Dynamic Separation Kernel for Safety- and Security-critical Applications

Henrik Karlsson (henrik10@kth.se)

Supervisors: Mads Dam & Roberto Guanciale
KTH Royal Institute of Technology

November 8, 2022

Cyber attacks

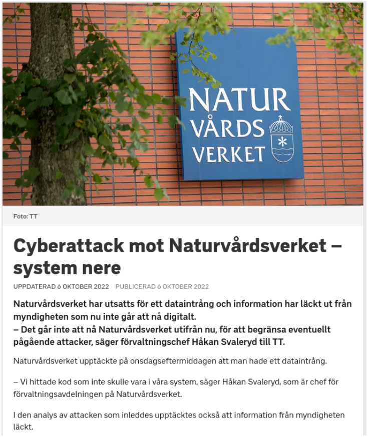


Foto: TT

Cyberattack mot Naturvårdsverket – system nere

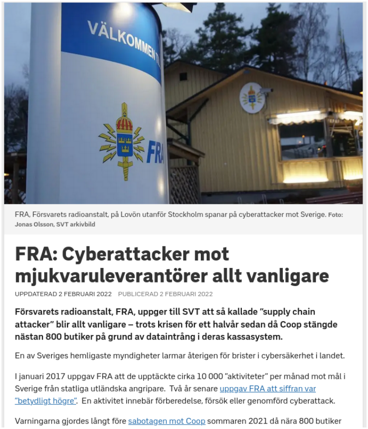
UPPDATERAD 6 OKTOBER 2022 PUBLICERAD 6 OKTOBER 2022

Naturvårdsverket har utsatts för ett dataintrång och information har läckt ut från myndigheten som nu inte går att nå digitalt.
– Det går inte att nå Naturvårdsverket utifrån nu, för att begränsa eventuellt pågående attacker, säger förvaltningschef Håkan Svaleryd till TT.

Naturvårdsverket upptäckte på onsdagseftermiddagen att man hade ett dataintrång.

– Vi hittade kod som inte skulle vara i våra system, säger Håkan Svaleryd, som är chef för förvaltningsavdelningen på Naturvårdsverket.

I den analys av attacken som inleddes upptäcktes också att information från myndigheten läckt.



FRA, Försvarets radioanstalt, på Lovön utanför Stockholm spanar på cyberattacker mot Sverige. Foto: Jonas Olsson, SVT arkivbild

FRA: Cyberattacker mot mjukvaruleverantörer allt vanligare

UPPDATERAD 2 FEBRUARI 2022 PUBLICERAD 2 FEBRUARI 2022

Försvarets radioanstalt, FRA, uppger till SVT att så kallade ”supply chain attacker” blir allt vanligare – trots krisen för ett halvår sedan då Coop stängde nästan 800 butiker på grund av dataintrång i deras kassasystem.

En av Sveriges hemligaste myndigheter larmar återigen för brister i cybersäkerhet i landet.

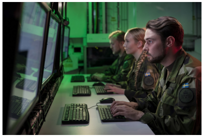
I januari 2017 uppgav FRA att de upptäckte cirka 10 000 ”aktiviteter” per månad mot mål i Sverige från statliga utländska angripare. Två år senare uppgav FRA att siffran var ”betydligt högre”. En aktivitet innebär förberedelse, försök eller genomförd cyberattack.

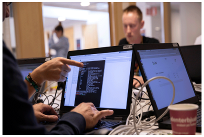
Varningarna gjordes långt före sabotagen mot Coop sommaren 2021 då nära 800 butiker

CDIS: Center for Cyber Defence and Information Security

Swedish
Armed
Forces

Center for
Cyber Defence and
Information Security


Cybersoldier education


Research projects

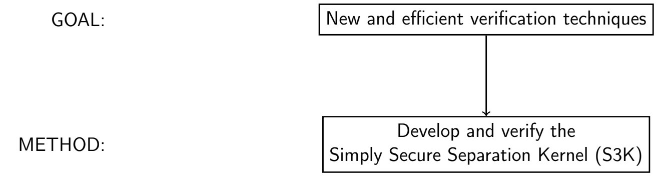
Provable Security

GOAL: $\boxed{\text{New and efficient verification techniques}}$

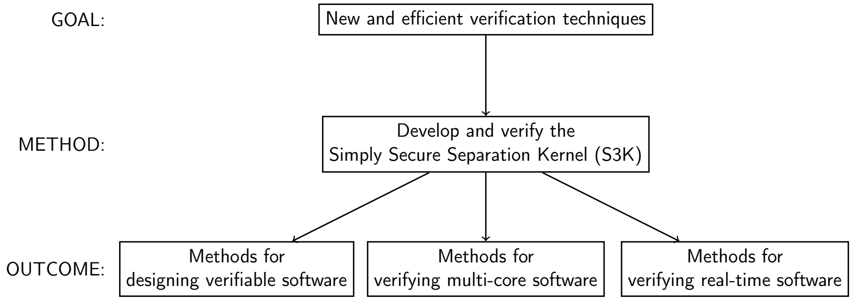
METHOD:

OUTCOME:

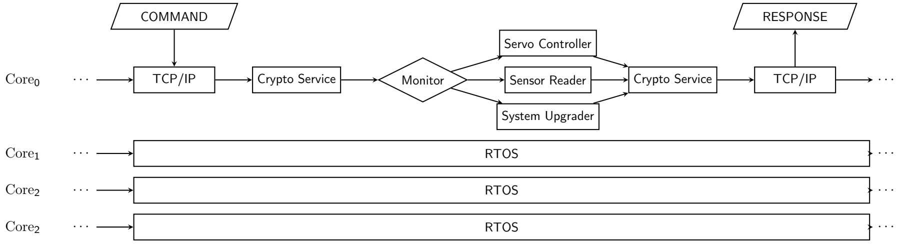
Provable Security


GOAL:
New and efficient verification techniques
METHOD:
Develop and verify the Simply Secure Separation Kernel (S3K)
OUTCOME:

