[v13,08/12] KVM: x86: Add Intel PT context switch for each vcpu

10654369	diff (/patch/10654369/raw/)	mbox (/patch/10654369/mbox/)	series (/series/34463/mbox/)

Message ID	1540368316-12998-9-git-send-email-luwei.kang@intel.com
State	New
Headers	show
Series	Intel Processor Trace virtualization enabling
Related	hide [v13,00/12] Intel Processor Trace virtualization enabling (/cover/10654355/) [v13,01/12] perf/x86/intel/pt: Move Intel PT MSRs bit defines to global header (/patch/10654357/) [v13,02/12] perf/x86/intel/pt: Export pt_cap_get() (/patch/10654359/) [v13,03/12] perf/x86/intel/pt: Introduce intel_pt_validate_cap() (/patch/10654361/) [v13,04/12] perf/x86/intel/pt: Add new bit definitions for PT MSRs (/patch/10654379/) [v13,05/12] perf/x86/intel/pt: add new capability for Intel PT (/patch/10654363/) [v13,06/12] KVM: x86: Add Intel PT virtualization work mode (/patch/10654365/) [v13,07/12] KVM: x86: Add Intel Processor Trace cpuid emulation (/patch/10654367/) [v13,08/12] KVM: x86: Add Intel PT context switch for each vcpu [v13,09/12] KVM: x86: Introduce a function to initialize the PT configuration (/patch/10654377/) [v13,10/12] KVM: x86: Implement Intel PT MSRs read/write emulation (/patch/10654371/) [v13,11/12] KVM: x86: Set intercept for Intel PT MSRs read/write (/patch/10654373/) [v13,12/12] KVM: x86: Disable Intel PT when VMXON in L1 guest (/patch/10654375/)

Commit Message

Kang, Luwei (/project/kvm/list/?submitter=168537) Oct. 24, 2018, 8:05 a.m. UTC

```
From: Chao Peng <chao.p.peng@linux.intel.com>

Load/Store Intel Processor Trace register in context switch.
MSR IA32_RTIT_CTL is loaded/stored automatically from VMCS.
In Host-Guest mode, we need load/resore PT MSRs only when PT
is enabled in guest.

Signed-off-by: Chao Peng <chao.p.peng@linux.intel.com>
Signed-off-by: Luwei Kang <luwei.kang@intel.com>
---
 arch/x86/include/asm/intel_pt.h |  2 +
 arch/x86/kvm/vmx.c              | 94 +++++++++++++++++++++++++++++++++++++++++
 2 files changed, 96 insertions(+)
```

Comments

Alexander Shishkin (/project/kvm/list/?submitter=46111) Oct. 24, 2018, 10:13 a.m. UTC | #1 (/comment/22282983/)

```
Luwei Kang <luwei.kang@intel.com> writes:

> +static void pt_guest_enter(struct vcpu_vmx *vmx)
> +{
> +       if (pt_mode == PT_MODE_SYSTEM)
> +               return;
> +
> +       /* Save host state before VM entry */
> +       rdmsrl(MSR_IA32_RTIT_CTL, vmx->pt_desc.host.ctl);
> +
> +       /*
> +        * Set guest state of MSR_IA32_RTIT_CTL MSR (PT will be disabled
> +        * on VM entry when it has been disabled in guest before).
> +        */
> +       vmcs_write64(GUEST_IA32_RTIT_CTL, vmx->pt_desc.guest.ctl);
> +
> +       if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
> +               wrmsrl(MSR_IA32_RTIT_CTL, 0);
> +               pt_save_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
> +               pt_load_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
> +       }
> +}

From my side this is still a NAK, because [1].

[1] https://marc.info/?l=kvm&m=153847567226248&w=2

Thanks,
--
Alex
```

