KVM Forum, Brno

Handling ~~Complex Guest~~ MMIO Exits with eBPF

June 2023

Will Deacon <will@kernel.org>

android

$ whoami

- Upstream kernel hacker
- Arm64 co-maintainer
- Android systems team at Google
- pKVM developer
- Homebrewer
- I'd rather be fishing

android

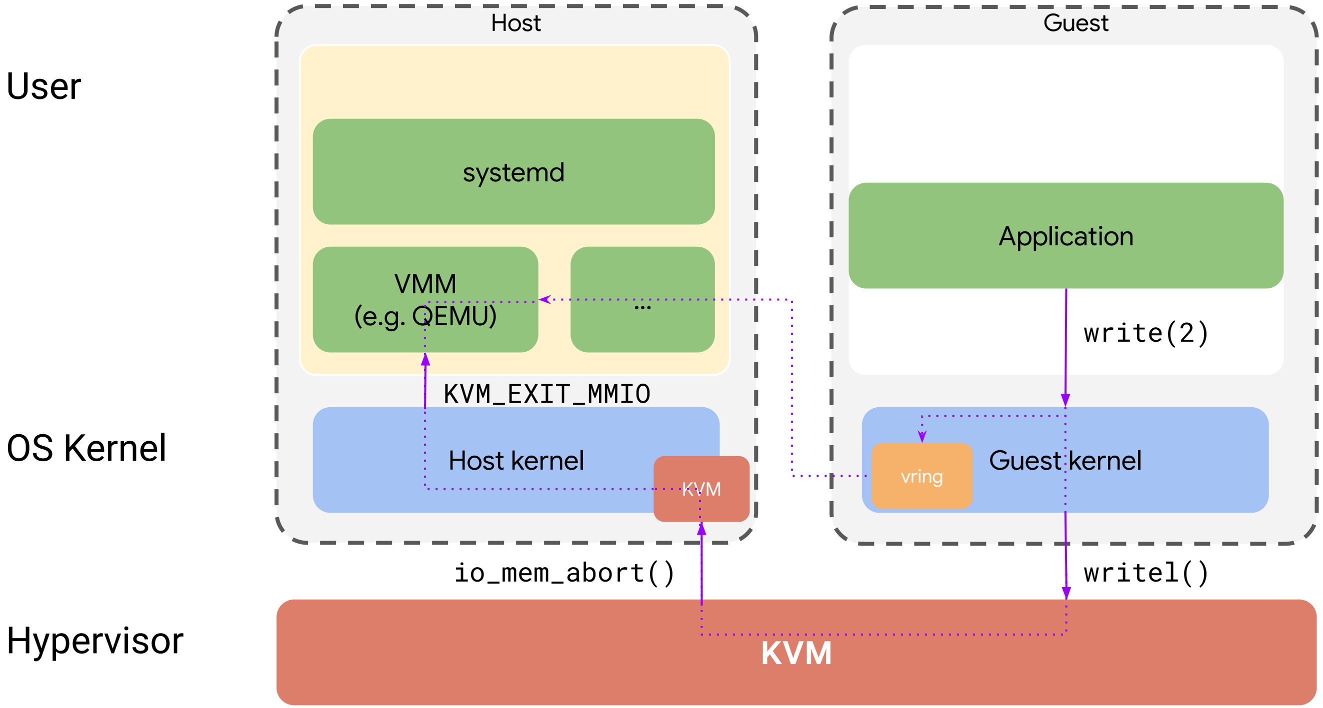
Disclaimer!

- I don't know anything about eBPF
- This is a work-in-progress; eBPF is a moving target
- I'm not convinced it's a sensible idea! Hoping to inspire…
- But it's cool and I fixed a bug
- "Conference-driven development" (I have a prototype)

