Designing a context switching service for Pip

Florian Vanhems

June 16, 2019 - *ENTROPY 2019*

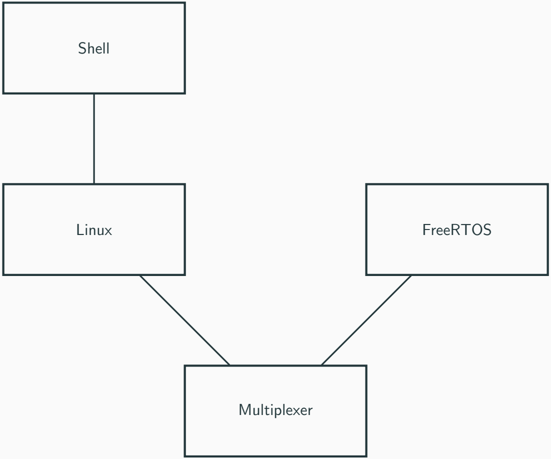
Lille University, France

Pip's overview

A minimalist kernel

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

Memory partitioning

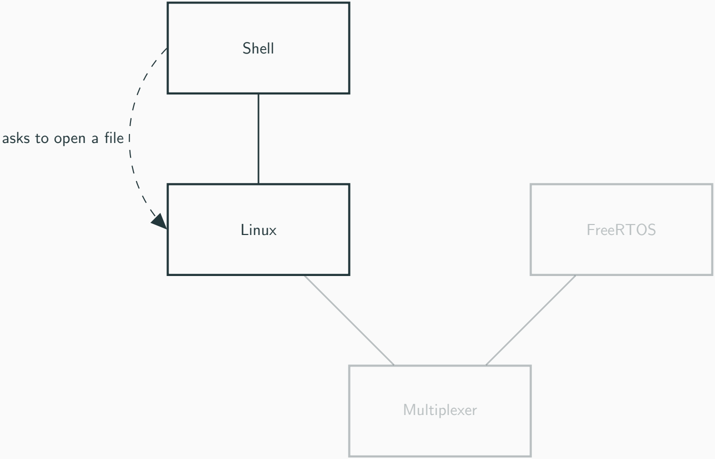


Stakes of context switching

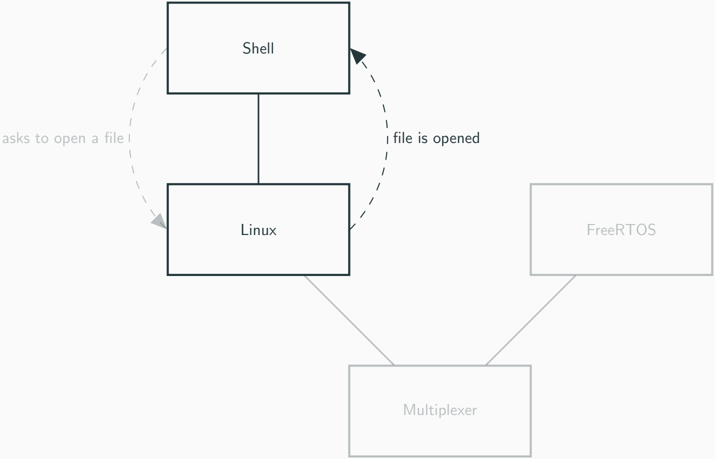
Usual software attacks target the control flow (e.g. buffer overflow)

