The Pip Protokernel

Narjes Jomaa, David Nowak
Joint work with the Pip team

Pip Club Meeting 2018

December 7, 2018

This work is partially supported by the European Celtic-Plus Project ODSI C2014/2-12.

The Pip protokernel: a brief system overview (*David Nowak*)

Pip design principles and security properties (*Narjes Jomaa*)

The Pip protokernel: a brief system overview (*David Nowak*)

Pip design principles and security properties (*Narjes Jomaa*)

On-Demand Secure Isolation

- This research is part of the European project ODSI.
  - Led by Orange
  - 1 academic partner: The university of Lille
  - 8 industrial partners from France, Romania, and Spain
- In Lille: 3 PhD students and 1 postdoctoral researcher.
- The Pip protokernel is one of the foundations of this project.
- Security protocols are designed on top of Pip.
- Case studies by industrial partners: IoT, M2M, SCADA
- Common Criteria certification

Memory isolation between applications

Why? For safety and security

How? By software (OS kernel), and hardware (MMU, CPU kernel mode)

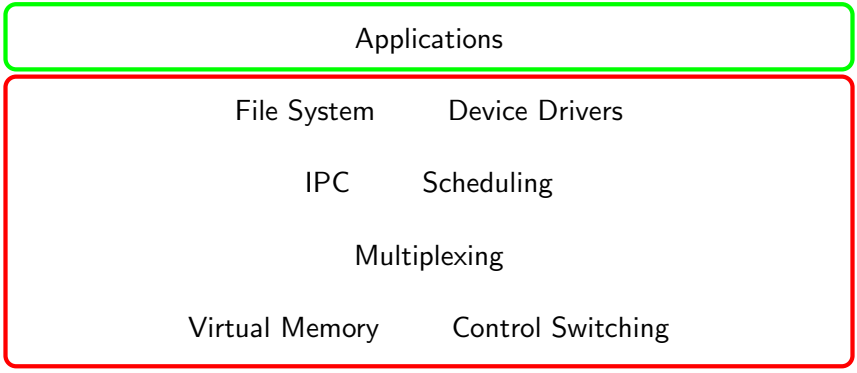
Correct? Ensured by a formal proof in Coq

Feasible? Yes, by reducing the trusted computing base to its bare bone

$$\text{reducing the TCB} \Rightarrow \text{increasing feasibility of a formal proof} \ \& \ \text{reducing the attack surface}$$

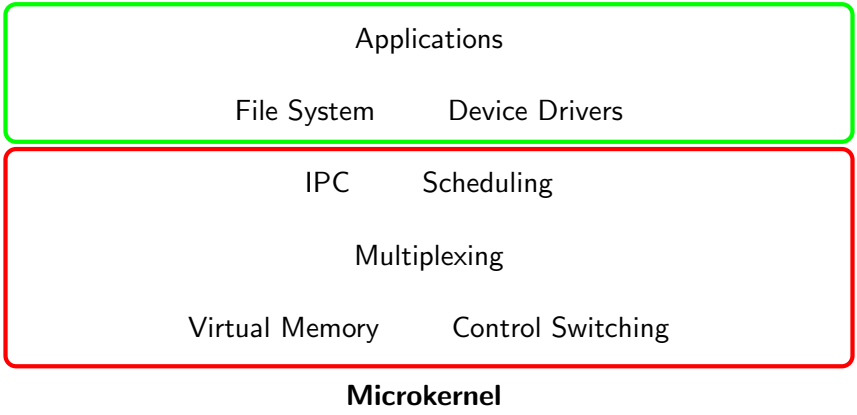
$$\text{simplifying the specification language} \Rightarrow \text{increasing feasibility of verified translation to C}$$