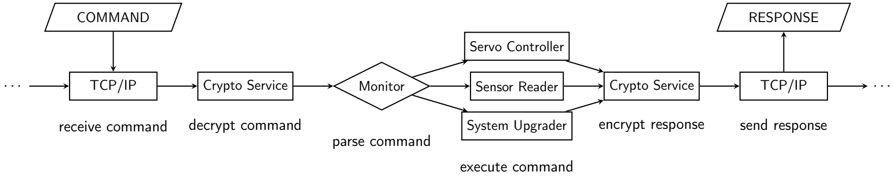
Provable Security


GOAL:
New and efficient verification techniques
METHOD:
Develop and verify the Simply Secure Separation Kernel (S3K)
OUTCOME:
Methods for designing verifiable software
Methods for verifying multi-core software
Methods for verifying real-time software

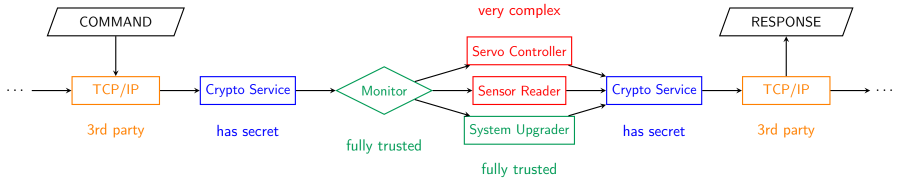
Scenario


COMMAND
RESPONSE
Core$_0$
Core$_1$
Core$_2$
Core$_2$
TCP/IP
Crypto Service
Monitor
Servo Controller
Sensor Reader
System Upgrader
Crypto Service
TCP/IP
RTOS
RTOS
RTOS

Scenario


COMMAND
TCP/IP
receive command
Crypto Service
decrypt command
Monitor
parse command
Servo Controller
Sensor Reader
System Upgrader
execute command
Crypto Service
encrypt response
TCP/IP
send response
RESPONSE
· · ·
· · ·

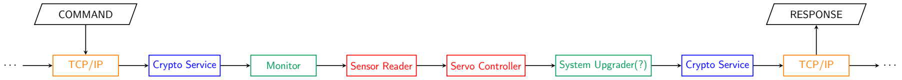
Scenario



Partition system using separation kernel

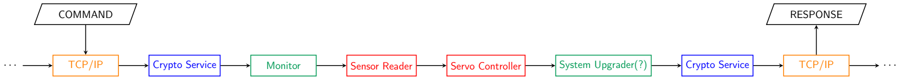
- TCP/IP – Handle TCP/IP communication
- Crypto Service – Encrypts & Decrypts Packages
- Monitor – Monitors, parse commands, upgrade system
- Functions – Interact with environment and output data

Separation Kernel



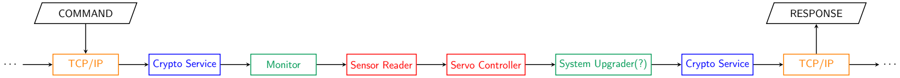
- Partitions has **fixed** round-robin scheduling

Separation Kernel



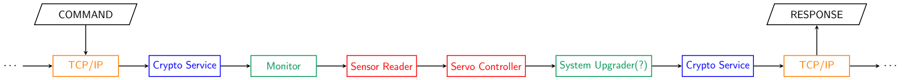
- Partitions has **fixed** round-robin scheduling
- Related Work

Separation Kernel



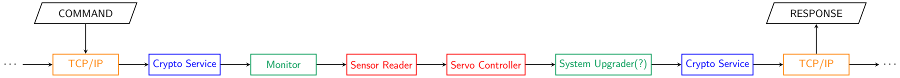
- Partitions has **fixed** round-robin scheduling
- Related Work
  - PikeOS by SYSGO

Separation Kernel



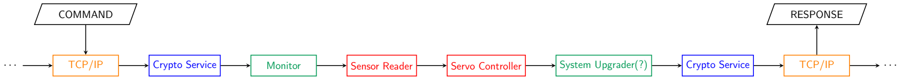
- Partitions has **fixed** round-robin scheduling
- Related Work
  - PikeOS by SYSGO
  - INTEGRITY-178B by Green Hills

Separation Kernel



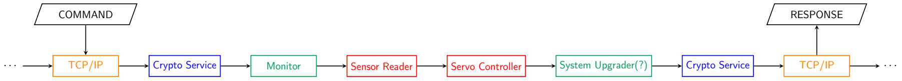
- Partitions has **fixed** round-robin scheduling
- Related Work
  - PikeOS by SYSGO
  - INTEGRITY-178B by Green Hills
  - seL4 by Heiser et al.

Separation Kernel



- Partitions has **fixed** round-robin scheduling
- Related Work
  - PikeOS by SYSGO
  - INTEGRITY-178B by Green Hills
  - seL4 by Heiser et al.
  - MultiZone Security by HEX-Five Security

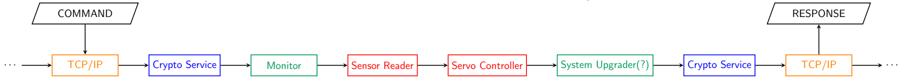
Separation Kernel



- Partitions has **fixed** round-robin scheduling
- Related Work
  - PikeOS by SYSGO
  - INTEGRITY-178B by Green Hills
  - seL4 by Heiser et al.
  - MultiZone Security by HEX-Five Security
  - OpenMZ (open-source MultiZone) by Henrik Karlsson

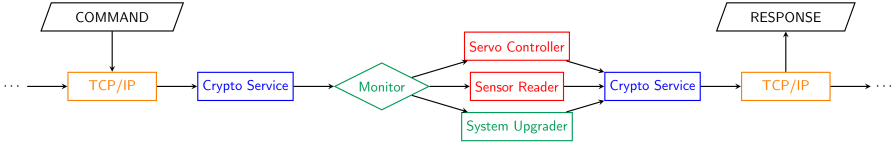
Contribution

Fixed partition scheduling

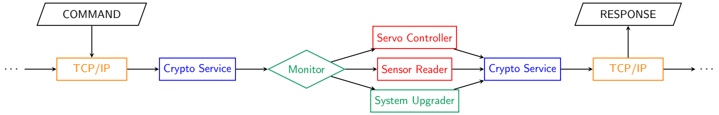


↓

Dynamic partition scheduling



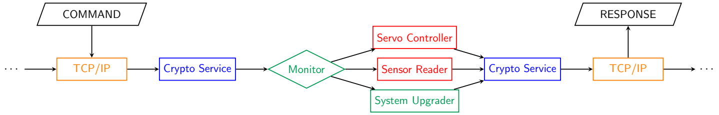
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with . . .

