**EPT简介**

* EPT is a feature that can be used to support the virtualization of physical memory.
* EPT是Intel VT-x对嵌套页表的支持，类似于操作系统页表（控制进程的虚拟地址到物理地址映射），EPT控制的是GPA到HPA的翻译。
* 类同于OS的页表GPT，EPT是多级的，提供R-W-X页控制，支持2M大页。（注意GPT不开PAE是不支持X位的设置的）
* OS的总页表指针被放在CR3中，同样的EPT的总页表指针被放在VMCS.EPTP里面
* 当发生GPT 的#PF ，CPU会调用中断表中的general protection exception handler(for R-W-X violation) or page fault handler(for non-present page access)。同样EPT在发生#NPF时，会首先发生#VMEXIT，然后调用nested page fault handler（统一的）
* 在AMD-V里，EPT类似于Rapid Virtualization Indexing（RVI）

**1、EPT原理**

Guest can have full control over Intel®64 page tables / events

– CR3, CR0, CR4 paging bits, INVLPG, page fault

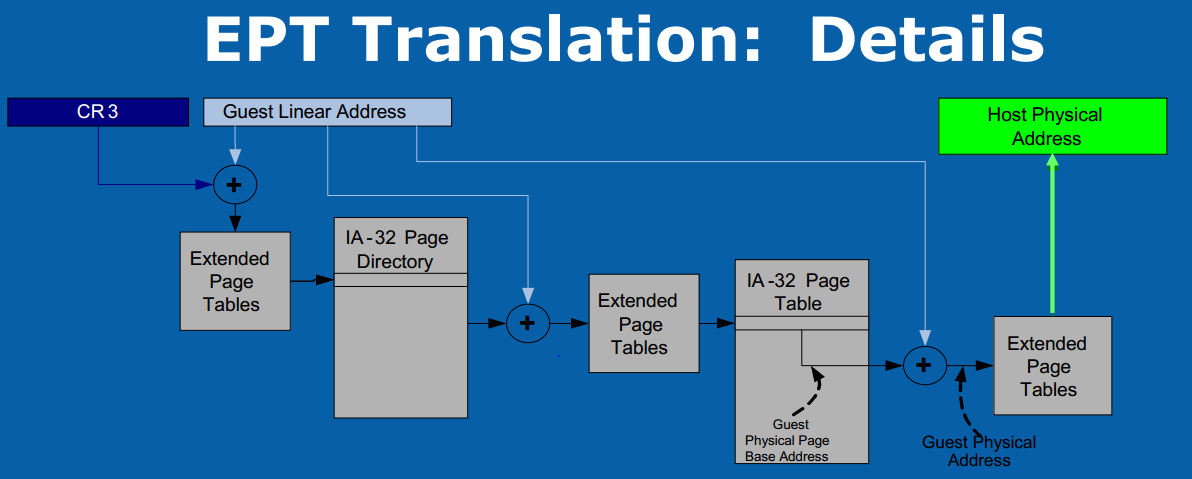
VMM controls Extended Page Tables

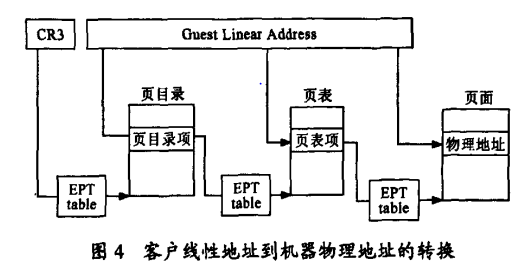
CPU uses both tables

EPT (optionally) activated on VM entry

– When EPT active, EPT base pointer (loaded on VM entry from VMCS) points to extended page tables

