虚拟内存

虚拟内存是计算机系统内存管理的一种技术。目前，大多数操作系统都使用了虚拟内存，如Windows家族的“虚拟内存”；Linux的“交换空间”等

虚拟内存可以被看作是存放在磁盘上的N个连续字节大小的单元组成的数组，每个字节都有一个唯一的虚拟地址作为数组的索引

为什么要有虚拟内存

在早期的计算机中，是没有虚拟内存的概念的。我们要运行一个程序，会把程序全部装入内存，然后运行。当运行多个程序时，经常会出现以下问题：

1）进程地址空间不隔离，没有权限保护

由于程序都是直接访问物理内存，所以一个进程可以修改其他进程的内存数据，甚至修改内核地址空间中的数据。

2）内存使用效率低

当内存空间不足时，要将其他程序（数据）暂时拷贝到硬盘，然后将新的程序（数据）装入内存运行

3）程序运行的地址不确定

因为内存地址是随机分配的，所以程序运行的地址也是不确定的

虚拟内存的优点

虚拟内存是单机系统最重要的几个底层原理之一，它由底层硬件和操作系统两者软硬件结合来实现，是硬件异常，硬件地址翻译，主存，磁盘文件和内核的完美

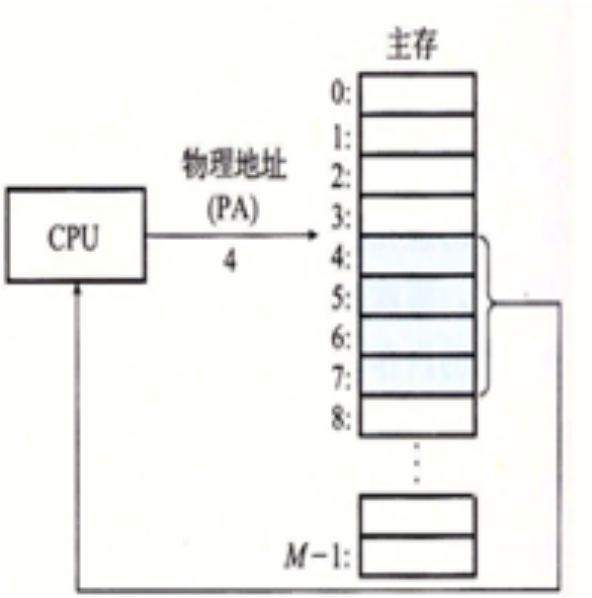
交互。它主要提供了3个能力：

1）给所有进程提供一致的地址空间，每个进程都认为自己是在独占使用单机系统的存储资源

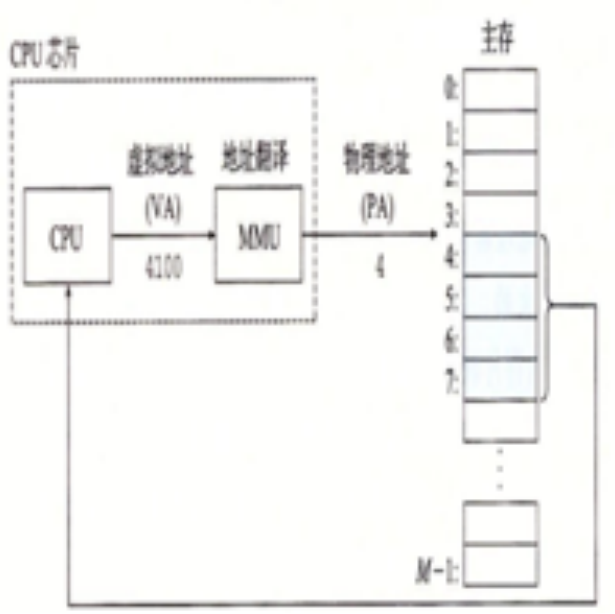
2）保护每个进程的地址空间不被其他进程破坏，隔离了进程的地址访问

3）根据缓存原理，上层存储是下层存储的缓存，虚拟内存把主存作为磁盘的高速缓存，在主存和磁盘之间根据需要来回传送数据，高效地使用了主存

虚拟地址和物理地址



物理寻址的系统



虚拟寻址的系统

对于每个进程来说，它使用到的都是虚拟地址，每个进程都看到一样的虚拟地址空间

对于32位计算机系统来说，它的虚拟地址空间是 0 - 2^32，也就是0 - 4G。对于64位的计算机系统来说，理论的虚拟地址空间是 0 - 2^64，远高于目前常见的物理内存空间。

