Solo5: A sandboxed, re-targetable execution environment for unikernels

Dan Williams (IBM Research), djwillia@us.ibm.com
Martin Lucina (robur.io / CCT), martin@lucina.net
Ricardo Koller (IBM Research), kollerr@us.ibm.com

FOSDEM 2019, Microkernel and Component-based OS devroom

Background: LibOS and unikernels

Library operating systems

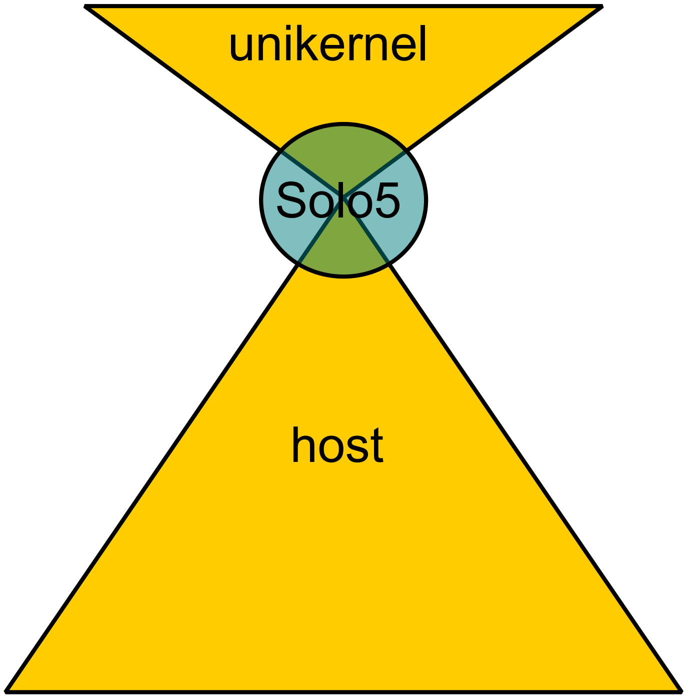
- A collection of **libraries** providing traditional OS functionality.
- No concept of process isolation.
- Generally use co-operative scheduling.

... these are combined at compile time with **application code** into a *unikernel*.

Unikernels

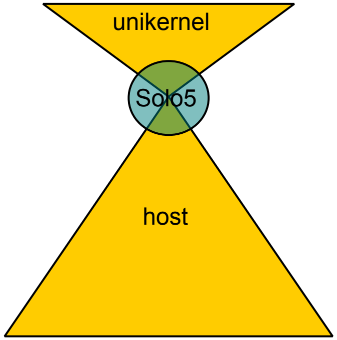
- **Minimal code size, minimal attack surface**.
- Single-purpose, single-application operating system.
- Perceived as something that must run in kernel space.

What is Solo5? (I)



What is Solo5? (II)

1. A minimalist, legacy-free ***interface***.



2. ***Bindings*** to this *interface* for:

    - microkernels (Genode), separation kernels (Muen)
    - virtio-based hypervisors
    - monolithic kernels (Linux, FreeBSD, OpenBSD)

3. On monolithic kernels a ***tender*** is used to strongly sandbox the unikernel:

    - `hvt`: hardware virtualized *tender*
    - `spt`: sandboxed process *tender*

What is Solo5? (III)

From the libOS point of view:

- "Middleware".
- **Integrated** into the libOS build system.
- **The developer does not interact with Solo5 directly.**

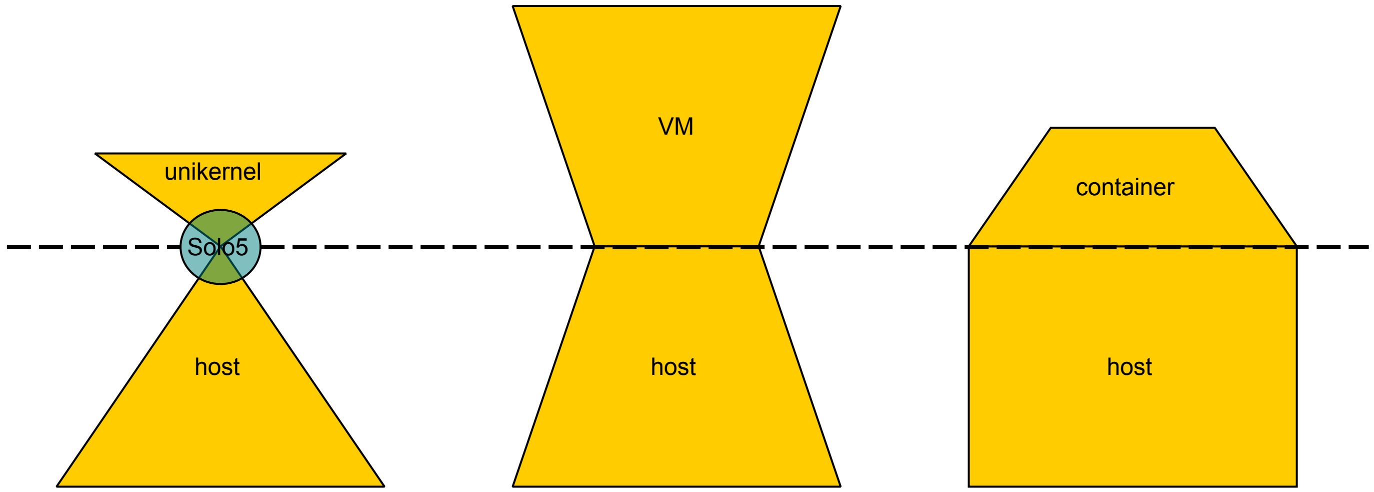
Example, for MirageOS:

```
mirage configure -t {hvt | spt |  muen | genode | ...}
make depend && make
```