Usual control flow transfers

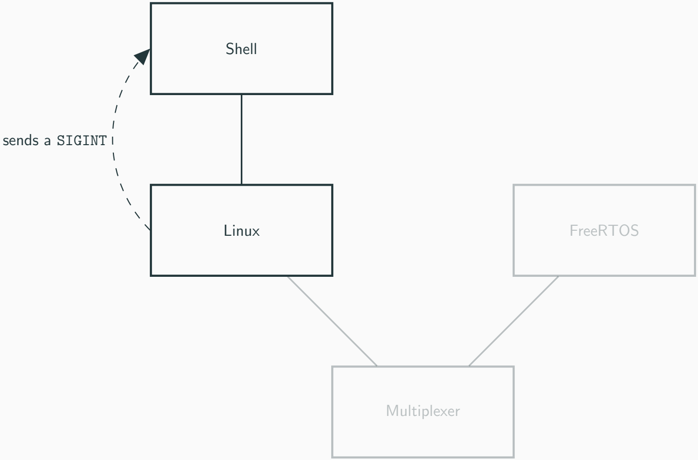
System calls



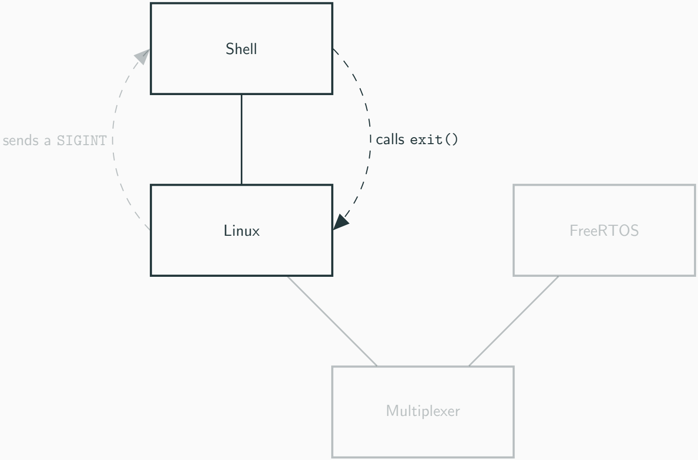
System calls



Signal sending



Signal sending



A versatile context switching service

Versatile context switching service

The service **unifies** all the previously showed control flow transfers and was designed to reduce the isolation proof.

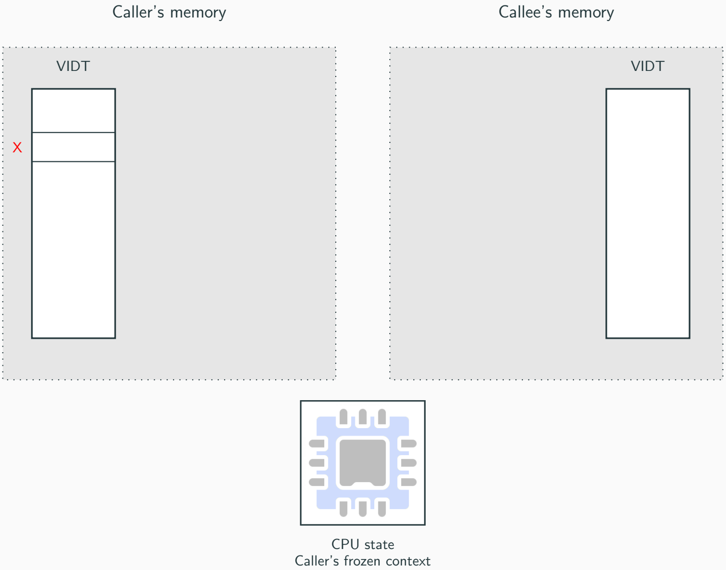
Allows to save your own CPU state before transfering the control flow, and restore CPU states from the target

A virtualization of the IDT

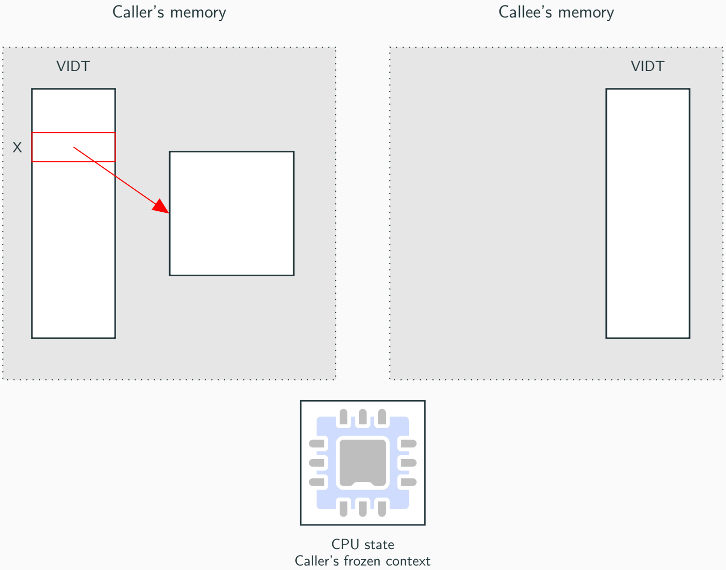
Per partition structure holding CPU state pointers

Accessible to userland code.

