第??卷 第?期 计 算 机 学 报 Vol. ?? No. ?
2023年12月 CHINESE JOURNAL OF COMPUTERS ???. 20??

虚拟机的进程的虚实地址转换综述

高正丰

摘 要

在单机换机之上，运行多种操作系统的需求越来越高，但有一些底层问题需要解决。面对单机单操作系统中，通常需要硬件配合完成虚实地址转换。在硬件环境缺乏的情况下，针对虚拟机之上的进程如何实现虚实地址转换需要进行解决。目前提出基本的解决方案是影子页表与嵌套页表。本文对这两种方式进行深入介绍，并且在此之上介绍新的机制，TPT框架。

关键词 虚拟机；虚实地址转换；嵌套页表；影子页表

Abstract

On top of a stand-alone replacement, the need to run multiple operating systems is getting higher and higher, but there are some underlying problems that need to be solved. In a single-machine and single-operating system, hardware is usually required to complete virtual-real address translation. In the case of lack of hardware environment, how to implement virtual-real address translation for processes on virtual machines needs to be solved. The basic solution proposed so far is a shadow page table and a nested page table. This article provides an in-depth introduction to these two approaches, and introduces a new mechanism, the TPT framework, on top of them.

1 引言

计算机在运行过程之中需要使用到指令以及数据，这些代码数据都被存放在内存之中。内存架构按地址进行区分。所以只需要给出内存的物理地址，CPU就能够准确找到想要的数据或者代码。但在操作系统之中，一方面，程序员在编写代码时候，并不能预知代码即将被装在内存什么地方，所以对于程序员，需要使用逻辑地址方便程序员的代码编写。另一方面，出于内存利用率方面考虑，现代计算机一般对内存以及进程进行分页管理，将进程相应页调入内存的页之中。正常运行在单机环境之下，分页管理使用到CR3基址寄存器，指出页表的起始地址，再根据这个起始地址进行计算，得到最终的物理地址。

但目前存在运行单机之上的虚拟机环境，在虚拟机上面运行进程这种情况。这种环境下，硬件条件较为苛刻，常规的单机地址转换过程可能不能够满足，需要研究新的方式满足以往的地址转换的要求。目前采用嵌套页表或者影子页表能够解决在硬件资源欠缺的情况下进行地址转换的问题。

1.1 逻辑地址与物理地址

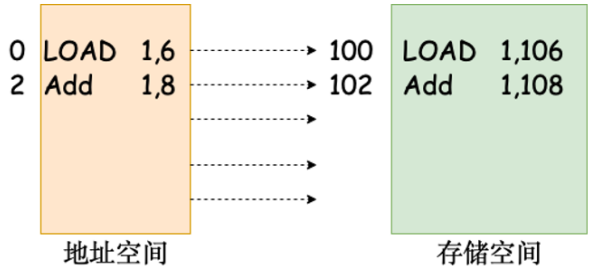

图1 地址转换

逻辑地址和物理地址是计算机中两个重要的地址概念，它们分别代表了不同的含义和作用。

逻辑地址（Logical Address），也称虚拟地址，是一个程序所使用的地址空间，是程序代码中使用的地址，它是由程序员或操作系统生成的。逻辑地址是指向内存中的一段地址空间，它是相对于程序的起始地址（即基址）的偏移量。在程序执行时，逻辑地址会被翻译成物理地址。

物理地址（Physical Address）是内存中真实的地址，它是由CPU生成的，用来访问实际的内存单元。物理地址是CPU传递给内存控制器的地址，通过地址总线传递到内存模块，访问实际的存储单元。

在计算机中，逻辑地址和物理地址的转换是由操作系统中的内存管理单元（Memory Management Unit，MMU）来完成的。MMU使用一个叫做页表（Page Table）的数据结构来将逻辑地址翻译成物理地址。

1.2 地址转换与分页管理

当程序要访问某个内存地址时，CPU会把逻辑地址发送给MMU进行转换。MMU根据页表将逻辑地址转换为物理地址，并将物理地址传递给内存控制器，访问相应的内存单元。如果页表中不存在该逻辑地址的映射，就会发生缺页异常（Page Fault），此时操作系统会将对应的物理页面从硬盘加载到内存中，再进行地址映射。

