Virtual Memory: Concepts

韩汶辰

2019 年 11 月 28 日

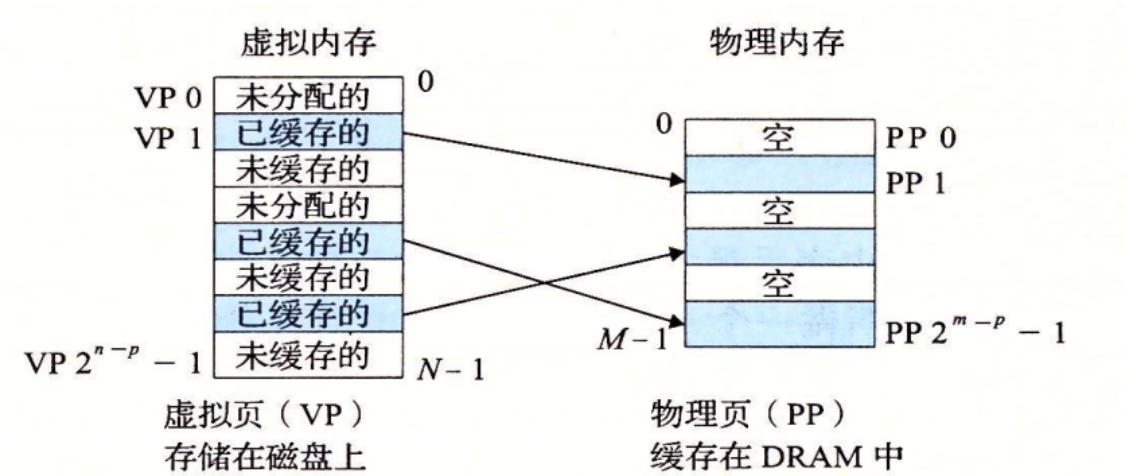
Contents

Basic Concepts

- 为什么使用虚拟内存? 多个进程同时运行时, 为每个进程提供一个假象, 仿佛他们独立地运行在系统中. 否则程序员直接同物理内存打交道很麻烦, 不同进程间容易产生冲突.
  - 某些嵌入式控制器, 数字信号处理器等仍然使用物理寻址方式.
- 由 os 和 mmu(内存管理单元) 共同完成.
- 虚拟地址空间 (2^n) 和物理地址空间.
- 通过地址翻译将虚拟地址空间转化为物理地址空间.

Contents

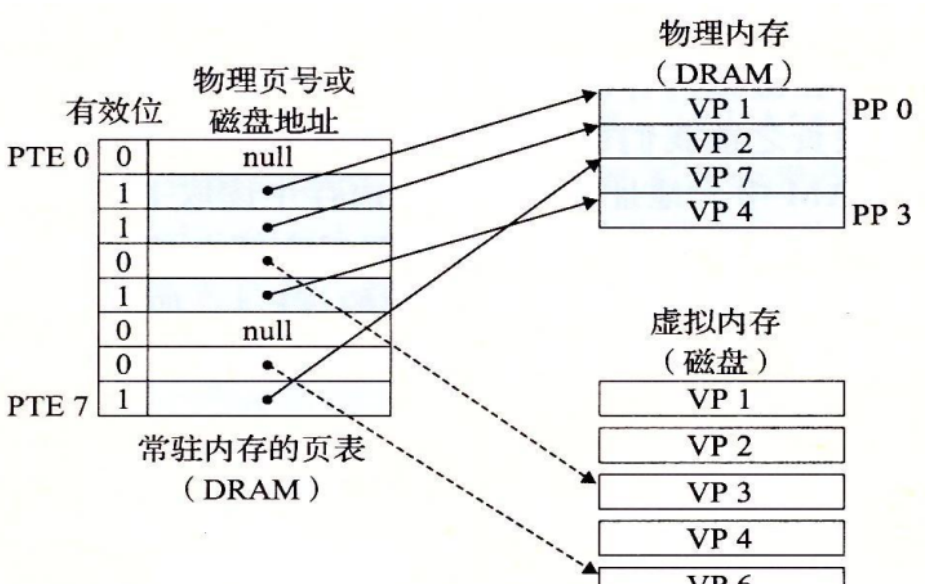
VM 可以看成 RAM 的缓存



- DRAM 可以看成为虚拟内存的全相连的缓存，如何映射由操作系统决定.
- 虚拟页 (VP) 和物理页 (PP)(二者大小相等)
- 对于虚拟内存，绝大多数是**未分配的**，剩下的是**已缓存的**：被映射到物理内存上，**未缓存的**：被映射在磁盘上.
  - 我对”未缓存”的理解：它**一定**被映射到了磁盘上，不过它可能是匿名文件 (没有实际东西，缺页时不会从磁盘到内存传输)，也可能是脏页 (有干货).

页表的结构

- 一个页表是页表条目 (PTE) 的数组，每个 PTE 会维护一个有效位 (0/1)，如果有效位为 1 的话还会维护虚拟页号 (VPN) 对应的物理页号 (PPN)；有效位为 0 可能是未分配页或者映射到交换页.
- 除此之外还会记录许可位等其它内容 (详见 9.7 节).
- 本质上相当于虚拟页到物理页和交换页的映射.
- 每个进程都有一个页表.
- 分工上，MMU 进行地址翻译，操作系统负责管理页表.
- 存储位置上，页表通常存在 RAM 中.



页命中和页不命中

- 页命中的情形比较简单，如果 PTE 为 1，则由 MMU 将 PTE 中的虚拟地址翻译成物理地址并访问 DRAM.
- 页不命中时会触发一个**缺页异常**这个”**故障**”，控制流转向内核缺页异常处理程序，该程序会从磁盘中”读取”所缺页，并选择一个牺牲页来替换.
- 这个牺牲页如果是”脏”的，还要将物理内存的牺牲页替换到磁盘.
  - Q: 如果是干净的，会将牺牲页”映射”到磁盘?
- 同时内核会更新页表.
- 接下来，控制流会转向导致缺页的命令**重新执行**.

一些注释

- 按需页面调度: 大多数操作系统会倾向于比较懒惰, 它会在直到一个缺失的页被访问时才会将交换页换回到磁盘中.
- 注意缺页不代表这个页在磁盘上, 还可能是不合法的地址 (段错误) 以及违反了权限 (见 9.7).
- 如何分配新的虚拟页: 例如 malloc. 将新的虚拟页映射到磁盘上并更新页表. 这些磁盘上的页直到被访问才会放入物理内存.

Contents

简化共享与内存保护

- 不同的进程之间多数数据是相互独立的; 某些数据可能对于某些进程共享; 一个程序不允许修改只读代码段等.
- PTE 中存在许可位, 分别代表: 是否有 SUP 权限, 是否有读和写权限.
- 共享机制的实现: 不同进程间共享的数据指向相同的物理页面并设定对应的许可位.
- 内存保护机制: 如果违反了许可位, 就会触发”段错误”这个故障.

简化链接和加载

- 每个进程的内存映像有着相同的形式 (e.g, 代码段从 0x400000 开始), 这样给链接工作带来方便.
- 在加载时, 无需将代码和数据段从磁盘搬到内存, 只需为其分配虚拟页, 映射到磁盘上的对应文件, 并标记为" 未缓存的" 即可. 运行时, 被访问的指令和数据会自动被搬到物理内存.
- 注: 感觉书上" 简化内存分配" 那里不太对, 网上说 malloc 时应该不会分配物理内存, 而是被映射到匿名文件, 被访问时才会分配物理也.

Contents

地址翻译的方法

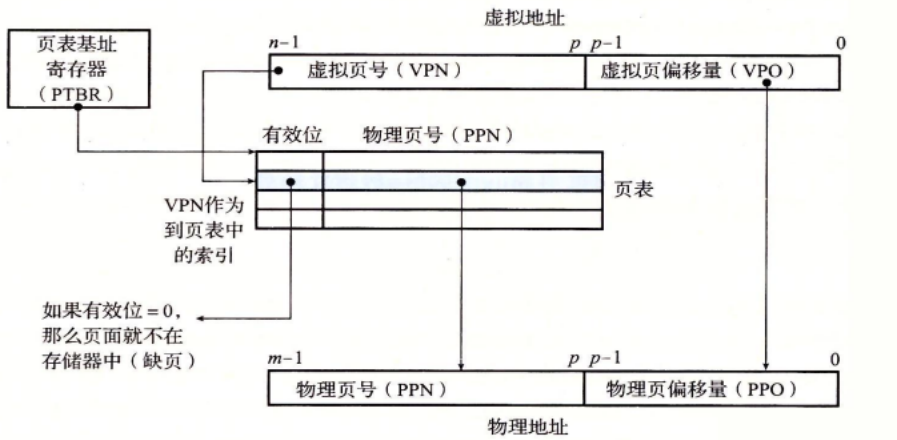


图 9-12 使用页表的地址翻译

- 虚拟页号 (VPN)(+PTBR) -> 查表得到物理页号 (PPN), 虚拟页偏移量 (VPO) 落下来得到物理页偏移量 (PPO), 合并后就得到了物理地址.
- 许多符号: VPN, PPN, VPO, PPO, CO(offset), CI(index), CT(tag).

recall: 页命中与页不命中的流程

- 第 1 步：处理器生成一个虚拟地址，并把它传送给 MMU。
- 第 2 步：MMU 生成 PTE 地址，并从高速缓存/主存请求得到它。
- 第 3 步：高速缓存/主存向 MMU 返回 PTE。
- 第 4 步：MMU 构造物理地址，并把它传送给高速缓存/主存。
- 第 5 步：高速缓存/主存返回所请求的数据字给处理器。

- 第 1 步到第 3 步：和图 9-13a 中的第 1 步到第 3 步相同。
- 第 4 步：PTE 中的有效位是零，所以 MMU 触发了一次异常，传递 CPU 中的控制到操作系统内核中的缺页异常处理程序。
- 第 5 步：缺页处理程序确定出物理内存中的牺牲页，如果这个页面已经被修改了，则把它换出到磁盘。
- 第 6 步：缺页处理程序页面调入新的页面，并更新内存中的 PTE。

- 第 7 步：缺页处理程序返回到原来的进程，再次执行导致缺页的指令。CPU 将引起缺页的虚拟地址重新发送给 MMU。因为虚拟页面现在缓存在物理内存中，所以就会命中，在 MMU 执行了图 9-13b 中的步骤之后，主存就会将所请求字返回给处理器。

加上高速缓存呢?

- 其实变化不大, 因为大多数系统是选择先进行地址翻译与权限检查, 再利用物理地址在 RAM 里面取数据的.
- 所以这里的流程是, 先进行地址翻译和权限检查等, 如果页命中, 则之后的操作等价于第六章在带高速缓存的存储系统中访问数据了, (逐级检查高速缓存, 如果 miss 则访问主存).
- 页表条目也可以缓存, 如同其他数据一样.

TLB 加速地址翻译

- 为什么？即使页表存储在 L1 cache 中也有延迟，并且还可能会被经常替换回更低层的 cache.
- TLB 与普通 cache 很像，也有 valid bit，标记 (TLBT) 和索引 (TLBI)，可以相联，存储的对象是 PTE，其结构大致如下：

TLB			
Index	Tag	PPN	Valid
0	0x13	0x30	1
	0x34	0x58	0
1	0x1F	0x80	0
	0x2A	0x72	1
2	0x1F	0x95	1
	0x20	0xAA	0
3	0x3F	0x20	1
	0x3E	0xFF	0

TLB hit, TLB miss

- TLB 命中时, 便不再从 cache/DRAM 中寻找 PTE 了, MMU 直接将其翻译成物理地址, 之后取出物理地址的数据.
- TLB 不命中时, 则需要 1. 先在 cache/DRAM 中找到 PTE 这个表项, 2. 之后放在 TLB 中的某个条目, 3. MMU 翻译成物理地址访问 cache/DRAM 的数据.
- 注: TLB 命中时, 要先看 valid bit 是否是 1, 可能会出现 valid bit 是 0 并且存储的 PPN 不是 0, 以及 TLBT 也匹配了, 这也算不命中 (也许是别的进程对应的 TLB, 并且刚刚进行 context switching, recall: 应对 context switching 可以通过 flush TLB 或者在 TLB 项中标记对应的 pid).

思考

- 我们知道通过页表访问一条虚拟地址可能会 hit, 也可能会发生 page fault. 假设一条虚拟地址该进程有权限访问, 不会发生段错误. 在 TLB 中查询时发现了 TLB hit. 那么接下来会不会发生到硬盘的 page fault 呢?

思考

- 我们知道通过页表访问一条虚拟地址可能会 hit, 也可能会发生 page fault. 假设一条虚拟地址该进程有权限访问, 不会发生段错误. 在 TLB 中查询时发现了 TLB hit. 那么接下来会不会发生到硬盘的 page fault 呢?
- 应该不会.
- 看一下书上 9.6 的例子和课后习题, 应该没有既 TLB hit 又 page fault 的.

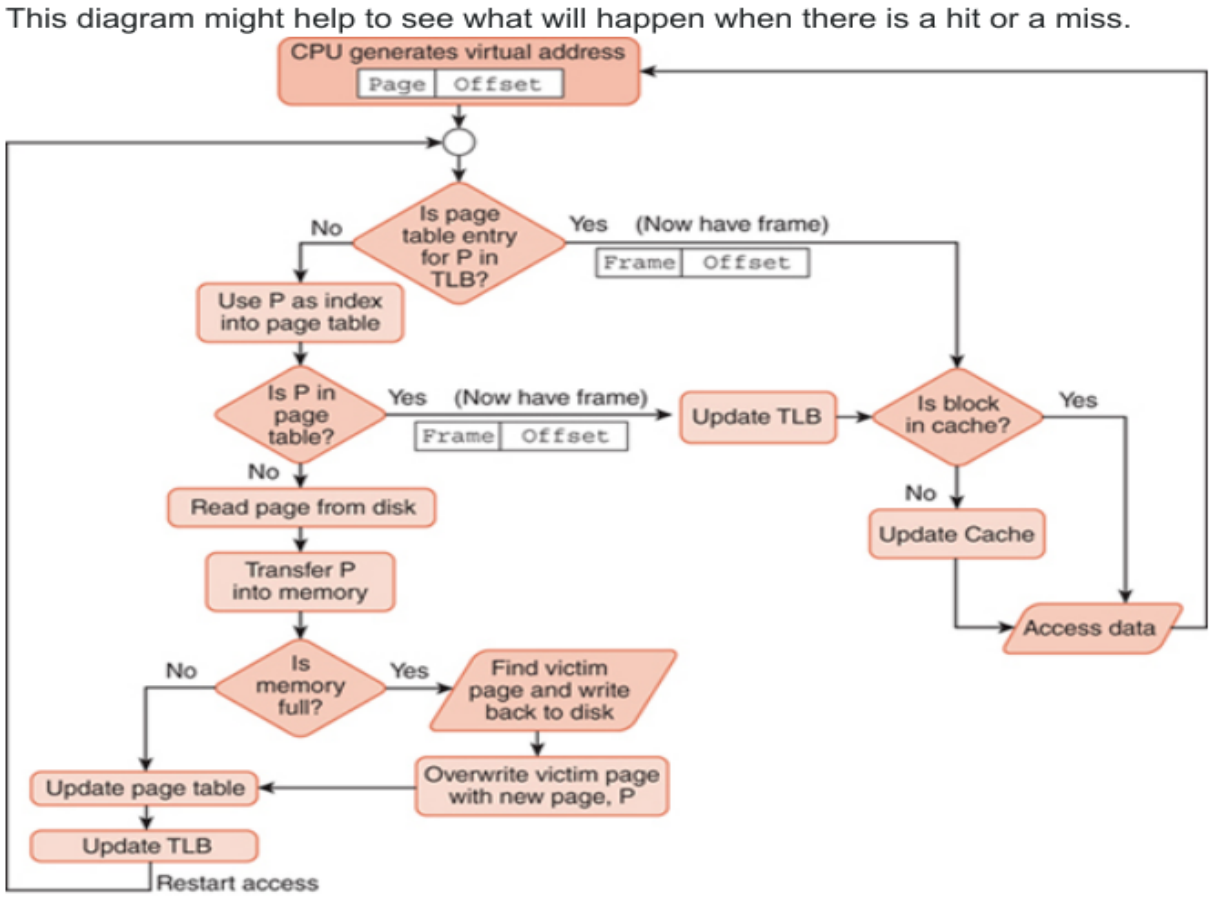
思考

- 我们知道通过页表访问一条虚拟地址可能会 hit, 也可能会发生 page fault. 假设一条虚拟地址该进程有权限访问, 不会发生段错误. 在 TLB 中查询时发现了 TLB hit. 那么接下来会不会发生到硬盘的 page fault 呢?
- 应该不会.
- 看一下书上 9.6 的例子和课后习题, 应该没有既 TLB hit 又 page fault 的.

One final point: last lecture we said that pages can be either in RAM or on disk, and we saw that there are two lengths of physical addresses – one for RAM and one for disk. The TLB does not store translations for pages on disk, however. If the program is trying to address a word that only lies on the hard disk, then (by design) it is impossible that the TLB will have the translation for that address; the TLB only holds translations for addresses that are in RAM. In order to have access to that word, a page fault will need to occur and the page will need to be copied from the hard disk to main memory. We will discuss how this is done in the next lecture.