– EPT deactivated on VM exit





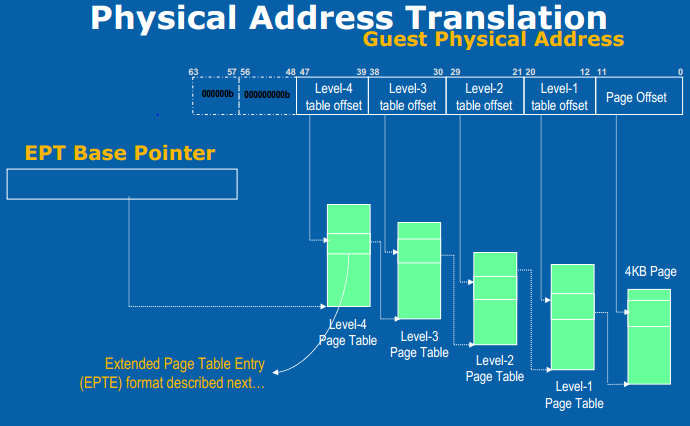
All guest-physical addresses go through extended page tables

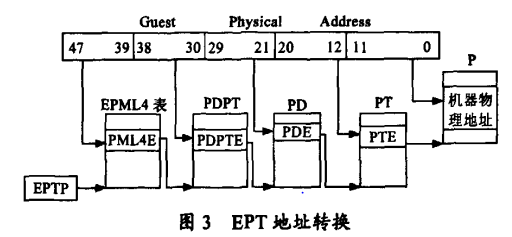
– Includes address in CR3, address in PDE, address in PTE, etc.

Example given is for basic 32-bit paging

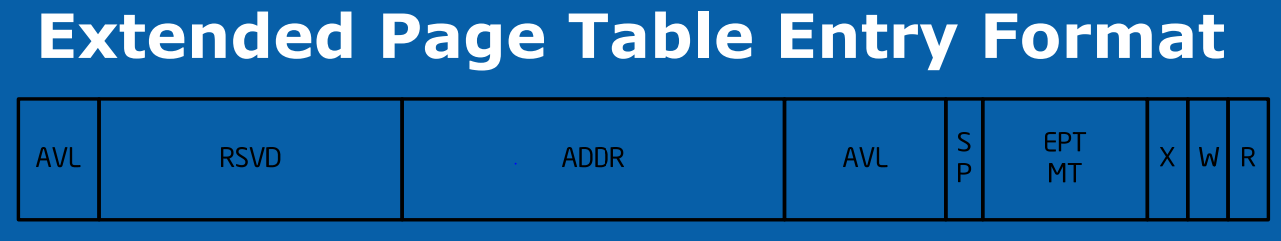
– Also applies to other paging modes (e.g., PAE and Intel® 64)

At leaf, Intel®64 page faults recognized before EPT violations.





EPTP结构



ADDR: Physical address of next table (if not super page and not EPTE) or of page frame (if super page or terminal entry)

SP: Super-page bit: if set, walk stops at a large page

Permission bits: read (R), write (W), execute (X)

EPT MT: Memory-typing controls

AVL: Software-available bit

EPT-based translations used (and cached) only when guest is running

EPT-based translations may be invalidated only when VMM is running

EPT-based translations are architecturally tagged with EPT Base Pointer:

– Tag based only on EPT base pointer

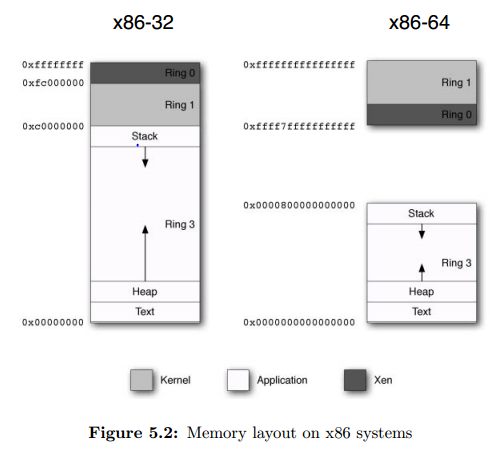
– Tag not based on guest CR3; guest CR loads still flush guest translations

**2、EPT的使用**

打开EPT功能后，虚拟机可以选择使用EPT，也可以选择不使用。默认情况是使用EPT，如果不使用EPT需要在虚拟机配置里添加hap=0

**3、EPT代码分析**

* 以xen的启动到domain0的创建整个流程为例，主要分析有关ept的机制。

经过xen的引导启动操作之后，跳转到第一个C函数\_\_start\_xen()完成后续的启动操作。\_\_start\_xen 会先建立va🡪pa映射表，然后设置xen地址空间，利用paging\_init()建立M2P页表。