From monolithic kernel to the Pip protokernel

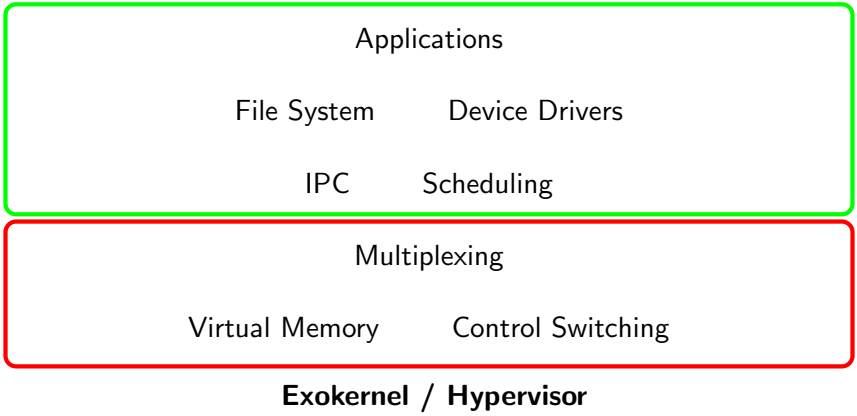


Monolithic Kernel

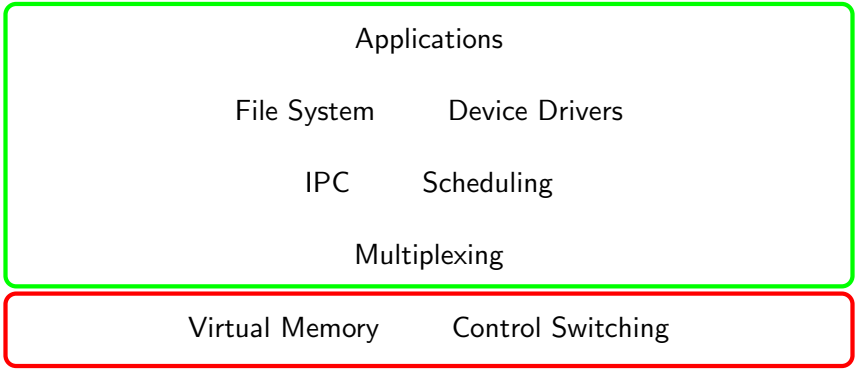
From monolithic kernel to the Pip protokernel



From monolithic kernel to the Pip protokernel



From monolithic kernel to the Pip protokernel

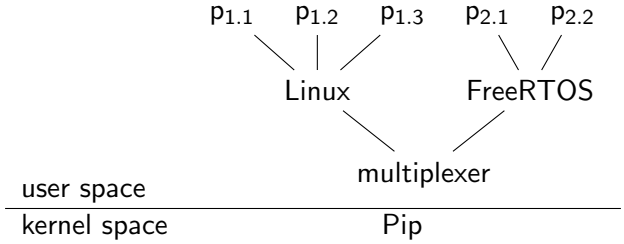


The Pip protokernel

Partition tree

Pip organizes the memory into hierarchical partitions.

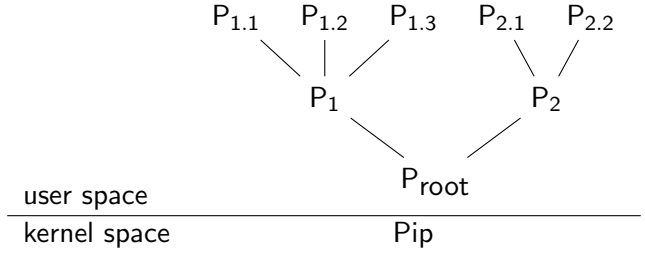
Example



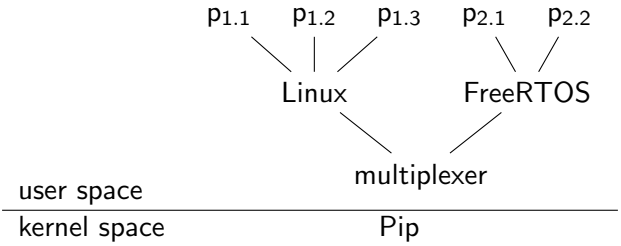
Partition tree: the point of view of Pip

The contents of each partition is not relevant for Pip.

- **Horizontal isolation**
  Partitions in different subtrees are isolated from each other, e.g. $P_{1.1}$ cannot access memory of $P_{1.2}$ or P_2.
- **Vertical sharing**
  A partition has access to the memory of its descendants.
- **Kernel isolation**
  Pip is isolated from all partitions.



Partition tree: dealing with interrupts



- **Software interrupts**
  - Pip deals with software interrupts to itself,
    e.g. FreeRTOS asks Pip to create a new partition.
  - Pip forwards other software interrupts to the caller's parent,
    e.g. $p_{1.2}$ make a system call to Linux.
- Pip forwards **hardware interrupts** to the root partition,
  e.g. a network packet has arrived.