[1] MPU - Memory Protection Unit, protects physical memory.

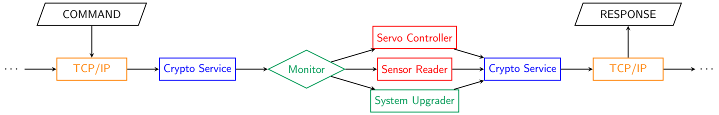
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with ...
  - Memory Protection and Management

[1]MPU - Memory Protection Unit, protects physical memory.

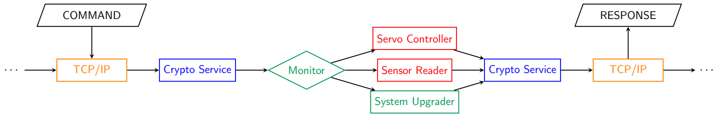
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems

[1]MPU - Memory Protection Unit, protects physical memory.

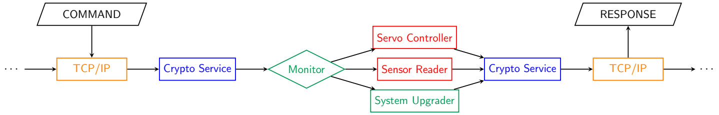
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)

[1]MPU - Memory Protection Unit, protects physical memory.

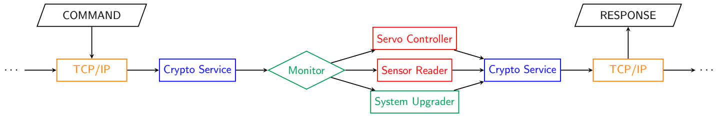
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with . . .
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring

[1]MPU - Memory Protection Unit, protects physical memory.

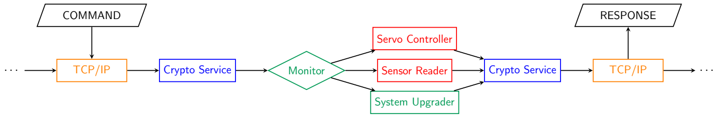
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring
- Targeting standard RISC-V 64-bit (RV64IMA) with MPU[1]

[1]MPU - Memory Protection Unit, protects physical memory.

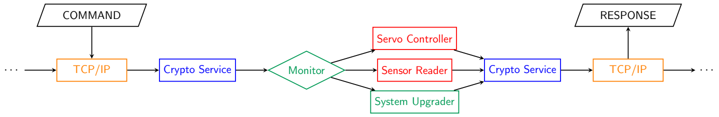
Simply Secure Separation Kernel (S3K)



- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring
- Targeting standard RISC-V 64-bit (RV64IMA) with MPU[1]
- Multi-core processor support

[1]MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)



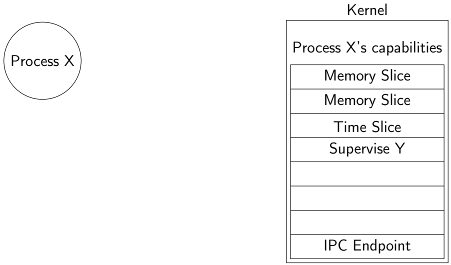
- Designed and implemented a separation kernel with . . .
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring
- Targeting standard RISC-V 64-bit (RV64IMA) with MPU[1]
- Multi-core processor support
- $\sim$ 2000 lines of C/Assembly

[1]MPU - Memory Protection Unit, protects physical memory.

Capabilities

Previously mentioned features are implemented using capabilities

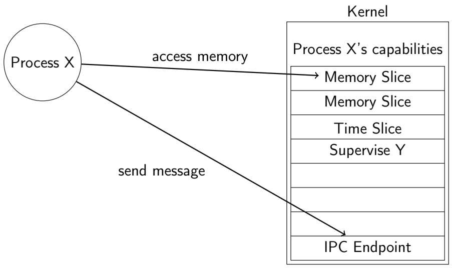
- Capability = object in kernel representing a resource



Capabilities

Previously mentioned features are implemented using capabilities

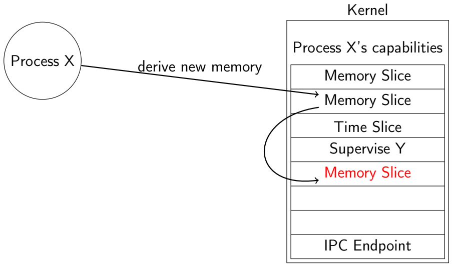
- Capability = object in kernel representing a resource
- Process owning a capability has access to corresponding resource



Capabilities

Previously mentioned features are implemented using capabilities

- Capability = object in kernel representing a resource
- Process owning a capability has access to corresponding resource
- Process can derive new capabilities from existing capabilities



S3K Capabilities

- Memory Slice – Manage access to a physical memory region.
    - PMP – Configure RISC-V's MPU, grants memory access.

S3K Capabilities

- Memory Slice – Manage access to a physical memory region.
  - PMP – Configure RISC-V's MPU, grants memory access.
- Time Slice – Manage and grant execution time on a core.

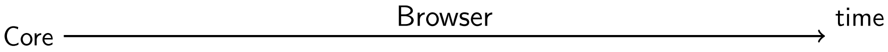
S3K Capabilities

- Memory Slice – Manage access to a physical memory region.
  - PMP – Configure RISC-V's MPU, grants memory access.
- Time Slice – Manage and grant execution time on a core.
- Channels – Manage IPC channels and endpoints.
  - Receiver/Sender – Unidirectional IPC channel.
  - Server/Client – Bidirectional IPC channel.