\_\_start\_xen调用do\_presmp\_initcalls();通过遍历\_\_initcall\_start至\_\_presmp\_initcall\_end之间所有的函数，在设置其他vcpu之前对CPU进行初始化设置，包括do\_presmp\_initcalls(hvm\_enable)调用hvm\_enable ，这个函数会调用start\_vmx();start\_vmx()会调用vmx\_cpu\_up，利用vmx\_init\_vmcs\_config()初始化vmcs结构。





vmx\_init\_vmcs\_config中：

通过adjust\_vmx\_controls初始化\_vmx\_cpu\_based\_exec\_control。主要是通过adjust\_vmx\_controls这个函数获取系统设置。

min = (CPU\_BASED\_HLT\_EXITING|CPU\_BASED\_VIRTUAL\_INTR\_PENDING

|CPU\_BASED\_INVLPG\_EXITING|CPU\_BASED\_CR3\_LOAD\_EXITING |CPU\_BASED\_CR3\_STORE\_EXITING |CPU\_BASED\_MONITOR\_EXITING ………..）

opt = (CPU\_BASED\_ACTIVATE\_MSR\_BITMAP |CPU\_BASED\_TPR\_SHADOW

|CPU\_BASED\_MONITOR\_TRAP\_FLAG|

CPU\_BASED\_ACTIVATE\_SECONDARY\_CONTROLS);

\_vmx\_cpu\_based\_exec\_control = adjust\_vmx\_controls(

"CPU-Based Exec Control", min, opt,

MSR\_IA32\_VMX\_PROCBASED\_CTLS, &mismatch);

static u32 adjust\_vmx\_controls(

const char \*name, u32 ctl\_min, u32 ctl\_opt, u32 msr, bool\_t \*mismatch)

{

u32 vmx\_msr\_low, vmx\_msr\_high, ctl = ctl\_min | ctl\_opt;

rdmsr(msr, vmx\_msr\_low, vmx\_msr\_high);

ctl &= vmx\_msr\_high; /\* bit == 0 in high word ==> must be zero \*/

ctl |= vmx\_msr\_low; /\* bit == 1 in low word ==> must be one \*/

if ( ctl\_min & ~ctl )

{

\*mismatch = 1;

printk("VMX: CPU%d has insufficient %s (%08x; requires %08x)\n",

smp\_processor\_id(), name, ctl, ctl\_min);

}

return ctl;

}

/\* The IA32\_VMX\_EPT\_VPID\_CAP MSR exists only when EPT or VPID available \*/

if ( \_vmx\_secondary\_exec\_control & (SECONDARY\_EXEC\_ENABLE\_EPT |

SECONDARY\_EXEC\_ENABLE\_VPID) )

{

rdmsrl(MSR\_IA32\_VMX\_EPT\_VPID\_CAP, \_vmx\_ept\_vpid\_cap);

/\*

\* Additional sanity checking before using EPT:

\* 1) the CPU we are running on must support EPT WB, as we will set

\* ept paging structures memory type to WB;

\* 2) the CPU must support the EPT page-walk length of 4 according to

\* Intel SDM 25.2.2.

\* 3) the CPU must support INVEPT all context invalidation, because we

\* will use it as final resort if other types are not supported.

\*

\* Or we just don't use EPT.

\*/

if ( !(\_vmx\_ept\_vpid\_cap & VMX\_EPT\_MEMORY\_TYPE\_WB) ||

!(\_vmx\_ept\_vpid\_cap & VMX\_EPT\_WALK\_LENGTH\_4\_SUPPORTED) ||

!(\_vmx\_ept\_vpid\_cap & VMX\_EPT\_INVEPT\_ALL\_CONTEXT) )

\_vmx\_secondary\_exec\_control &= ~SECONDARY\_EXEC\_ENABLE\_EPT;

/\*

\* the CPU must support INVVPID all context invalidation, because we

\* will use it as final resort if other types are not supported.

