Towards a Provably Secure Separation Kernel
with Dynamic Time and Memory Management

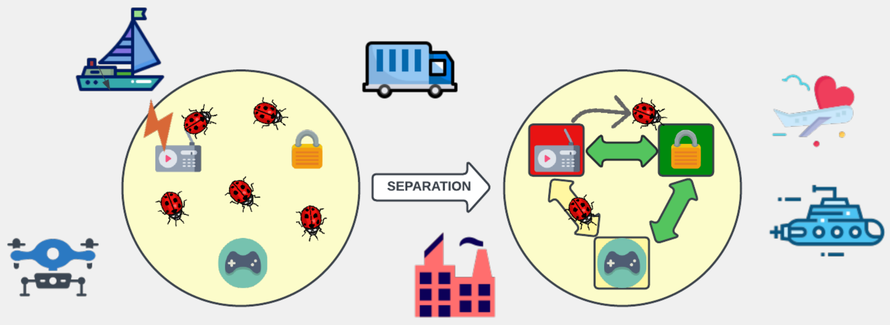
CDIS Spring Conference, 24 May 2022

Henrik Karlsson (henrik10@kth.se)

Center for Cyber Defense and Information Security (CDIS)
KTH Royal Institute of Technology

Motivation

A system's software components may have **different levels of trust and assurance**, this can cause serious **availability and security problems**, but it can be solved using a **separation kernel** that isolates components as below.

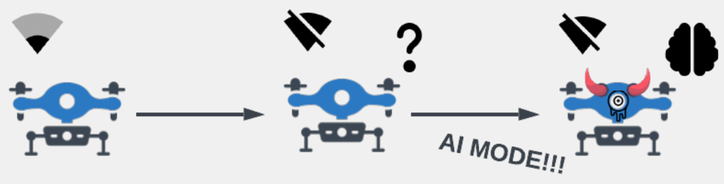


Currently, the avionics industry uses **static separation kernels** to solve the above issue, so what does our project offer?

We offer a **dynamic separation kernel** having both dynamic memory and time management. These features opens new possibilities for systems as it allows them to safely and securely change modes of operation, exchange memory, and more. Moreover, we aim to formally verify the kernel, providing a proof that our separation kernel works as intended!

Use cases:

- **Ships and airplanes** are exposed events such as adverse conditions that may require a mode change for safety. In a static system, every mode of operation must be handled by a single process, with our kernel, every mode can be handled by separate processes that can be swapped on demand.



- We can securely multiplex a **communication channel** using a timely separation kernel. With dynamic time management, we can also dynamically decide the bandwidth of channels.

Properties of the Kernel

Predictability. The kernel schedules processes with a dynamic and deterministic round-robin scheduler. We provide predictability using scheduling priorities, slack time, and state flushes (hardware dependant).

Low hardware requirements. The kernel is designed to run on a RISC-V processor with machine- and user-mode, and a memory protection unit, no virtual memory.

Multicore. The kernel supports multicore processors, making it suitable for applications with constraints on space, weight and power.

Dynamic time and memory. Our main result, the kernel allow processes to dynamically manage both time and memory.

Formally verifiable. We have designed and implemented the kernel to be formally verifiable as we intend to produce a mathematical proof of its correctness and security.

Results

First separation kernel with strong timing isolation and dynamic scheduling.

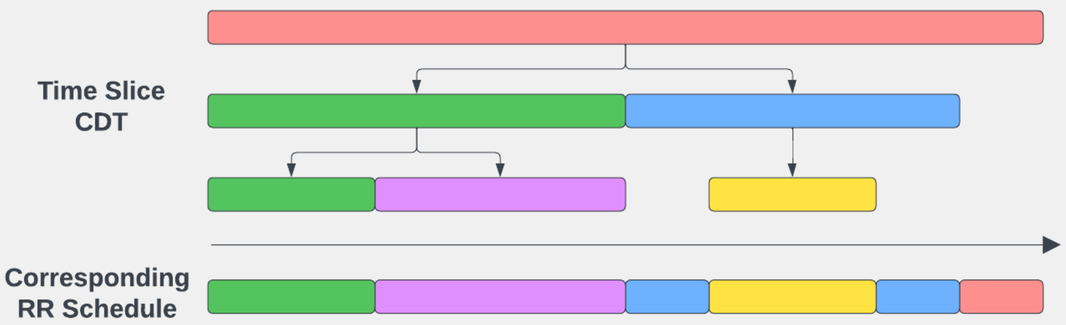
We have implemented the separation kernel using 2kLoC of C/Assembly for 64-bit RISC-V. We are now testing, debugging, benchmarking, and tweaking the kernel. We have one master's student evaluating the kernel's performance and design choices.

Design Details

Microkernel. We design the kernel as a microkernel as they are simpler than corresponding monolithic and hybrid kernels. The simplicity of software is essential for secure systems as they are easier to understand, implement correctly, and consequently, easier to analyze and verify.

Capabilities. We implement resource management using capabilities. Capabilities are tokens residing inside the kernel that lets processes manage resources. We use capabilities to control and use memory slices, memory protection, time slices, communication endpoints, and other processes. Let us take a closer look at time slices.

Time slices. The kernel uses a round-robin scheduler which we can dynamically modify using time slices. A time slice gives a process run-time on a core. A process can derive new time slices from existing slices. The derived slices must be a subset of the original. This rule gives us a capability derivation tree (CDT) illustrated below. Time slices further down a CDT have a higher priority in the scheduling but the slices further up have the power to revoke time by deleting the children. The revoke operation is crucial as it lets a process temporally give time to another process and reclaim it if necessary.



Preemption. We have designed the kernel to be (mostly) preemptive. The exceptions are parts of capability management, exception/interrupt entry and exit, and the scheduler. For the time predictability of the scheduler, we use slack time that should cover the non-preemptive parts of the kernel.

Hardware. We run the kernel in RISC-V's machine mode giving complete control over the system. Processes run in user mode and are isolated using an MPU. A process can control its MPU configuration using capabilities derived from memory slices.

Future plans

Immediate plan. Our immediate plan is to make the kernel publishable, and for this purpose, we will deploy the kernel to a real system, benchmark the kernel, and measure the WCET of the non-preemptive code. Once completed, we can publish the kernel design, and proceed with the modelling and verification of the kernel.

Long-term plan. Our long-term development goals for the project are below. Green is complete, yellow is work-in-progress, red is future stages, and blue is for external projects. Note that the development process is iterative, meaning that the kernel design and implementation are final when we have finished the formal verification.