01

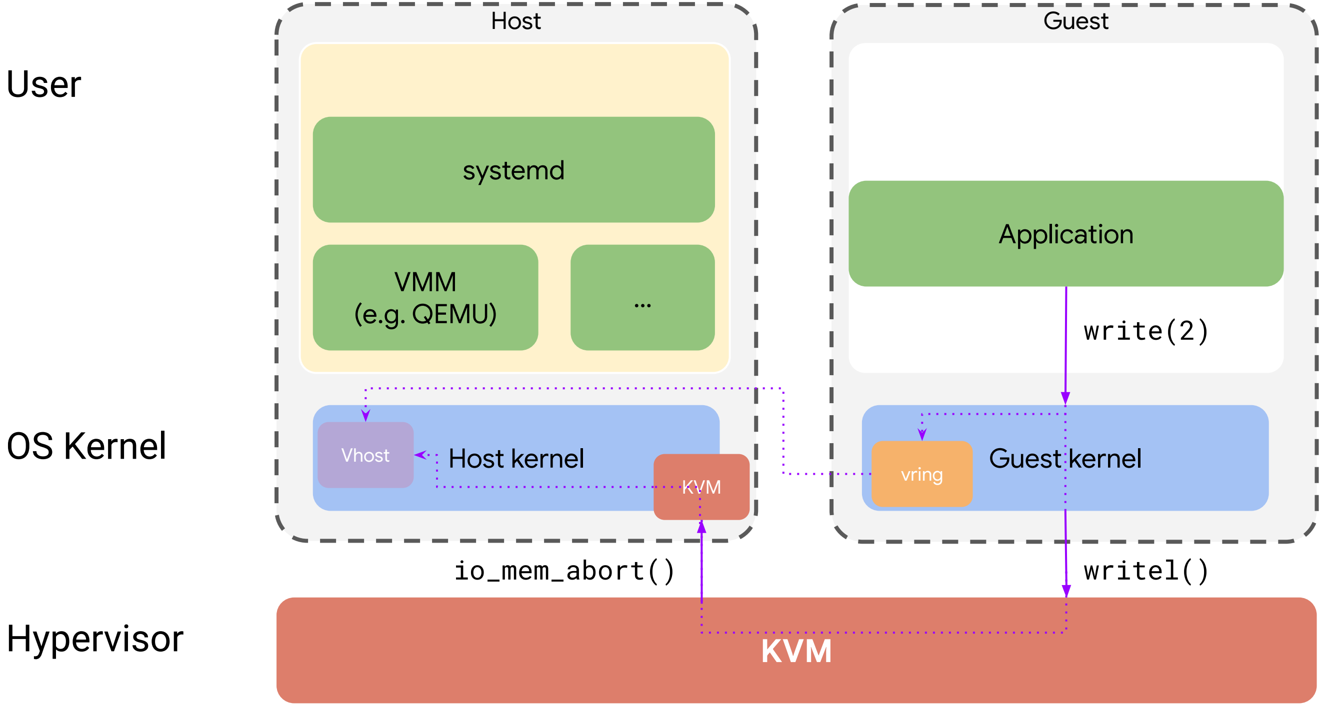
Motivation

Basic model for I/O handling in KVM



android

Vhost model for I/O handling in KVM

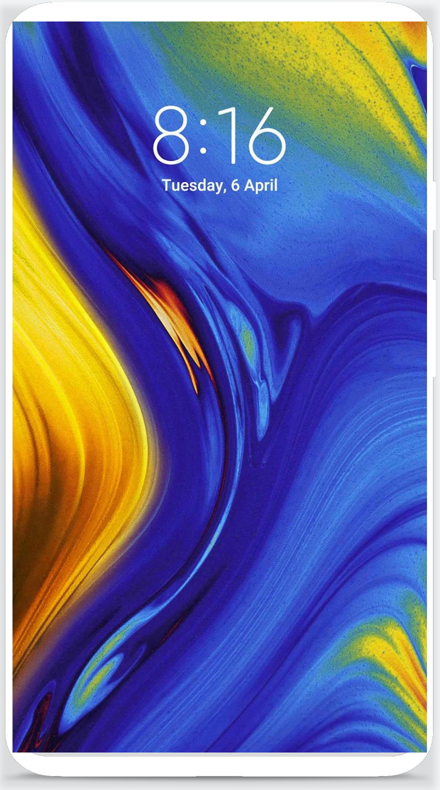


android

Limitations of vhost

Vhost is widely used to accelerate virtio devices, but it has some limitations:

- Thousands of lines of device-specific C code running in the host kernel
- Only supports virtio; other devices are handled either in userspace or via device-specific `KVM_CREATE_DEVICE` emulation
- The VMM still needs built-in device knowledge to instantiate and manage the in-kernel state
- Hard/impossible to update at runtime
- In-kernel emulation code is privileged and cannot be sandboxed



“Haha, maybe we should use eBPF to handle guest exits!”

“No, seriously.”

Can eBPF save the day?