\*

\* Or we just don't use VPID.

\*/

if ( !(\_vmx\_ept\_vpid\_cap & VMX\_VPID\_INVVPID\_ALL\_CONTEXT) )

\_vmx\_secondary\_exec\_control &= ~SECONDARY\_EXEC\_ENABLE\_VPID;

}

vmx\_init\_vmcs\_config()初始化vmcs结构结束后，会执行vmxon进入vmx模式。

\_\_vmxon(virt\_to\_maddr(this\_cpu(vmxon\_region)))

进入VMX模式后，接着执行：

if ( cpu\_has\_vmx\_ept )

ept\_sync\_all();

//ept\_sync\_all()主要是调用\_\_invept(int type, u64 eptp, u64 gpa)设置所有缓存映射无效，因为刚开启VMX模式。

到此，vmx\_cpu\_up结束，start\_vmx 结束，do\_presmp\_initcalls(hvm\_enable)结束。

\_\_start\_xen继续执行。do\_presmp\_initcalls进行一系列初始化后，执行创建domaon0操作如下：

/\* Create initial domain 0. \*/

dom0 = domain\_create(0, DOMCRF\_s3\_integrity, 0);

这里dom0能够感知虚拟化环境，能够利用P2M页表（P2M页表映射一个虚拟机，而M2P页表由xen创建，由xen维护，映射整个物理内存地址）操作直接寻址机器物理地址，或是执行hypercall与设备交互，属于PV型domain，不执行HVM创建，不创建shadow页表或ept页表。

注：start\_xen 执行完domain\_create创建dom0之后，调用construct\_dom0函数利用一些预先存放的模块进一步初始化dom0。

/\*

\* We're going to setup domain0 using the module(s) that we stashed safely

\* above our heap. The second module, if present, is an initrd ramdisk.

\*/

if ( construct\_dom0(dom0, mod, modules\_headroom,

(initrdidx > 0) && (initrdidx < mbi->mods\_count)

? mod + initrdidx : NULL,

bootstrap\_map, cmdline) != 0)

panic("Could not set up DOM0 guest OS\n");

construct\_dom0调用alloc\_vcpu：

for ( i = 1; i < opt\_dom0\_max\_vcpus; i++ )

{

cpu = cpumask\_cycle(cpu, cpupool0->cpu\_valid);

(void)alloc\_vcpu(d, i, cpu);

}

alloc\_vcpu首先利用 sched\_init\_vcpu(v, cpu\_id)对vcpu时间调度参数进行设置，把vcpu排入调度队列，然后利用vcpu\_initialise(v) 初始化v，利用

if ( is\_hvm\_domain(d) )

{

rc = hvm\_vcpu\_initialise(v);

goto done;

}

判断是不是hvm，如果不是则利用

v->arch.schedule\_tail = continue\_nonidle\_domain;

v->arch.ctxt\_switch\_from = paravirt\_ctxt\_switch\_from;

v->arch.ctxt\_switch\_to = paravirt\_ctxt\_switch\_to;

进行调度，如果是则利用hvm\_vcpu\_initialise(v)🡪 vm\_funcs.vcpu\_initialise(v))进行设置，利用以下转换：

.vcpu\_initialise = vmx\_vcpu\_initialise,

在vmx\_vcpu\_initialise中设置v->arch.schedule\_tail = vmx\_do\_resume;

（schedule\_tail可以理解为调度结尾所做的工作，这样每次调度一个vcpu时结尾就会调用vmx\_do\_resume。

如void continue\_running(struct vcpu \*same)

{

schedule\_tail(same);

BUG();

}）//直接执行schedule\_tail函数，跳过vcpu切换

在vmx\_do\_resume 中利用reset\_stack\_and\_jump(***vmx\_asm\_do\_vmentry***);

就可以VMEntry进入VMX Non-root模式运行。

每个VCPU都对应一个VMCS结构，所以vmxon、vmxentry、vmxexit、vmxoff不会引起vcpu切换。（A VMM could use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM could use a different VMCS for each virtual processor.）

