# vSVA

vSVA的目标是在虚拟机里(qemu)，使的IO设备可以直接使用进程VA。所以，我们这里的假设是物理IO设备已经通过host上vfio驱动直通给虚拟机。要实现vSVA的目标，我们需要同时使能SMMU的S1,S2地址翻译，S1进行VA->IPA翻译，S2进行IPA->PA翻译，如果是host vfio使能，我们认为S2的翻译已经通过vfio配置在SMMU里。所以，vSVA的方案需要把虚拟机系统里的进程页表同步到host SMMU上。因为是vSVA，就有可能出现设备发起内存访问的时候，host SMMU上虚拟机里的进程页表项不存在的情况，  
所以，host上的SMMU要可以支持S1缺页。因为，S2用vfio支持，vfio采用pin内存的方式，暂时我们不需要S2的缺页。这里说的host上SMMU支持S1缺页，并不是在host系统上做S1缺页，我们这里讨论的是nested SMMU, 所以在host SMMU硬件检测到S1缺页的时候，应该把这个信息上报给guest里的SMMU，guest里使用和host一样的SMMU驱动处理缺页，当guest处理完这个缺页后，应该把对应的页表信息同步到SMMU的物理硬件上(SMMU.CD.TT0里)。因为，guest里的进程页表和SMMU CD上的页表物理上不是一个，很明显这里有一个设备和vcpu页表的同步问题，在host SVA上这个问题不存在，因为host SVA上cpu和SMMU是物理上共用相同页表。因此，在需要在vcpu无效化页表的时候，需要把信息同步到host的SMMU上，这个信息包括页表项和TLB。host SVA上也有这个问题，但是如果用SMMU stall mode, 可以配置DVM，把CPU侧TLB invalidate广播到SMMU，这样就不需要软件同步。在guest里多进程使用一个设备的资源，就需要支持PASID。这里的逻辑和上面的是一样的，只不过扩展到多进程。

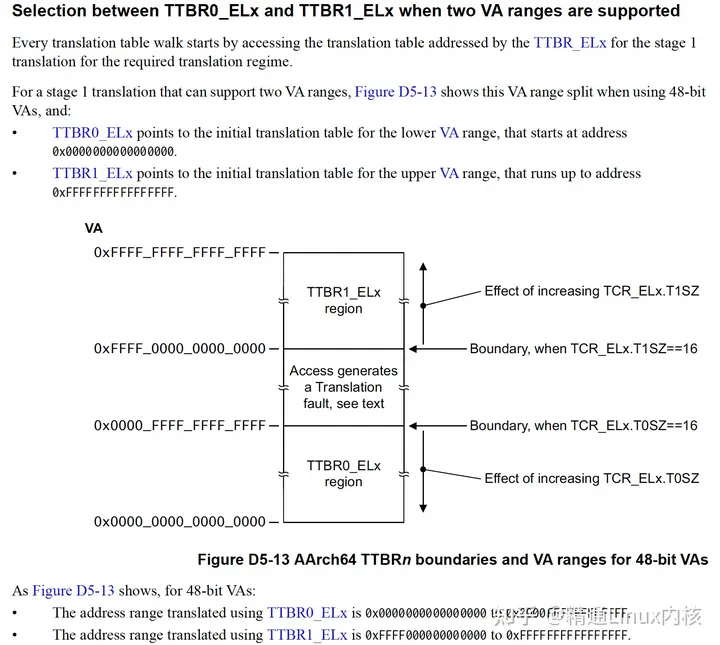
# LINUX ASID/VMID/PASID 编程接口

一般来说，我们使用ASID来标识进程的ID， 使用VMID来标识虚拟机ID，那么具体CPU/加速器/PCIe发起的一次操作，如何从硬件角度识别，并透传到后续模块呢？即如何标识不同的流？软硬件的编程接口是如何配合的？

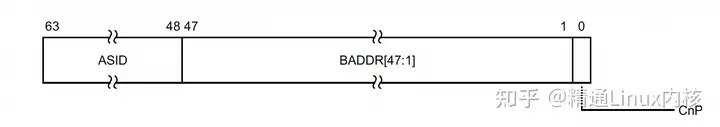
CPU视角

## ASID

首先，站在CPU视角，CPU通过TTBR0/TTBR1来控制不同EL等级下的内存域。当CPU下发操作的时候，通过访问地址的VA，确定当前应该使用TTBR0还是TTBR1:



简而言之，换做Linux，TTBR0管用户态，TTBR1管内核态（包括EL2），无论哪个level的TTBR寄存器，基本的组成形式如下：



其中寄存器域63-48合计16个bit标识ASID（这里存在一个卷绕的问题，内核巧妙的进行了处理）。

通过TTBR寄存器，可以识别不同的进程使用的页表，也就是说页表信息跟随进程绑定，我们在内核的task\_struct里面可以看到这一点：

struct task\_struct {

...

