内存虚拟化

学号： N/A

姓名： N/A

专业： 计算机科学与技术

1 调研部分

1.1 通过页表实现主机的虚拟内存

虚拟内存的基本原理是将虚拟地址空间映射到物理地址空间。这种地址转换的过程是通过页表来实现的。页表是一种数据结构，用于存储虚拟地址和物理地址之间的映射关系。当应用程序访问虚拟地址时，CPU 会根据虚拟地址的索引部分获得虚拟页号，然后通过页表查询得到对应的物理页号，最终将物理页号和页内偏移组合成物理地址。这种设计允许进程访问逻辑上连续的虚拟地址空间，而不需要关心物理内存的实际分布。

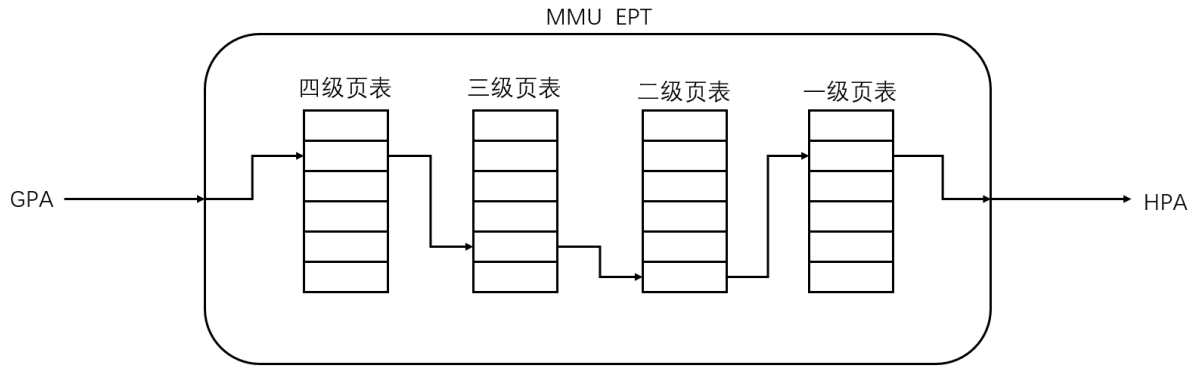
为了优化内存使用，页表的结构通常是分级的。分级页表可以很好的利用内存访问的时空局部性，不必将整个页表一次性加载到内存中，而是根据需要逐级加载。由于每一级页表只需要存储较小范围的虚拟地址映射，故而减少了整个页表的内存开销。

在地址转换过程中，CPU 依赖内存管理单元 MMU 来执行虚拟地址到物理地址的映射。每当程序访问虚拟地址时，MMU 会根据虚拟地址的各个部分逐级查询页表，最终找到对应的物理地址。硬件支持的地址转换机制不需要 CPU 的干预，因此可以在硬件层面实现高效的地址转换。为了提高查询效率，也引入了 TLB 机制，缓存最近的查询结果，利用内存访问的时空局部性减少对页表的访问次数。

1.2 EPT 地址翻译过程

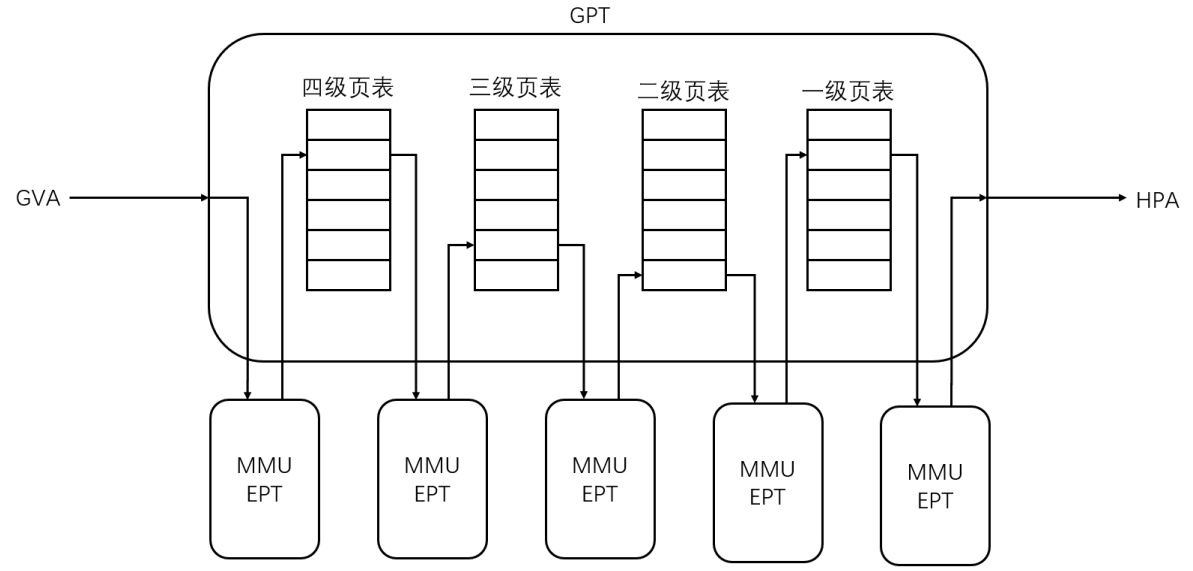
扩展页表通过硬件支持的 EPT 机制，可以利用 MMU 直接将将客户机物理地址 GPA 转换为主机物理地址 HPA，无需访问主机的页表 HPT。这样虚拟机的内存访问可以分成 GPT 译码和 EPT 译码两个阶段，提高了虚拟机的内存访问效率。