VMExit

利用vmx\_vcpu\_initialise调用vmx\_create\_vmcs，vmx\_create\_vmcs调用construct\_vmcs，construct\_vmcs中执行\_\_vmwrite(**HOST\_RIP**, (unsigned long)vmx\_asm\_vmexit\_handler);将**HOST\_RIP所在地址指向**vmx\_asm\_vmexit\_handler，其中HOST\_RIP = 0x00006c16,这样VMExit发生时硬件可以直接转换到vmx\_asm\_vmexit\_handler分析处理, vmx\_asm\_vmexit\_handler处理后会调用vmx\_vmexit\_handler。

* 当我们在dom0中利用xl create命令创建HVM domain时，会依次执行以下操作：



在xl.c的main函数中执行cmdtable\_lookup(cmd) ，根据以下映射，因为是create命令，所以会接着执行main\_create函数。

struct cmd\_spec cmd\_table[] = {

{ "create",

&main\_create, 1, 1,

"Create a domain from config file <filename>",

"<ConfigFile> [options] [vars]",

"-h Print this help.\n"

"-p Leave the domain paused after it is created.\n"

"-c Connect to the console after the domain is created.\n"

"-f FILE, --defconfig=FILE\n Use the given configuration file.\n"

"-q, --quiet Quiet.\n"

"-n, --dryrun Dry run - prints the resulting configuration\n"

" (deprecated in favour of global -N option).\n"

"-d Enable debug messages.\n"

"-e Do not wait in the background for the death of the domain.\n"

"-V, --vncviewer Connect to the VNC display after the domain is created.\n"

"-A, --vncviewer-autopass\n"

" Pass VNC password to viewer via stdin."

}

main\_create🡪create\_domain🡪libxl\_domain\_create\_new🡪do\_domain\_create🡪initiate\_domain\_create🡪libxl\_\_domain\_make🡪xc\_domain\_create

通过xc\_domain\_create()创建控制结构体变量domctl；通过do\_domctl()生成超级调用请求；传递请求到OS内核：do\_xen\_hypercall()do\_privcmd通过ioctl来完成由3环到1环的转变，并完成超级调用。如下：

int xc\_domain\_create(xc\_interface \*xch,

uint32\_t ssidref,

xen\_domain\_handle\_t handle,

uint32\_t flags,

uint32\_t \*pdomid)

{

int err;

DECLARE\_DOMCTL;

domctl.cmd = XEN\_DOMCTL\_createdomain;

domctl.domain = (domid\_t)\*pdomid;

domctl.u.createdomain.ssidref = ssidref;（SSIDrefs：configure security，系统相关http://old-list-archives.xenproject.org/archives/html/xense-devel/2006-01/msg00002.html）

domctl.u.createdomain.flags = flags;

memcpy(domctl.u.createdomain.handle, handle, sizeof(xen\_domain\_handle\_t));

if ( (err = do\_domctl(xch, &domctl)) != 0 )

return err;

\*pdomid = (uint16\_t)domctl.domain;

return 0;

}

在do\_domctl函数中，根据传入参数，相应的会调用domain\_create如下。

case XEN\_DOMCTL\_createdomain:

d = domain\_create(dom, domcr\_flags, op->u.createdomain.ssidref);

其中domcr\_flags参数会指示创建的为HVM虚拟机，接着domain\_create 🡪arch\_ domain\_create，在arch\_ domain\_create中：

首先paging\_domain\_init 🡪hap\_domain\_init(d);初始化hap（硬件辅助页表，即ept）空闲页表。

void hap\_domain\_init(struct domain \*d)

{

INIT\_PAGE\_LIST\_HEAD(&d->arch.paging.hap.freelist);

}

然后hvm\_domain\_initialise🡪 paging\_enable 🡪hap\_enable分配页面供hap所用。

domain\_pause(d);

/\* Allow p2m and log-dirty code to borrow our memory \*/

d->arch.paging.alloc\_page = hap\_alloc\_p2m\_page;

d->arch.paging.free\_page = hap\_free\_p2m\_page;

if ( mode & PG\_translate )

/\* Allocate a new p2m table for a domain.