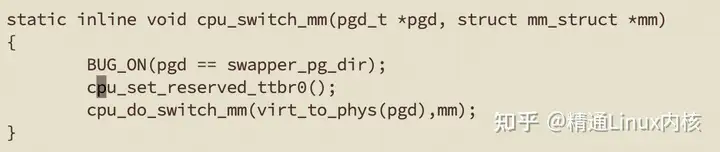
struct mm\_struct \*mm;

struct mm\_struct \*active\_mm;

...

}

在进程切换的时候，我们可以看到ASID的切换：



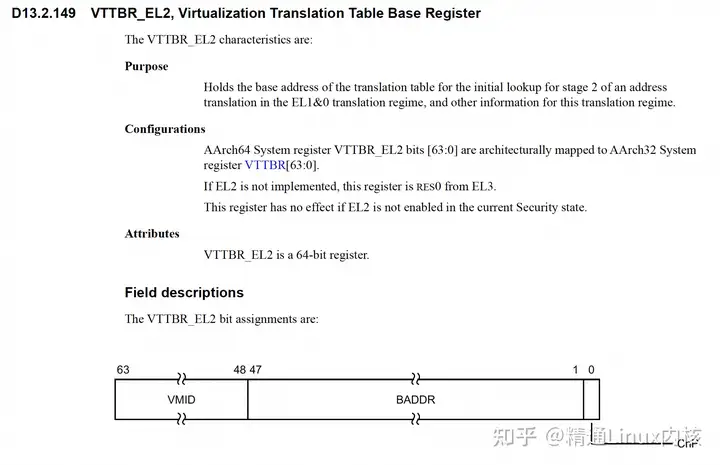


因此ASID通过页表，TTBR，task\_struct全部串联起来。

## VMID

如果在虚拟化场景，多个虚拟机之间的ASID可能存在一样的情况，这个时候，系统主要通过VMID区分两者。

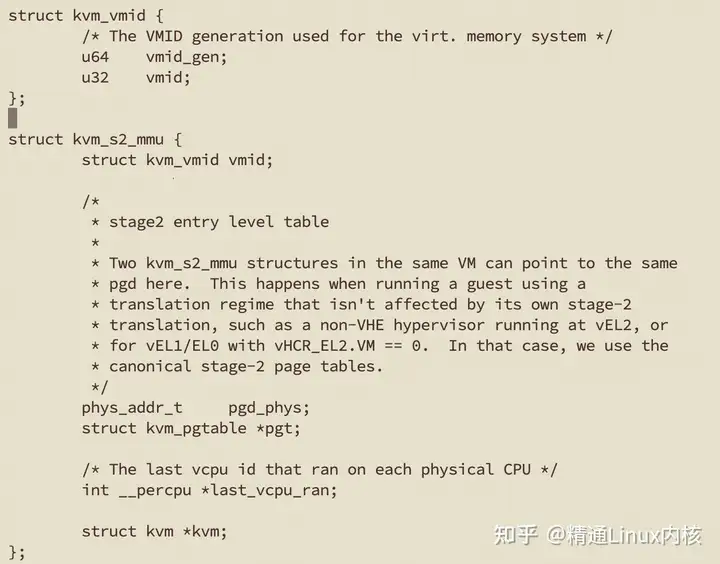
在ARM SPEC中，VMID主要设置于：

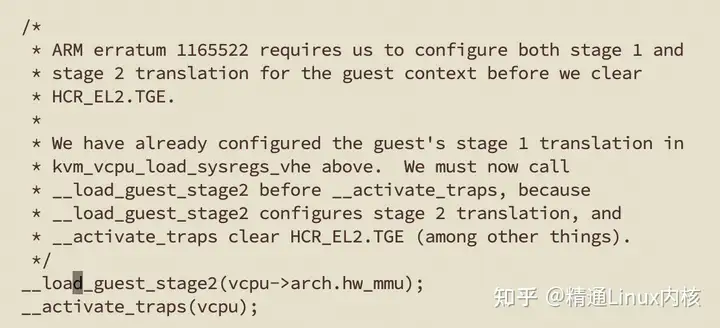


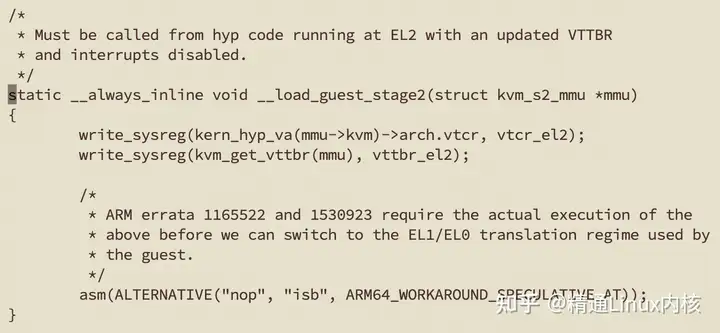
那么多个虚拟机之间的流怎么区分呢？

虚拟化场景下，CPU发出的请求是需要通过VA->IPA->PA的过程，在VA->IPA 的时候不需要关心VMID，因为此时在虚拟机内部，当IPA到PA的时候，就需要通过VTTBR获取VMID，从而标识这条流的ASID以及VMID。