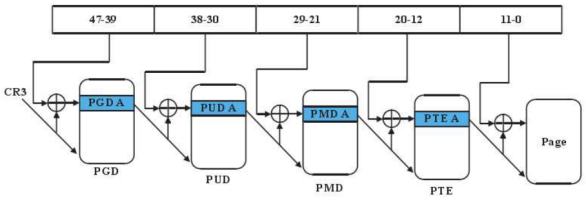

图2 四级页表结构

出了四级页表结构，多的项数还会导致查找、更新等操作变得耗时。为了解决这一问题，研究者们提间的页表本身也非常庞大。庞大的页表不仅占据了相当一部分内存空间，由于过现代计算机系统支持的物理内存空间越来越大，用于管理如此巨大的内存空

现代计算机系统支持的物理内存空间越来越大，用于管理如此巨大的内存空间的页表本身也非常庞大。庞大的页表不仅占据了相当一部分内存空间，由于过多的项数还会导致查找、更新等操作变得耗时。为了解决这一问题，研究者们提出了四级页表结构。

在四级页表结构中，对页表进行分级，由高级到低级分别称之为PGD(Page Global Directory)、PUD(Page Upper Directory)、PMD(Page Middle Directory)和PTE(Page Table Entry)。每个页表大小为4KB，内含512个页表项（PTE），每个页表项占用8字节。最高级的PGD只包含一个页表，它的起始物理地址被存放在特定的CR3寄存器中。四级页表中的每个页表项都相当于一个索引，指向某个三级页表的起始物理地址。以此类推，直到在最后一级页表的页表项中找到数据所在的物理页框号，最终和虚拟地址中的低12位偏移信息拼合形成物理地址。使用多级页表结构不但可以加快页表的查询速度，而且可以让内存中的页表利用虚拟内存机制，将部分低级页表（如二级页表和一级页表）放在磁盘中，当程序需要时再通过内存缺页机制将这些页表从磁盘换入内存，减少页表使用的内存空间。

1.3 虚拟机

虚拟化技术能够在一台物理计算机上创建多个虚拟机，每个虚拟机具有各自的操作系统(OS)和应用。虚拟机无法与物理计算机直接交互。而是需要借助一个叫做虚拟机管理器的轻量级软件层，在虚拟机与底层物理硬件之间进行协调。虚拟机管理器将物理计算资源（例如处理器、内存和存储）分配给每个虚拟机。它使虚拟机之间相互隔离，互不干扰。

虚拟机分类：

虚拟机管理器主要有两种类型。

类型1 虚拟机管理器直接在物理硬件上运行，作用是取代操作系统。可将一个虚拟机作为模板，通过复制模板以创建新的虚拟机。可针对不同用途（例如软件测试、生产数据库和开发环境），按需创建多个虚拟机模板。

类型2 虚拟机管理器作为应用在主机操作系统中运行。如果使用类型2 虚拟机管理器，则需要手动创建虚拟机，然后在其中安装访客操作系统。可以使用虚拟机管理器为虚拟机分配物理数量和内存容量。根据虚拟机资源，并手动设置虚拟机可以使用的处理器核心管理器的功能。

1.4 问题引出

那么针对于第二类虚拟机，建立在于VMM之上的虚拟机的操作系统中的进程，如何实现虚实地址的转换？

如果这个操作系统是运行在虚拟机上的，那么这只是一个中间的物理地址（Intermediate Phyical Address - IPA），需要经过VMM/hypervisor的转换，才能得到最终的物理地址（Host Phyical Address - HPA）。从VMM的角度，guest VM中的虚拟地址就成了GVA(Guest Virtual Address)，IPA就成了GPA(Guest Phyical Address)。

一个简单的思路就是，先在VM的操作系统中，使用操作系统的页表机制将GVA转换为GPA，通过VMM，将GPA映射为HVA，最后再由宿主机的操作系统将HVA映射为HPA。但这又存在硬件上的问题。传统的IA32架构从硬件上只支持一次地址转换，即由CR3寄存器指向进程第一级页表的首地址，通过MMU查询进程的各级页表，获得物理地址。如果按照刚才的思路进行下去，一开始在GVA转换为GPA时候，按照正常的逻辑，他需要GPA的“物理地址”（实质上是逻辑地址）存在CR3中，然后进行页表查询。但GPA页表实际上也应该映射为一个内存物理地址空间。在单机上，一般存储在CR3寄存器当中。但在虚拟机的逻辑下，他又需要存储GPA的地址，也需要将这个GPA页表地址真正映射为内存的物理地址，这时候在物理元器件上就得不到满足。因此目前提出了影子页表与嵌套页表来解决此问题。

