Linux的内存管理机制

内存映射

- 将虚拟内存地址映射到物理内存地址
- 内核为每个进程都维护了一张页表, 记录虚拟地址与物理地址的映射关系
- 页表存储在CPU的内存管理单元中(MMU)
  - 当程序在CPU上执行时, CPU可以直接通过MMU查询到虚拟内存对应的物理内存
  - 如果CPU发现MMU上没有对应的映射关系, 就会产生一个缺页异常, 进入内核空间分配物理内存, 更新进程页表, 最后再返回用户空间, 回复进程的运行

内存分页与分段

- 早期程序是直接运行在物理内存上的，产生了使用效率低, 多进程间相互影响等问题, 于是发展出了内存分段
- 内存分段是根据程序的逻辑角度，分成了栈段、堆段、数据段、代码段等，这样可以分离出不同属性的段，同时是一块连续的空间。但是每个段的大小都不是统一的，这就会导致内存碎片和内存交换效率低的问题。
- 为了解决内存分段的内存碎片和内存交换效率低的问题，就出现了内存分页。
- 分页是把整个虚拟和物理内存空间切成一段段固定尺寸的大小, 这样一个连续并且尺寸固定的内存空间，我们叫页（Page）
- 分页是怎么解决分段的内存碎片、内存交换效率低的问题?
  - 采用了分页，那么释放的内存都是以页为单位释放的，也就不会产生无法给进程使用的小内存。
  - 如果内存空间不够，操作系统会把其他正在运行的进程中的「最近没被使用」的内存页面给释放掉，也就是暂时写在硬盘上，称为换出（Swap Out）。一旦需要的时候，再加载进来，称为换入（Swap In）。所以，一次性写入磁盘的也只有少数的一个页或者几个页，不会花太多时间，内存交换的效率就相对比较高。
  - 更进一步地，分页的方式使得我们在加载程序的时候，不再需要一次性都把程序加载到物理内存中。我们完全可以在进行虚拟内存和物理内存的页之间的映射之后，并不真的把页加载到物理内存里，而是只有在程序运行中，需要用到对应虚拟内存页里面的指令和数据时，再加载到物理内存里面去。
- 内存分段和内存分页并不是对立的，它们是可以组合起来在同一个系统中使用的，那么组合起来后，通常称为段页式内存管理。
  - 先将程序划分为多个有逻辑意义的段，也就是前面提到的分段机制；
  - 接着再把每个段划分为多个页，也就是对分段划分出来的连续空间，再划分固定大小的页；

多级页表

- 简单的分页有什么缺陷吗? — 因为操作系统是可以同时运行非常多的进程的，那这不就意味着页表会非常的庞大。
- TLB
  - 多级页表虽然解决了空间上的问题，但是虚拟地址到物理地址的转换就多了几道转换的工序，这显然就降低了这俩地址转换的速度，也就是带来了时间上的开销。
  - 程序是有局部性的，即在一段时间内，整个程序的执行仅限于程序中的某一部分。相应地，执行所访问的存储空间也局限于某个内存区域。
  - 我们就可以利用这一特性，把最常访问的几个页表项存储到访问速度更快的硬件，于是计算机科学家们，就在 CPU 芯片中，加入了一个专门存放程序最常访问的页表项的 Cache，这个 Cache 就是 TLB（Translation Lookaside Buffer），通常称为页表缓存、转址旁路缓存、快表等。