Pros:

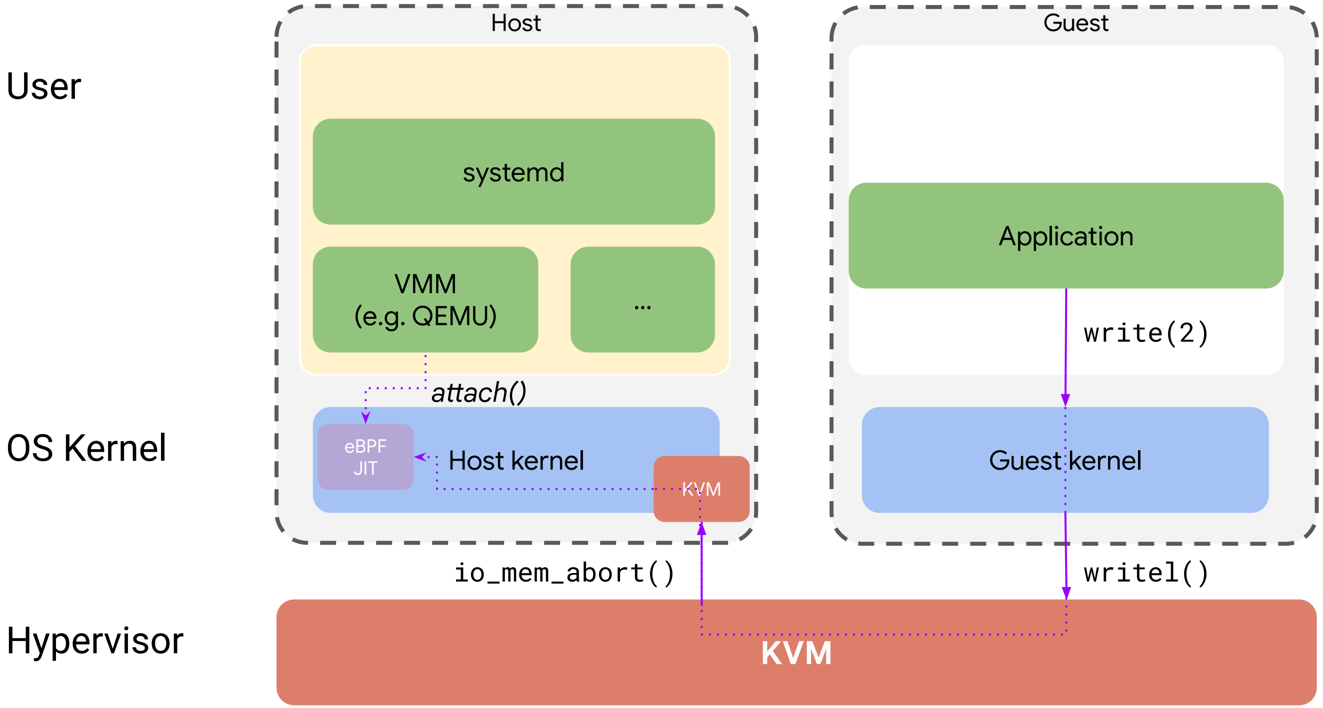
- In-kernel sandbox using verifier
- Programs uploaded at runtime
- Flexible/portable ABIs (user and kernel)
- It's fashionable (good for conference submissions ;))

Cons:

- Atypical use-case
- Fairly rigid permissions/ACL model
- It's fashionable (moving *very* quickly)

KVM_DEV_TYPE_BPF

eBPF model for I/O handling in KVM



KVM_DEV_TYPE_BPF: Programming interface

Managing the new device type

- Device instantiated via KVM_CREATE_DEVICE VM ioctl()
  - KVM_DEV_BPF_ATTR_GROUP_REGION attribute to set a new MMIO range and attach bpf progs:

```
#define KVM_DEV_BPF_ATTR_GROUP_REGION   1
struct kvm_bpf_user_region {
        __u64   addr;
        __u64   size;
        __s32   bpf_readfd;
        __s32   bpf_writefd;
};
```

File handles returned by bpf(2) BPF_PROG_LOAD system call. (libbpf makes this easy)

  - Envisage a similar approach for vIRQs (eventfds)
    - i.e. Associate eventfds with a region and allow them to be signalled from the eBPF programs