以 GPT 和 EPT 都使用四级页表为例，EPT 的地址翻译过程如下：



在支持 EPT 的 MMU 中，EPT 表的结构和 GPT 表的结构类似，都是分级的。当虚拟机访问客户机物理地址时，MMU 会根据 EPT 表将 GPA 转换为 HPA。

在使用这样的 EPT 的情况下，虚拟机的内存访问过程如下：



虚拟机每次访问虚拟内存，都需要经过 EPT 的地址转换。访问一个四级虚拟页表需要经过 4 次页表基地址的查询，还需要 1 次 GPA 到 HPA 的转换，故一共需要 5 次对 EPT 的访问。由于主机也使用 4 级页表，每次访问 EPT 需要 4 次物理内存访问，此外还有虚拟页表的 4 次物理内存访问，故一共需要 24 次物理内存访问。

1.3 QEMU 通过 KVM 接口建立 EPT

为了建立 EPT 表，QEMU 首先需要向 KVM 告知虚拟机使用的内存空间。通过 `KVM_SET_USER_MEMORY_REGION` 接口，QEMU 可以将虚拟机的

内存区域映射到主机的物理内存中。这一操作使得虚拟机的内存能够与主机的物理内存互通。接着，QEMU 还需要向 KVM 提供虚拟机的页表结构信息，以确保 KVM 能够正确处理页表的层级、页大小、访问权限等参数。

在配置 vCPU 时，QEMU 通过 `struct kvm_mmu` 结构体启用 EPT（扩展页表）功能，并通过 `ioctl` 接口传递相关的配置，如 EPT 的基地址、页表层级等信息。此外，QEMU 还需设置全局变量 `enable_ept` 为 1，以确保 KVM 在执行地址转换时使用 EPT 机制。启用 EPT 后，KVM 将自动管理虚拟机的地址转换，无需 QEMU 进一步干预。

2 实验目的

- 了解虚拟内存的基本原理
- 了解 GVA、GPA、HPA 之间的映射关系
- 了解 EPT 的地址翻译过程

3 实验步骤

3.1 实验环境

- 操作系统：Ubuntu 22.04.3 LTS
- 内核版本：5.15.0-86-generic x86_64
- 编译环境：gcc version 11.4.0
- 编辑环境：Visual Studio Code