因此VMID只有在虚拟机切换的时候会需要进行装载，因此在内核代码中，我们可以看到：







\_\_kvm\_vcpu\_run->\_\_kvm\_vcpu\_run\_vhe->\_\_load\_guest\_stage2

因此，Linux将ASID跟task\_struct进程绑定，VMID跟kvm\_vcpu 虚拟机绑定，完成了CPU 侧的stream id的编程。

## PASID

IO视角其实与CPU非常一致，可以将PCIe设备当成一个CPU来处理，传统的DMA编程模型下，如果CPU填充PCIe的是PA，此时SMMU要么是关闭，要么是直接透传。

在关闭的情况下，设备发出的请求是没有携带标识的（除非在出口处做了处理），在透传的情况下，相当于进行了stage1的转换。

在SMMU的spec中，使用了类似ASID以及VMID的标识来区分不同等级的流，即STREAMID,SUBSTREAMID（PCIe体系中，命名为RequesterID和PASID）。

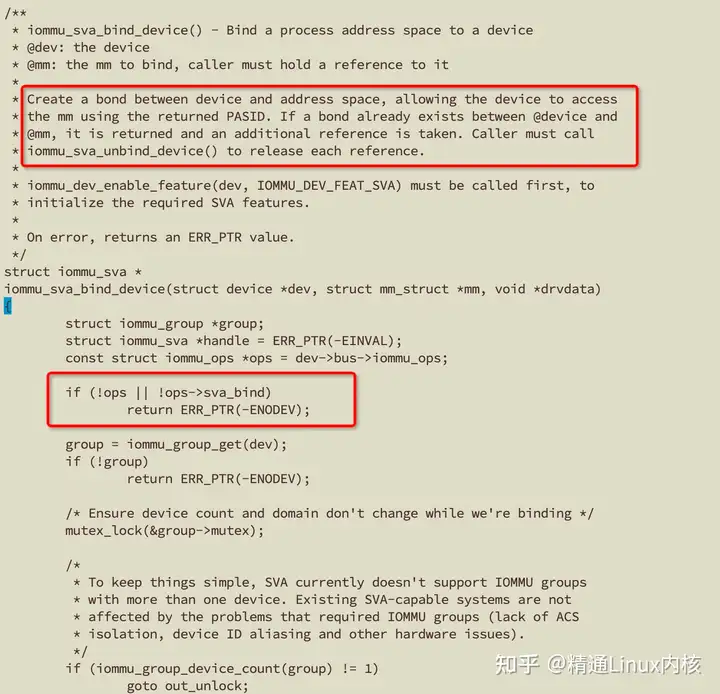
当只进行了一级的转换，只需要给转换的流携带STREAMID即可，而SUBSTREAMID在stage2的时候使用，目前看到的场景，仅在虚拟化中使用SVA可能会使用到。