S3K Capabilities

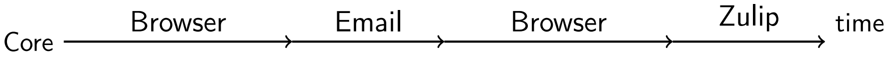
- Memory Slice – Manage access to a physical memory region.
    - PMP – Configure RISC-V's MPU, grants memory access.
- Time Slice – Manage and grant execution time on a core.
- Channels – Manage IPC channels and endpoints.
    - Receiver/Sender – Unidirectional IPC channel.
    - Server/Client – Bidirectional IPC channel.
- Supervisor – Manage a set of processes.

What is Process Scheduling?



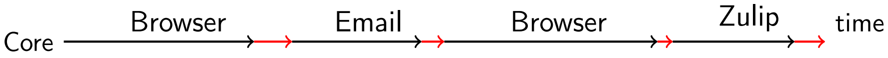
- Cores can only run one process at the time

What is Process Scheduling?



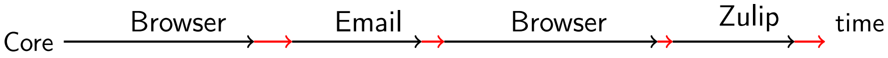
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel

What is Process Scheduling?



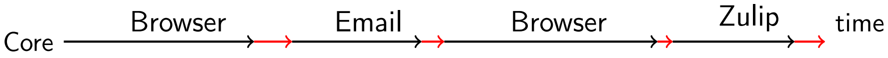
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel
- Context switch $\implies$ overhead

What is Process Scheduling?



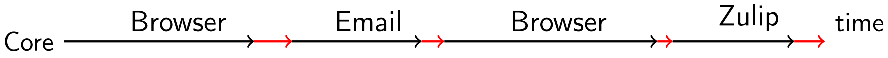
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel
- Context switch $\implies$ overhead
- Core multiplexing technique = Process Scheduling

What is Process Scheduling?



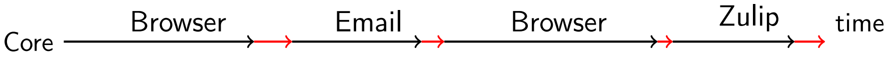
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel
- Context switch $\implies$ overhead
- Core multiplexing technique = Process Scheduling
- Best scheduler?

What is Process Scheduling?



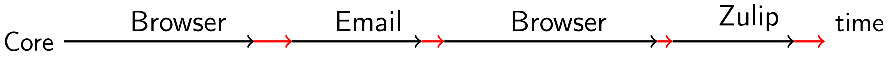
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel
- Context switch $\implies$ overhead
- Core multiplexing technique = Process Scheduling
- Best scheduler?
  - Supercomputer, Desktop, ... $\rightarrow$ Performance

What is Process Scheduling?



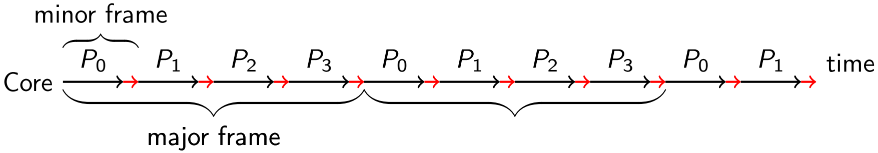
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel
- Context switch $\implies$ overhead
- Core multiplexing technique = Process Scheduling
- Best scheduler?
  - Supercomputer, Desktop, ... $\rightarrow$ Performance
  - Airplane, Car, Railroad, ... $\rightarrow$ Safety

What is Process Scheduling?



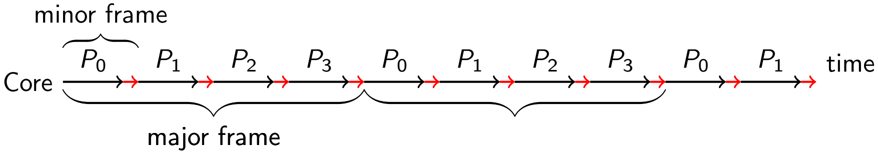
- Cores can only run one process at the time
- Multiplexing the Core – main duty of the kernel
- Context switch $\implies$ overhead
- Core multiplexing technique = Process Scheduling
- Best scheduler?
  - Supercomputer, Desktop, ... $\rightarrow$ Performance
  - Airplane, Car, Railroad, ... $\rightarrow$ Safety
  - VPN servers, routers, ... $\rightarrow$ Security

S3K Scheduling



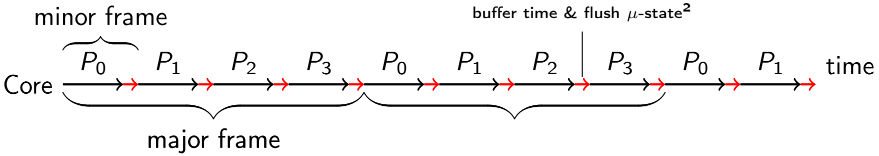
- Modified RR – Minor frames defined by time slice capabilities.

S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.

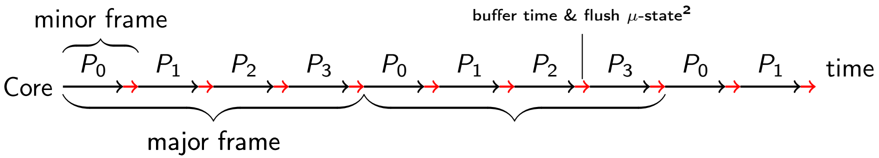
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.

[2]Flush cache, branch predictors, etc., support is hardware dependant.

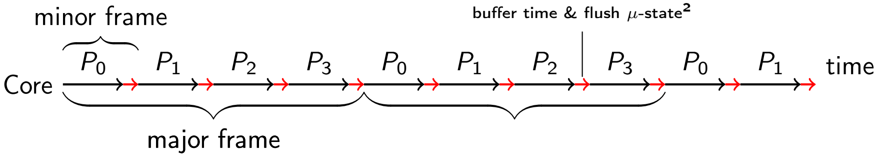
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.
- Temporal Isolation – A process's execution time depends only on its capabilities.

[2]Flush cache, branch predictors, etc., support is hardware dependant.

