sufficient authority** to allocate kernel objects
- All abstractions are provided via **first class kernel objects**
- Allocate on explicit user request
 - Creator gets the full authority
 - Distribute capabilities to allow other access the service

IIES08/sel4 8A

Memory Management Model

NICTA

- No implicit allocations within the kernel
- Physical memory is divided into **untyped objects**
 - Authority conferred via capabilities
 - Untyped capability is **sufficient authority** to allocate kernel objects
- All abstractions are provided via **first class kernel objects**
- Allocate on explicit user request
 - Creator gets the full authority
 - Distribute capabilities to allow other access the service

Kernel objects

- Untyped
- TCB (Thread Control Blocks)
- Capability tables (CT)
- Comm. ports

IIES08/sel4 8B

Memory Management Model

NICTA

- No implicit allocations within the kernel
- Physical memory is divided into **untyped objects**
 - Authority conferred via capabilities
 - Untyped capability is **sufficient authority** to allocate kernel objects
- All abstractions are provided via **first class kernel objects**
- Allocate on explicit user request
 - Creator gets the full authority
 - Distribute capabilities to allow other access the service

Kernel objects

- Untyped
- TCB (Thread Control Blocks)
- Capability tables (CT)
- Comm. ports
- Objects are managed by user-level

IIES08/sel4 8C

Memory Management Model ...

NICTA

- Delegate authority
 - Allow others to obtain services
 - Delegate resource management
- Memory management policy is **completely in user-space**
- Isolation of physical memory = Isolation of authority (capabilities)**
 - Capability dissemination is controlled by a "Take-Grant" like protection model

IIES08/sel4 8D

Memory Management Model ...

NICTA

- De-allocation upon explicit user request
- Call revoke on the Untyped capability
- Memory can be reused
- Kernel tracks capability derivations
 - Recorded in capability derivation tree (CDT)
 - Need bookkeeping
 - Doubly-linked list through capabilities
 - Space allocated with capability tables

CDT

copy

IIES08/sel4 9

Capability Derivation Tree

NICTA

- For allocation:**
 - The untyped capability should not have any CDT children
 - Guarantees that there are no previously allocated objects
 - Size of the object(s) must be small or equal to untyped object

CDT

copy

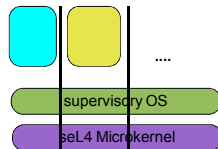
IIES08/sel4 10

Evaluation



Formal properties:

- Formalised the protection model in Isabelle/HOL
 - Machine checked, abstract model of the kernel
- Formal, machine checked proof that mechanisms are sufficient for enforcing spatial partitioning
- Proof also identify the invariants the "supervisory OS" needs to enforce for isolation to hold



IIES08/seL4

11A

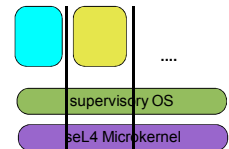
Evaluation



Formal properties:

- Formalised the protection model in Isabelle/HOL
 - Machine checked, abstract model of the kernel
- Formal, machine checked proof that mechanisms are sufficient for enforcing spatial partitioning
- Proof also identify the invariants the "supervisory OS" needs to enforce for isolation to hold

- Can not share modifiable page/capability tables
- Can not share thread control blocks
- Can not have communication channels that allow capability propagation



IIES08/seL4

11B

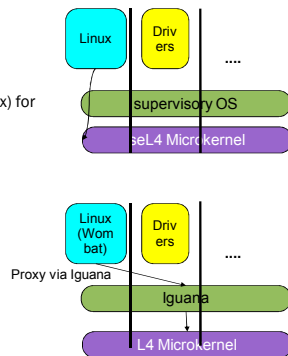
Evaluation ...



Performance

- Used paravirtualised Linux as an example
- Compared with L4/Wombat (Linux) for running LMBench

Bench mark	L4 (µs)	seL4 (µs)	Gain(%)
fork	4570	3083	32.5
exec	5022	3440	31.5
shell	29729	19999	32.7
page faults	34	18.7	45.4
Null Syscall	3.4	2.9	11
ctx	10.7	9.3	7.6



IIES08/seL4

12

Status

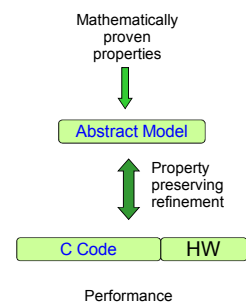


Empirical work

- Runs on ARM11
- Investigate performance as a virtualisation platform

Formal work

- Information flow properties (example: Clark-Wilson)
- Formal refinement work in progress



IIES08/seL4

13

Conclusion



No implicit allocations within the kernel

- Users explicitly allocate kernel objects
- No heap, slab .. (no hidden bookkeeping)
- Authority confinement guarantees control of kernel memory

All kernel memory management policy is outside the kernel

- Different isolation/integration configurations
- Support diverse, co-existing policies
- No modification to the kernel (remains verified)

Hard guarantees on kernel memory consumption

- Facilitate formal reasoning of physical memory consumption

Improve performance by controlled delegation

Similar performance in other case

IIES08/seL4

14