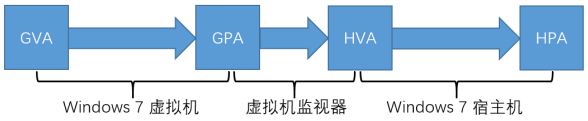

图3 虚拟机上的地址转换

2 解决方法

2.1 影子页表

影子页表，新建并维护着一张页表，就是影子页表，进行客户机虚拟地址和宿主机物理地址（机器地址）之间的映射.在进行虚实地址转换时候，真正被VMM载入到物理MMU中的页表是影子页表，而不是虚拟机中的页表。通过影子页表，则可以实现客户机虚拟地址到宿主机物理地址的直接转换。通过维护记录GVA-¿HPA的影子页表SPT，减少了地址转换带来的开销，可以直接将GVA 转换为HPA。

一个关键点需要思考的是，引入SPT之后，系统如何使用SPT进行调用，并且能够达到CR3重复占用的情况。

在加入了SPT 技术后，当Guest 访问CR3 时，VMM 会捕获到这个操作EXIT_REASON_CR_ACCESS，之后VMM 会载入特殊的CR3 和影子页表，欺骗Guest 这就是真实的CR3。之后就和传统的访问内存方式一致，当需要访问物理内存的时候，只会经过一层影子页表的转换。基于这种软件实现方式进行地址转换时候，实质上只使用到了一个CR3，满足物理条件进行内存转换。

另外一个需要考虑的问题是，在引入了SPT后，他的内容是何时更新的？SPT在虚拟机进程运行之中，肯定不知道进程所需要的地址空间大小如何。因此，SPT依旧保持了虚拟机进程在虚拟机操作系统之上的页表。并且根据这个页表进行跟踪，将其转换为物理地址。

KVM 会将Guest 的页表设置为只读，当Guest OS 对页表进行修改时就会触发Page Fault，VM-EXIT 到KVM，之后KVM 会对GVA 对应的页表项进行访问权限检查，结合错误码进行判断：

如果是Guest OS 引起的，则将该异常注入回去，Guest OS 将调用自己的缺页处理函数，申请一个Page，并将Page 的GPA 填充到上级页表项中

如果是Guest OS 的页表和SPT 不一致引起的，则同步SPT，根据Guest 页表和mmap 映射找到GPA 到HVA 的映射关系，然后在SPT 中增加/更新GVA-HPA表项

由此就可以使用SPT建立虚拟机之上的进程的逻辑地址到宿主机的物理地址的映射关系，它解决了CR3寄存器在转换之中的占用问题。但与此同时，他引入了额外的问题，即内存占用问题。因为SPT是针对开辟的单位虚拟机进程，需要为每一个虚拟机进程都开辟SPT，这就带来了严重的内存开销。并且在虚拟机进程切换时候，可能也会带来

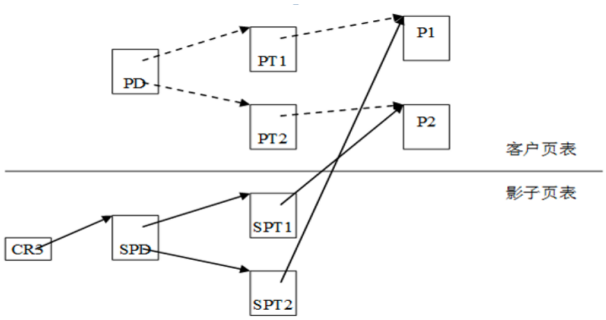

图4 影子页表

查找的问题，虽然可以使用hash table来提升检索速度，但也是一部分的时间开销。

最终呈现的影子页表的结构如图所示。会让进程维持自己的页表结构，同时添加影子页表结构。在地址转换的时候，实际上使用影子页表装填CR3内容，但是计算时候的偏移量采用进程的逻辑地址，从而计算出真正的物理地址。