Builds a *unikernel* for your target of choice.

Solo5 compared

Solo5 compared to common isolation *interfaces* and units of execution:



(From left to right: Solo5, traditional VMs, Linux containers)

Philosophy of Solo5 (I)

The *interface* must be:

1. **Minimal**.
2. **Stateless**.
3. **Portable**.

The *implementation* must:

- Do one thing and do it well:
  - Be an **engine** for running unikernels.
- Orchestration, configuration management, monitoring, etc. are done elsewhere.

Philosophy of Solo5 (II)

Minimal

- Simplest useful abstraction.
  - Not Linux!
- No "device discovery" at run time.

Leads to:

- **Small implementation size**:
  - Typical configuration: **~3 kLOC**.
  - 12 kLOC in total (all combinations!).
- **Clarity** of implementation.
- **Fast startup time**:
  - Solo5 `hvt/spt`: < 50 ms
  - `qemu` Linux VM: ~ 1000 ms
  - Cloud-managed VMs: Seconds.

Philosophy of Solo5 (III)

Stateless

Very little state in the *interface* itself:

- Guest cannot change host state:
  - No dynamic resource allocation.
- Host cannot change guest state:
  - No interrupts.

Results in a system that:

- Is **deterministic** and easy to reason about.
- Is **static**.
- Enables **strong isolation**:
  - On monolithic *and* component-based / high assurance systems.

Philosophy of Solo5 (IV)

Portable

Easy to port libOS *to* Solo5:

- **MirageOS** (Ocaml-based), **IncludeOS** (C++), **Rumprun** (NetBSD).

Easy to port Solo5 *to* new targets:

- **OpenBSD vmm**, **Muen Separation Kernel**, **Genode OS framework**.
- Contributed by folks who are not Solo5 "experts".

Solo5: Limitations

Minimal

- Does not run Linux applications.
- But, there are POSIX-ish libOSes (Rumprun, LKL) that do.

Stateless

- "No interrupts" implies single core.
- Not intended for interfacing to hardware.
- Drivers are "some other component's" problem.

Portable

- Performance (copying semantics, number of "calls per IOP").
  - Not intended for HPC or millions of PPS.

The Solo5 *interface* (I)

```
struct solo5_start_info {
    const char *cmdline;
    uintptr_t heap_start;
    size_t heap_size;
}
int solo5_app_main(const struct solo5_start_info *info) /* entry point */

void solo5_exit(int status)
void solo5_abort(void)

void solo5_console_write(const char *buf, size_t size)

solo5_time_t solo5_clock_monotonic()
solo5_time_t solo5_clock_wall()

bool solo5_yield(solo5_time_t deadline)
```

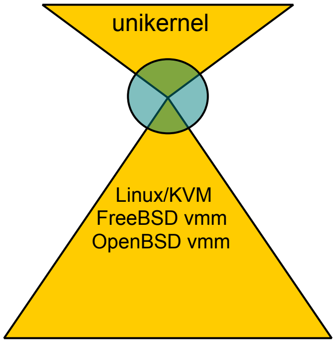
The Solo5 *interface* (II)

```
bool solo5_yield(solo5_time_t deadline)

typedef enum {
    SOLO5_R_OK, SOLO5_R_AGAIN, SOLO5_R_EINVAL, SOLO5_R_EUNSPEC
} solo5_result_t

solo5_result_t solo5_block_read(solo5_off_t offset, uint8_t *buf, size_t size)
solo5_result_t solo5_block_write(solo5_off_t offset, const uint8_t *buf, size_t size)
void solo5_block_info(struct solo5_block_info *info)

solo5_result_t solo5_net_read(uint8_t *buf, size_t size, size_t *read_size)
solo5_result_t solo5_net_write(const uint8_t *buf, size_t size)
void solo5_net_info(struct solo5_net_info *info)
```

(Demo: Solo5 in action)

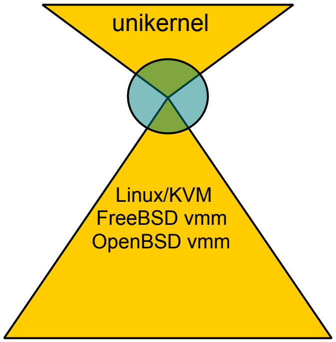
hvt: "Hardware virtualized tender" (I)

- Uses **hardware virtualization** as an **isolation layer**.
    - KVM, FreeBSD (Bhyve), OpenBSD (vmm).
- **Not** a traditional VMM:
    - **10 hypercalls**.
    - Modular, typical configuration **~1.5 kLOC**.
    - Compare QEMU: ~1000 kLOC, crosvm: ~100 kLOC.
- Supports `x86_64` and `arm64` architectures.
- Mature implementation, around since 2015.
    - Formerly known as `ukvm`.



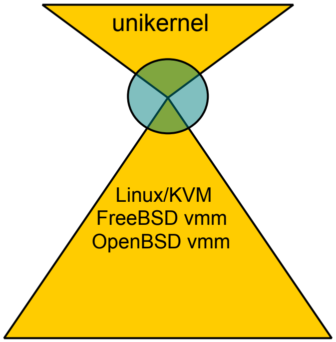
hvt: "Hardware virtualized tender" (II)

- Loads the unikernel.
- Sets up host resources.
- Sets up VCPU, page tables.
- Handles guest hypercalls (VMEXITs).



hvt: Hypercalls

- Hybrid PIO/MMIO-like approach.
- Transfer a 32-bit pointer to a `struct`.



On `x86_64`:

```
static inline void hvt_do_hypercall(int n, volatile void *arg)
{
    __asm__ __volatile__("outl %0, %1"
            :
            : "a" ((uint32_t)((uint64_t)arg)),
              "d" ((uint16_t)(HVT_HYPERCALL_PIO_BASE + n))
            : "memory");
}
```

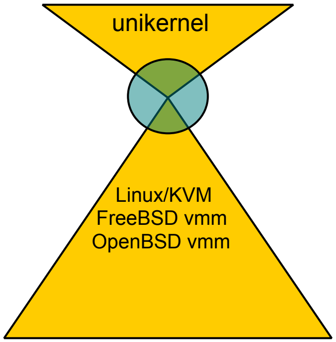
On `arm64`:

```
static inline void hvt_do_hypercall(int n, volatile void *arg)
{
    __asm__ __volatile__("str %w0, [%1]"
            :
            : "rZ" ((uint32_t)((uint64_t)arg)),
              "r" ((uint64_t)HVT_HYPERCALL_ADDRESS(n))
            : "memory");
}
```

hvt: Bindings

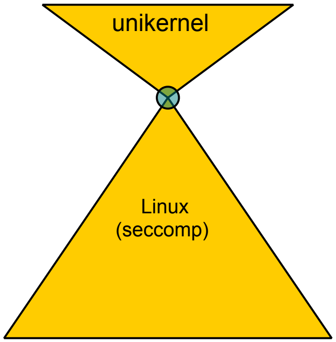
Implement the Solo5 *interface*:

- Using **hypercalls** to *tender*.
- Handle VCPU trap vectors.
  - Which just "report and abort".
- Provide monotonic time.
  - Via RDTSC or equivalent.




```
solo5_result_t solo5_net_write(const uint8_t *buf, size_t size)
{
    volatile struct hvt_netwrite wr;

    wr.data = buf;
    wr.len = size;
    wr.ret = 0;

    hvt_do_hypercall(HVT_HYPERCALL_NETWRITE, &wr);

    return (wr.ret == 0 && wr.len == size) ? SOLO5_R_OK : SOLO5_R_EUNSPEC;
}
```

spt: "Sandboxed process tender" (I)



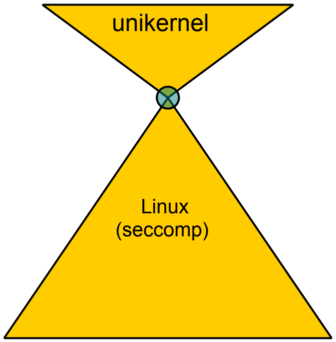
- Uses **process isolation** with **seccomp-BPF** as an **isolation layer**.
  - The system call filter is a **strict whitelist**.
  - ~7 system calls needed for the **entire** Solo5 *interface*.
- Should be possible to port to other kernels.
  - FreeBSD: Capsicum.
  - OpenBSD: `pledge(2)`.
- See our ACM SoCC 2018 paper:
  - https://dl.acm.org/citation.cfm?id=3267845

spt: "Sandboxed process tender" (II)

- Loads the unikernel.
- Sets up host resources.
- Applies the seccomp-BPF sandbox.