虚拟地址空间不需要和物理地址空间一样大小

虚拟内存作为缓存工具

虚拟内存技术将虚拟内存分割为了大小固定的块，这些块被称为虚拟页(Virtual Page, VP)。类似地，物理内存也被分割为了大小相同的物理页(Physical Page, PP)，物理页有时候也被称为页帧(page frame)

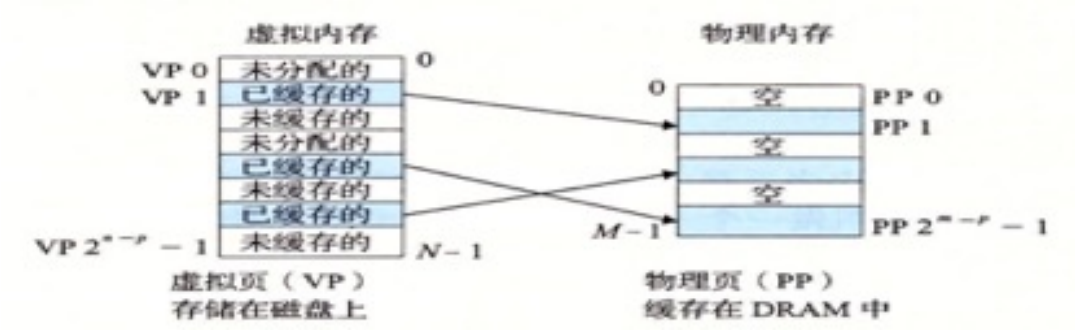
虚拟内存也就是虚拟页的集合可以被分为以下三类：

1) 未映射的虚拟页：这种虚拟页是属于虚拟内存系统还没有创建的页，因此

它不会占据任何磁盘上的空间

2) 缓存的虚拟页：这种虚拟页已经被虚拟内存系统创建(映射到磁盘上)，同时也被缓存到了物理内存上

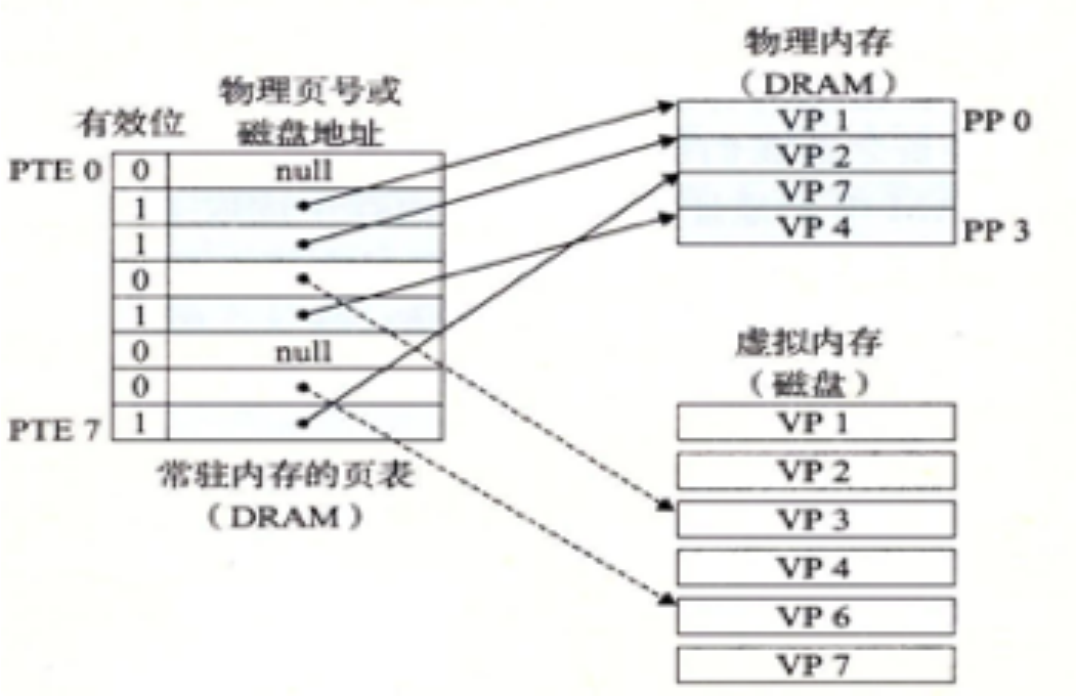
3) 未缓存的虚拟页：这种虚拟页已经被虚拟内存系统创建(映射到磁盘上)，但没被缓存到了物理内存上