2.2 嵌套页表

另外一种采用方式是通过硬件EPT的方式解决。最大的问题是在转换过程中，可能出现寄存器的占用切换问题。另外一个硬件解决方法，就是直接增添此类硬件即可。

Intel EPT 技术引入了EPT（Extended Page Table）和EPTP（EPT base pointer）的概念。EPT 中维护着GPA 到HPA 的映射，而EPTP 负责指向EPT。EPT 技术在原有客户机页表对客户机虚拟地址到客户机物理地址映射的基础上，又引入了EPT 页表来实现客户机物理地址到宿主机物理地址的另一次映射。这两次地址映射都是由硬件自动完成。客户机运行时，客户机页表被载入物理CR3，而EPT 页表被载入专门的EPT 页表指针寄存器EPTP。于是在进行地址转换时，首先通过CR3 指向的页表实现GVA 到GPA 的转换，再通过EPTP 指向的EPT 完成GPA 到HPA 的转换。当发生EPT Page Fault 时，需要VM-EXIT 到KVM，更新EPT。

他的优点是虚拟机进程页与底层页没有关系，虚拟机进程页表与EPT保持独立性，各自进行维护即可。同时相较于影子页表，他也不需要针对虚拟机进程就维护一个SPT，他只需要针对于一个虚拟机维护一个EPT即可。此外，他在地址转换时候，通过硬件直接完成，效率较高。

但他相较于影子页表有一个额外缺点。他看似通过硬件直接完成，效率更高，时间较短。但这个效率，针对的是这一次地址转换效率较高。如果存在多次的嵌套页表，EPT反而效率并没有很高，反而更低下。因为在虚拟机进程进行地址转换时候，

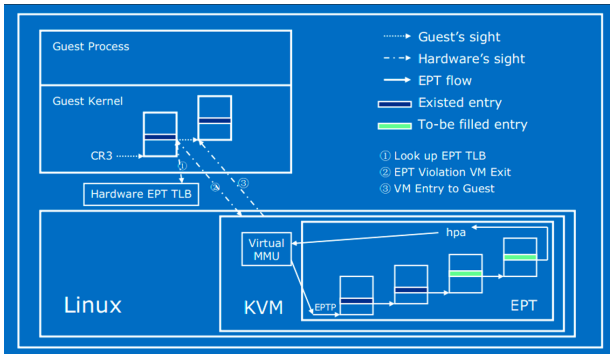

图5　嵌套页表

页表本质上是一个逻辑地址，他需要进行一次地址转换，这时候，他需要一次EPT。在通过页表找到一级页表的逻辑地址时候，又需要一次EPT转换，才能够找到一级页表的物理地址，进行访问。这就导致了在虚拟机进程页表为多层嵌套页表的情况下，在虚拟机进程进行逻辑地址到物理地址转换的过程中，出现多次进行EPT转换的过程，从而导致效率的低下。

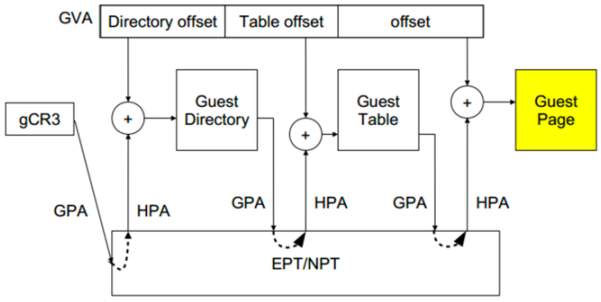

图6　嵌套页表地址转换过程

2.3　建立映射关系

无论是SPT还是EPT，另外一个需要深入思考的点是，如何建立GPA到HPA的映射关系？前面都提及了SPT的全新机制，以及EPT引入新硬件能够分别使得VMM实现GVA到HPA的映射，以及GPA到HPA的映射。但并没有描述针对EPT以及SPT的填充的内容到底是如何获得的。

VMM是建立在宿主机操作系统上的，可以认为他就是宿主机上面的一个进程，他会有自己的逻辑地址空间。那么运行在VMM之上的虚拟机，可以认为是VMM自己的一个线程，他可以将自己的逻辑地址分配出去，分配给建立在自己之上的虚拟机。并且记录下来，这时候分配出去的空间，在虚拟机看来，就是自己的PA，也就是前文提及的GPA。那么只要将VMM自己的逻辑地址，交由给宿主机操作系统，让他进行一个逻辑地址到物理地址的转换，就能够建立起GPA到HPA的映射。所以在EPT在进行GPA到HPA的映射时候，就需要将刚才的GPA到HVA记录下来，进行这一步的转换之后，就可以交由宿主机操作系统完成HVA到HPA的转换，第一步转换只需要哈希映射就能解决，不需要额外硬件支持。