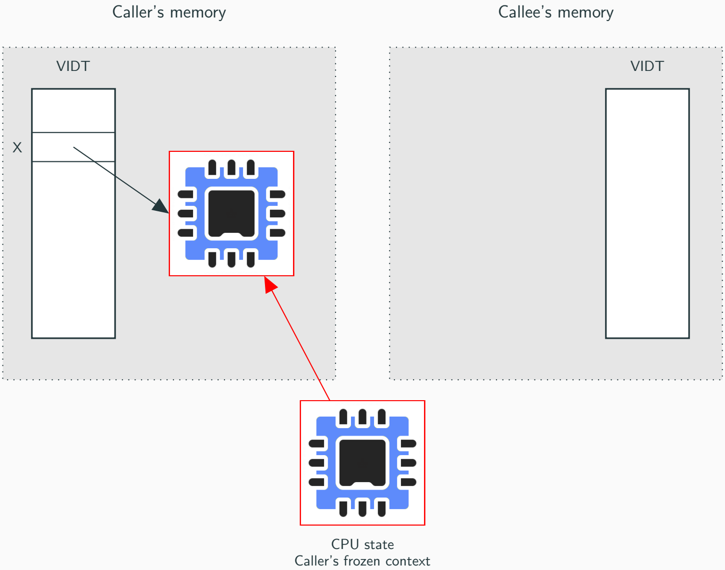
Service illustration



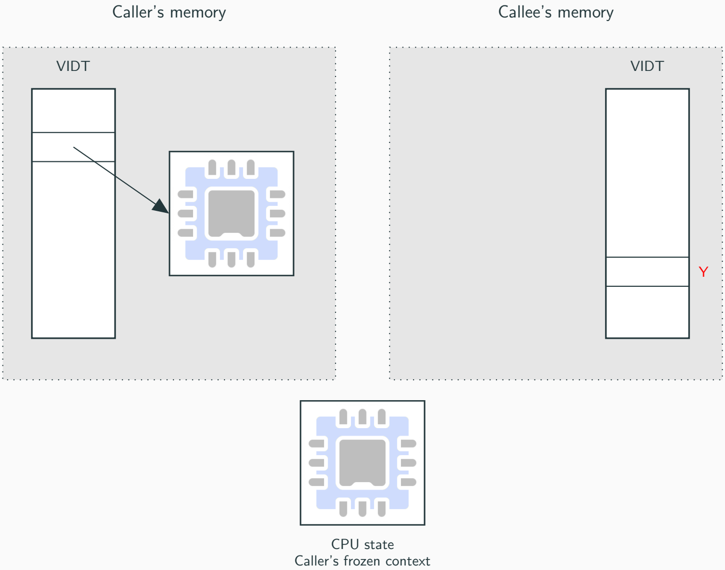
Service illustration



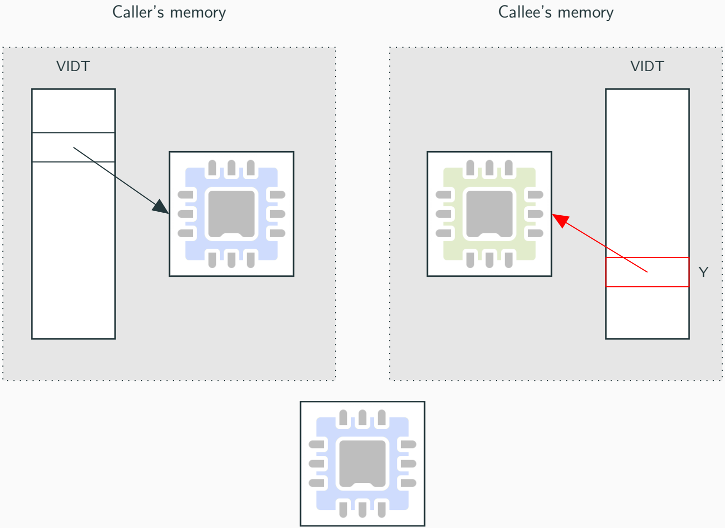
Service illustration



Service illustration

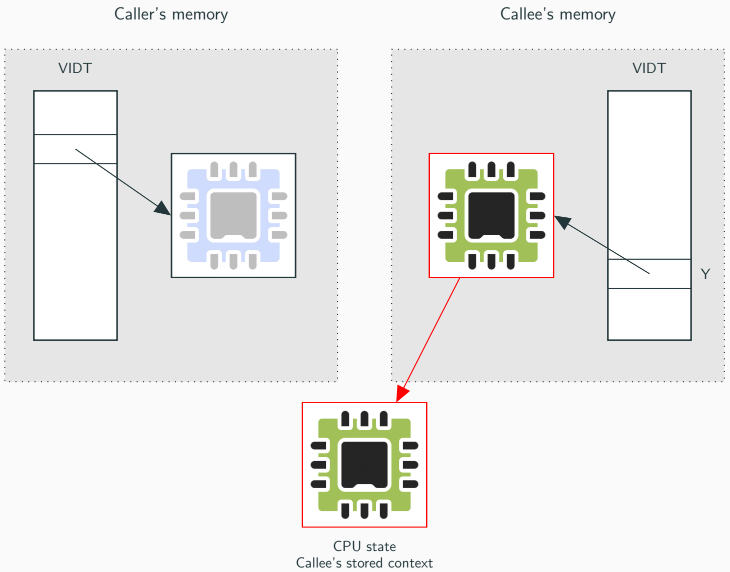


Service illustration



CPU state
Caller's frozen context

Service illustration



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.