图：虚拟内存是如何使用主存作为缓存的

页表

页表实际就是一个包含有多个页表条目(Page Table Entry, PTE)的数组，存放着各虚拟页的状态，是否映射，是否缓存

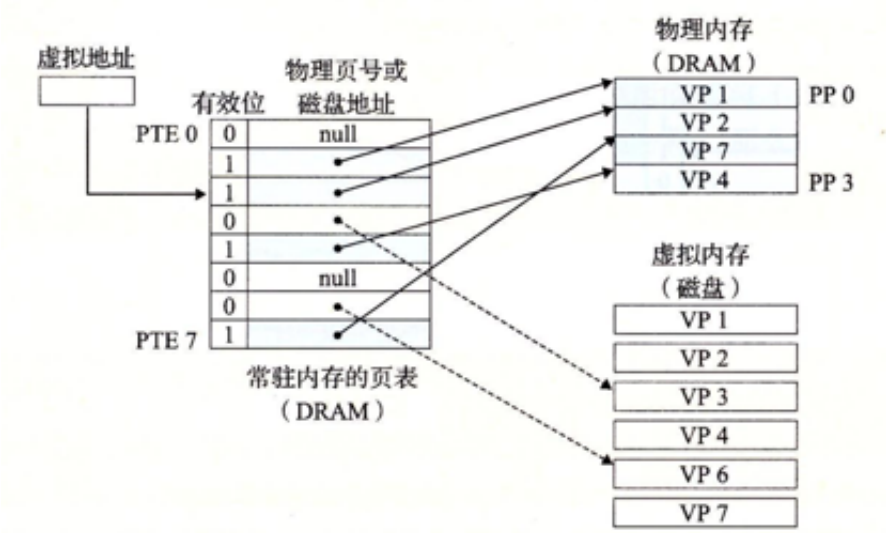


图：页表结构

页命中

如果CPU要访问的虚拟页已经被缓存到物理内存上的物理页，那么CPU就可以直接从物理内存中读取需要的数据，我们将这种情形称为页命中

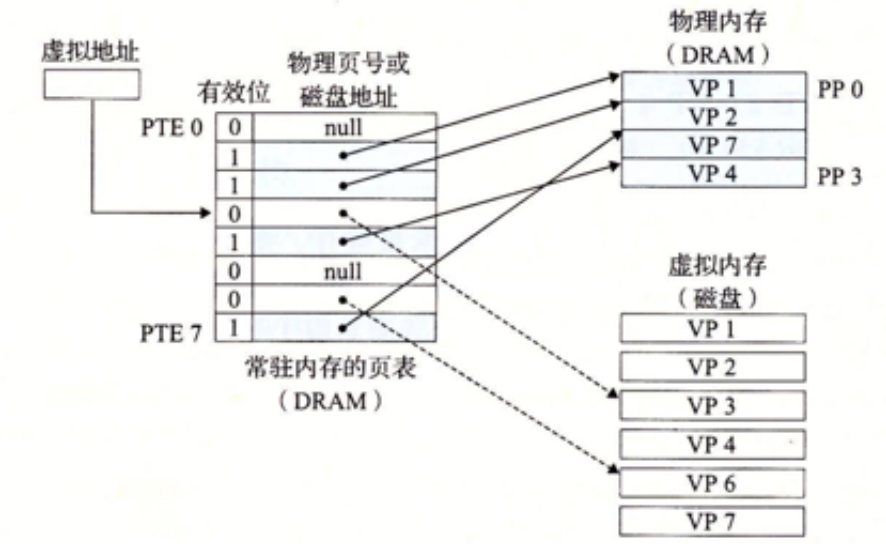
下图展示了CPU想要读取虚拟页VP2中的数据时，发生了什么：
地址翻译硬件将虚拟地址作为索引来定位PTE2，并从内存中读取它。由于PTE2中的有效位为1，因此地址翻译硬件就知道VP2已经被缓存到了物理内存上了。因此它可以直接使用PTE2中地址字段中的地址，构建出一个需要访问的数据的物理地址



缺页

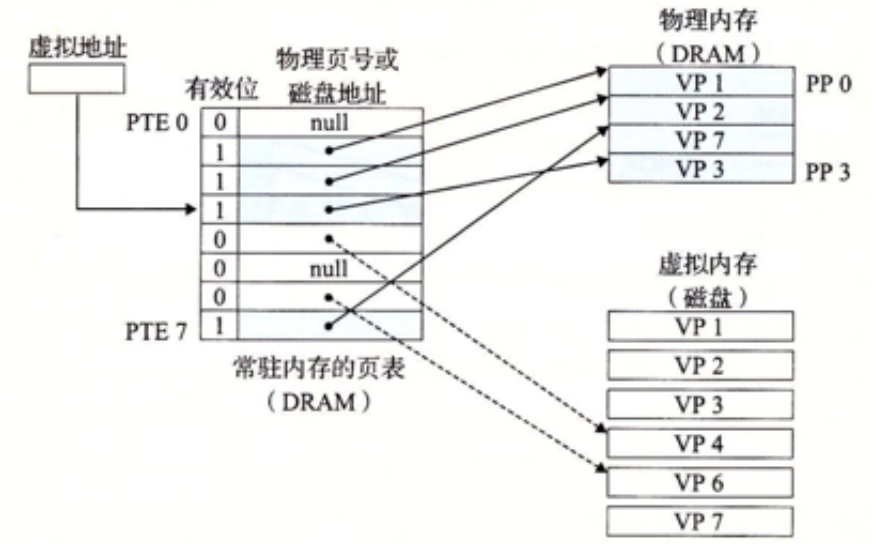
如果CPU要访问的虚拟页在物理内存上未命中，我们称这种情形为缺页(page fault)

下图展示了CPU想要访问VP3中的数据，触发了缺页异常：



CPU需要访问VP3中的数据，因此地址翻译硬件在内存中读取PTE3，从有效位为0判断出VP3并没有被缓存到物理内存中，此时触发一个缺页异常

下图展示了面对缺页异常，内核采取的措施：

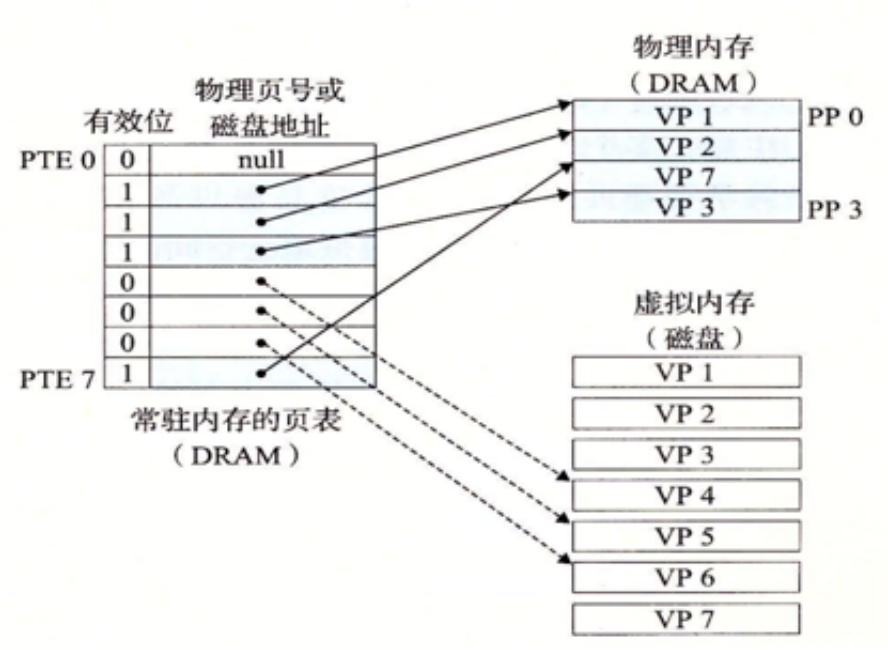


缺页异常的触发会导致内核中缺页异常处理程序的调用，首先该程序会在虚拟页中选择一个牺牲页，在这个例子中选择的是原本被缓存在物理页PP3上的虚拟页VP4。如果之前VP4的数据在物理内存中被修改了，内核会将被修改后的VP4内容复制回磁盘上。之后内核会修改页表条目PTE4有效位为0，地址字段指向VP4在磁盘中的位置。

紧接着内核会从磁盘上复制VP3到物理内存中的PP3，并更新PTE3有效位为1，地址字段指向PP3的位置，此时缺页异常处理程序完成了它的功能，开始返回。返回时它会重新启动之前导致缺页异常的那条指令，该指令会把导致缺页异常的虚拟地址再次发送给地址翻译硬件。此时，VP3已经缓存在物理内存的PP3上了，因此该执行页命中后正常逻辑了