- **Effectively ceases to exist!**

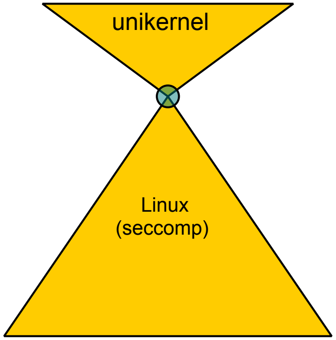
- Treats the monolithic kernel as a **hypervisor**!



spt: Bindings

Implement the Solo5 *interface*:

- By directly invoking system calls.
- **No `libc` involved**.

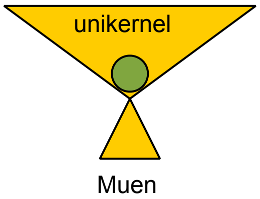



```
solo5_result_t solo5_net_write(const uint8_t *buf, size_t size)
{
    assert(netfd >= 0);

    long nbytes = sys_write(netfd, (const char *)buf, size);

    return (nbytes == (int)size) ? SOLO5_R_OK : SOLO5_R_EUNSPEC;
}
```

- Supports `x86_64` and `arm64`.
- Trivial to add more architectures.

muen: Native component

Muen: An x86_64 **Separation Kernel** for High Assurance.



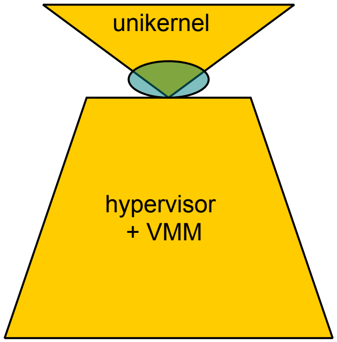
- Guarantees that components communicate **exclusively** according to given security policy.
- Isolation using **hardware virtualization**.
- Implemented in **ADA/SPARK**.
- **Formally proven** to contain **no runtime errors** at the source code level.
- Open Source, https://muen.sk/.

Solo5 *bindings* shared with `hvt`:

- No hypercalls.
- Communication via shared memory rings.

virtio: Cloud hypervisors

- The first Solo5 implementation.
- No longer consistent with our philosophy:
  - Still too much legacy in interface.
  - **2.5 kLOC**, and still not complete.
  - Anyone for SCSI?
- But, runs on **existing** cloud hypervisors:
  - e.g. Google Compute Engine.



Debugging

Just use gdb!

```
(1) $ solo5-hvt --gdb test_hello.hvt
...
(2) $ gdb --ex="target remote localhost:1234" test_hello.hvt
```

Lessons learned (I)

Usable modularity

- Wanted a *tender* that is **specialized** to the *unikernel*.
- Original `hvt` approach:
  - Specialize at **unikernel compile time**.
  - This is **not practical**:
    - Supply chain is wrong.
- Removed for `spt`.

Lessons learned (II)

Usable modularity

- We still want to **enforce a contract** (*tender*/*unikernel*).
  - *unikernel* wants *A*, *B*, *C*.
  - *tender* must match or refuse to run.
  - *tender* **must not** accidentally provide *D*.
- Embed a "manifest" into the *unikernel*:
  - Enforce at run time.
  - Can we still **specialize** the *tender*?

Future: Plans and challenges (I)

Security

Implement more best practices:

- ASLR (static PIE), SSP, W^X.
  - Undocumented ABIs.
  - `hvt`: Hypervisor support lacking.
    - EPT `mprotect`

Defense in depth:

- Further de-privilege *tenders*.
- Initial setup stage runs with full privileges.

Future: Plans and challenges (II)

Portability

- Can we do dynamic linking *safely*?
- Leads to a *tender*-independent unikernel binary.

Performance

- Can we define an *interface* that allows **asynchronicity**, yet is consistent with our philosophy? (**Minimal**, **Stateless**, **Portable**).

Future: Ideas

More languages and libOS

- Go, Rust, ...
  - Further validation of the Solo5 interface.

More targets

- Mac OS `Hypervisor.framework`.
- Secure enclaves (e.g. SGX).

Architecture independence

- Webassembly as a target.
  - Build once, run on any CPU architecture.

Related work

Library operating systems

- MSR Drawbridge
- Graphene

Lightweight VMs

- kvmtool, qemu-lite
- novm
- crosvm
- Firecracker

Securing the Linux interface

- gVisor

Conclusion

Minimalism!

- Tiny API (**13 functions**), legacy-free. **ISC licensed**.
- **Low resource usage**. Run 1500 VMs on your 3 year old laptop.

Apply unikernels everywhere!

- Linux, FreeBSD, OpenBSD, Muen, Genode, Cloud, ...

No compromises!

- **Strong isolation** on all targets.

https://github.com/Solo5/solo5