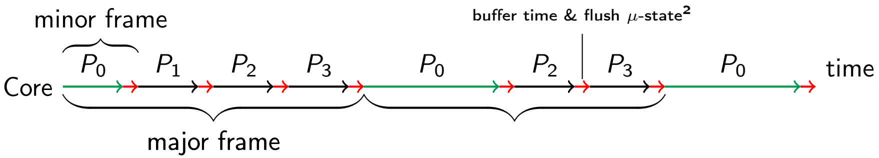
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.
- Temporal Isolation – A process's execution time depends only on its capabilities.
- Low-overhead – Scheduling decided by a lookup table.

[2]Flush cache, branch predictors, etc., support is hardware dependant.

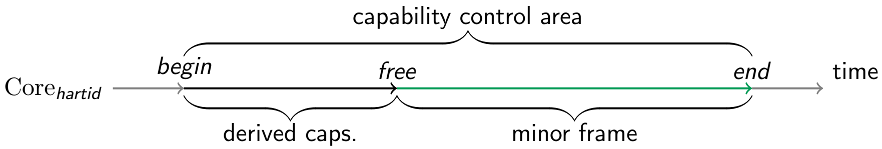
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.
- Temporal Isolation – A process's execution time depends only on its capabilities.
- Low-overhead – Scheduling decided by a lookup table.
- **Dynamic – Process can alter their time slices.**

[2]Flush cache, branch predictors, etc., support is hardware dependant.

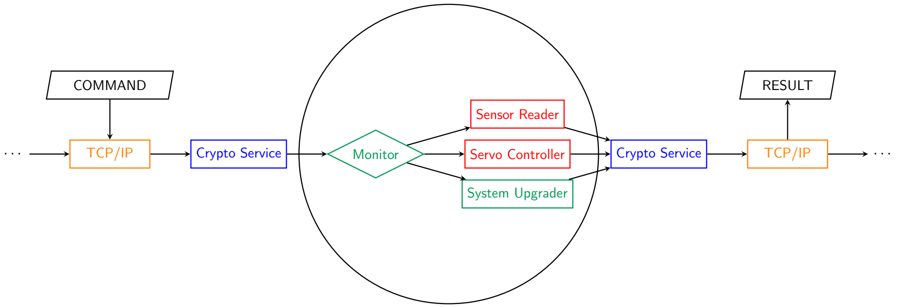
Time Slice Capability



- hartid – ID of a hardware thread.[3]
- begin – start of a time slice
- free – start of minor frame
- end – end of a time slice and minor frame

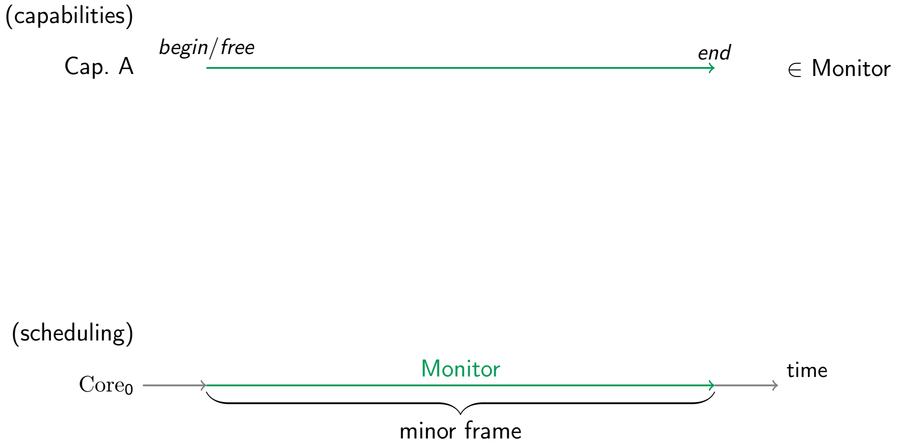
[3]Hardware Thread – Logically separate processor.

Application with Time Slices



Application with Time Slice

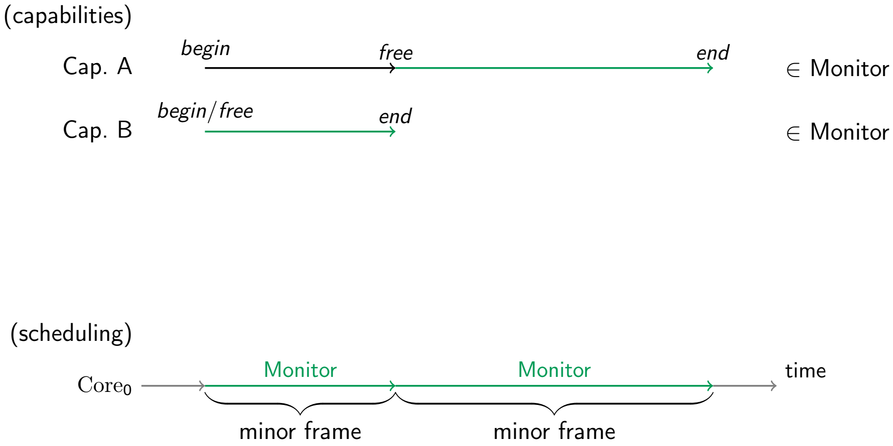
Monitor has the initial time slice



Application with Time Slice

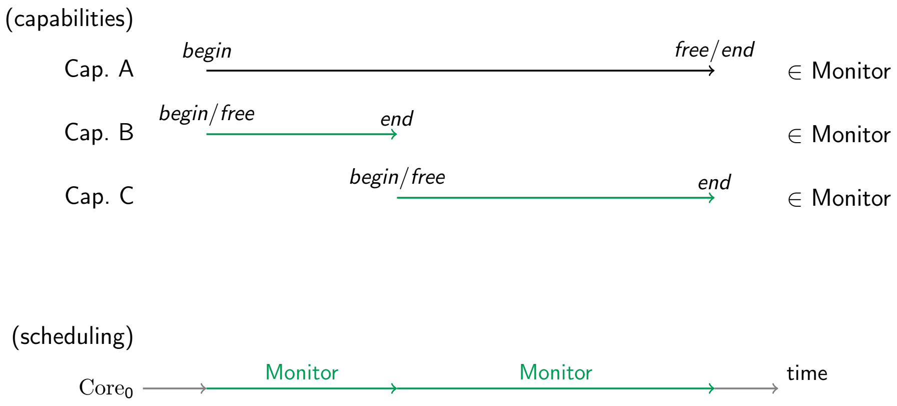
Monitor derives capability B

(only create slices from *free* to *end*)

