EECS 262a
Advanced Topics in Computer Systems
Lecture 26

seL4 Kernel verification
April 27th, 2016

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

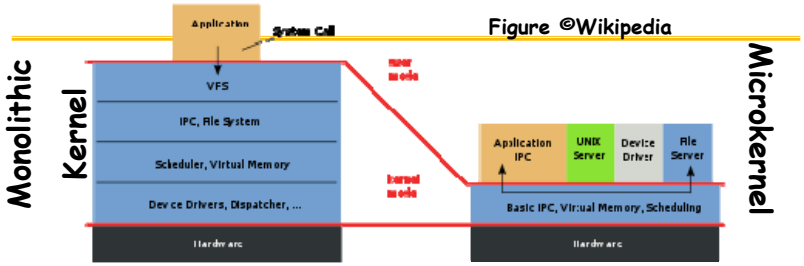
http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Paper

- **seL4: Formal Verification of an OS Kernel**, Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, Simon Winwood. Appeared in Proceedings of the ACM Symposium on Operating Systems Principles, 2009

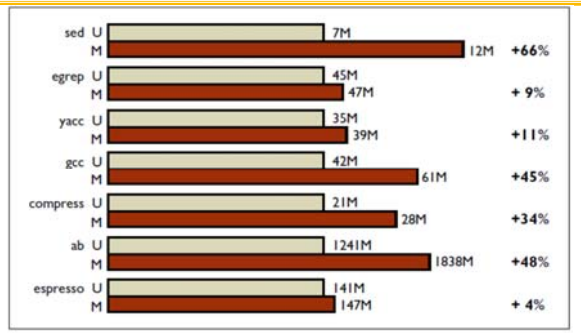
- Thoughts?

Microkernel Structure



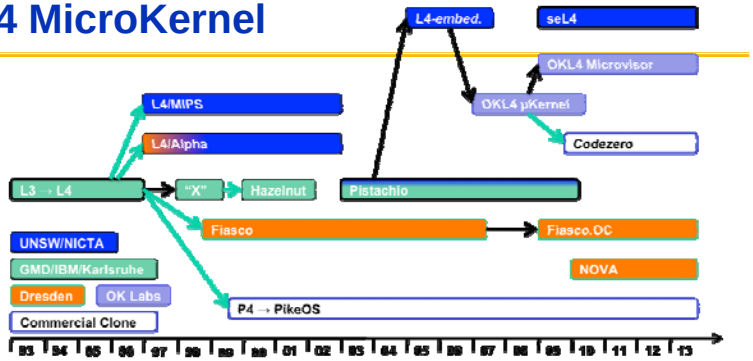
- Moves as much from the kernel into "*user*" space
  - Small core OS running at kernel level
  - OS Services built from many independent user-level processes
  - Communication between modules with message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port OS to new architectures
  - More reliable (less code is running in kernel mode)
  - Fault Isolation (parts of kernel protected from other parts)
  - More secure
- Detriments:
  - Performance overhead severe for naïve implementation

On the cost of micro-kernels:



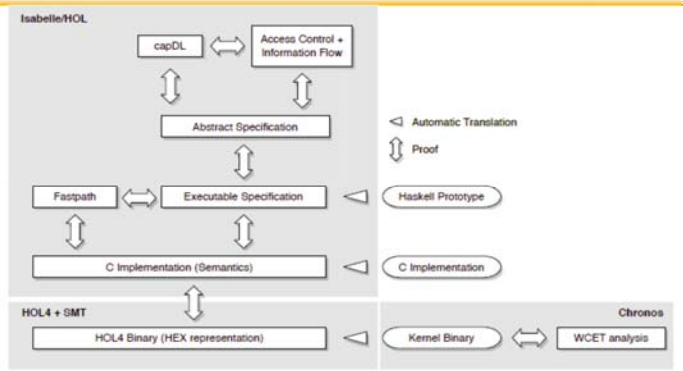
- Above: Ultrix (TAN) vs Mach (Reddish-brown)
- Cost of external pagers: page fault of first-generation microkernels took up to 1000µs!
- See optional paper "Toward Real Microkernels" by Leidtke
- After many iterations, L4 became a "best-in-class" microkernel
  - Fast IPC
  - Better resource control than simple pagers ("Address Space Managers")
  - Other optimizations