KVM_DEV_TYPE_BPF: Programming interface

View from the eBPF program

- Passed a single context pointer argument by the kernel:
  -
    ```
    struct bpf_kvm_io_ctx {
            __u8    buf[8];
            __u64   offset;
            __u8    len;
            __u32   :24;
            __u32   vcpu_id;
    };
    ```

This structure is *fake* and never allocated! JIT generates accesses to the real structures underneath (e.g. the internal vCPU structure)

  - Verifier enforces fine-grained permissions on the struct members (e.g. buf is read-only for the MMIO write handler).
  - Return value from handler:
    - 0: return to guest (skipping faulting instruction)
    - Non-zero: MMIO exit to the VMM

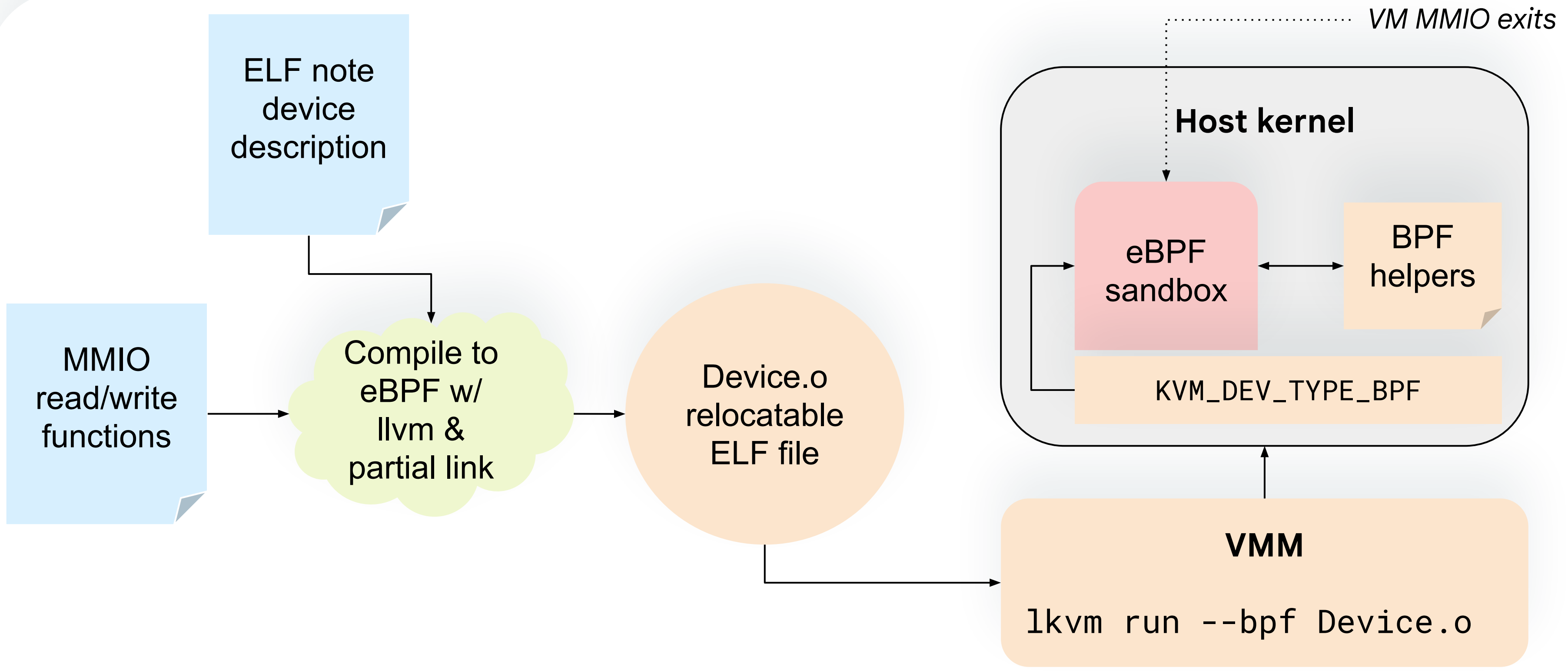
BYOD: ELF encapsulation

Wrap the device in an ELF file for libbpf

- Implement read/write callbacks in C (or rust)
- eBPF maps for global device state
- ELF note to describe the device configuration such as device-tree compatible string, MMIO size, number of IRQs etc.
- `Device.o: ELF 64-bit LSB relocatable, eBPF, version 1 (SYSV), with debug_info, not stripped`
- Different to the usual “skeleton” header approach

Warning: linkers really don’t seem to like linking this, so I did terrible things with objcopy 😒

Putting it all together



Live demo

Wish me luck.

ABSOLUTELY NO WARRANTY etc. etc.

android

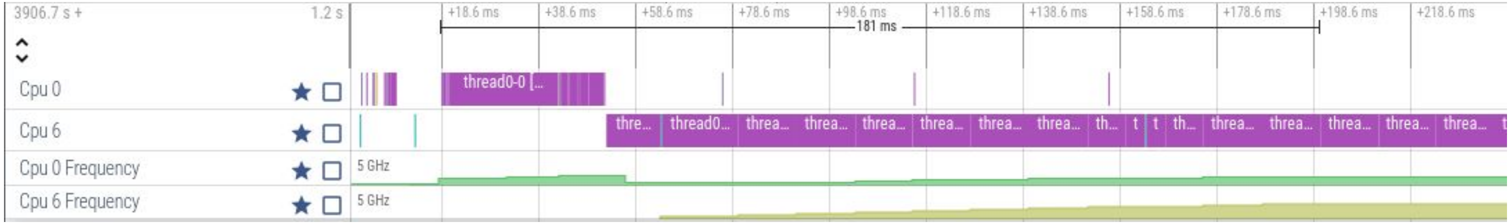
Scheduler hooks (with help)

Saravana!

David!

Set capacity for guest thread to migrate

Host - 181ms to Fmax on big CPU.



VM - 140ms to Fmax on little CPU. Guest thread never migrates to vCPU1 pinned to big CPU.

Source: Saravana’s LPC ‘22 talk: https://lpc.events/event/16/contributions/1195/

android

Problem:

“Workloads running in a guest VM get terrible task placement and DVFS behavior when compared to running the same workload in the host”

https://lore.kernel.org/all/20230330224348.1006691-1-davidai@google.com/

Guest frequency requests

Add a new cpufreq driver in the guest:

- VMM pins the vCPUs
- Guest cpufreq driver advertises host CPU properties (e.g. available frequencies, capacity)
- Guest frequency requests result in uclamp utilization requests on the host

Communication channel

The guest frequency requests need to reach the host:

- New hypercall(s)?
- MMIO device?
- Guess what’s coming…

Latency

It is *critical* to minimise the latency when processing a guest request:

- Fast-path accesses (e.g. reading current frequency every context-switch)
- Pure overhead: the guest is runnable
