Improving Interrupt Response Time in a Verifiable Protected Microkernel

Blackham, Bernard and Shi, Yao and Heiser, Gernot

EuroSys ‘12

James Marshall, GW-SSL Fall 2013

Resources

Blackham, Bernard, Yao Shi, and Gernot Heiser. "Improving interrupt response time in a verifiable protected microkernel." ***Proceedings of the 7th ACM european conference on Computer Systems*****. ACM, 2012.**

Elphinstone, Kevin, and Gernot Heiser. "From L3 to seL4 What Have We Learnt in 20 Years of L4 Microkernels?."

Mehnert, Frank, Michael Hohmuth, and Hermann Hartig. "Cost and benefit of separate address spaces in real-time operating systems." *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*. IEEE, 2002.

Blackham, Bernard, et al. "Timing analysis of a protected operating system kernel." *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE, 2011.

Blackham, Bernard, Vernon Tang, and Gernot Heiser. "To preempt or not to preempt, that is the question." *Proceedings of the Asia-Pacific Workshop on Systems*. ACM, 2012.

Klein, Gerwin, et al. "seL4: Formal verification of an OS kernel." ***Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*****. ACM, 2009.**

Domain

Hard Real-Time

- Worst Case Execution Time

Growing more complex

Mixed-criticality systems

Current Real-Time OSes

Focus on lowest possible WCET

Small, simple RT kernels

Mixed-criticality dealt with like RTLinux

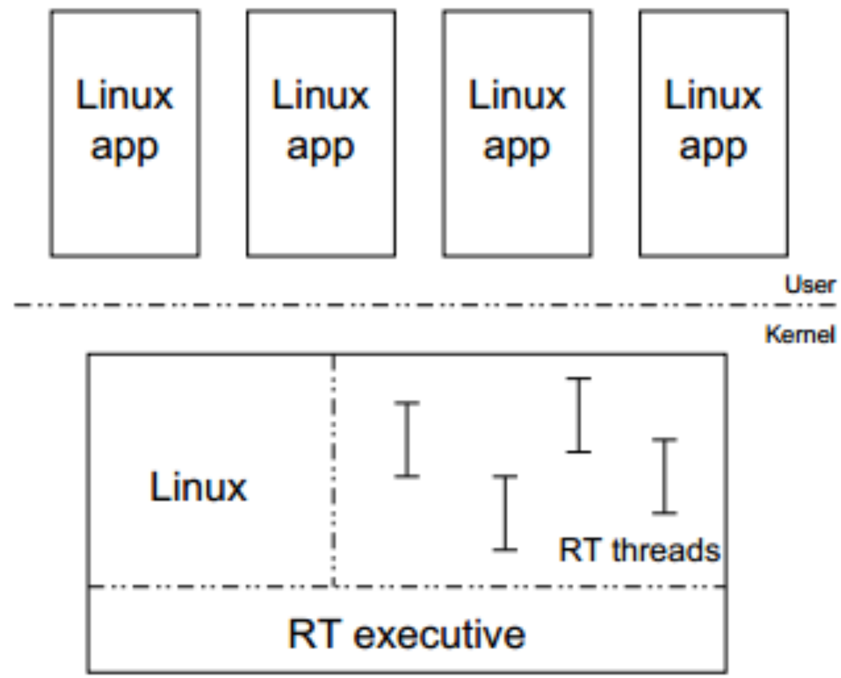

Figure 1. RTLinux structure

History

80’s - L3 by Jochen Liedtke

90’s - L4: fast IPC, microkernels work

Many variants

00’s - commercial success

seL4

Redesigned L4 to be verified

WCET analysis

* Improvements for WCET of interrupts

Verification (How)