Kang, Luwei (/project/kvm/list/?submitter=168537) Oct. 25, 2018, 12:06 a.m. UTC | #2 (/comment/22284039/)

```
> > +static void pt_guest_enter(struct vcpu_vmx *vmx) {
> > +	if (pt_mode == PT_MODE_SYSTEM)
> > +		return;
> > +
> > +	/* Save host state before VM entry */
> > +	rdmsrl(MSR_IA32_RTIT_CTL, vmx->pt_desc.host.ctl);
> > +
> > +	/*
> > +	 * Set guest state of MSR_IA32_RTIT_CTL MSR (PT will be disabled
> > +	 * on VM entry when it has been disabled in guest before).
> > +	 */
> > +	vmcs_write64(GUEST_IA32_RTIT_CTL, vmx->pt_desc.guest.ctl);
> > +
> > +	if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
> > +		wrmsrl(MSR_IA32_RTIT_CTL, 0);
> > +		pt_save_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
> > +		pt_load_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
> > +	}
> > +}
>
> From my side this is still a NAK, because [1].
>
> [1] https://marc.info/?l=kvm&m=153847567226248&w=2
>

This place is save the host PT status and load the guest PT status before VM-entry when working in Host-Guest mode.

Thanks,
Luwei Kang
```

Paolo Bonzini (/project/kvm/list/?submitter=2536) Oct. 29, 2018, 5:48 p.m. UTC | #3 (/comment/22289839/)

```
On 24/10/2018 12:13, Alexander Shishkin wrote:
> Luwei Kang <luwei.kang@intel.com> writes:
>
>> +static void pt_guest_enter(struct vcpu_vmx *vmx)
>> +{
>> +	if (pt_mode == PT_MODE_SYSTEM)
>> +		return;
>> +
>> +	/* Save host state before VM entry */
>> +	rdmsrl(MSR_IA32_RTIT_CTL, vmx->pt_desc.host.ctl);
>> +
>> +	/*
>> +	 * Set guest state of MSR_IA32_RTIT_CTL MSR (PT will be disabled
>> +	 * on VM entry when it has been disabled in guest before).
>> +	 */
>> +	vmcs_write64(GUEST_IA32_RTIT_CTL, vmx->pt_desc.guest.ctl);
>> +
>> +	if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
>> +		wrmsrl(MSR_IA32_RTIT_CTL, 0);
>> +		pt_save_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
>> +		pt_load_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
>> +	}
>> +}
>
> From my side this is still a NAK, because [1].
>
> [1] https://marc.info/?l=kvm&m=153847567226248&w=2

Then you should have replied to
https://marc.info/?l=kvm&m=153865386015249&w=2 instead of having Luwei
do the work for nothing.

Quoting from there:

>> One shouldn't have to enable or disable anything in KVM to stop it from
>> breaking one's existing workflow. That makes no sense.
>
> If you "have to enable or disable anything" it means you have to
> override the default.  But the default in this patches is "no change
> compared to before the patches", leaving tracing of both host and guest
> entirely to the host, so I don't understand your remark.  What workflow
> is broken?
>
>> There already are controls in perf that enable/disable guest tracing.
>
> You are confusing "tracing guest from the host" and "the guest can trace
> itself".  This patchset is adding support for the latter, and that
> affects directly whether the tracing CPUID leaf can be added to the
> guest.  Therefore it's not perf that can decide whether to turn it on;
> KVM must know it when /dev/kvm is opened, which is why it is a module
> parameter.

I'd be happier if we found an agreement, but without discussion that
just won't happen.

Also, is there an existing interface to write a record into a tracing
buffer?

Paolo
```

Thomas Gleixner (/project/kvm/list/?submitter=107) Oct. 30, 2018, 10 a.m. UTC | #4 (/comment/22290933/)

```
On Mon, 29 Oct 2018, Paolo Bonzini wrote:
> On 24/10/2018 12:13, Alexander Shishkin wrote:
> > Luwei Kang <luwei.kang@intel.com> writes:
> >> +  /*
> >> +   * Set guest state of MSR_IA32_RTIT_CTL MSR (PT will be disabled
> >> +   * on VM entry when it has been disabled in guest before).
> >> +   */
> >> +  vmcs_write64(GUEST_IA32_RTIT_CTL, vmx->pt_desc.guest.ctl);
> >> +
> >> +  if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
> >> +          wrmsrl(MSR_IA32_RTIT_CTL, 0);
> >> +          pt_save_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
> >> +          pt_load_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
> >> +  }
> >> +}
> >
> > From my side this is still a NAK, because [1].
> >
> > [1] https://marc.info/?l=kvm&m=153847567226248&w=2
>
> Then you should have replied to
> https://marc.info/?l=kvm&m=153865386015249&w=2 instead of having Luwei
> do the work for nothing.
>
> Quoting from there:
>
> >> One shouldn't have to enable or disable anything in KVM to stop it from
> >> breaking one's existing workflow. That makes no sense.
> >
> > If you "have to enable or disable anything" it means you have to
> > override the default.  But the default in this patches is "no change
> > compared to before the patches", leaving tracing of both host and guest
> > entirely to the host, so I don't understand your remark.  What workflow
> > is broken?
> >
> >> There already are controls in perf that enable/disable guest tracing.
> >
> > You are confusing "tracing guest from the host" and "the guest can trace
> > itself".  This patchset is adding support for the latter, and that
> > affects directly whether the tracing CPUID leaf can be added to the
> > guest.  Therefore it's not perf that can decide whether to turn it on;
> > KVM must know it when /dev/kvm is opened, which is why it is a module
> > parameter.
>
> I'd be happier if we found an agreement, but without discussion that
> just won't happen.

So at least we need a way for perf on the host to programmatically detect,
that 'guest traces itself' is enabled, so it can inject that information
into the host data and post processing can tell that. W/o something like
```

```
that it's going to be a FAQ.

Thanks,

        tglx
```

