File 20111025.0708: Notes from Security Reading Group:

Security Reading Group met Tuesday to discuss 'seL4: Formal Verification of an OS Kernel' by Klein, et al. (2009), suggested by Justin. This is the front page paper from the research group Justin just left in Sydney. They argue that this is the only sane way to verify an OS kernel. The cool/terrifying part is that after bootstrap, the kernel never does any memory allocation ever after that first time; subsequently all memory allocation is done by user level processes.

Pronounced 'ess-eee-ell-four' or 'sell-four'.

What the authors call the 'executable specification' is a Haskell programme. What they proved was functional correctness; that the C implementation is sane and does what the specification says it does.

Interesting things: the subset of C they permit. One significant thing they prohibit is taking the address of local variables; the reason is because they maintain the fiction of where the stack is kept in real memory.

The assumptions at the beginning of the paper are funny. Huge emphasis on access control to memory, but nothing on kernel threading. Justin was working on a new paper, not submitted, about what makes programming in seL4 hard: writing memory allocation routines on this kernel is obscenely difficult.

[Skype video freezing; audio dropped out for about half a minute]

Question from Shamal: is there a paper that describes seL4 itself? Answer from Justin: Yes, reference 20 in the paper [1, ref. 20] describes the seL4 API.

Question from John: comparison with INTEGRITY? Answer from Justin: capabilities in seL4 should be more powerful than the separation guarantees in Green Hills' OS. The powerful, fundamental object in seL4 is the untyped object; then capabilities are built on top of those.

The interesting thing is destruction; you can actually tear down entire subsystems with a single system call. The destruction operation is particularly interesting because it is unbounded in duration, meaning you can't keep interrupts disabled throughout the operation, so it must be pre-emptable, so you have to pay careful attention to global preconditions and postconditions at every yield point.

Problem with formal verification: making changes is insanely hard. The designers want a static kernel. Shamal made a comparison to microcode, mentioned Dr Simpson's interests.

Justin: one of the goals of the project was to yield a kernel that is verifiable, implemented on actual hardware, and fast. Performance was a strong consideration; that is why they used a microkernel—to get good scheduling performance, at the cost of losing IPC performance.

Shamal asked about the difference between verification and validation: validation means building the right system; verification means building the system right. (See the Wikipedia article at `http://en.wikipedia.org/wiki/Verification_and_validation` for more on this.)

Justin: you must trust the bootstrapper. If the bootstrapper is a simple static one that only starts a bunch of processes and leaves, then you can probably trust it. Question from John about verified compilers.

I asked where is a good place to start. Reference 20 is a good place; begin from the research group web site at `http://ertos.nicta.com.au/research/sel4/` (University of New South Wales, Sydney); can find a copy of the paper there, and also the software reference manual.

Performance: the interrupt latency of seL4 is crap, compared to its IPC latency which is excellent. According to Justin, where seL4 should be put is at the core of another OS.

Question from John re: Singularity OS. Justin doesn't trust it.

References

[1] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *22nd ACM Symposium on Principles of Operating Systems (SOSP)*, pages 207–220, Big Sky, Montana, USA, 11–14 October 2009.