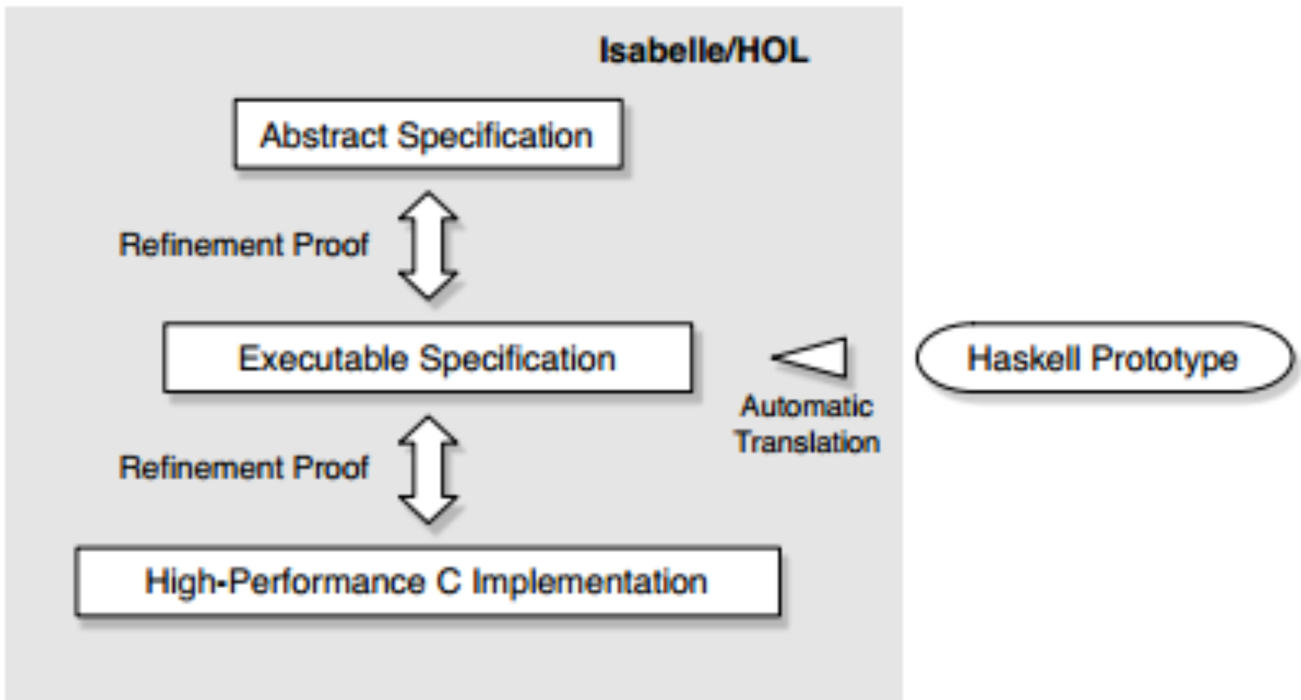

Figure 2: The refinement layers in the verification of seL4

Verification (What)

Functional Correctness: The implementation the abstract specification of the kernel.

Implications:

- No buffer overflows
- Well-formed data structures
- No non-termination
- many more...

Assumptions: C-compiler, assembly, hardware, and kernel initialization.

Drawback

Concurrency is the Verification Killer

- Non-Preemptive kernel
- Event based kernel

Verification is very expensive

- Changes are hard to make

First WCET

Table III
COMPUTED WCET VERSUS OBSERVED WCET FOR FEASIBLE WORST-CASE PATHS IN SEL4.

Event handler	Computed	Observed	Ratio
Syscall (open)	1634.8 μs	305.2 μs	5.36
Syscall (closed)	387.4 μs	46.4 μs	8.21
Unknown syscall	173.3 μs	17.9 μs	9.68
Undefined instruction	173.4 μs	17.1 μs	10.15
Page fault	175.5 μs	18.9 μs	9.27
Interrupt	104.7 μs	13.1 μs	8.01

Microseconds; too slow

approx. 800 Mhz system

Lazy Scheduling

Optimization for better average case execution

WCET is king now.

Open vs. Closed systems

What are the slow system calls?

- Kernel object creation and deletion
- Example: deleting an IPC port.

How do we speed them up?

- Original solution: Don’t do it.

Data Structure Manipulation

Allow preemption points

Progress must be made between points

Preemption Point

Kernel checkpoints progress

Checks for interrupts

Resumes system call

WCET

Difficult to compute

Observations can not be trusted

WCET Problems

Cache policies

Loops

Execution paths

2nd WCET Results

Event handler	Before changes; L2 disabled	After changes; L2 disabled		
	Computed	Computed	Observed	Ratio
System call	3851 μs	332.4 μs	101.9 μs	3.26
Undefined instruction	394.5 μs	44.4 μs	42.6 μs	1.04
Page fault	396.1 μs	44.9 μs	42.9 μs	1.05
Interrupt	143.1 μs	23.2 μs	17.7 μs	1.31

Approx. 500 Mhz system

With better WCET analysis:

- ~300 microsecond computed WCET
- Much closer to observed

Significance

Verified micro-kernel

Supports address spaces

Protected-mode kernel

Still manages sub-millisecond worst case interrupt execution time

Future Work

- Re-verification
- A few more optimizations
  - IPC send-receive
  - capability policy

That’s all folks!

L4RTL

Shows that RT Tasks can be run in user space along side of normal Linux applications (provides separate address spaces).

The performance hit is there, but not that bad.

RTLinux keeps the RT Tasks in the kernel space.

Stack Blocking

seL4 is even-based

means single stack (unlike thread based. a stack for every process, easy to preempt, just swap stacks).

Can not preempt, because the stack is shared. Stack blocking occurs if a process^1 holds an exclusive object, and process^2 preempts it. Process^2 takes the top of the stack, effectively blocking process^1 from running.

There are solutions, but this would force a policy on seL4.