Alexander Shishkin (/project/kvm/list/?submitter=46111) Oct. 30, 2018, 11:26 a.m. UTC | #5 (/comment/22291025/)

```
Paolo Bonzini <pbonzini@redhat.com> writes:

>> If you "have to enable or disable anything" it means you have to
>> override the default.  But the default in this patches is "no change
>> compared to before the patches", leaving tracing of both host and guest
>> entirely to the host, so I don't understand your remark.  What workflow
>> is broken?
>>
>>> There already are controls in perf that enable/disable guest tracing.
>>
>> You are confusing "tracing guest from the host" and "the guest can trace
>> itself".  This patchset is adding support for the latter, and that

I'm not confusing anything. In the terminology that you're using, the
latter breaks the former. This cannot happen.

>> affects directly whether the tracing CPUID leaf can be added to the
>> guest.  Therefore it's not perf that can decide whether to turn it on;
>> KVM must know it when /dev/kvm is opened, which is why it is a module
>> parameter.

There is a control in the perf event attribute that enables tracing the
guest. If this control is enabled, the kvm needs to stay away from any
PT related MSRs. Conversely, if kvm is using PT (or, as you say, "the
guest is tracing itself"), the host should not be allowed to ask for
tracing the guest at the same time.

Regards,
--
Alex
```

Paolo Bonzini (/project/kvm/list/?submitter=2536) Oct. 31, 2018, 10:43 a.m. UTC | #6 (/comment/22293129/)

```
On 30/10/2018 11:00, Thomas Gleixner wrote:
> On Mon, 29 Oct 2018, Paolo Bonzini wrote:
>> On 24/10/2018 12:13, Alexander Shishkin wrote:
>>> Luwei Kang <luwei.kang@intel.com> writes:
>>>> +  /*
>>>> +   * Set guest state of MSR_IA32_RTIT_CTL MSR (PT will be disabled
>>>> +   * on VM entry when it has been disabled in guest before).
>>>> +   */
>>>> +  vmcs_write64(GUEST_IA32_RTIT_CTL, vmx->pt_desc.guest.ctl);
>>>> +
>>>> +  if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
>>>> +          wrmsrl(MSR_IA32_RTIT_CTL, 0);
>>>> +          pt_save_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
>>>> +          pt_load_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
>>>> +  }
>>>> +}
>>>
>>> From my side this is still a NAK, because [1].
>>>
>>> [1] https://marc.info/?l=kvm&m=153847567226248&w=2
>>
>> Then you should have replied to
>> https://marc.info/?l=kvm&m=153865386015249&w=2 instead of having Luwei
>> do the work for nothing.
>>
>> Quoting from there:
>>
>>>> One shouldn't have to enable or disable anything in KVM to stop it from
>>>> breaking one's existing workflow. That makes no sense.
>>>
>>> If you "have to enable or disable anything" it means you have to
>>> override the default.  But the default in this patches is "no change
>>> compared to before the patches", leaving tracing of both host and guest
>>> entirely to the host, so I don't understand your remark.  What workflow
>>> is broken?
>>>
>>>> There already are controls in perf that enable/disable guest tracing.
>>>
>>> You are confusing "tracing guest from the host" and "the guest can trace
>>> itself".  This patchset is adding support for the latter, and that
>>> affects directly whether the tracing CPUID leaf can be added to the
>>> guest.  Therefore it's not perf that can decide whether to turn it on;
>>> KVM must know it when /dev/kvm is opened, which is why it is a module
>>> parameter.
>>
>> I'd be happier if we found an agreement, but without discussion that
>> just won't happen.
>
> So at least we need a way for perf on the host to programmatically detect,
> that 'guest traces itself' is enabled, so it can inject that information
> into the host data and post processing can tell that. W/o something like
> that it's going to be a FAQ.

In guest-tracing mode there will be already a TIP.PGD and TIP.PGE packet
respectively before vmentry and after vmexit, caused by the RTIT_CTL
WRMSRs in pt_guest_enter and pt_guest_exit.  The target IP of the
packets will come from kvm-intel.ko.

In system mode instead you get a Paging Information Packet on
vmentry/vmexit, with bit 0 set in the third byte.  You won't get it if
guest-side tracing is on (because tracing has been disabled by
pt_guest_enter and won't be re-enabled until pt_guest_exit).  I don't
think it's correct to "fake" the PIP in guest-tracing mode, because
TIP.PGD should be followed immediately by TIP.PGE.

Is this okay for perf users?

Paolo
```