函数中p2m->phys\_table = pagetable\_from\_mfn(page\_to\_mfn(p2m\_top));

为p2m->phys\_table赋值，并利用各种ept页表操作函数完成ept页表创建

arch\_domain\_create🡪p2m\_init🡪p2m\_initialise🡪ept\_p2m\_init设置成ept操作函数\*/

rv = p2m\_alloc\_table(p2m\_get\_hostp2m(d));

/\* Now let other users see the new mode \*/

d->arch.paging.mode = mode | PG\_HAP\_enable;

最后，利用rc = hvm\_funcs.domain\_initialise(d); 确定eptp。

在vmx.c中：.domain\_initialise = vmx\_domain\_initialise,

static int vmx\_domain\_initialise (struct domain \*d)

{ int rc;

/\* Set the memory type used when accessing EPT paging structures. \*/

d->arch.hvm\_domain.vmx.ept\_control.ept\_mt = EPT\_DEFAULT\_MT;

/\* set EPT page-walk length, now it's actual walk length - 1, i.e. 3 \*/

d->arch.hvm\_domain.vmx.ept\_control.ept\_wl = 3;

d->arch.hvm\_domain.vmx.ept\_control.asr =

pagetable\_get\_pfn(p2m\_get\_pagetable(p2m\_get\_hostp2m(d)));

/\* #define p2m\_get\_pagetable(p2m) ((p2m)->phys\_table) \*/和上面形成对应。asr即为ept页表最顶层的一页，EPML4表。

if ( !zalloc\_cpumask\_var(&d->arch.hvm\_domain.vmx.ept\_synced) )

return -ENOMEM;

if ( (rc = vmx\_alloc\_vlapic\_mapping(d)) != 0 )

{ free\_cpumask\_var(d->arch.hvm\_domain.vmx.ept\_synced);

return rc;

} return 0;}

参考ept\_control定义可知union结构ept\_mt，ept\_wl，asr均已赋值，rsvd默认为6。eptp即已经赋值。

union {

struct {

u64 ept\_mt :3,

ept\_wl :3,

rsvd :6,

asr :52;

};

u64 eptp;

} ept\_control;

EPTP切换：vmx\_vcpu\_initialise 🡪vmx\_create\_vmcs🡪construct\_vmcs.

construct\_vmcs中执行：

if ( paging\_mode\_hap(d) )

\_\_vmwrite(EPT\_POINTER, d->arch.hvm\_domain.vmx.ept\_control.eptp);

其中EPT\_POINTER = 0x0000201a,一个eptp对应一个domain，这样每次初始化vcpu时就会设置EPTP值，指向EPT页表。



construct\_vmcs之后会调用paging\_update\_paging\_modes(v); 相应的更新Host页表和HOST\_CR3。在VMX模式中，VMCS调度运行相当于一般系统中进程的调度运行，每个VMCS结构对应各自的HOST\_CR3、GUEST\_CR3。

/\* will update HOST & GUEST\_CR3 as reqd \*/

/\* Update all the things that are derived from the guest's CR0/CR3/CR4.

\* Called to initialize paging structures if the paging mode

\* has changed, and when bringing up a VCPU for the first time. \*/

static inline void paging\_update\_paging\_modes(struct vcpu \*v)

{

paging\_get\_hostmode(v)->update\_paging\_modes(v);

}

hap\_paging\_real\_mode{

.update\_paging\_modes = hap\_update\_paging\_modes,

}

hap\_update\_paging\_modes中：

if ( pagetable\_is\_null(v->arch.monitor\_table) )

{

mfn\_t mmfn = hap\_make\_monitor\_table(v);

v->arch.monitor\_table = pagetable\_from\_mfn(mmfn);

make\_cr3(v, mfn\_x(mmfn));

hvm\_update\_host\_cr3(v);

}

static inline void

hvm\_update\_host\_cr3(struct vcpu \*v)