3.2 实验过程

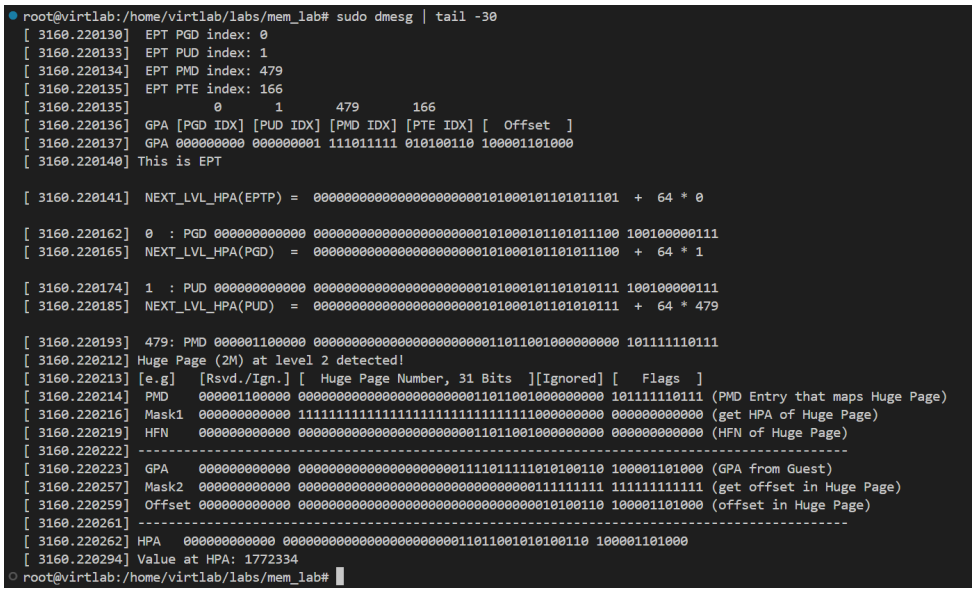
在 L1 虚拟机内启动基于 QEMU 的 L2 虚拟机：

```
Ubuntu 18.04.6 LTS ubuntu ttyS0

ubuntu login: ubuntu
Password:
Last login: Sat Oct  7 06:07:02 UTC 2023 on ttyS0
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-213-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 1.0

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

30 updates can be applied immediately.
1 of these updates is a standard security update.
To see these additional updates run: apt list --upgradable


ubuntu@ubuntu:~$
```

在 L2 虚拟机内编译并加载 gpt-dump 内核模块。内核模块中动态分配了一个值为 1772333 的 int，并打印出相应的 GPT 译码信息：

```
ubuntu@ubuntu:~/Book/Chapter3$ cat gpt-dump.txt
gpt_dump: loading out-of-tree module taints kernel.
gpt_dump: module verification failed: signature and/or required key missing - tainting kernel
Value at GVA: 1772334
 GPT PGD index: 287
 GPT PUD index: 478
 GPT PMD index: 479
 GPT PTE index: 166
          287       478       479       166
 GVA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]
 GVA 100011111 111011110 111011111 010100110 100001101000

 NEXT_LVL_GPA(CR3)  =  0000000000000000000001111001111000011110  +  64 * 287

 287: PGD 000000000000 0000000000000000000001001100011110011111 000001100111
 NEXT_LVL_GPA(PGD)  =  0000000000000000000001001100011110011111  +  64 * 478

 478: PUD 000000000000 0000000000000000000001001100011110100011 000001100111
 NEXT_LVL_GPA(PUD)  =  0000000000000000000001001100011110100011  +  64 * 479

 479: PMD 000000000000 0000000000000000000001111011111010111000 000001100011
 NEXT_LVL_GPA(PMD)  =  0000000000000000000001111011111010111000  +  64 * 166

 166: PTE 100000000000 0000000000000000000001111011111010100110 000001100011
 GPA           =       0000000000000000000001111011111010100110 100001101000
ubuntu@ubuntu:~/Book/Chapter3$
```

退出到 L1 虚拟机，L1 虚拟机已修改了 kvm 中的 hypercall 22，L2 虚拟机通过 hypercall 通知 L1 虚拟机打印相应的 EPT 译码信息：



3.3 实验结果

gpt-dump 中的 GPT 译码信息：

```
gpt_dump: loading out-of-tree module taints kernel.
gpt_dump: module verification failed: signature and/or required key missing - tainting kernel
Value at GVA: 1772334
 GPT PGD index: 287
 GPT PUD index: 478
 GPT PMD index: 479
 GPT PTE index: 166
          287       478       479       166
 GVA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]
 GVA 100011111 111011110 111011111 010100110 100001101000

 NEXT_LVL_GPA(CR3)  =  0000000000000000000001111001111000011110  +  64 * 287

 287: PGD 000000000000 000000000000000000000100110001111001111 000001100111
 NEXT_LVL_GPA(PGD)  =  0000000000000000000001001100011110011111  +  64 * 478

 478: PUD 000000000000 000000000000000000000100110001111010011 000001100111
 NEXT_LVL_GPA(PUD)  =  0000000000000000000001001100011110100011  +  64 * 479

 479: PMD 000000000000 000000000000000000000111101111101011000 000001100011
 NEXT_LVL_GPA(PMD)  =  0000000000000000000001111011111010111000  +  64 * 166

 166: PTE 100000000000 000000000000000000000111101111101010110 000001100011
 GPA            =      000000000000000000000111101111101010110 100001101000
```