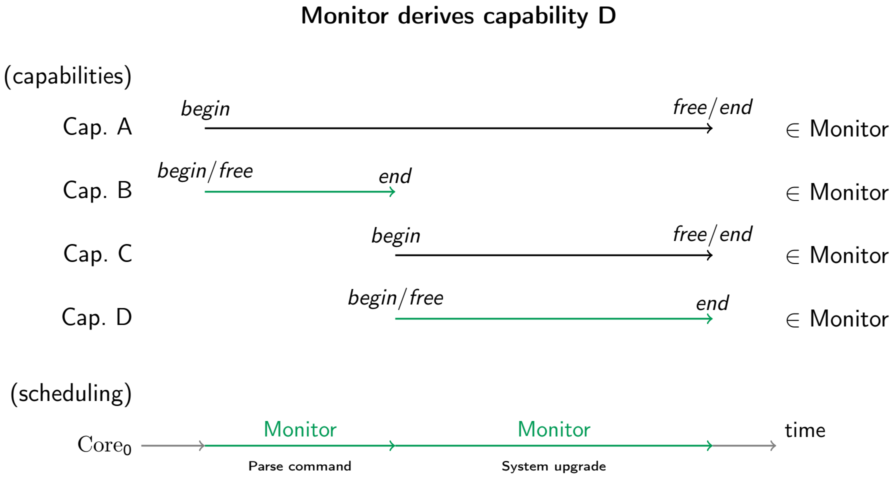


Application with Time Slice

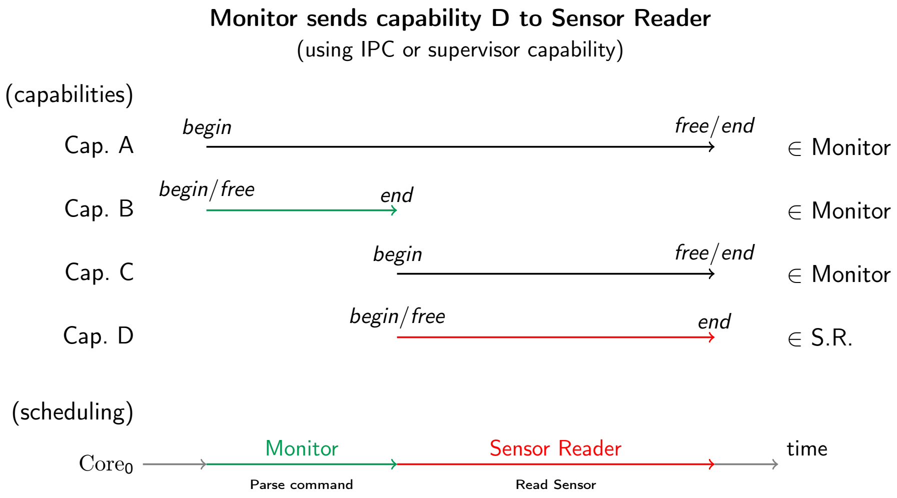
Monitor derives capability C



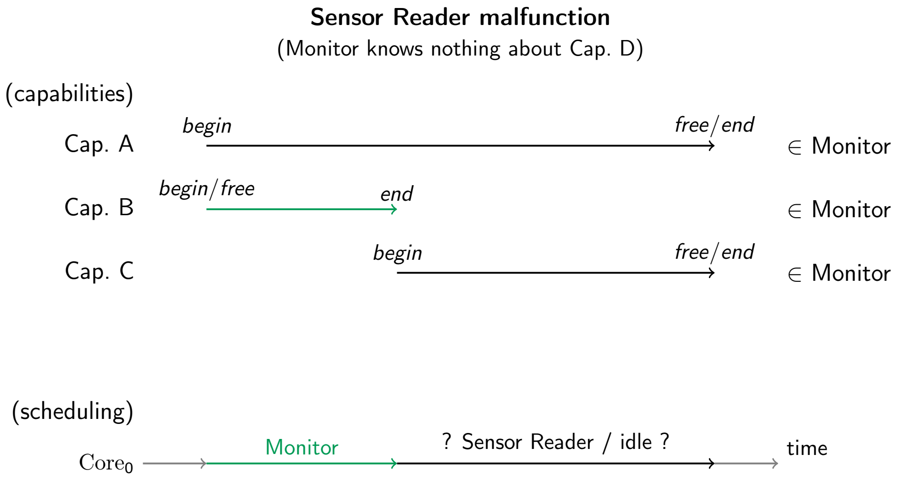
Application with Time Slice



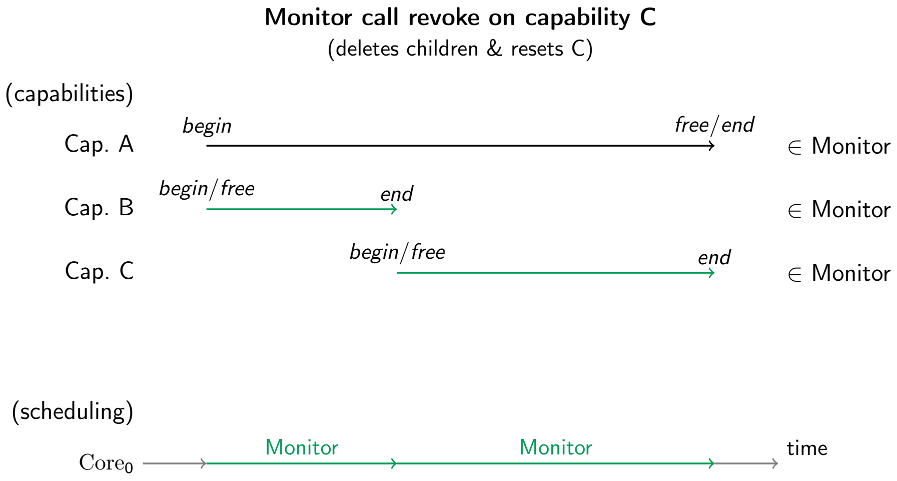
Application with Time Slice



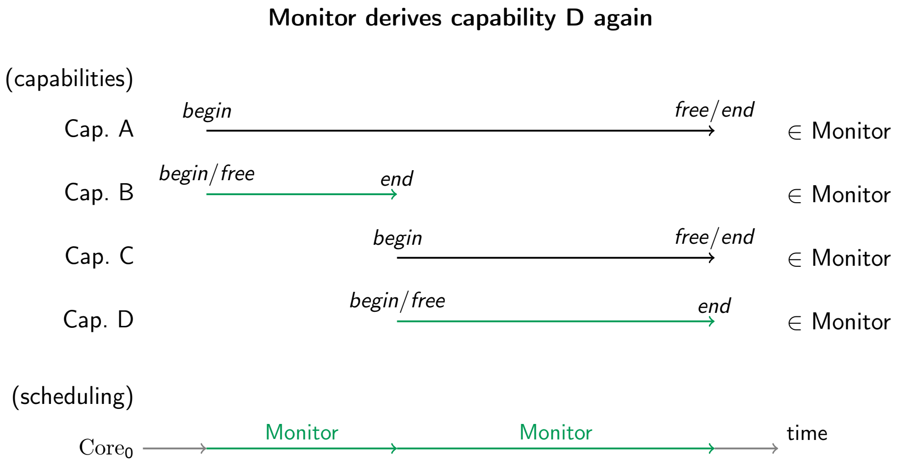
Application with Time Slice



Application with Time Slice

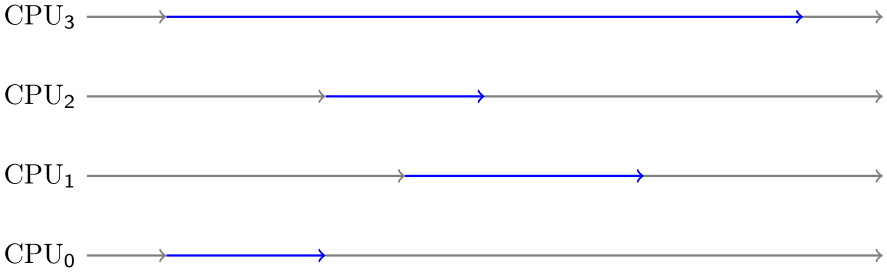


Application with Time Slice



Multicore Scheduling

Where does the blue process execute?



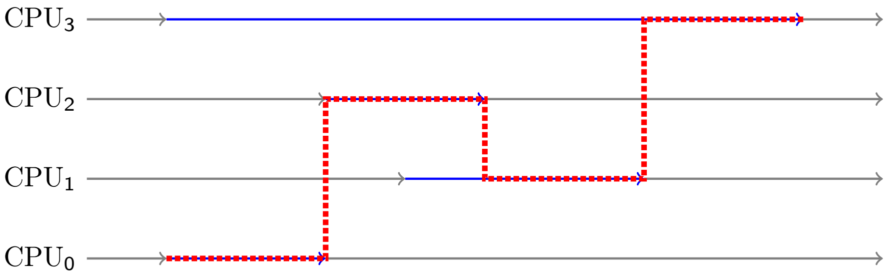
Process runs on one core, for determinism,[4] we need priority rules.

- currently running core
- core with smallest ID.

[4]non-determinism may leak information

Multicore Scheduling

Where does the blue process execute?



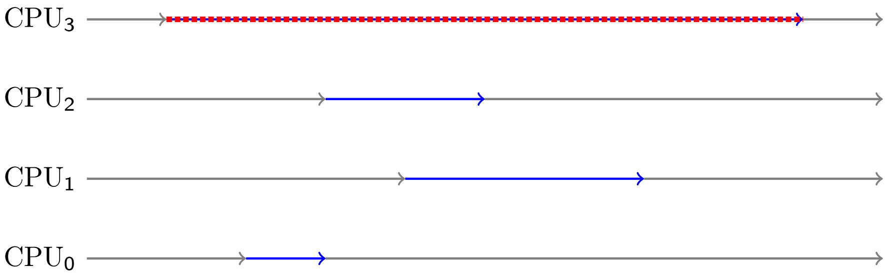
Process runs on one core, for determinism,[4] we need priority rules.

- currently running core
- core with smallest ID.

[4]non-determinism may leak information

Multicore Scheduling

Where does the blue process execute?



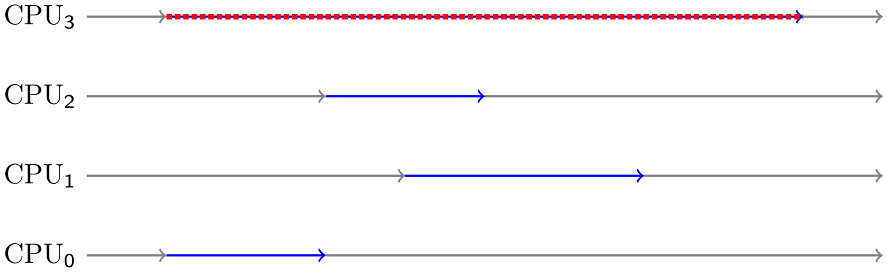
Process runs on one core, for determinism,[4] we need priority rules.

- currently running core
- core with smallest ID.

[4]non-determinism may leak information

Multicore Scheduling

Where does the blue process execute?



Process runs on one core, for determinism,[4] we need priority rules.

- currently running core
- longest quantum (not implemented)
- core with smallest ID

[4]non-determinism may leak information

Multicore HolBA

Initial values

$\{x = 0; y = 0\}$

P0	P1
(a) $y = 1$	(c) $r1 = x$
(b) $x = 1$	(d) $r2 = y$

Multicore HolBA

Initial values

$\{x = 0;\ y = 0\}$

$$\begin{tikzcd}
\underline{\text{P0}} & \underline{\text{P1}} \\
\text{(a) } y = 1 & \text{(c) } r1 = x \\
\text{(b) } x = 1 \arrow[ur] & \text{(d) } r2 = y
\end{tikzcd}$$

(c) reads "$x = 1$" from (b).
(d) reads?

