Kernel Design for Isolation and Assurance of Physical Memory

Dhammika Elkaduwe
Philip Derrin
Kevin Elphinstone

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

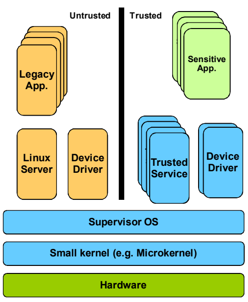
Embedded Systems

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

Reliability is paramount

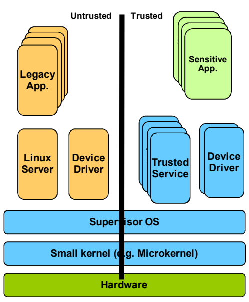
Small Kernel Approach

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



Small Kernel Approach

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

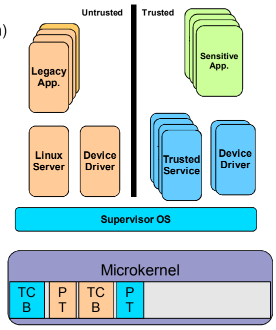


- Microkernel should:
  - Provide sufficient API
  - Correct realisation of API
  - **Adhere to isolation/integration requirements of the system**

Issue

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

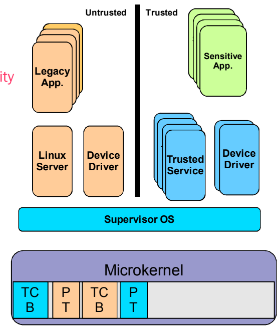
Example:
  - threads – thread control block,
  - address space – page-tables
  - bookkeeping to reclaim memory



Possible Approaches

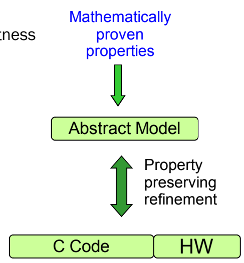
How do we manage kernel metadata?

- 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?



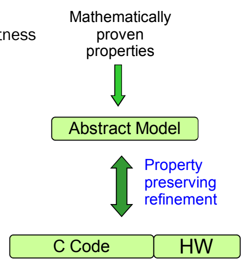
Modified ≠ Verified

- **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



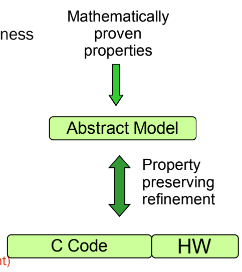
Modified ≠ Verified

- **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

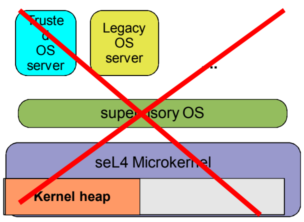


Modified ≠ Verified

- **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



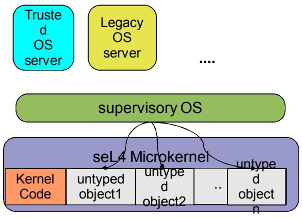
Approach in a nutshell



- 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

Memory Management Model

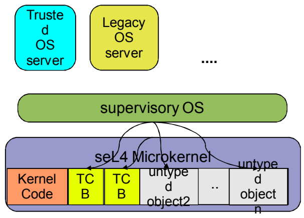
NICTA



- 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

Memory Management Model

NICTA

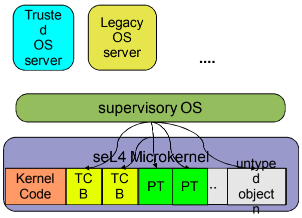


- Kernel objects
  - Untyped
  - TCB (Thread Control Blocks)
  - Capability tables (CT)
  - Comm. ports ....

- 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

Memory Management Model

NICTA

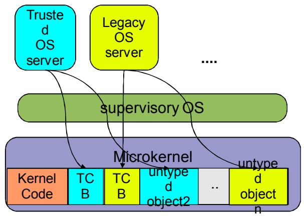


- Kernel objects
  - Untyped
  - TCB (Thread Control Blocks)
  - Capability tables (CT)
  - Comm. ports ....
- Objects are managed by user-level

- 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

Memory Management Model ...

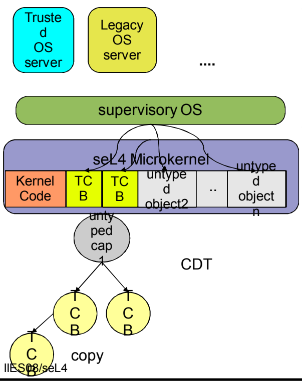
NICTA



- 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

Memory Management Model ...

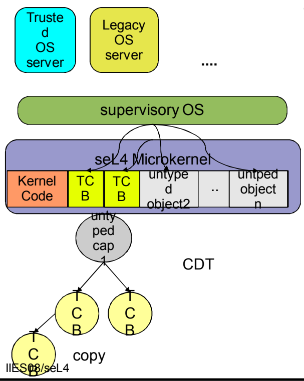
NICTA



- 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

Capability Derivation Tree

NICTA

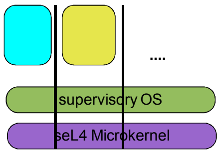


- **For allocation:**
  - The untyped capability should not have any CDT children
    - Guarantees that there are no previously allocated objects
  - Size of the object(s) must be small or equal to untyped object

Evaluation

NICTA

- **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

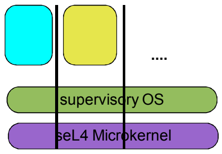


Evaluation

NICTA

- **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

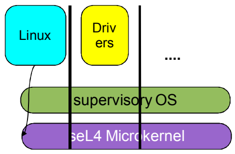
- Can not share modifiable page/capability tables
- Can not share thread control blocks
- Can not have communication channels that allow capability propagation



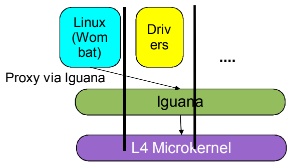
Evaluation ...

NICTA

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



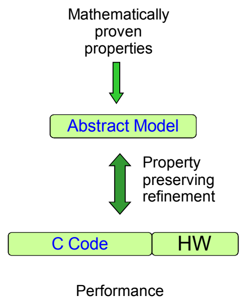
Bench mark	L4 (µs)	seL4(s)	Gain(%)
fork	4570	3083	32.5
exec	5022	3440	31.5
shell	29729	19999	32.7
page faults	34	18.7	45.4
Null Syscall	3.4	2.9	11
ctx	10.7	9.3	7.6



Status

NICTA

- **Empirical work**
  - Runs on ARM11
  - Investigate performance as a virtualisation platform
- **Formal work**
  - Information flow properties (example: Clark-Wilson)
  - Formal refinement work in progress



Conclusion

NICTA

- 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

NICTA


NICTA
From imagination to impact