- State of the system can change

VCPUFreq device in eBPF

A tiny amount of eBPF code (< 80 lines)!

New eBPF helper functions for:

- Querying CPU state:
  - `bpf_get_cpu_freq(cpu)`
  - `bpf_get_cpu_max_hw_freq(cpu)`
  - `bpf_get_cpu_scale(cpu)`
- Setting desired uclamp values:
  - `bpf_set_current_uclamp(min,max)`

These all have corresponding user-accessible interfaces already (`sysfs`, `sched_setattr()`).

How does it perform? →

Preliminary results in pKVM (higher is better)

FIO test	Baseline	Userspace MMIO	eBPF MMIO
Seq write	1.0	1.10	1.15
Rand write	1.0	1.13	1.23
Seq read	1.0	1.03	1.05
Rand read	1.0	1.05	1.09

04

Show me the code

I have hacks!

Host kernel

git://git.kernel.org/pub/scm/linux/kernel/git/will/linux.git kvm/bpf

- Partial `KVM_DEV_TYPE_BPF` implementation
  - One memory region per device instance
  - vIRQs not functional yet
  - New program types instead of 'BPF `struct_ops`'
- eBPF verifier codegen fix
- Scheduler helpers and minor `sched_setattr()` rework

eBPF devices

git://git.kernel.org/pub/scm/linux/kernel/git/will/bpf-devices.git

- Partial PL031 RTC emulation
- vCPUFreq device implementation
- ELF note generation
- Nasty build system hacks to avoid linker crashes
- Completely standalone

Kvmtool

https://android-kvm.googlesource.com/kvmtool willdeacon/bpf

- ELF note parsing and device-tree generation
- Libbpf to extract and load programs
- Instantiation of KVM_DEV_TYPE_BPF device
- Program attachment

Guest kernel

https://android-review.googlesource.com/c/kernel/common/+/2239182/21

- Guest driver for vCPUFreq device
- Currently per-vCPU register region
  - Banking an alternative?
- AMUs preferred if available

05

Amplify the crazy

android

With great power, comes great… uncertainty?

This all feels quite powerful, but I'm nervous about the ABI and security implications of some of these:

- Asynchronous device behaviour: blocking and signalling?
- `bpf_copy_from_user()` is bad, but what about bpf *guest* accessors? To specific windows?
- Vhost as a bpf program
- Finer-grained permissions for BPF programs (a la seccomp?)
- PCI devices (i.e. x86 support)
- Device migration (between VMMs!) using JSON map state
- Guest uploads devices as firmware… (too far?!)
- **⇒ Your idea here ⇐**

Conclusion

I think this is cool but I'm not precious about it.

I'd love it if other folks could have a play and see where they can take it.

The security story needs figuring out properly for some future extensions.

What next?

android

Thank you