Methodology

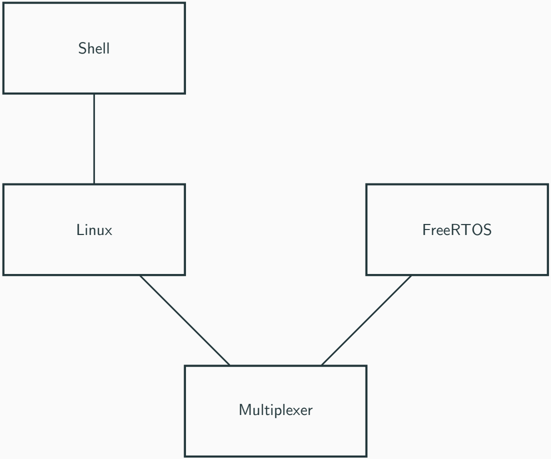
The service was written in Gallina (with imperative style), uses a shallow embedding to produce C code.

Some figures

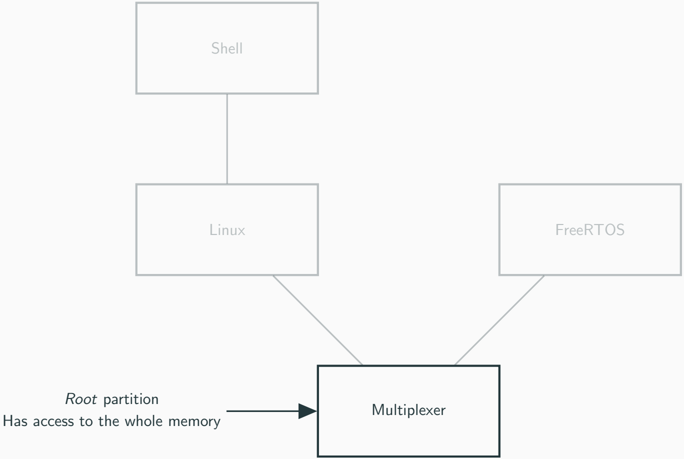
- Service ˜340 LoC
- Initial isolation proof ˜1800 LoP
- (about 80% of the proof uses lemmas already proven for our project)
- Initial isolation proof ˜3 weeks

Questions?

