Designing a context switching service for Pip

Florian Vanhems
June 16, 2019 - ENTROPY 2019

Lille University, France
Pip’s overview
A minimalist kernel

Designed to provide *formally proven* memory isolation to applications.
Memory partitioning

- Shell
- Linux
- FreeRTOS
- Multiplexer
Usual software attacks target the control flow (e.g. buffer overflow)
Usual control flow transfers
System calls

Shell

asks to open a file

Linux

file is opened

FreeRTOS

Multiplexer
Shell asks to open a file

file is opened

Multiplexer

Linux

FreeRTOS
Signal sending

Shell

sends a SIGINT

Linux

FreeRTOS

Multiplexer
Signal sending

Shell

Linux

calls exit()

sends a SIGINT

FreeRTOS

Multiplexer
A versatile context switching service
The service **unifies** all the previously showed control flow transfers and was designed to reduce the isolation proof.

Allows to save your own CPU state before transferring the control flow, and restore CPU states from the target.
A virtualization of the IDT

Per partition structure holding CPU state pointers

*Accessible* to userland code.
Service illustration

Caller's memory

Callee's memory

VIDT

X

VIDT

CPU state
Caller's frozen context
Service illustration

Caller’s memory

Callee’s memory

VIDT

Callers memory

Caller’s frozen context

Callee’s stored context

CPU state

Caller’s frozen context

8
Service illustration

 Caller’s memory

 VIDT

 X

 CPU state
 Caller’s frozen context

 Callee’s memory

 VIDT

 CPU state
 Callee’s stored context
Service illustration

Caller’s memory

Callee’s memory

CPU state
Callee’s stored context
About the proof

Unification *eases* the proof

Isolation proof is almost done, we expect no significant obstacles on the way.

Our intention is to prove the functional correctness of the function, but that has not started yet.
The service was written in Gallina (with imperative style), uses a shallow embedding to produce C code.
Some figures

- Service ~340 LoC
- Initial isolation proof ~1800 LoP
- (about 80% of the proof uses lemmas already proven for our project)
- Initial isolation proof ~3 weeks
Questions?
Memory Partitioning

Shell

Linux

FreeRTOS

Multiplexer

Children can't access the memory that wasn't shared with them.

A child can't access the memory of another (the shared memory is disjoint).

Root partition

Parent of both process partitions

Still has access to the memory shared to its children

Root partition

Created from the memory of the root partition

Another child partition

Created from the memory of the root partition

Child partition

Has access to the whole memory

Root partition

Created from the memory of the root partition

Child partition
Memory Partitioning

- **Root partition** has access to the whole memory.

  - Root partition is the parent of both process partitions.
  - Children can't access the memory that wasn't shared with them.
  - A child can't access the memory of another (the shared memory is disjoint).

  - Multiplexer
    - Shell
      - Linux
      - FreeRTOS
Memory Partitioning

- **Root partition**: Has access to the whole memory.
- **Child partition**: Created from the memory of the root partition.
- **Multiplexer**: Parent of both process partitions and still has access to the memory shared to its children. Children can't access the memory that wasn't shared with them. A child can't access the memory of another (the shared memory is disjoint).

- **Shell**, **Linux**, and **FreeRTOS** are depicted as nodes in the diagram.
Memory Partitioning

- **Shell**
- **Linux**
- **FreeRTOS**
- **Multiplexer**
- **Root partition**
- Another child partition
  Created from the memory of the root partition
  Parent of both process partitions
  Still has access to the memory shared to its children
  Children can't access the memory that wasn't shared with them
  A child can't access the memory of another (the shared memory is disjoint)
  Another child partition
  Created from the memory of the root partition
Memory Partitioning

Root partition

Has access to the whole memory

Child partition

Created from the memory of the root partition

Another child partition

Created from the memory of the root partition

Parent of both process partitions

Still has access to the memory shared to its children

Root partition

Children can't access the memory that wasn't shared with them

A child can't access the memory of another (the shared memory is disjoint)

Shell

Linux

FreeRTOS

Multiplexer
Memory Partitioning

- **Shell**
- **Linux**
- **FreeRTOS**

- **Root partition**
  - Has access to the whole memory
  - Parent of both process partitions
  - Still has access to the memory shared to its children

- **Children**
  - Can’t access the memory that wasn’t shared with them

- Multiplexer
  - A child can’t access the memory of another (the shared memory is disjoint)
  - Created from the memory of the root partition
Memory Partitioning

Shell

Linux

FreeRTOS

Multiplexer

A child can’t access the memory of another (the shared memory is disjoint)

Root partition

Children can’t access the memory that wasn’t shared with them

A child can’t access the memory of another (the shared memory is disjoint)
What to do with hardware interrupts?

Usually the kernel configures the IDT.

![Diagram showing IDT and Kernel](image-url)
What to do with hardware interrupts?

No multiplexer - Pip can’t handle the interrupts

IDT

Kernel priviledges

User priviledges

Multiplexer
Interrupt Descriptor Table and partitions

We can’t let partitions configure the IDT at will.

- they could bypass the kernel
- unique handler per interrupt
Configuring the IDT

No multiplexer - Pip can’t handle the interrupts

IDT

Pip

Multiplexer

Kernel privileges

User privileges