Linux KVM Hypercall

X86:

KVM Hypercalls have a three-byte sequence of either the vmcall or the vmmcall instruction. The hypervisor can replace it with instructions that are guaranteed to be supported.

Up to four arguments may be passed in rbx, rcx, rdx, and rsi respectively. The hypercall number should be placed in rax and the return value will be placed in rax. No other registers will be clobbered unless explicitly stated by the particular hypercall.

S390:

R2-R7 are used for parameters 1-6. In addition, R1 is used for hypercall number. The return value is written to R2.

S390 uses diagnose instruction as hypercall (0x500) along with hypercall number in R1.

For further information on the S390 diagnose call as supported by KVM, refer to Documentation/virt/kvm/s390-diag.rst.

PowerPC:

It uses R3-R10 and hypercall number in R11. R4-R11 are used as output registers. Return value is placed in R3.

KVM hypercalls uses 4 byte opcode, that are patched with 'hypercall-instructions' property inside the device tree's /hypervisor node. For more information refer to Documentation/virt/kvm/ppc-pv.rst

MIPS:

KVM hypercalls use the HYPCALL instruction with code 0 and the hypercall number in $2 (v0). Up to four arguments may be placed in $4-$7 (a0-a3) and the return value is placed in $2 (v0).

KVM Hypercalls Documentation

The template for each hypercall is: 1. Hypercall name. 2. Architecture(s) 3. Status (deprecated, obsolete, active) 4. Purpose

1. KVM_HC_VAPIC_POLL_IRQ

Architecture: x86
Status: active
Purpose: Trigger guest exit so that the host can check for pending interrupts on reentry.

2. KVM_HC_MMU_OP

Architecture: x86
Status: deprecated.
Purpose: Support MMU operations such as writing to PTE, flushing TLB, release PT.

3. KVM_HC_FEATURES

Architecture: PPC
Status: active
Purpose: Expose hypercall availability to the guest. On x86 platforms, cpuid used to enumerate which hypercalls are available. On PPC, either device tree based lookup (which is also what EPAPR dictates) OR KVM specific enumeration mechanism (which is this hypercall) can be used.

4. KVM_HC_PPC_MAP_MAGIC_PAGE

Architecture: PPC
Status: active
Purpose: To enable communication between the hypervisor and guest there is a shared page that contains parts of supervisor visible register state. The guest can map this shared page to access its supervisor register through memory using this hypercall.

5. KVM_HC_KICK_CPU

Architecture: x86
Status: active
Purpose: Hypercall used to wakeup a vcpu from HLT state
Usage example:

A vcpu of a paravirtualized guest that is busywaiting in guest kernel mode for an event to occur (ex: a spinlock to become available) can execute HLT instruction once it has busy-waited for more than a threshold time-interval. Execution of HLT instruction would cause the hypervisor to put the vcpu to sleep until occurrence of an appropriate event. Another vcpu of the same guest can wakeup the sleeping vcpu by issuing KVM_HC_KICK_CPU hypercall, specifying APIC ID (a1) of the vcpu to be woken up. An additional argument (a0) is used in the hypercall for future use.

6. KVM_HC_CLOCK_PAIRING

Architecture: x86
Status: active
Purpose: Hypercall used to synchronize host and guest clocks.

Usage:

a0: guest physical address where host copies "struct kvm_clock_offset" structure.

a1: clock_type, ATM only KVM_CLOCK_PAIRING_WALLCLOCK (0) is supported (corresponding to the host's CLOCK_REALTIME clock).

```
struct kvm_clock_pairing {
        __s64 sec;
        __s64 nsec;
        __u64 tsc;
        __u32 flags;
        __u32 pad[9];
};
```

Where:

- sec: seconds from clock_type clock.
- nsec: nanoseconds from clock_type clock.
- tsc: guest TSC value used to calculate sec/nsec pair
- flags: flags, unused (0) at the moment.

The hypercall lets a guest compute a precise timestamp across host and guest. The guest can use the returned TSC value to compute the CLOCK_REALTIME for its clock, at the same instant.

Returns KVM_EOPNOTSUPP if the host does not use TSC clocksource, or if clock type is different than KVM_CLOCK_PAIRING_WALLCLOCK.

6. KVM_HC_SEND_IPI

Architecture: x86
Status: active
Purpose: Send IPIs to multiple vCPUs.

- a0: lower part of the bitmap of destination APIC IDs
- a1: higher part of the bitmap of destination APIC IDs
- a2: the lowest APIC ID in bitmap
- a3: APIC ICR

The hypercall lets a guest send multicast IPIs, with at most 128 128 destinations per hypercall in 64-bit mode and 64 vCPUs per hypercall in 32-bit mode. The destinations are represented by a bitmap contained in the first two arguments (a0 and a1). Bit 0 of a0 corresponds to the APIC ID in the third argument (a2), bit 1 corresponds to the APIC ID a2+1, and so on.

Returns the number of CPUs to which the IPIs were delivered successfully.

7. KVM_HC_SCHED_YIELD

Architecture: x86
Status: active
Purpose: Hypercall used to yield if the IPI target vCPU is preempted

a0: destination APIC ID

Usage example: When sending a call-function IPI-many to vCPUs, yield if any of the IPI target vCPUs was preempted.

8. KVM_HC_MAP_GPA_RANGE

Architecture: x86
Status: active
Purpose: Request KVM to map a GPA range with the specified attributes.

a0: the guest physical address of the start page a1: the number of (4kb) pages (must be contiguous in GPA space) a2: attributes

Where 'attributes' :

- bits 3:0 - preferred page size encoding 0 = 4kb, 1 = 2mb, 2 = 1gb, etc...
- bit 4 - plaintext = 0, encrypted = 1
- bits 63:5 - reserved (must be zero)

Implementation note: this hypercall is implemented in userspace via the KVM_CAP_EXIT_HYPERCALL capability. Userspace must enable that capability before advertising KVM_FEATURE_HC_MAP_GPA_RANGE in the guest CPUID. In addition, if the guest supports KVM_FEATURE_MIGRATION_CONTROL, userspace must also set up an MSR filter to process writes to MSR_KVM_MIGRATION_CONTROL.