2.4　TPT架构

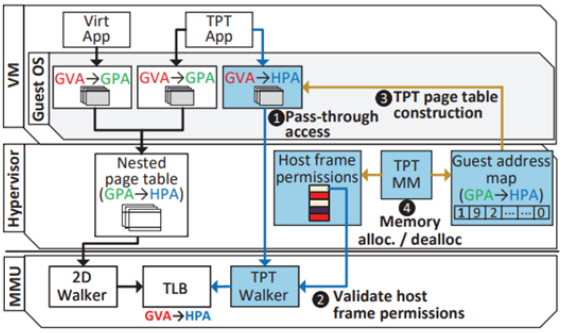

图7　TPT架构

首先，TPT架构他直接让进程设置，是否支持TPT。若需要TPT架构，为其开辟右侧蓝色部分。若其要求不支持，就继续按照以往的方式进行虚实地址转换，这里论文提及是可以使用任何方式，图片仅以嵌套分页作为一个展示。

TPT思路也是建立出一个名为TPT的页表能够直接完成GVA到HPA的映射一步到位的方式。里面内容来自于TPTMM生成的用户地址映射。TPTMM中可以根据自己将自己的HVA分配给虚拟机的地址即GPA，然后根据自己的HVA可以让其映射为HPA，那么可以认为他在这里可以获得所有的GPA到HPA的映射，这时候就可以根据TPTMM为虚拟机进程生成对应的GPA到HPA的映射，根据这个内容对TPT进行填充。

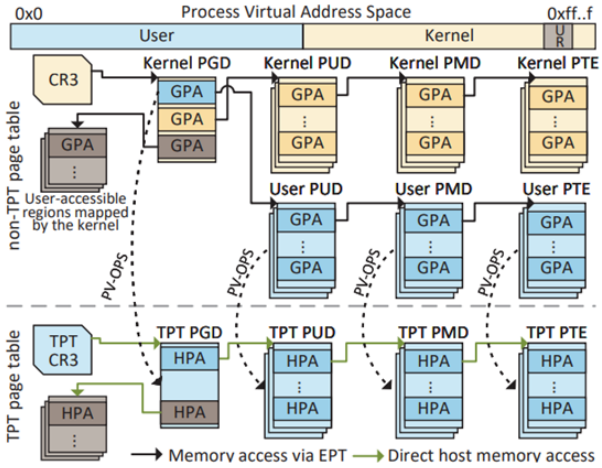

图8　TPT页表

在映射的过程中，使用TPT页表映射时，由于TPTMM交付给map进行映射，再到map更新TPT可能与TPTMM内容并不是完全同步，所以在最终进行转换时候，还需要一个越界判断，判定当前地址是否为这个虚拟机进程允许访问的，这个表直接由TPTMM进行生成。校验完成就完成了地址的转换。

可以发现TPT结构与影子页表非常相似。目前看来，我感受到他跟影子页表的区别主要在于填充部分。影子页表填充过程是，第一次由于虚拟机进程缺页进行虚拟机页表进行填充，第二次由于页表与影子页表不同步进行填充影子页表。但是TPT是直接将页表进行设置，在修改页表得到GPA直接根据MAP内容查找，获得其对应的HPA，这样子相较于影子页表两次处理，在构造影子页表的效率上有明显提升。但我并没有感受到他与影子页表有什么显著上的不同，他依旧没有解决影子页表需要针对每一个进程都需要开辟，在切换上面的消耗。只是说他在建立的时候速度有一定提升。

参 考 文 献

[1] 魏金晖. 面向多GPU系统的高效虚实地址转换机制研究[D]. 国防科技大学, 2021. DOI:10.27052/d.cnki.gzjgu.2021.000182

[2] Shai Bergman, Mark Silberstein, Takahiro Shinagawa, Peter R. Pietzuch, Lluís Vilanova:Translation Pass-Through for Near-Native Paging Performance in VMs. USENIX Annual Technical 2023: 753-768

[3] Jayneel Gandhi, Mark D Hill, and Michael M Swift. Agile Paging: Exceeding the Best of Nested and Shadow Paging. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pages 707–718. IEEE, 2016.