dmesg 中的 EPT 译码信息：

```
[ 3160.220130]  EPT PGD index: 0
[ 3160.220133]  EPT PUD index: 1
[ 3160.220134]  EPT PMD index: 479
[ 3160.220135]  EPT PTE index: 166
[ 3160.220135]           0       1       479       166
[ 3160.220136]  GPA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]
[ 3160.220137]  GPA 000000000 000000001 111011111 010100110 100001101000
[ 3160.220140] This is EPT
```

```
[ 3160.220141]  NEXT_LVL_HPA(EPTP) =  0000000000000000000000010100010110101110 1  + 64 * 0

[ 3160.220162]  0  : PGD 000000000000 000000000000000000000000010100010110101110 0 100100000111
[ 3160.220165]  NEXT_LVL_HPA(PGD)  =  0000000000000000000000010100010110101110 0  + 64 * 1

[ 3160.220174]  1  : PUD 000000000000 000000000000000000000000010100010110101011 1 100100000111
[ 3160.220185]  NEXT_LVL_HPA(PUD)  =  0000000000000000000000010100010110101011 1  + 64 * 479

[ 3160.220193]  479: PMD 000001100000 000000000000000000000000011011001000000000 1 011111110111
[ 3160.220212] Huge Page (2M) at level 2 detected!
[ 3160.220213] [e.g]   [Rsvd./Ign.] [  Huge Page Number, 31 Bits  ][Ignored] [   Flags  ]
[ 3160.220214]  PMD    000001100000 000000000000000000000000011011001000000000 1 011111110111 (PMD
Entry that maps Huge Page)
[ 3160.220216]  Mask1  000000000000 111111111111111111111111111111000000000 0 000000000000 (get
HPA of Huge Page)
[ 3160.220219]  HFN    000000000000 000000000000000000000000011011001000000000 0 000000000000 (HFN
of Huge Page)
[ 3160.220222]
-----------------------------------------------------------------------------------------------
[ 3160.220223]  GPA    000000000000 000000000000000000000011110111111010100110 1 000001101000 (GPA
from Guest)
[ 3160.220257]  Mask2  000000000000 000000000000000000000000000000111111111 1 111111111111 (get
offset in Huge Page)
[ 3160.220259]  Offset 000000000000 000000000000000000000000000000010100110 1 000001101000
(offset in Huge Page)
[ 3160.220261]
-----------------------------------------------------------------------------------------------
[ 3160.220262] HPA   000000000000 000000000000000000000000011011001010100110 1 000001101000
[ 3160.220294] Value at HPA: 1772334
```

4 实验分析

通过 gpt-dump 和 dmesg 的输出信息，可以看到一个 GVA 地址先经过 GPT 译码转换为 GPA 地址，再经过 EPT 译码转换为 HPA 地址。GPA 和 HPA 地址的转换过程中，都各经过了 4 级页表的查询。

在 GPT 译码过程中，先从 GVA 地址提取出 4 级页表的索引 PGD、PUD、PMD、PTE，然后根据这些索引逐级查询页表，最终得到 GPA 地址。首先通过 CR3 寄存器找到 PGD 页表的基地址，加上 PGD 索引的 64 倍（页表项大小 64B）找到这个索引的页表项，它保存了下一级 PUD 页表的基地址。以此类推，最终得到 PTE 页表项，读取页表项中的物理页地址，加上页内偏移，得到 GPA 地址。页表项中除了保存下一级页表的基地址或者物理页号外，还保存了一些控制位，如是否有效、可读可写、访问权限、缓存策略等。

在 EPT 译码过程中，与 GPT 译码类似，也是 4 级页表的查询过程，从 GPA 提取索引依次计算 PGD、PUD、PMD、PTE 的地址，最终得到 HPA 地址。与 GPT 译码不同的是，EPT 译码的页表基地址是通过 EPTP 提供的，而不是 CR3 寄存器。在 EPT 译码过程中，还可能遇到

大页表项，此处的 PMD 页表项是一个大页表项，直接映射了一个 2M 的物理页而不是下一级页表的基地址。在这种情况下，只需要将 GPA 的低 21 位作为偏移，加上大页表项的地址，即可得到 HPA 地址，无需进一步查询下一级页表。因此这里只经过了 3 次页表查询，而不是普通页情况下的 4 次。

最终，我们读取到了 HPA 地址对应的值，即 1772334，与虚拟机中 GVA 地址对应的值一致，说明 GPT 和 EPT 的译码过程是正确的。

附录

参考文献

- 教材 深入浅出系统虚拟化：原理与实践
- KVM API Documentation：https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt