The seL4 Microkernel Operating System

By Penny Rowe

Encapsulated by the motto, “Security is no excuse for bad performance,” the seL4 operating system promises comprehensive formal verification while delivering high performance (Heiser, 2020). seL4 was developed in 2006 as a microkernel in the L4 family with the goal of providing the highest level of security. (A microkernel provides only the minimum mechanisms for the system’s required functionality; Liedtke, 1995; see Fig. 1). seL4 is the first application to be mathematically proven to be bug-free in its specification (Klein et al, 2009), and secure (Klein et al, 2014). In particular, applications are isolated so that vulnerabilities cannot be transmitted among them. Here we discuss several aspects of the seL4 operating system, including thread management, the CPU scheduling policy, and memory management, and how these relate to the high security provided by seL4.

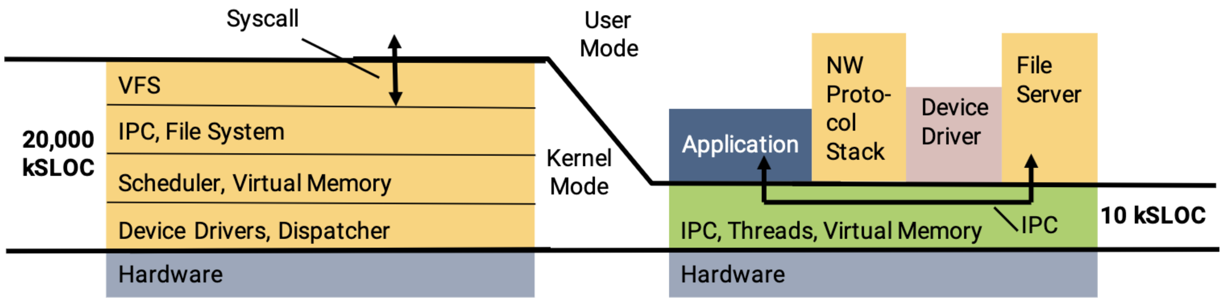

Figure 1. Monolithic kernel (left) vs microkernel (right). Reproduced from Heiser (2020).

seL4 supports multicore on some architectures, using a “giant lock”. A giant lock (or big-lock) is a single lock that is used during a system call or other instance when a thread is in kernel mode. A giant lock can slow down performance because it eliminates concurrency in kernel mode - multiple threads can run in user mode, but only one thread can run in kernel mode (https://en.wikipedia.org/wiki/Giant_lock). This is mitigated in seL4 by the ability to break a system operation into smaller sub-operations, the larger operation is protected by the ability to abort and restart if all sub-operations are not successfully run (Heiser 2020). While using a giant lock can degrade performance on a system like Linux, it is suitable for microkernels such as seL4, which run on cores that are closely-coupled, sharing an L2 cache (https://docs.sel4.systems/projects/sel4/frequently-asked-questions.html; Peters et al 2015)

Managing processor time is done in seL4 using threads. Each thread has a thread control block (TCB) that includes a priority and maximum control priority (described below) as well as other parameters (register state and floating-point context, CSpace capability, VSpace capability, and endpoint capability to send fault messages to and reply capability slot; https://docs.sel4.systems/Tutorials/threads.html).

seL4 uses a priority-based scheduler to select threads that are runnable and non-blocked. Priority is set between 0 and 255, and the thread with the maximum priority is selected, with the caveat that the priority must be lower than the thread's maximum control priority (MCP), given on the TCB. The priority and MCP of the root task (at system start) are both set to the maximum of 255. In case of ties in priority, the scheduler uses round robin, or first-in-first-out (https://docs.sel4.systems/Tutorials/threads.html).

The CPU uses a periodic interrupt based on its time quantum, called a "tick". Ticks are used to determine how long a thread can run before being preempted. Each thread receives a certain number of ticks, specified by a field on the TCB called a timeslice, after which scheduling reverts to round robin. Threads can also yield their timeslice (although not between domains, as discussed below).

The seL4 scheduler also has multi-level capability through its top-level hierarchical domain scheduler. In order to provide the high security promised by seL4, the scheduler allows threads to be assigned to separate domains. Threads can only be scheduled when their domains are active, and the domains are run on a cyclic, non-preemptible schedule, set at compile time, that is therefore deterministic. Yields between domains are not possible, and if a scheduled domain has no more threads to run, an idle thread specific to that domain will run until the switch to the next domain. This therefore isolates threads in different domains from each other, allowing confidentiality (https://docs.sel4.systems/Tutorials/threads.html), where confidentiality means access or modification of data can only be done by authorized processes (Fruhlinger, 2020). However, it presumably comes at the cost of decreased performance, since the CPU spends time in idle.

Memory management in seL4 is primarily done by the user through threads. The only memory management done by the kernel is at boot time, after which control of all free resources passes to the user (https://docs.sel4.systems/projects/sel4/frequently-asked-questions.html). Memory management is thus done by the user, who implements a Virtual Memory Manager, via the threads. The user allocates a physical memory frame, maps it to the thread's page directory, and manually allocates page tables (McLeod et al, 2021; GitHub repository readme).