{

hvm\_funcs.update\_host\_cr3(v);

}

vmx\_function\_table中：

.update\_host\_cr3 = vmx\_update\_host\_cr3,

static void vmx\_update\_host\_cr3(struct vcpu \*v)

{

vmx\_vmcs\_enter(v);

\_\_vmwrite(HOST\_CR3, v->arch.cr3);

vmx\_vmcs\_exit(v);

}

arch\_vcpu几个页表对比：

struct arch\_vcpu{

pagetable\_t guest\_table; /\* (MFN) guest notion of cr3 \*/

/\* guest\_table holds a ref to the page, and also a type-count unless

\* shadow refcounts are in use \*/

pagetable\_t shadow\_table[4]; /\* (MFN) shadow(s) of guest \*/

pagetable\_t monitor\_table; /\* (MFN) hypervisor PT (for HVM) \*/

unsigned long cr3; /\* (MA) value to install in HW CR3 \*/

}

**4、EPT转换错误处理**

EPT在发生#NPF时，会首先发生#VMEXIT（前文介绍过VMEXIT设置）。调用vmx\_vmexit\_handler处理。

* case EXIT\_REASON\_EPT\_VIOLATION:

ept\_handle\_violation🡪 hvm\_hap\_nested\_page\_fault

1. nestedhvm\_enabled(v->domain)（是否开启嵌套虚拟化并处理）
2. p2m\_mem\_access\_check （处理against access错误）

/\* Check access permissions first, then handle faults \*/

/\* If the access is against the permissions, then send to mem\_event \*/

/\* First, handle rx2rw conversion automatically \*/

/\* Otherwise, check if there is a memory event listener, and send the message along \*/

如果有的话：/\* Pause the current VCPU \*/

/\* Send request to mem event \*/

/\* VCPU may be paused, return \*/

注：xen提供接口int xc\_hvm\_set\_mem\_access(

xc\_interface \*xch, domid\_t dom, hvmmem\_access\_t mem\_access, uint64\_t first\_pfn, uint64\_t nr)来设置页的访问权限。

1. /\*\* If this GFN is emulated MMIO or marked as read-only, pass the fault

to the mmio handler.\*/

1. /\* Check if the page has been paged out \*/
2. /\* Mem sharing: unshare the page and try again \*/
3. /\* Spurious fault? PoD and log-dirty also take this path. \*/
4. /\* Shouldn't happen: Maybe the guest was writing to a r/o grant mapping? \*/

hvm\_hap\_nested\_page\_fault处理完后打印有关错误。

* case EXIT\_REASON\_INVEPT:

hvm\_inject\_hw\_exception(TRAP\_invalid\_op, HVM\_DELIVER\_NO\_ERROR\_CODE);

hvm\_inject\_trap🡪inject\_trap🡪 vmx\_inject\_trap（向虚拟机注入异常，如下）

* case TRAP\_page\_fault
* case TRAP\_debug:
* case TRAP\_int3:

**5、EPT函数调用关系**

\_\_start\_xen（setup.c）调用domain\_create，domain\_create（Domain.c）调用arch\_domain\_create（xen-4.3.0/xen/arch/x86/domain.c），arch\_domain\_create调用paging\_domain\_init（xen-4.3.0\xen\arch\x86\mm\paging.c），paging\_domain\_init调用p2m\_init（xen-4.3.0\xen\arch\x86\mm\paging.c），p2m\_init调用p2m\_init\_hostp2m（xen-4.3.0\xen\arch\x86\mm\p2m.c），p2m\_init\_hostp2m调用p2m\_init\_one（xen-4.3.0\xen\arch\x86\mm\p2m.c），p2m\_init\_one调用p2m\_initialise（xen-4.3.0\xen\arch\x86\mm\p2m.c）

p2m\_initialise会判断hap是否开启且CPU是否支持vmx

if ( hap\_enabled(d) && cpu\_has\_vmx )

ret = ept\_p2m\_init(p2m);

else

p2m\_pt\_init(p2m);

开启则调用ept\_p2m\_init，否则调用p2m\_pt\_init

调用关系如下图

