KVM-specific MSRs

Author: Glauber Costa <glommer@redhat.com>, Red Hat Inc, 2010

KVM makes use of some custom MSRs to service some requests.

Custom MSRs have a range reserved for them, that goes from 0x4b564d00 to 0x4b564dff. There are MSRs outside this area, but they are deprecated and their use is discouraged.

Custom MSR list

The current supported Custom MSR list is:

MSR_KVM_WALL_CLOCK_NEW:

0x4b564d00

data:

4-byte alignment physical address of a memory area which must be in guest RAM. This memory is expected to hold a copy of the following structure:

```
struct pvclock_wall_clock {
       u32   version;
       u32   sec;
       u32   nsec;
 } __attribute__((__packed__));
```

whose data will be filled in by the hypervisor. The hypervisor is only guaranteed to update this data at the moment of MSR write. Users that want to reliably query this information more than once have to write more than once to this MSR. Fields have the following meanings:

version:

guest has to check version before and after grabbing time information and check that they are both equal and even. An odd version indicates an in-progress update.

sec:

number of seconds for wallclock at time of boot.

nsec:

number of nanoseconds for wallclock at time of boot.

In order to get the current wallclock time, the system_time from MSR_KVM_SYSTEM_TIME_NEW needs to be added.

Note that although MSRs are per-CPU entities, the effect of this particular MSR is global.

Availability of this MSR must be checked via bit 3 in 0x4000001 cpuid leaf prior to usage.

MSR_KVM_SYSTEM_TIME_NEW:

0x4b564d01

data:

4-byte aligned physical address of a memory area which must be in guest RAM, plus an enable bit in bit 0. This memory is expected to hold a copy of the following structure:

```
struct pvclock_vcpu_time_info {
      u32   version;
      u32   pad0;
      u64   tsc_timestamp;
      u64   system_time;
      u32   tsc_to_system_mul;
      s8    tsc_shift;
      u8    flags;
      u8    pad[2];
} __attribute__((__packed__)); /* 32 bytes */
```

whose data will be filled in by the hypervisor periodically. Only one write, or registration, is needed for each VCPU. The interval between updates of this structure is arbitrary and implementation-dependent. The hypervisor may update this structure at any time it sees fit until anything with bit0 == 0 is written to it.

Fields have the following meanings:

version:

guest has to check version before and after grabbing time information and check that they are both equal and even. An odd version indicates an in-progress update.

tsc_timestamp:

the tsc value at the current VCPU at the time of the update of this structure. Guests can subtract this value from current tsc to derive a notion of elapsed time since the structure update.

system_time:

a host notion of monotonic time, including sleep time at the time this structure was last updated. Unit is nanoseconds.

tsc_to_system_mul:

multiplier to be used when converting tsc-related quantity to nanoseconds

tsc_shift:

shift to be used when converting tsc-related quantity to nanoseconds. This shift will ensure that multiplication with tsc_to_system_mul does not overflow. A positive value denotes a left shift, a negative value a right shift.

The conversion from tsc to nanoseconds involves an additional right shift by 32 bits. With this information, guests can derive per-CPU time by doing:

```
time = (current_tsc - tsc_timestamp)
if (tsc_shift >= 0)
        time <<= tsc_shift;
else
        time >>= -tsc_shift;
time = (time * tsc_to_system_mul) >> 32
time = time + system_time
```

flags:

bits in this field indicate extended capabilities coordinated between the guest and the hypervisor. Availability of specific flags has to be checked in 0x40000001 cpuid leaf. Current flags are:

flag bit	cpuid bit	meaning
0	24	time measures taken across multiple cpus are guaranteed to be monotonic
1	N/A	guest vcpu has been paused by the host See 4.70 in api.txt

Availability of this MSR must be checked via bit 3 in 0x4000001 cpuid leaf prior to usage.

MSR_KVM_WALL_CLOCK:

0x11

data and functioning:

same as MSR_KVM_WALL_CLOCK_NEW. Use that instead.

This MSR falls outside the reserved KVM range and may be removed in the future. Its usage is deprecated.

Availability of this MSR must be checked via bit 0 in 0x4000001 cpuid leaf prior to usage.

MSR_KVM_SYSTEM_TIME:

0x12

data and functioning:

same as MSR_KVM_SYSTEM_TIME_NEW. Use that instead.

This MSR falls outside the reserved KVM range and may be removed in the future. Its usage is deprecated.

Availability of this MSR must be checked via bit 0 in 0x4000001 cpuid leaf prior to usage.

The suggested algorithm for detecting kvmclock presence is then:

```
if (!kvm_para_available())    /* refer to cpuid.txt */
        return NON_PRESENT;

flags = cpuid_eax(0x40000001);
if (flags & 3) {
        msr_kvm_system_time = MSR_KVM_SYSTEM_TIME_NEW;
        msr_kvm_wall_clock = MSR_KVM_WALL_CLOCK_NEW;
        return PRESENT;
} else if (flags & 0) {
        msr_kvm_system_time = MSR_KVM_SYSTEM_TIME;
        msr_kvm_wall_clock = MSR_KVM_WALL_CLOCK;
        return PRESENT;
} else
        return NON_PRESENT;
```

MSR_KVM_ASYNC_PF_EN:

0x4b564d02

data:

Asynchronous page fault (APF) control MSR.

Bits 63-6 hold 64-byte aligned physical address of a 64 byte memory area which must be in guest RAM and must be zeroed. This memory is expected to hold a copy of the following structure:

```
struct kvm_vcpu_pv_apf_data {
      /* Used for 'page not present' events delivered via #PF */
      __u32 flags;

      /* Used for 'page ready' events delivered via interrupt notification */
      __u32 token;

      __u8 pad[56];
      __u32 enabled;
};
```

Bits 5-4 of the MSR are reserved and should be zero. Bit 0 is set to 1 when asynchronous page faults are enabled on the vcpu, 0 when disabled. Bit 1 is 1 if asynchronous page faults can be injected when vcpu is in cpl == 0. Bit 2 is 1 if asynchronous page faults are delivered to L1 as #PF vmexits. Bit 2 can be set only if KVM_FEATURE_ASYNC_PF_VMEXIT is present in CPUID. Bit 3 enables interrupt based delivery of 'page ready' events. Bit 3 can only be set if KVM_FEATURE_ASYNC_PF_INT is present in CPUID.

'Page not present' events are currently always delivered as synthetic #PF exception. During delivery of these events APF CR2 register contains a token that will be used to notify the guest when missing page becomes available. Also, to make it possible to distinguish between real #PF and APF, first 4 bytes of 64 byte memory location ('flags') will be written to by the hypervisor at the time of injection. Only first bit of 'flags' is currently supported, when set, it indicates that the guest is dealing with asynchronous 'page not present' event. If during a page fault APF 'flags' is '0' it means that this is regular page fault. Guest is supposed to clear 'flags' when it is done handling #PF exception so the next event can be delivered.

Note, since APF 'page not present' events use the same exception vector as regular page fault, guest must reset 'flags' to '0' before it does something that can generate normal page fault.

Bytes 5-7 of 64 byte memory location ('token') will be written to by the hypervisor at the time of APF 'page ready' event injection. The content of these bytes is a token which was previously delivered as 'page not present' event. The event indicates the page in now available. Guest is supposed to write '0' to 'token' when it is done handling 'page ready' event and to write 1' to MSR_KVM_ASYNC_PF_ACK after clearing the location; writing to the MSR forces KVM to re-scan its queue and deliver the next pending notification.

Note, MSR_KVM_ASYNC_PF_INT MSR specifying the interrupt vector for 'page ready' APF delivery needs to be written to before enabling APF mechanism in MSR_KVM_ASYNC_PF_EN or interrupt #0 can get injected. The MSR is available if KVM_FEATURE_ASYNC_PF_INT is present in CPUID.

Note, previously, 'page ready' events were delivered via the same #PF exception as 'page not present' events but this is now deprecated. If bit 3 (interrupt based delivery) is not set APF events are not delivered.

If APF is disabled while there are outstanding APFs, they will not be delivered.

Currently 'page ready' APF events will be always delivered on the same vcpu as 'page not present' event was, but guest should not rely on that.