Pip system calls

10 elementary system calls

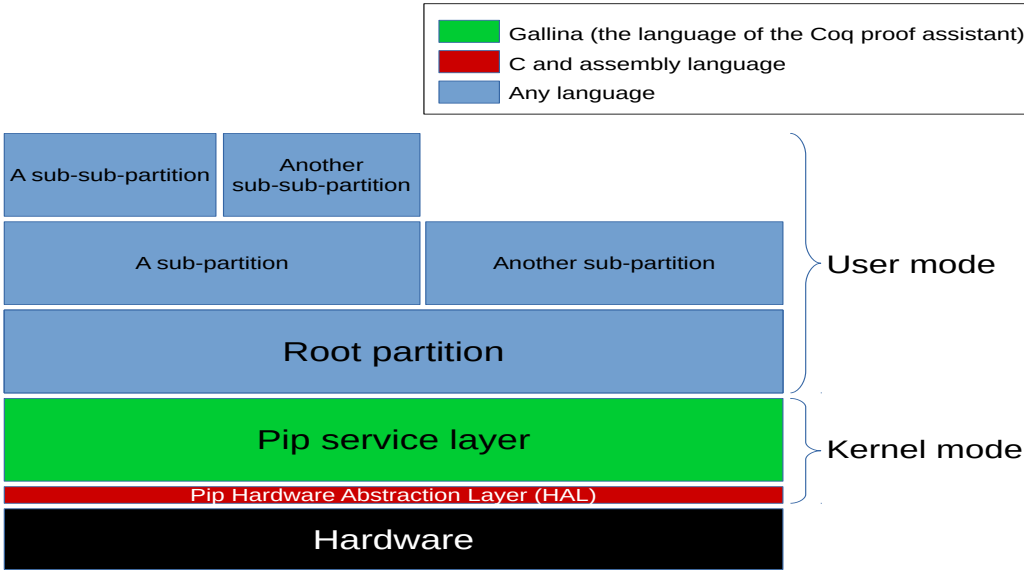
- **Memory management**

`createPartition`	creates a child partition
`removePartition`	deletes a child partition
`addVaddr`	lends a memory page to a child
`removeVaddr`	removes a memory page from a child
`pageCount`	the number of needed configuration pages
`prepare`	gives needed configuration pages
`collect`	takes back unused configuration pages
`mappedInChild`	returns the child using a given page

- **control switching**

`dispatch`	notifies a partition about an interrupt
`resume`	restores the context of a partition

Software layers



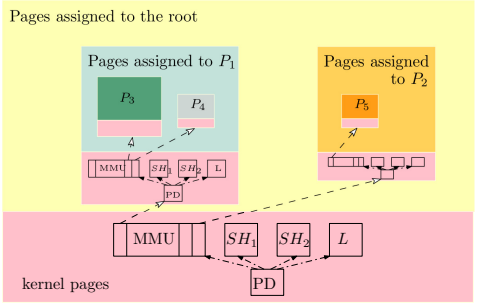
Applications

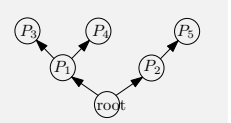
- The HAL of Pip has been ported to:
  - QEMU (x86)
  - x86
  - The Galileo board (Intel Pentium-compliant embedded board)
- Kernels ported on Pip
  - FreeRTOS: Tasks can be isolated in sibling partitions.
  - Linux 4.10.4: More involved because Linux configures MMU.
- Porting a kernel to Pip essentially consists of:
  - removing privileged instructions and operations, and
  - replacing them with system calls to Pip (paravirtualization).
- Drhystone benchmark: low overhead of 2,6% in terms of CPU cycles

The Pip protokernel: a brief system overview (*David Nowak*)

Pip design principles and security properties (*Narjes Jomaa*)

Partition tree management





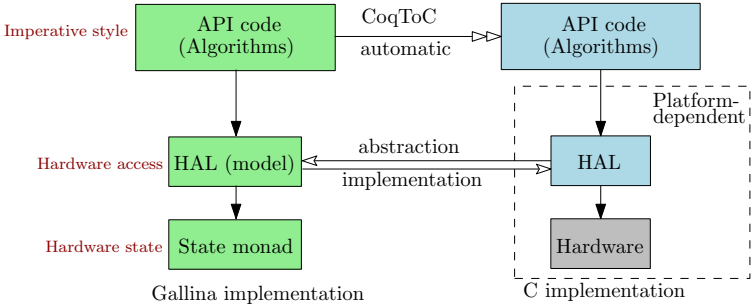
The configuration of a partition

- Partition descriptor (*PD*)
- MMU tables
- Shadow 1 (*SH1*) and Shadow 2 (*SH2*)
- Linked list (*L*)

Data structure of partitions

- MMU structure: Define assigned pages and access control
- Mirror the MMU structure
  - Shadow 1: Find out which pages are assigned to children and which pages are used as a partition descriptor identifier *(security)*
  - Shadow 2: Ease getting back the ownership of assigned pages *(efficiency)*
- List (L): Ease getting back the ownership of pages lent to the kernel *(efficiency)*

Pip design principles



- Hardware state: the part that is relevant to model the partition tree
  - the partition that is currently active
  - the physical memory where Pip stores its own data

