虚拟化内存

假象：看起来每个程序都拥有私有的内存
实际：许多程序同时在共享内存

灵活性：如何尽可能的让程序可以简单的操作内存

访问受限：如何保证程序之间的内存互不影响，也不会影响操作系统

高效

地址转换

实现

硬件支持：CPU中添加MMU(内存单元管理)，包括一对基址寄存器base和界限寄存器bound

进程看到的是虚拟地址virtual address，操作系统读取的是转换后的实际地址physical address

physical address = virtual address + base

bound保证进程只能访问自己的内存空间

进程切换时，操作系统会将当前进程的base和bound的值保存在进程中，例如PCB中，以便切换

缺陷

base和bound粒度太大，会导致已分配的内存中有未被使用的空间，称为内部碎片

分段

实现

将进程空间分为代码段、堆、栈进行分别映射

寄存器从一对变成了三对，每一段都有一对base和bound，称为段寄存器

虚拟地址由段+偏移量构成(显式方式)

寄存器中除了base和bound还记录保护位等其他信息，支持更多功能，例如共享内存段

缺陷

每个段的大小都不一样，很快内存中会充满不连续的空间，称为外部碎片

如何有效管理空闲空间？

空闲列表

最优匹配，选择大小最匹配的空闲块，性能代价高

最差匹配，选择最大的空闲块进行分割，不仅也需要遍历整个列表，还会产生过量碎片

首次匹配，找到第一个足够大的空闲块，速度快，但是列表开头可能会有很多小块

下次匹配，每次从上一次查找的位置往下查找，性能与首次匹配接近

分离空闲列表，将经常被申请的固定大小的空闲块单独管理

伙伴系统，每次分配2^n次方大小的内存块，回收比较方便，但是会产生内部碎片

分页

实现

将内存分配为相同长度的分片，每一片为一页

每个进程都有一份页表，保存虚拟页与实际物理内存页的映射关系，每一条映射关系称为PTE

页表

VP0 -> PF3
VP1 -> PF7
...
physical address = VPN(虚拟页号) + offset(页内偏移量)

优点

不需要考虑进程如何使用内存，不用像分段中还要记录内存增长方向

空闲空间管理方便

缺陷

读取数据可能很慢

页表可能非常大

如何尽量避免额外的那一次内存访问？

地址转换缓存TLB

原理

TLB位于CPU附近，访问速度很快

TLB中存储VPN:PFN的映射

每次访问内存，硬件先检查是否存在映射

如果存在，可以直接读取PFN计算实际物理地址，避免了读取PTE获取PFN那一次内存访问

如果不存在，读取PTE中的PFN，存入TLB，再重新尝试内存访问

应用

在有时间局部性(例如循环变量)和空间局部性(数组，相邻元素通常位于同一页)特点的程序中表现很好

仍需考虑的问题

进程上下文切换时，TLB原有的映射如何处理

可以在TLB中添加更多位，用来标记属于哪个进程

TLB替换策略

内存页换出到磁盘策略

策略

比起分段直接从寄存器中获取基址再加上偏移量可以得到physical address，分页会多一次内存读取

1. 根据虚拟地址获得VPN，offset
2. 根据VPN以及页表基址寄存器PTBR获取PTE的物理地址
3. 读取PTE(额外的内存读取)
4. 根据PTE中的记录，找到PFN，结合offset获取physical address
5. 读取地址

原因

32位系统，页大小4K(2^12)，VPN会有20位，可以表示为2^20个地址，如果一个PTE需要4个字节，那么光是一个进程就需要2^22(4M)内存的页表

如何解决？

方案

更大的页

偏移量变大，VPN变小，从而页表变小

带来页内浪费——内部碎片【注意和外部碎片区分】

分页和分段混合

段寄存器记录页表起始地址和最大有效页号。节省了内存，结合了两者的优点，段之间未被使用的页不占用页表空间

分段方式不够灵活；仍然会带来外部碎片

原因

对于普通的分页来说，一个进程空间的页表是固定大小的，由位数决定。这样对于操作系统来说，分配页表是比较方便的。但是段页式，每个段的页表大小都不一样，是PTE的倍数，那么意味着一个进程的页表大小不是固定的。分配内存时，仍然可能会出现外部碎片。

多级页表

将传统的线性页表变成了树状(VPN中分离出几位当做页目录索引)，页表紧凑并且支持稀疏的地址空间。

但是带来了成本(未命中TLB时，更多次的内存查找)和复杂性。

反向页表

整个系统中只有一个页表，记录所有页当前被哪个进程所使用

页表交换到磁盘

物理内存有限，进程的地址空间无法放入，将不需要的页表【暂时】换出到磁盘，需要时换入，营造出内存很大的假象。

深入

细节

内存满了应该将哪些页交换出去？

页交换策略

目标

尽可能让未命中次数减少，增加命中次数

策略

最优替换OPT

替换内存中最久才会被访问的页。难以实现，但是可以作为衡量其他策略的基准

先进先出FIFO

发生替换时，将最先进入的页换出到磁盘。实现简单，但是忽略了页的信息，可能把经常使用的页因为最先进入队列替换出去

随机Random

取决于运气

最少使用LRU

能够利用页的历史访问信息，有一定提升。但是需要做额外的操作，比如移动页，更新页的访问时间，替换页时需要遍历所有页的访问时间。代价高昂

近似LRU

时钟算法，定时将使用位为1的页的该位反转为0，并将使用位为0的页替换出去。

考虑脏页

脏页落盘需要写入操作，替换未被修改的页就没有副作用。因此倾向于踢出未被修改的干净页，可以使用PTE的dirty位标识

交换出去的页位于哪里？

在硬盘中开辟一块交换空间swap space

怎么确定页在内存中还是在磁盘上？

在PTE中使用存在位present来标识当前页是否存在物理内存中，如果不存在会触发页错误page fault

触发page fault后，如何处理？

PTE中存储PFN的位置也可以用于存储硬盘地址。当操作系统发现page fault时，根据PTE查找硬盘地址，将页读取到内存中，并且更新页表中PTE的present位，再重试指令。

交换的时机一定要等到内存满了吗？

大多数操作系统会设置高水位HW和低水位LW，当后台运行的守护进程检测到少于LW个页可用，会执行内存释放，将部分页交换出去，直到存在HW个页可用。

页选择策略

从磁盘预取页到内存

聚集写入到磁盘

进程内存需求超过物理内存，频繁换页导致抖动

准入控制，只让部分进程运行

杀死内存密集型进程