Generic vcpu interface

The virtual cpu "device" also accepts the ioctls KVM_SET_DEVICE_ATTR, KVM_GET_DEVICE_ATTR, and KVM_HAS_DEVICE_ATTR. The interface uses the same struct kvm_device_attr as other devices, but targets VCPU-wide settings and controls.

The groups and attributes per virtual cpu, if any, are architecture specific.

1. GROUP: KVM_ARM_VCPU_PMU_V3_CTRL

Architectures: ARM64

1.1. ATTRIBUTE: KVM_ARM_VCPU_PMU_V3_IRQ

Parameters: in kvm_device_attr.addr the address for PMU overflow interrupt is a pointer to an int

Returns:

-EBUSY	The PMU overflow interrupt is already set
-EFAULT	Error reading interrupt number
-ENXIO	PMUv3 not supported or the overflow interrupt not set when attempting to get it
-ENODEV	KVM_ARM_VCPU_PMU_V3 feature missing from VCPU
-EINVAL	Invalid PMU overflow interrupt number supplied or trying to set the IRQ number without using an in-kernel irqchip.

A value describing the PMUv3 (Performance Monitor Unit v3) overflow interrupt number for this vcpu. This interrupt could be a PPI or SPI, but the interrupt type must be same for each vcpu. As a PPI, the interrupt number is the same for all vcpus, while as an SPI it must be a separate number per vcpu.

1.2 ATTRIBUTE: KVM_ARM_VCPU_PMU_V3_INIT

Parameters: no additional parameter in kvm_device_attr.addr

Returns:

-EEXIST	Interrupt number already used
-ENODEV	PMUv3 not supported or GIC not initialized
-ENXIO	PMUv3 not supported, missing VCPU feature or interrupt number not set
-EBUSY	PMUv3 already initialized

Request the initialization of the PMUv3. If using the PMUv3 with an in-kernel virtual GIC implementation, this must be done after initializing the in-kernel irqchip.

1.3 ATTRIBUTE: KVM_ARM_VCPU_PMU_V3_FILTER

Parameters: in kvm_device_attr.addr the address for a PMU event filter is a pointer to a struct kvm_pmu_event_filter

Returns:

-ENODEV	PMUv3 not supported or GIC not initialized
-ENXIO	PMUv3 not properly configured or in-kernel irqchip not configured as required prior to calling this attribute
-EBUSY	PMUv3 already initialized or a VCPU has already run
-EINVAL	Invalid filter range

Request the installation of a PMU event filter described as follows:

```
struct kvm_pmu_event_filter {
        __u16       base_event;
        __u16       nevents;

#define KVM_PMU_EVENT_ALLOW 0
#define KVM_PMU_EVENT_DENY  1

        __u8        action;
        __u8        pad[3];
};
```

A filter range is defined as the range [@base_event, @base_event + @nevents), together with an @action (KVM_PMU_EVENT_ALLOW or KVM_PMU_EVENT_DENY). The first registered range defines the global policy (global ALLOW if the first @action is DENY, global DENY if the first @action is ALLOW). Multiple ranges can be programmed, and must

fit within the event space defined by the PMU architecture (10 bits on ARMv8.0, 16 bits from ARMv8.1 onwards).

Note: "Cancelling" a filter by registering the opposite action for the same range doesn't change the default action. For example, installing an ALLOW filter for event range [0:10) as the first filter and then applying a DENY action for the same range will leave the whole range as disabled.

Restrictions: Event 0 (SW_INCR) is never filtered, as it doesn't count a hardware event. Filtering event 0x1E (CHAIN) has no effect either, as it isn't strictly speaking an event. Filtering the cycle counter is possible using event 0x11 (CPU_CYCLES).

1.4 ATTRIBUTE: KVM_ARM_VCPU_PMU_V3_SET_PMU

Parameters: in kvm_device_attr.addr the address to an int representing the PMU identifier.

Returns:

-EBUSY	PMUv3 already initialized, a VCPU has already run or an event filter has already been set
-EFAULT	Error accessing the PMU identifier
-ENXIO	PMU not found
-ENODEV	PMUv3 not supported or GIC not initialized
-ENOMEM	Could not allocate memory

Request that the VCPU uses the specified hardware PMU when creating guest events for the purpose of PMU emulation. The PMU identifier can be read from the "type" file for the desired PMU instance under /sys/devices (or, equivalent, /sys/bus/even_source). This attribute is particularly useful on heterogeneous systems where there are at least two CPU PMUs on the system. The PMU that is set for one VCPU will be used by all the other VCPUs. It isn't possible to set a PMU if a PMU event filter is already present.