Sample translation

– Low-level HAL primitives
– Higher-level monadic code

Pip monadic code in Coq

```
Definition getFstShadow (partition : page) : LLI page :=
  perform idx := getSh1idx in
  perform idxSucc := Index.succ idx in
  readPhysical partition idxSucc.
```

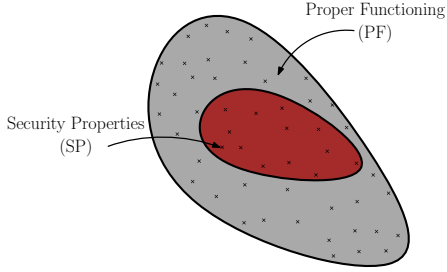
Its generated translation to C

```
uintptr_t getFstShadow (const uintptr_t partition) {
  const uint32_t idx = getSh1idx ();
  const uint32_t idxSucc = succ (idx);
  return readPhysical (partition, idxSucc); }
```

Security properties

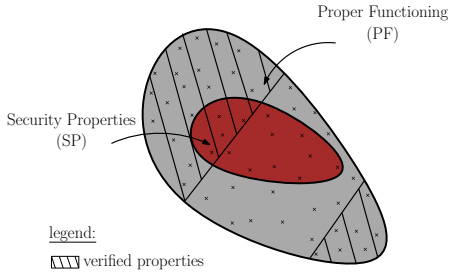
Classification of properties

Proper functioning : a property related to a one or more service of the system
Security : a property that should be preserved by all the services of the system



- Prove all SP $\not\Rightarrow$ Prove all PF
- Prove all PF $\Rightarrow$ Prove all SP;

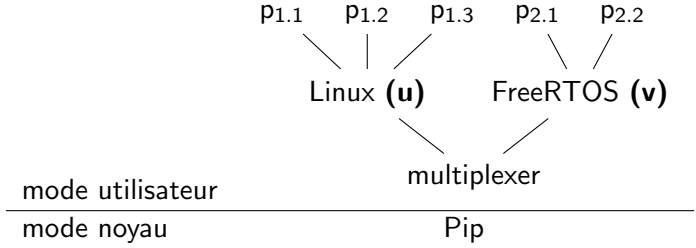
Reduce the number of properties to prove



- identify security properties;
- identify the subset of correctness properties that shoud be verified to ensure security properties

✓ Non interference property[1]

"a security domain **u** *is noninterfering with domain* **v** *if no action performed by* **u** *can influence subsequent outputs seen by v."*

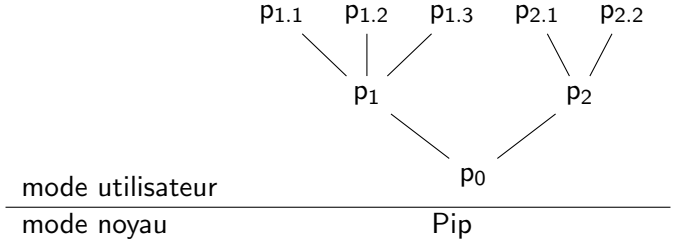


[1] John Rushby. *Noninterference, transitivity, and channel-control security policies*. SRI International, Computer Science Laboratory, 1992.

Reduce to elementary properties

✓ Non interference property

"a security domain **u** *is noninterfering with domain* **v** *if no action performed by* **u** *can influence subsequent outputs seen by v."*



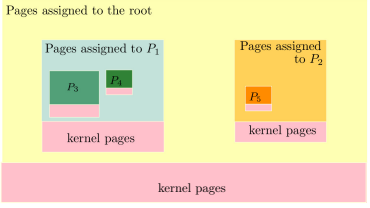
☞ isolation property ensured by Pip does not depend on actions performed by partitions

☞ reason about the memory of partitions

☞ isolation properties imply non interference property

The horizontal isolation property

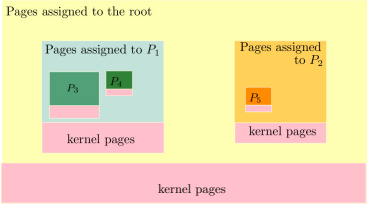
```
Definition HI s: Prop :=

∀ parent child1 child2 : page,

  parent ∈ (partitionTree s)→

  child1 ∈ (children parent s) →

  child2 ∈ (children parent s) →

  child1 ≠ child2 →

  (allocatedPages child1 s) ∩ (allocatedPages child2 s) = ∅.
```



- Sibling partitions cannot access each others memory.