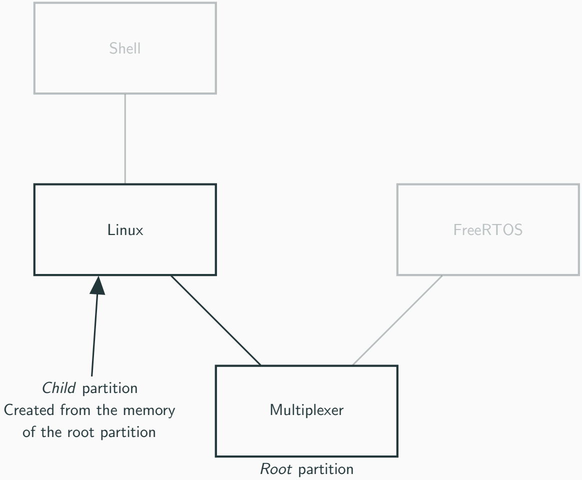
Memory Partitioning



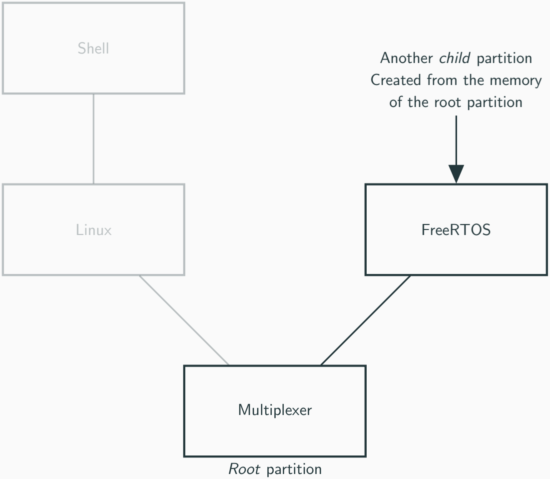
Memory Partitioning



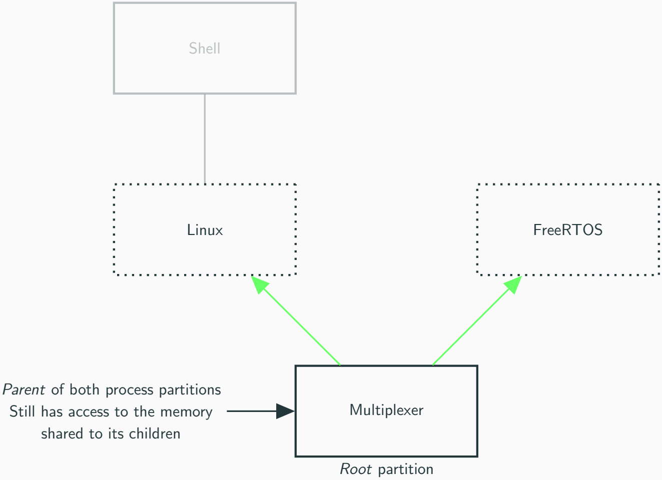
Memory Partitioning



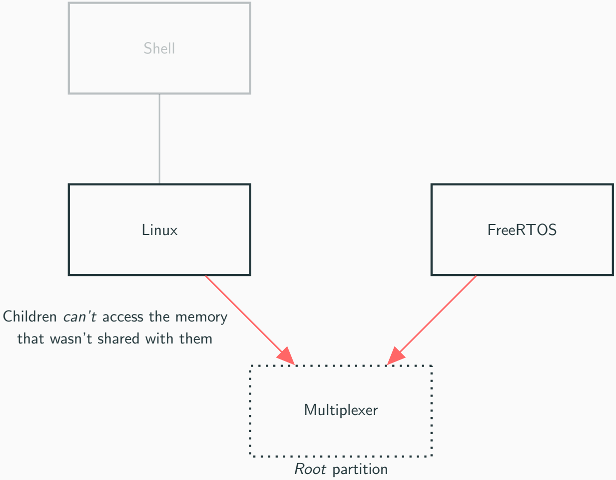
Memory Partitioning



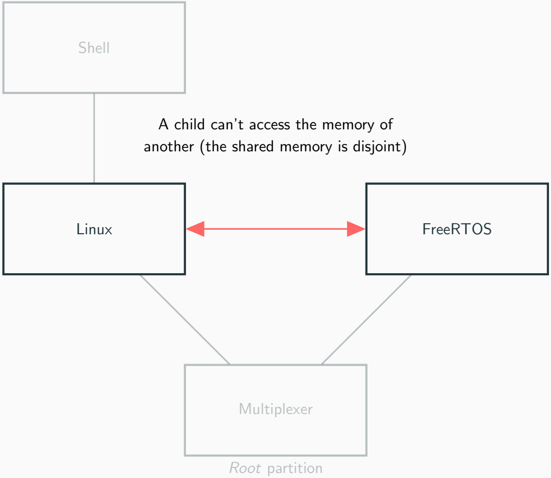
Memory Partitioning



Memory Partitioning

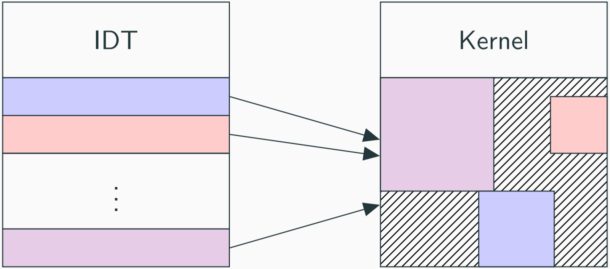


Memory Partitioning



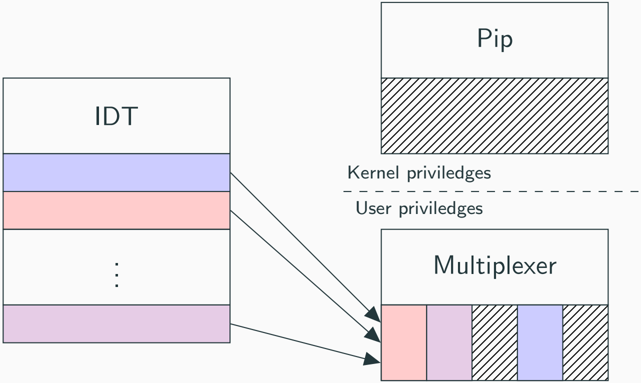
What to do with hardware interrupts ?

Usually the kernel configures the IDT.



What to do with hardware interrupts ?

No multiplexer - Pip can't handle the interrupts



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