Multicore HolBA

Initial values

$$\{x = 0;\ y = 0\}$$

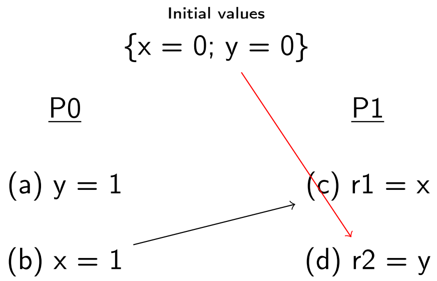
$$\begin{tikzcd}
\underline{\text{P0}} & \underline{\text{P1}} \\
\text{(a) } y = 1 \arrow[dr] & \text{(c) } r1 = x \\
\text{(b) } x = 1 \arrow[ur] & \text{(d) } r2 = y
\end{tikzcd}$$

(c) reads "x = 1" from (b)

$$\Longrightarrow$$

(d) reads "y = 1" from (a)?

Multicore HolBA

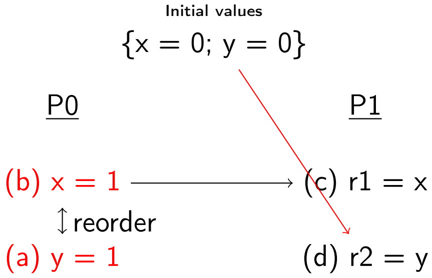


RISC-V lets writes appear *reordered* for *other cores*.[5]

(d) can read initial value, "y = 0"!

[5] All operations always appear in-order for the local core.

Multicore HolBA



Writes reordered.

Multicore HolBA

$$\begin{tikzcd}
 & \text{Initial values} & \\
 & \{x = 0;\ y = 0\} \arrow[ddr] & \\
\underline{\text{P0}} & & \underline{\text{P1}} \\
\text{(a) } y = 1 & & \text{(d) } r2 = y \arrow[d, leftrightarrow, "\text{reorder}"] \\
\text{(b) } x = 1 \arrow[rr] & & \text{(c) } r1 = x
\end{tikzcd}$$

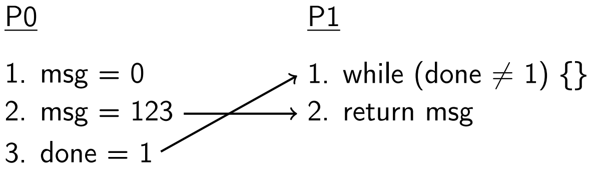
Reads reordered.

Multicore HolBA

$$\begin{tikzcd}
 & \text{Initial values} & \\
 & \{x = 0; y = 0\} \arrow[ddr] & \\
\underline{\text{P0}} & & \underline{\text{P1}} \\
\text{(a) } y = 1 & & \text{(d) } r2 = y \arrow[d, leftrightarrow, "\text{reorder}"] \\
\text{(b) } x = 1 \arrow[rr] & & \text{(c) } r1 = x
\end{tikzcd}$$

Verification of multicore RISC-V need all reorderings.
Multicore HolBA provides this!

Separation Kernel Example



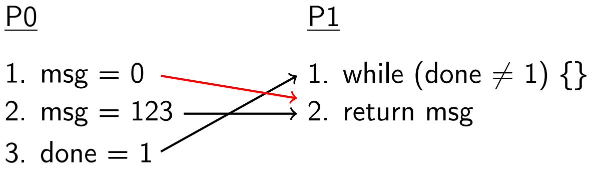
P0

1. msg = 0
2. msg = 123
3. done = 1

P1

1. while (done $\neq$ 1) {}
2. return msg

Separation Kernel Example



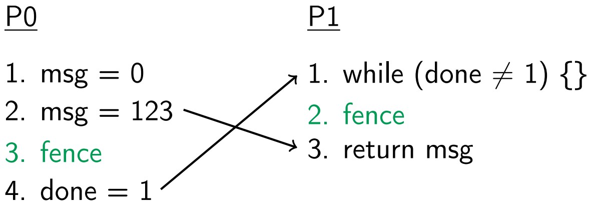
P0

1. msg = 0
2. msg = 123
3. done = 1

P1

1. while (done $\neq$ 1) {}
2. return msg

Separation Kernel Example (fix)



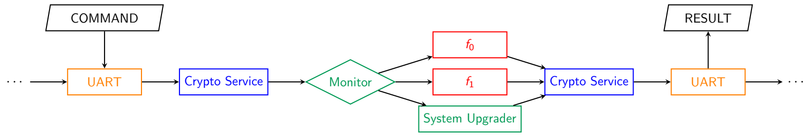
P0

1. msg = 0
2. msg = 123
3. fence
4. done = 1

P1

1. while (done $\neq$ 1) {}
2. fence
3. return msg

Research Plan



- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept

[5] Worst-case execution time

Research Plan

- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept
- Spring 2023
  - Publication of multicore HolBA
  - Workshop publication of OpenMZ (older kernel)
  - Measurements of WCET[5] and jitter of non-preemptive kernel parts
  - Implement secure interrupts, optimized scheduler, and 32-bit kernel version

[5] Worst-case execution time

Research Plan

- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept
- Spring 2023
  - Publication of multicore HolBA
  - Workshop publication of OpenMZ (older kernel)
  - Measurements of WCET[5] and jitter of non-preemptive kernel parts
  - Implement secure interrupts, optimized scheduler, and 32-bit kernel version
- Summer 2023
  - Model and proofs on some concurrent kernel code using multicore HolBA
  - Finish sequential high-level model of kernel

[5]Worst-case execution time

Research Plan

- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept
- Spring 2023
  - Publication of multicore HolBA
  - Workshop publication of OpenMZ (older kernel)
  - Measurements of WCET[5] and jitter of non-preemptive kernel parts
  - Implement secure interrupts, optimized scheduler, and 32-bit kernel version
- Summer 2023
  - Model and proofs on some concurrent kernel code using multicore HolBA
  - Finish sequential high-level model of kernel
- Autumn 2023 – Finish concurrent high-level model of kernel

[5] Worst-case execution time

Find out more

https://kth-step.github.io/projects/separation-kernel/