With its combination of high security and performance, the seL4 operating system is a must for highly sensitive applications. For example, when used in an autonomous helicopter (see Fig. 2), it proved its ability to withstand cyber-attacks. This high security is aided by the use of giant lock that only allows one thread in kernel mode at a time, as well as a deterministic domain scheduler that can be used to isolate threads from each other.

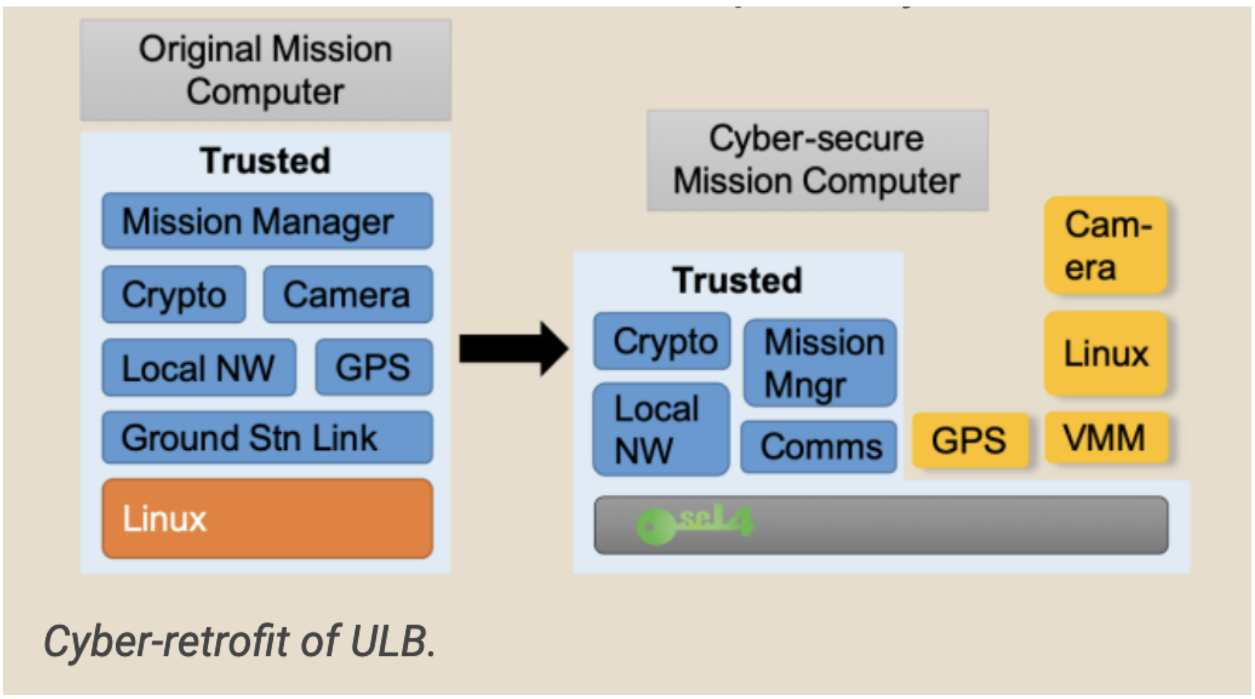

Figure 2. Retrofit of the Unmanned Little Bird (ULB) autonomous helicopter computer system to ensure security. Image from https://microkerneldude.org/2019/08/06/10-years-sel4-still-the-best-still-getting-better/.

Bibliograpy

https://sel4.systems/

https://docs.sel4.systems/projects/sel4/frequently-asked-questions.html;

https://docs.sel4.systems/Tutorials/threads.html

https://microkerneldude.org/2019/08/06/10-years-sel4-still-the-best-still-getting-better/.

Fruhlinger, J. (2020), The CIA triad: Definition, components and examples, CSO online, https://www.csoonline.com/article/3519908/the-cia-triad-definition-components-and-examples.html#/.

Heiser, G. (2020). The seL4 Microkernel – An Introduction. White paper. The seL4 Foundation, Revision 1.2 of 2020-06-10.

Klein, Gerwin; Elphinstone, Kevin; Heiser, Gernot; Andronick, June; Cock, David; Derrin, Philip; Elkaduwe, Dhammika; Engelhardt, Kai; Kolanski, Rafal; Norrish, Michael; Sewell, Thomas; Tuch, Harvey; Winwood, Simon (October 2009). "seL4: Formal verification of an OS kernel"

(PDF). *22nd ACM Symposium on Operating System Principles*. Big Sky, MT, USA. Archived (PDF) from the original on 28 July 2011.

Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. Comprehensive formal verification of an OS microkernel. ACM Transactions on Computer Systems, 32(1):2:1–2:70, February 2014. URL `https://ts.data61.csiro.au/publications/nicta_full_text/7371.pdf`.

Liedtke, Jochen (December 1995). "On µ-Kernel Construction". *Proceedings 15th ACM Symposium on Operating Systems Principles (SOSP)*. pp. 237–250. Archived from the original on 25 October 2015.

McLeod, K., Pingerino, A., and Klein, G. (2021). seL4 Dynamic Libraries: IPC, GitHub readme.

Peters, Sean, Adrian Danis, Kevin Elphinstone, and Gernot Heiser. "For a microkernel, a big lock is fine." In *Proceedings of the 6th Asia-Pacific Workshop on Systems*, pp. 1-7. 2015.