思考

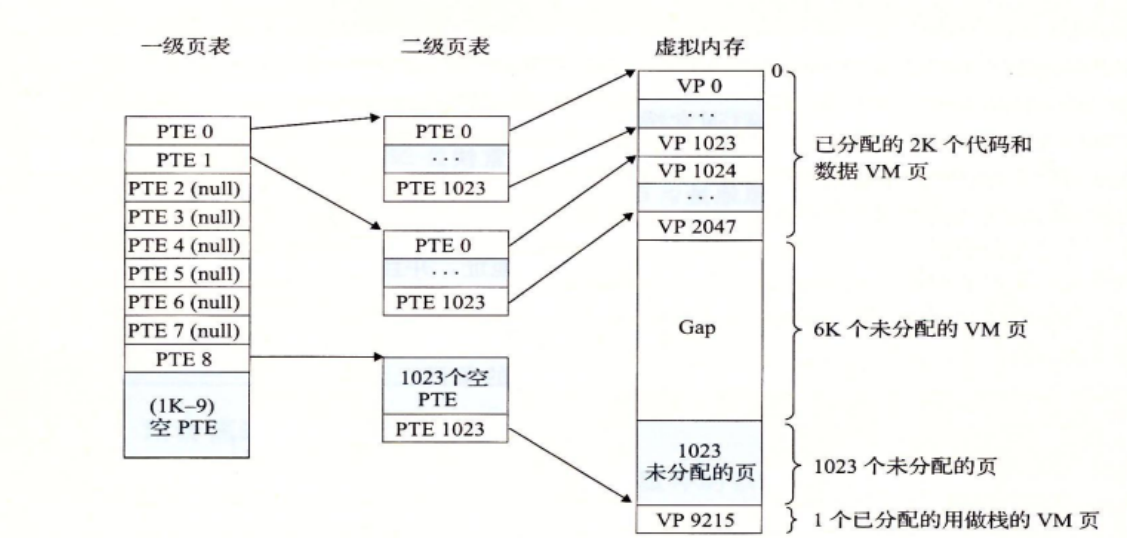
- 这张图总结了各种 miss 和 hit 问题.

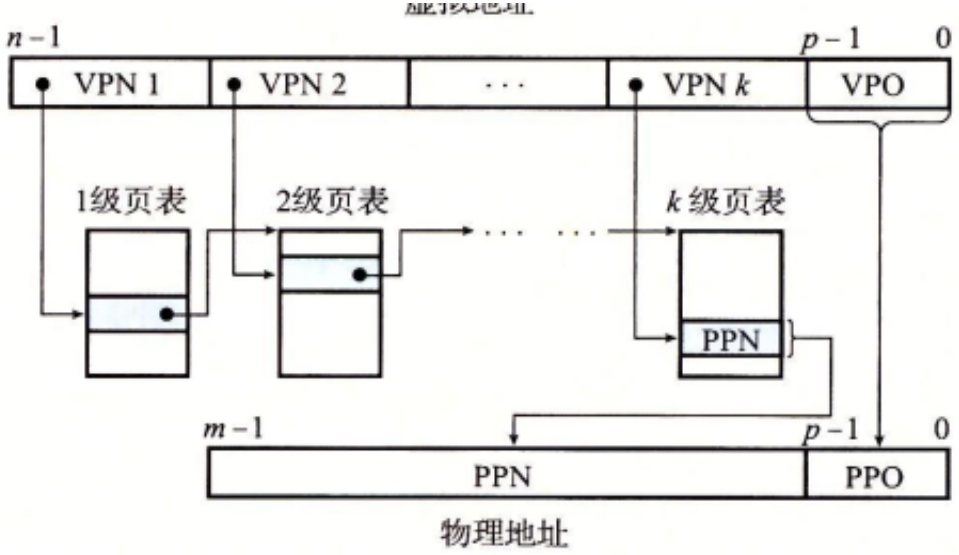


多级页表

- 原因：即使是一个 32 位的系统，单级页表也太大了，有 4MB，每个进程还分别有自己的页表．64 位系统就更麻烦了．然而，大部分虚拟内存都是未被分配的，且虚拟内存的分配一般有良好的局部性．
- 解决方案：多级页表
- 如同普通的 PTE，上一级页表指向下一级页表的物理地址，或者是未分配的．
  - 实际上，绝大多数一级页表项都是空的，从而对应的二级页表的空间被节省下来了．
- 此外只有一级页表需要存储在主存中，其它的页表可以搬到内存中．

多级页表


一级页表
二级页表
虚拟内存
PTE 0
PTE 1
PTE 2 (null)
PTE 3 (null)
PTE 4 (null)
PTE 5 (null)
PTE 6 (null)
PTE 7 (null)
PTE 8
(1K–9) 空 PTE
PTE 0
PTE 1023
PTE 0
PTE 1023
1023个空 PTE
PTE 1023
VP 0
VP 1023
VP 1024
VP 2047
Gap
1023 未分配的页
VP 9215
0
已分配的 2K 个代码和数据 VM 页
6K 个未分配的 VM 页
1023 个未分配的页
1 个已分配的用做栈的 VM 页


n – 1
p – 1
0
VPN 1
VPN 2
VPN k
VPO
1级页表
2级页表
k 级页表
PPN
m – 1
p – 1
0
PPN
PPO
物理地址