Paolo Bonzini (/project/kvm/list/?submitter=2536) Oct. 31, 2018, 10:49 a.m. UTC | #7 (/comment/22293145/)

```
On 30/10/2018 12:26, Alexander Shishkin wrote:
>>> affects directly whether the tracing CPUID leaf can be added to the
>>> guest.  Therefore it's not perf that can decide whether to turn it on;
>>> KVM must know it when /dev/kvm is opened, which is why it is a module
>>> parameter.
>
> There is a control in the perf event attribute that enables tracing the
> guest. If this control is enabled, the kvm needs to stay away from any
> PT related MSRs.

This cannot happen once the guest has been told it can trace itself.
There is no standard way to tell the guest that the host overrode its
choice to use PT.  However, the host will get a PGD/PGE packet around
vmentry and vmexit, so there _will_ be an indication that the guest
owned the MSRs for that period of time.

If PT context switching is enabled with the module parameter, we could
also reject creation of events with the attribute set.  However that
won't help if the event is created before KVM is even loaded.

Paolo

> Conversely, if kvm is using PT (or, as you say, "the
> guest is tracing itself"), the host should not be allowed to ask for
> tracing the guest at the same time.
```

Alexander Shishkin (/project/kvm/list/?submitter=46111) Oct. 31, 2018, 11:38 a.m. UTC | #8 (/comment/22293217/)

```
Paolo Bonzini <pbonzini@redhat.com> writes:

> On 30/10/2018 12:26, Alexander Shishkin wrote:
>> There is a control in the perf event attribute that enables tracing the
>> guest. If this control is enabled, the kvm needs to stay away from any
>> PT related MSRs.
>
> This cannot happen once the guest has been told it can trace itself.

So, they need to be made mutually exclusive.

> There is no standard way to tell the guest that the host overrode its
> choice to use PT.  However, the host will get a PGD/PGE packet around
> vmentry and vmexit, so there _will_ be an indication that the guest
> owned the MSRs for that period of time.

Not if they are not tracing the kernel.

> If PT context switching is enabled with the module parameter, we could
> also reject creation of events with the attribute set.  However that
> won't help if the event is created before KVM is even loaded.

In that case, modprobe kvm should fail.

Regards,
--
Alex
```

Alexander Shishkin (/project/kvm/list/?submitter=46111) Oct. 31, 2018, 11:46 a.m. UTC | #9 (/comment/22293237/)

```
Paolo Bonzini <pbonzini@redhat.com> writes:

> On 30/10/2018 11:00, Thomas Gleixner wrote:
>> So at least we need a way for perf on the host to programmatically detect,
>> that 'guest traces itself' is enabled, so it can inject that information
>> into the host data and post processing can tell that. W/o something like
>> that it's going to be a FAQ.
>
> In guest-tracing mode there will be already a TIP.PGD and TIP.PGE packet
> respectively before vmentry and after vmexit, caused by the RTIT_CTL
> WRMSRs in pt_guest_enter and pt_guest_exit.  The target IP of the
> packets will come from kvm-intel.ko.

Most people aren't tracing the kernel, so they'd just get a PGD with no
address and a PGE after the kvm is done without any indication of what
happened in between.

> In system mode instead you get a Paging Information Packet on
> vmentry/vmexit, with bit 0 set in the third byte.  You won't get it if
> guest-side tracing is on (because tracing has been disabled by
> pt_guest_enter and won't be re-enabled until pt_guest_exit).  I don't
> think it's correct to "fake" the PIP in guest-tracing mode, because
> TIP.PGD should be followed immediately by TIP.PGE.

Indeed, we should most definitely not fake PIP. Perf has RECORD_AUX,
which already has PARTIAL flag that was introduced specifically because
of kvm.

Regards,
--
Alex
```

Paolo Bonzini (/project/kvm/list/?submitter=2536) Oct. 31, 2018, 12:07 p.m. UTC | #10 (/comment/22293283/)

```
On 31/10/2018 12:38, Alexander Shishkin wrote:
>> There is no standard way to tell the guest that the host overrode its
>> choice to use PT.  However, the host will get a PGD/PGE packet around
>> vmentry and vmexit, so there _will_ be an indication that the guest
>> owned the MSRs for that period of time.
>
> Not if they are not tracing the kernel.

If they are not tracing the kernel why should they be tracing the guest
at all?  If you only choose to trace userspace, anything that happens
between syscall entry and syscall exit is hidden and ioctl(KVM_RUN) _is_
a syscall.

>> If PT context switching is enabled with the module parameter, we could
>> also reject creation of events with the attribute set.  However that
>> won't help if the event is created before KVM is even loaded.
>
> In that case, modprobe kvm should fail.

Does that mean that an unprivileged user can effectively DoS
virtualization for everyone on the machine?  (Honest question).

I am aware that guest-side tracing blocks host-side guest tracing, but
that's exactly why it is entirely opt-in.  Only root can enable it by
switching the module parameter.  I don't see any way to change that,
other than rejecting this feature completely.

Paolo
```

Alexander Shishkin (/project/kvm/list/?submitter=46111) Oct. 31, 2018, 2:21 p.m. UTC | #11 (/comment/22293551/)

```
Paolo Bonzini <pbonzini@redhat.com> writes:

> On 31/10/2018 12:38, Alexander Shishkin wrote:
>>> There is no standard way to tell the guest that the host overrode its
>>> choice to use PT.  However, the host will get a PGD/PGE packet around
>>> vmentry and vmexit, so there _will_ be an indication that the guest
>>> owned the MSRs for that period of time.
>>
>> Not if they are not tracing the kernel.
>
> If they are not tracing the kernel why should they be tracing the guest
> at all?

To trace the guest userspace, perhaps?

>>> If PT context switching is enabled with the module parameter, we could
>>> also reject creation of events with the attribute set.  However that
>>> won't help if the event is created before KVM is even loaded.
>>
>> In that case, modprobe kvm should fail.
>
> Does that mean that an unprivileged user can effectively DoS
> virtualization for everyone on the machine?  (Honest question).

Would the leave-PT-to-the-host still be allowed? Would ignoring the
module parameter in that case and falling back to this mode still be
fine?

I'm not really the one to brainstorm solutions here. There are
possibilities of solving this, and the current patchset does not even
begin to acknowledge the existence of the problem, which is what my ACK
depends on.

Regards,
--
Alex
```

Paolo Bonzini (/project/kvm/list/?submitter=2536) Oct. 31, 2018, 2:43 p.m. UTC | #12 (/comment/22293635/)

```
On 31/10/2018 15:21, Alexander Shishkin wrote:
> Paolo Bonzini <pbonzini@redhat.com> writes:
>
>> On 31/10/2018 12:38, Alexander Shishkin wrote:
>>>> There is no standard way to tell the guest that the host overrode its
>>>> choice to use PT.  However, the host will get a PGD/PGE packet around
>>>> vmentry and vmexit, so there _will_ be an indication that the guest
>>>> owned the MSRs for that period of time.
>>>
>>> Not if they are not tracing the kernel.
>>
>> If they are not tracing the kernel why should they be tracing the guest
>> at all?
>
> To trace the guest userspace, perhaps?

Tracing the guest userspace and not the kernel is pretty much useless.
I'd also be surprised if it worked at all, and/or would consider it a
bug if it worked.

IMO tracing the kernel in system-wide mode should trace either all or
none of the guest, but certainly not just the guest kernel.  Tracing
userspace should trace none of the guest.

>>>> If PT context switching is enabled with the module parameter, we could
>>>> also reject creation of events with the attribute set.  However that
>>>> won't help if the event is created before KVM is even loaded.
>>>
>>> In that case, modprobe kvm should fail.
>>
>> Does that mean that an unprivileged user can effectively DoS
>> virtualization for everyone on the machine?  (Honest question).
>
> Would the leave-PT-to-the-host still be allowed? Would ignoring the
> module parameter in that case and falling back to this mode still be
> fine?

That would still prevent the feature from being accessed, until someone
with root access can rmmod kvm-intel.

> I'm not really the one to brainstorm solutions here. There are
> possibilities of solving this, and the current patchset does not even
> begin to acknowledge the existence of the problem, which is what my ACK
> depends on.

Well, one way it does acknowledge the existence of the problem is by not
turning the option on by default.

BTW, Intel (not you) also doesn't acknowledge the existence of the
problem, by not suggesting a solution in the SDM.  The SDM includes
```

```
examples of host-only, guest-only and combined tracing, but not separate
host and guest tracing.

Paolo
```

Patch

10654369	diff (/patch/10654369/raw/)	mbox (/patch/10654369/mbox/)	series (/series/34463/mbox/)

```
diff --git a/arch/x86/include/asm/intel_pt.h b/arch/x86/include/asm/intel_pt.h
index 4727584..eabbdbc 100644
--- a/arch/x86/include/asm/intel_pt.h
+++ b/arch/x86/include/asm/intel_pt.h
@@ -8,6 +8,8 @@
 #define PT_MODE_SYSTEM         0
 #define PT_MODE_HOST_GUEST     1
 
+#define RTIT_ADDR_RANGE                4
+
 enum pt_capabilities {
        PT_CAP_max_subleaf = 0,
        PT_CAP_cr3_filtering,
diff --git a/arch/x86/kvm/vmx.c b/arch/x86/kvm/vmx.c
index 692154c..d8480a6 100644
--- a/arch/x86/kvm/vmx.c
+++ b/arch/x86/kvm/vmx.c
@@ -978,6 +978,24 @@  struct vmx_msrs {
        struct vmx_msr_entry    val[NR_AUTOLOAD_MSRS];
 };
 
+struct pt_ctx {
+       u64 ctl;
+       u64 status;
+       u64 output_base;
+       u64 output_mask;
+       u64 cr3_match;
+       u64 addr_a[RTIT_ADDR_RANGE];
+       u64 addr_b[RTIT_ADDR_RANGE];
+};
+
+struct pt_desc {
+       u64 ctl_bitmask;
+       u32 addr_range;
+       u32 caps[PT_CPUID_REGS_NUM * PT_CPUID_LEAVES];
+       struct pt_ctx host;
+       struct pt_ctx guest;
+};
+
 struct vcpu_vmx {
        struct kvm_vcpu       vcpu;
        unsigned long         host_rsp;
@@ -1071,6 +1089,8 @@  struct vcpu_vmx {
        u64 msr_ia32_feature_control;
        u64 msr_ia32_feature_control_valid_bits;
        u64 ept_pointer;
+
+       struct pt_desc pt_desc;
 };
 enum segment_cache_field {
@@ -2899,6 +2919,69 @@  static unsigned long segment_base(u16 selector)
 }
 #endif
 
+static inline void pt_load_msr(struct pt_ctx *ctx, u32 addr_range)
+{
+	u32 i;
+
+	wrmsrl(MSR_IA32_RTIT_STATUS, ctx->status);
+	wrmsrl(MSR_IA32_RTIT_OUTPUT_BASE, ctx->output_base);
+	wrmsrl(MSR_IA32_RTIT_OUTPUT_MASK, ctx->output_mask);
+	wrmsrl(MSR_IA32_RTIT_CR3_MATCH, ctx->cr3_match);
+	for (i = 0; i < addr_range; i++) {
+		wrmsrl(MSR_IA32_RTIT_ADDR0_A + i * 2, ctx->addr_a[i]);
+		wrmsrl(MSR_IA32_RTIT_ADDR0_B + i * 2, ctx->addr_b[i]);
+	}
+}
+
+static inline void pt_save_msr(struct pt_ctx *ctx, u32 addr_range)
+{
+	u32 i;
+
+	rdmsrl(MSR_IA32_RTIT_STATUS, ctx->status);
+	rdmsrl(MSR_IA32_RTIT_OUTPUT_BASE, ctx->output_base);
+	rdmsrl(MSR_IA32_RTIT_OUTPUT_MASK, ctx->output_mask);
+	rdmsrl(MSR_IA32_RTIT_CR3_MATCH, ctx->cr3_match);
+	for (i = 0; i < addr_range; i++) {
+		rdmsrl(MSR_IA32_RTIT_ADDR0_A + i * 2, ctx->addr_a[i]);
+		rdmsrl(MSR_IA32_RTIT_ADDR0_B + i * 2, ctx->addr_b[i]);
+	}
+}
+
+static void pt_guest_enter(struct vcpu_vmx *vmx)
+{
+	if (pt_mode == PT_MODE_SYSTEM)
+		return;
+
+	/* Save host state before VM entry */
+	rdmsrl(MSR_IA32_RTIT_CTL, vmx->pt_desc.host.ctl);
+
+	/*
+	 * Set guest state of MSR_IA32_RTIT_CTL MSR (PT will be disabled
+	 * on VM entry when it has been disabled in guest before).
+	 */
+	vmcs_write64(GUEST_IA32_RTIT_CTL, vmx->pt_desc.guest.ctl);
+
+	if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
+		wrmsrl(MSR_IA32_RTIT_CTL, 0);
+		pt_save_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
+		pt_load_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
+	}
+}
+
+static void pt_guest_exit(struct vcpu_vmx *vmx)
+{
+	if (pt_mode == PT_MODE_SYSTEM)
+		return;
+
+	if (vmx->pt_desc.guest.ctl & RTIT_CTL_TRACEEN) {
+		pt_save_msr(&vmx->pt_desc.guest, vmx->pt_desc.addr_range);
+		pt_load_msr(&vmx->pt_desc.host, vmx->pt_desc.addr_range);
+	}
+
+	/* Reload host state (IA32_RTIT_CTL will be cleared on VM exit). */
+	wrmsrl(MSR_IA32_RTIT_CTL, vmx->pt_desc.host.ctl);
+}
+
 static void vmx_prepare_switch_to_guest(struct kvm_vcpu *vcpu)
 {
 	struct vcpu_vmx *vmx = to_vmx(vcpu);
@@ -6749,6 +6832,13 @@  static void vmx_vcpu_setup(struct vcpu_vmx *vmx)

 	if (cpu_has_vmx_encls_vmexit())
 		vmcs_write64(ENCLS_EXITING_BITMAP, -1ull);
+
+	if (pt_mode == PT_MODE_HOST_GUEST) {
+		memset(&vmx->pt_desc, 0, sizeof(vmx->pt_desc));
+		/* Bit[6~0] are forced to 1, writes are ignored. */
+		vmx->pt_desc.guest.output_mask = 0x7F;
+		vmcs_write64(GUEST_IA32_RTIT_CTL, 0);
+	}
 }

 static void vmx_vcpu_reset(struct kvm_vcpu *vcpu, bool init_event)
@@ -11260,6 +11350,8 @@  static void __noclone vmx_vcpu_run(struct kvm_vcpu *vcpu)
 	    vcpu->arch.pkru != vmx->host_pkru)
 		__write_pkru(vcpu->arch.pkru);

+	pt_guest_enter(vmx);
+
 	atomic_switch_perf_msrs(vmx);

 	vmx_update_hv_timer(vcpu);
@@ -11459,6 +11551,8 @@  static void __noclone vmx_vcpu_run(struct kvm_vcpu *vcpu)
 				  | (1 << VCPU_EXREG_CR3));
 	vcpu->arch.regs_dirty = 0;

+	pt_guest_exit(vmx);
+
 	/*
 	 * eager fpu is enabled if PKEY is supported and CR4 is switched
	 * back on host, so it is safe to read guest PKRU from current
```