pasid同样在结构体mm\_struct中定义：

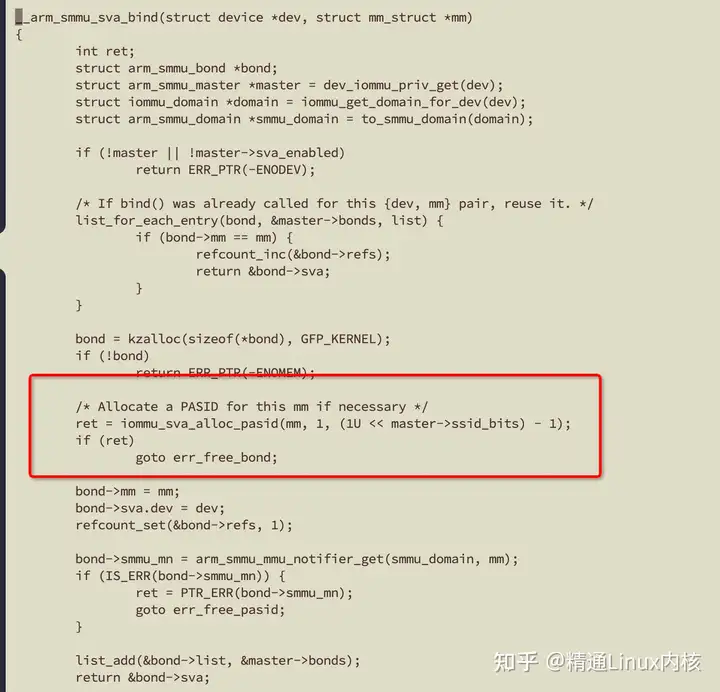


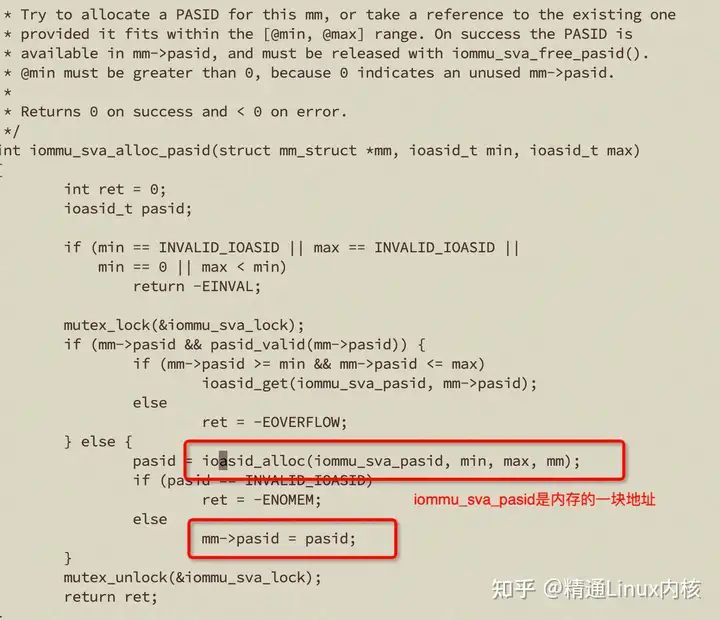
那么在虚拟机中的设备如何绑定某个进程的呢？（因为SVM跟进程相关，不同进程使用的地址空间不一样）

* 首先，进程 DMA前会通过bind，将进程地址空间跟设备绑定起来：



* 其次，会通sva\_bind分配一个pasid给对应的进程：





注意，iommu\_bond这个结构并不对外, 用iommu\_sva\_bind\_device/unbind接口时，函数参数都是mm\_struct(进程地址空间)。使用SVA的设备可以把一个设备的一些资源和一个进程地址空间绑定，这种绑定关系是灵活的，比如可以一个设备上的不同资源和不用的进程地址空间绑定(bond 1, bond 2)，还可以同一个设备上的资源都绑定在一个进程的地址空间上(bond 3,bond 4)。从进程地址空间的角度看，一个进程地址空间可能和多个设备资源绑定。iommu\_bond指的就是一个绑定，io\_mm指的是绑定了外设资源的一个进程地址空间。io\_pgtables是指内核dma接口申请内存的页表。