分配虚拟页

之前例子中页表的PTE0和PTE5都处于未分配状态，下图展示虚拟内存系统分配一个新的虚拟页时对页表的操作：

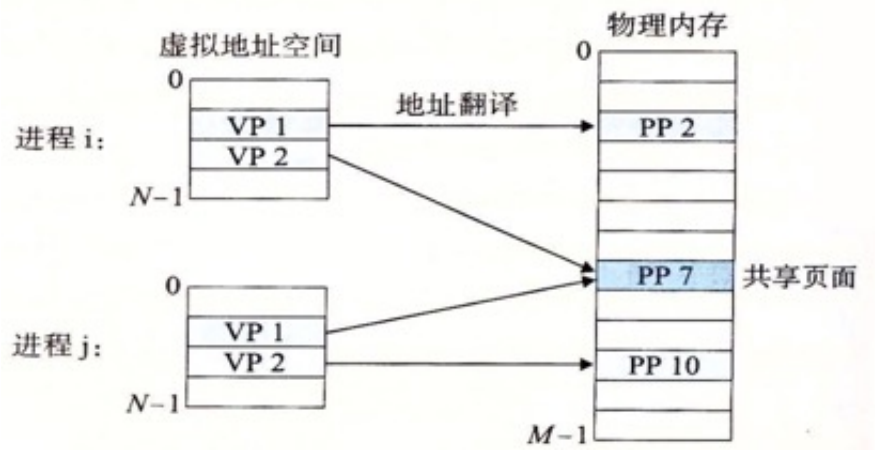


本质上来说，分配新的虚拟页VP5的过程，就是在磁盘上创建空间，并更新页表PTE5，使它指向磁盘上新建空间的位置

虚拟内存作为内存管理的工具

虚拟内存同样为操作系统提供了一种自然的管理内存的机制

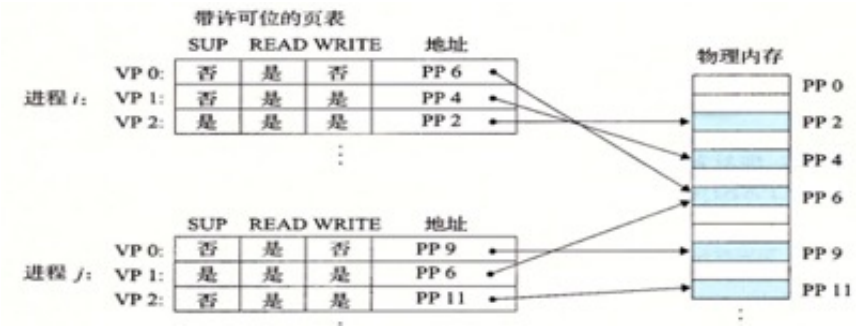
在虚拟内存作为缓存的工具中，都在假设在内存中有一个单独的页表，用来将虚拟内存的虚拟地址空间映射到物理内存的物理内存上。事实上，如下图所示，操作系统为每一个进程都提供了一个独立的页表，因而每个进程都拥有一个一致的却独立的虚拟地址空间了



如上图所示，进程i的页表将VP1和VP2分别映射到物理内存的PP2和PP7上去；而进程j的页表将VP1和VP2分别映射到物理内存的PP7和PP10上。
因此，当我们在两个进程中使用相同的虚拟地址去定位VP1时，实际上访问到的是不同物理内存上的数据，从而保证了每个进程拥有一致的并且独立的虚拟内存空间。为了方便进程间共享某些内存上的数据，多个虚拟页面是可以被映射到相同物理页上的，如上图中的物理页面PP7

虚拟内存作为内存保护的工具

所有的操作系统都需要提供一种机制用来控制对内存的访问。比如，不应该允许一个用户进程修改它的只读代码段；也不应该允许它读取或是修改任何内核中的代码和数据结构；不应该允许它读取或者写其它进程的私有内存等等。
正如我们上一节所提到的，独立的地址空间使得区分不同进程的私有内存变得相当容易。但是要想严密地控制对内存的访问，光靠地址空间还是不够的。由于地址翻译硬件在翻译地址时都会读取页表中的PTE，因此一种简单的方法就是在PTE上添加一些额外的许可位来控制对虚拟页的访问。下图展示了大致的思想



在这个实例中，每个PTE中额外添加了三个控制许可位。SUP位表示进程是否必须运行在超级用户模式下才能访问该页；READ位和WRITE位控制对页面的读和写。比如进程i运行在用户模式下，那么它有对页面VP0的读权限，VP1的读写权限，但是没有访问VP2的权限