Note that KVM will not make any attempts to run the VCPU on the physical CPUs associated with the PMU specified by this attribute. This is entirely left to userspace. However, attempting to run the VCPU on a physical CPU not supported by the PMU will fail and KVM_RUN will return with exit_reason = KVM_EXIT_FAIL_ENTRY and populate the fail_entry struct by setting hardare_entry_failure_reason field to KVM_EXIT_FAIL_ENTRY_CPU_UNSUPPORTED and the cpu field to the processor id.

2. GROUP: KVM_ARM_VCPU_TIMER_CTRL

Architectures: ARM64

2.1. ATTRIBUTES: KVM_ARM_VCPU_TIMER_IRQ_VTIMER, KVM_ARM_VCPU_TIMER_IRQ_PTIMER

Parameters: in kvm_device_attr.addr the address for the timer interrupt is a pointer to an int

Returns:

-EINVAL	Invalid timer interrupt number
-EBUSY	One or more VCPUs has already run

A value describing the architected timer interrupt number when connected to an in-kernel virtual GIC. These must be a PPI (16 <= intid < 32). Setting the attribute overrides the default values (see below).

KVM_ARM_VCPU_TIMER_IRQ_VTIMER	The EL1 virtual timer intid (default: 27)
KVM_ARM_VCPU_TIMER_IRQ_PTIMER	The EL1 physical timer intid (default: 30)

Setting the same PPI for different timers will prevent the VCPUs from running. Setting the interrupt number on a VCPU configures all VCPUs created at that time to use the number provided for a given timer, overwriting any previously configured values on other VCPUs. Userspace should configure the interrupt numbers on at least one VCPU after creating all VCPUs and before running any VCPUs.

3. GROUP: KVM_ARM_VCPU_PVTIME_CTRL

Architectures: ARM64

3.1 ATTRIBUTE: KVM_ARM_VCPU_PVTIME_IPA

Parameters: 64-bit base address

Returns:

-ENXIO	Stolen time not implemented
-EEXIST	Base address already set for this VCPU
-EINVAL	Base address not 64 byte aligned

Specifies the base address of the stolen time structure for this VCPU. The base address must be 64 byte aligned and exist within a valid guest memory region. See Documentation/virt/kvm/arm/pvtime.rst for more information including the layout of the stolen time

structure.

4. GROUP: KVM_VCPU_TSC_CTRL

Architectures: x86

4.1 ATTRIBUTE: KVM_VCPU_TSC_OFFSET

Parameters: 64-bit unsigned TSC offset

Returns:

-EFAULT	Error reading/writing the provided parameter address.
-ENXIO	Attribute not supported

Specifies the guest's TSC offset relative to the host's TSC. The guest's TSC is then derived by the following equation:

guest_tsc = host_tsc + KVM_VCPU_TSC_OFFSET

This attribute is useful to adjust the guest's TSC on live migration, so that the TSC counts the time during which the VM was paused. The following describes a possible algorithm to use for this purpose.

From the source VMM process:

1. Invoke the KVM_GET_CLOCK ioctl to record the host TSC (tsc_src), kvmclock nanoseconds (guest_src), and host CLOCK_REALTIME nanoseconds (host_src).
2. Read the KVM_VCPU_TSC_OFFSET attribute for every vCPU to record the guest TSC offset (ofs_src[i]).
3. Invoke the KVM_GET_TSC_KHZ ioctl to record the frequency of the guest's TSC (freq).

From the destination VMM process:

4. Invoke the KVM_SET_CLOCK ioctl, providing the source nanoseconds from kvmclock (guest_src) and CLOCK_REALTIME (host_src) in their respective fields. Ensure that the KVM_CLOCK_REALTIME flag is set in the provided structure.

   KVM will advance the VM's kvmclock to account for elapsed time since recording the clock values. Note that this will cause problems in the guest (e.g., timeouts) unless CLOCK_REALTIME is synchronized between the source and destination, and a reasonably short time passes between the source pausing the VMs and the destination executing steps 4-7.

5. Invoke the KVM_GET_CLOCK ioctl to record the host TSC (tsc_dest) and kvmclock nanoseconds (guest_dest).
6. Adjust the guest TSC offsets for every vCPU to account for (1) time elapsed since recording state and (2) difference in TSCs between the source and destination machine:

   ofs_dst[i] = ofs_src[i] -
   (guest_src - guest_dest) * freq + (tsc_src - tsc_dest)

   ("ofs[i] + tsc - guest * freq" is the guest TSC value corresponding to a time of 0 in kvmclock. The above formula ensures that it is the same on the destination as it was on the source).

7. Write the KVM_VCPU_TSC_OFFSET attribute for every vCPU with the respective value derived in the previous step.