Hierarchical TCB (vertical sharing)

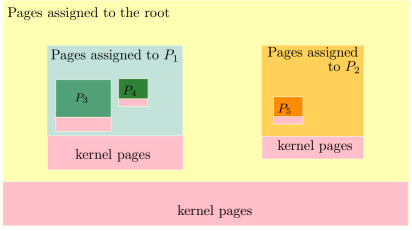
```
Definition VS s : Prop :=
∀ parent child : page,
 parent ∈ ( partitionTree s) →
 child ∈ (children parent s) →
 (allocatedPages child s) ⊆ (assignedPages parent s).
```



- All the pages allocated for a partition are included in the pages assigned to its ancestors

The kernel isolation property

```
Definition KI s: Prop :=
∀ partition1 partition2 : page,
  partition1 ∈ (partitionTree s) →
  partition2 ∈ (partitionTree s) →
  (ownedPages partition1 s) ∩ (kernelPages partition2 s) = ∅.
```



- No partition can access to the pages owned by the kernel.

Verification approach

Verification approach

Hoare logic on top of the LLI (Low Level Interface) monad

```
Definition hoareTriple {A : Type} (P : state → Prop) (m : LLI A)
(Q : A → state → Prop) : Prop :=
∀ s, P s → match m s with
  | val (a, s') ⇒ Q a s'
  | undef _ _⇒ False
  end.
```

States that if the precondition holds then

- the postcondition holds; and
- there is no undefined behavior

```
{{P}} m {{Q}}
```

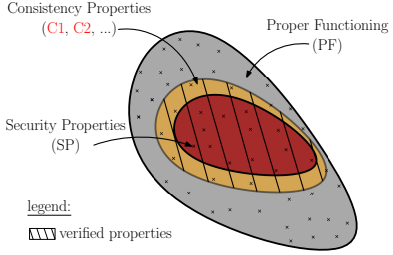
☞ `{{HI & VS & KI}} API_service {{HI & VS & KI}}`

The need of consistency properties

✕ {{HI & VS & KI}} API_service {{HI & VS & KI}}

☞ consistency : C1, C2 ... ≈ well-formedness of Pip's data structures

{{HI & VS & KI & C1 & C2 & ..}} API_service {{HI & VS & KI & C1 & C2 & ..}}



- discovered incrementally during the proof
- must be preserved in order for isolation to be preserve

Example: createPartition invariant

✓ {{HI & VS & KI & C}} createPartition v1 v2 v3 v4 v5 {{HI & VS & KI & C}}

Proceed forward using transitivity (1/2)

✓ {{HI & VS & KI & C}} createPartition v1 v2 v3 v4 v5 {{HI & VS & KI & C}}

{{HI & VS & KI & C}}

```
perform currentPart := getCurPartition in
perform ptv1FromPD := getTableAddr currentPart v1 nbL in
        ...
if negb accessv1 then ret false else
writeAccessible ptv1FromPD idxv1 false ;;
...
```

{{HI & VS & KI & C}}

Proceed forward using transitivity (2/2)

✓ {{HI & VS & KI & C}} createPartition v1 v2 v3 v4 v5 {{HI & VS & KI & C}}

First sub-goal:

{{HI & VS & KI & C}}

getCurPartition

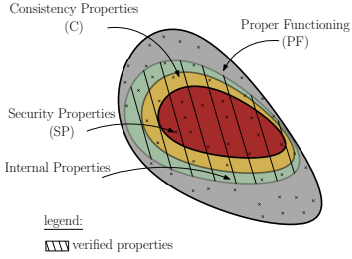
{{HI & VS & KI & C & P currentPart }}

Second sub-goal:

{{HI & VS & KI & C & P currentPart}}

```
perform ptv1FromPD := getTableAddr currentPart v1 nbL in
        ...
if negb accessv1 then ret false else
writeAccessible ptv1FromPD idxv1 false ;;
        ...
```

{{HI & VS & KI & C}}



☞ P currentPart : an internal property of createPartition

Verification overview

✓ one person year

Invariants	**lines of proof**
createPartition ($\approx 300 loc$)	$\approx$ 60000
createPartition + addVaddr ($\approx 110 loc$)	$\approx$ 78000
createPartition + addVaddr + mappedInChild ($\approx 40 loc$)	$\approx$ 78300
createPartition + addVaddr + mappedInChild + removeVaddr ($\approx 100 loc$)	$\approx$ 88300

Table: Size of the proof

Invariant	**duration**
createPartition	$\approx$ 10 months
addVaddr	$\approx$ 2 months
mappedInChild	$\approx$ 4 hours
removeVaddr	$\approx$ 2 weeks

Table: Duration of the verification

To find out more

http://pip.univ-lille1.fr