L4 MicroKernel



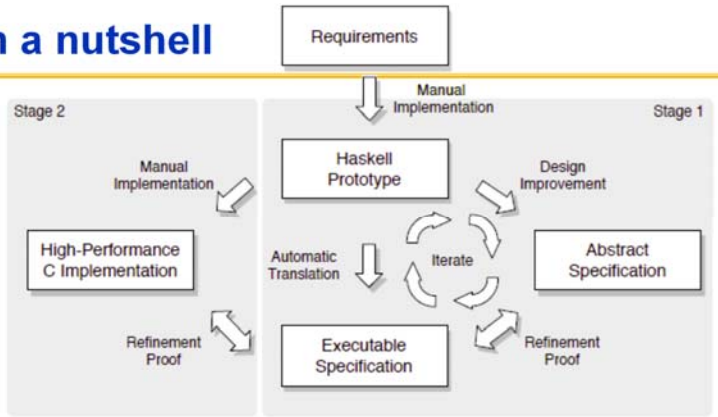
- Original L4 kernel: Jochen Liedtke
  - Derived from L3 kernel
  - Developed at GMD (Germany)
  - Designed to have fast IPC, better memory management,
- Picked up in several commercial usages

Layers of Proof Techniques



- Results (including follow-on results)
  - Functional correctness verification in Isabelle/HOL framework
  - Functional correctness for hand-optimized IPC
  - Correctness of access-control enforcement
  - Automatic poof of refinement between compiled binary and C implementation
  - Automatic static analysis of worst-case execution time for sys calls

In a nutshell



- Apply automatic techniques for "proof" of correctness
  - Series of *refinements* from Abstract Specification ⇒ Executable Specification ⇒ C implementation
  - Utilized Isabelle/HOL theorem prover
- Three different implementations
  - Spanning "obvious to humans" and "efficient"

Three Implementations

- Abstract Specification: Isabelle/HOL code
  - Describes *what* system does without saying *how* it is done
  - Written as series of assertions about what should be true
- Executable Specification: Haskel
  - Fill in details left open at abstract level
  - Specify *how* the kernel works (as opposed to what I does)
  - Deterministic execution (except for underlying machine)
  - Data structures have explicit data types
  - Controlled subset of Haskel
- The Detailed Implementation: C
  - Translation from C into Isabelle requires controlled subset of C99
    - » No address-of operator & on local (stack) variables
    - » Only 1 function call in expressions or need proof of side-effect freedom
    - » No function calls through pointers
    - » No goto, switch statements with fall-through
- Machine model: need model of how hardware behaves

Example: Scheduling

```
schedule ≡ do
  threads ← all_active_tcbs;
  thread ← select threads;
  switch_to_thread thread
od OR switch_to_idle_thread
```

- Figure 3: Isabell/High-order logic (HOL) code for scheduler at abstract level

```
schedule = do
  action <- getSchedulerAction
  case action of
    ChooseNewThread -> do
      chooseThread
      setSchedulerAction ResumeCurrentThread
      ...
chooseThread = do
    r <- findM chooseThread' (reverse [minBound .. maxBound])
    when (r == Nothing) $ switchToIdleThread
chooseThread' prio = do
     q <- getQueue prio
     liftM isJust $ findM chooseThread'' q
chooseThread'' thread = do
     runnable <- isRunnable thread
     if not runnable then do
            tcbSchedDequeue thread
            return False
     else do
            switchToThread thread
            return True
```

- Figure 4: Haskell Code for Scheduler

Challenges

- Smaller Kernel ⇒ Increased Interdependence of components
  - Radical size reduction in microkernel leads to high degree of dependence of remaining components
  - Proof techniques still possible
- Design special versions of each of the languages
- Design multiple implementations each in different languages
- Huge number of invariants is moving from Abstract specification to Executable specification
  - Low-level memory invariants
  - Typing invariants
  - Data structure Invariants
  - Algorithmic Invariants

Experience and Lessons Learned

- Performance
  - Optimized IPC path is well performing (via micro-benchmarks)
  - Other performance not characterized in this paper
- Verification effort: extremely high!
  - Three phases, each with non-trivial effort
  - Overall size of proof (including automatically generated proofs): 200,000 lines of Isabelle script
  - Abstract Spec: 4 person months (4 pm)
  - Haskell prototype: 2 person years (2 py)
  - Executable spec translated from Haskel prototype: 3 pm
  - Initial C translation+implementation: 2pm
  - Cost of the proof: 20py (seL4 specific, 11py)
- What about future: 8py for new kernel from scratch!
- Cost of change
  - Simple or independent small features not huge, say 1 person week
  - Cost to add new features can be very large: up to 2py!

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

Final Project Timing

- Poster Session:
  - Next Thursday (5/5) from 10:30 – 12:30
  - Plan on staying whole time, but might be shorter
- Final paper:
  - Due Tuesday 5/10 @ AOE (by 5am)
  - 10 pages, 2- column, conference format
  - Must have a related work section!
  - Also, plan on a future work and/or discussion section
  - Make sure that your METRICs of success are clear and available

Thank you! You have all been great!