MSR_KVM_STEAL_TIME:

0x4b564d03

data:

64-byte alignment physical address of a memory area which must be in guest RAM, plus an enable bit in bit 0. This memory is expected to hold a copy of the following structure:

```
struct kvm_steal_time {
      __u64 steal;
      __u32 version;
      __u32 flags;
      __u8  preempted;
      __u8  u8_pad[3];
      __u32 pad[11];
}
```

whose data will be filled in by the hypervisor periodically. Only one write, or registration, is needed for each VCPU. The interval between updates of this structure is arbitrary and implementation-dependent. The hypervisor may update this structure at any time it sees fit until anything with bit0 == 0 is written to it. Guest is required to make sure this structure is initialized to zero.

Fields have the following meanings:

version:

a sequence counter. In other words, guest has to check this field before and after grabbing time information and

make sure they are both equal and even. An odd version indicates an in-progress update.

flags:

At this point, always zero. May be used to indicate changes in this structure in the future.

steal:

the amount of time in which this vCPU did not run, in nanoseconds. Time during which the vcpu is idle, will not be reported as steal time.

preempted:

indicate the vCPU who owns this struct is running or not. Non-zero values mean the vCPU has been preempted. Zero means the vCPU is not preempted. NOTE, it is always zero if the the hypervisor doesn't support this field.

MSR_KVM_EOI_EN:

0x4b564d04

data:

Bit 0 is 1 when PV end of interrupt is enabled on the vcpu; 0 when disabled. Bit 1 is reserved and must be zero. When PV end of interrupt is enabled (bit 0 set), bits 63-2 hold a 4-byte aligned physical address of a 4 byte memory area which must be in guest RAM and must be zeroed.

The first, least significant bit of 4 byte memory location will be written to by the hypervisor, typically at the time of interrupt injection. Value of 1 means that guest can skip writing EOI to the apic (using MSR or MMIO write); instead, it is sufficient to signal EOI by clearing the bit in guest memory - this location will later be polled by the hypervisor. Value of 0 means that the EOI write is required.

It is always safe for the guest to ignore the optimization and perform the APIC EOI write anyway.

Hypervisor is guaranteed to only modify this least significant bit while in the current VCPU context, this means that guest does not need to use either lock prefix or memory ordering primitives to synchronise with the hypervisor.

However, hypervisor can set and clear this memory bit at any time: therefore to make sure hypervisor does not interrupt the guest and clear the least significant bit in the memory area in the window between guest testing it to detect whether it can skip EOI apic write and between guest clearing it to signal EOI to the hypervisor, guest must both read the least significant bit in the memory area and clear it using a single CPU instruction, such as test and clear, or compare and exchange.

MSR_KVM_POLL_CONTROL:

0x4b564d05

Control host-side polling.

data:

Bit 0 enables (1) or disables (0) host-side HLT polling logic.

KVM guests can request the host not to poll on HLT, for example if they are performing polling themselves.

MSR_KVM_ASYNC_PF_INT:

0x4b564d06

data:

Second asynchronous page fault (APF) control MSR.

Bits 0-7: APIC vector for delivery of 'page ready' APF events. Bits 8-63: Reserved

Interrupt vector for asynchnonous 'page ready' notifications delivery. The vector has to be set up before asynchronous page fault mechanism is enabled in MSR_KVM_ASYNC_PF_EN. The MSR is only available if KVM_FEATURE_ASYNC_PF_INT is present in CPUID.

MSR_KVM_ASYNC_PF_ACK:

0x4b564d07

data:

Asynchronous page fault (APF) acknowledgment.

When the guest is done processing 'page ready' APF event and 'token' field in 'struct kvm_vcpu_pv_apf_data' is cleared it is supposed to write '1' to bit 0 of the MSR, this causes the host to re-scan its queue and check if there are more notifications pending. The MSR is available if KVM_FEATURE_ASYNC_PF_INT is present in CPUID.

MSR_KVM_MIGRATION_CONTROL:

0x4b564d08

data:

This MSR is available if KVM_FEATURE_MIGRATION_CONTROL is present in CPUID. Bit 0 represents whether live migration of the guest is allowed.

When a guest is started, bit 0 will be 0 if the guest has encrypted memory and 1 if the guest does not have encrypted

memory. If the guest is communicating page encryption status to the host using the `KVM_HC_MAP_GPA_RANGE` hypercall, it can set bit 0 in this MSR to allow live migration of the guest.