链滴

Linux 内存管理 Q&A

作者：Yukibana

原文链接：https://ld246.com/article/1589428361410

来源网站：链滴

这篇文章记录了我在学习Linux内存管理时比较关心的几个问题，很多回答可能不是很精确，属于我完以后还能留在脑子里的内容。 如果以后有应用的机会的话，会再拿起书复习复习的。

内存寻址部分

什么是段？ 为什么Linux中很少使用段?

首先明确段的意义，段的意义在于逻辑地址加上段选择子的地址可以产生不同的地址。 也就是如果面段编程的话， 拥有不同段选择子的段就会被分散到不同的线性地址上，从而达到 **所有的段的使用者享一套地址空间的目的。** 此外段编程鼓励程序员将代码也分成不同的段，如全局数据和局部数据等。由于我没有接触过面向段的编程，所以上面的描述可能有错误。

但是Linux每一个进程都有独立的地址空间，没有共享地址空间，页表的必要。 所以段的切换仅仅在核和用户空间切换的时候进行。

而且，段的使用是不跨平台的。仅在x86上使用。

简单讲一下Linux的分页系统

首先我们要明确，分页很大一部分程度上是依赖体系结构的。有些体系结构只有两级页表，有些体系构则有三级甚至四级页表。 Linux提供了一个通用的分页方案使得可以跨体系结构。

具体怎么实现的？ 老实说我没有看懂。总之可以跨就对了，在不同的体系结构上可以有不同层数的页结构。

一些常见的组件

- CR3寄存器存放 主页表的物理地址。进程切换会更改CR3，并刷新TLB。
- TLB Translation Lookaside Buffer。 存储转换关系，加快访存速度。
- 进程页表。 分为用户空间和内核空间。前半段私有，后半段与内核页表等同，在访问发现为0时触缺页中断，从而得以复制内核页表的内容到自己的内核部分页表中。
- 内核页表。 称为Master kernel page table。 一开始会提前映射除高端内存外的其他所有动态内，所以之后可以不用再改变页表， 但是vmalloc会对内核页表进行改动，从而触发flush_tlb。 性能差。

页表存储与量化计算

以一个页4KB来看，一个4KB的页表，每个页表项是32bit，4字节，刚好可以寻址1000页，也就是4M的内存。

那么，页目录只需要1000项，4KB就可以寻址4GB内存。如果全部分配了，那么总内存消耗在4MB。在64位的情况下，页表级别增加，最耗内存的页表叠加起来也不会超过消耗太多内存。 而且，页表配是懒惰的，只有在需要的时候才创建页表项，这也极大地节省了内存。

于是，我们可以得出一个比较粗浅的结论，页表可以长时间驻留物理内存中，不被交换出去。 这样，MU就可以安全地存储和使用页表的物理地址，无论是页目录项的也好，还是页目录项中存储的页表址也好，都可以直接用物理地址表示。 当然，想要修改，还是得靠映射到地址空间里的虚拟地址。

Linux的页表

分为用户空间，内核空间两个部分。

内存管理部分

缺页异常都做哪些处理?

我就说一些我能记得住的。 首先判断线性地址是内核空间还是用户空间。 内核空间下有几条路径，核错误路径，用户参数传达错误路径，以及vmalloc懒分配调页的路径。 在用户空间下，最简单的路就是段错误，没有权限访问之类的。 如果是出现了没有分配物理页的情况，有文件内存映射分配，新分配（匿名文件映射），从磁盘读取交换文件几种。 我们可以很轻松地看到，最后一种的效率很低，以被称为主错误，其他的被称为小错误。 主错误会导致进程的阻塞， 而次错误不会。ps可以查看这个类型的缺页异常发生的情况。

什么是请求调页？ 什么是写时复制？ 什么是零页？应用场景是什么?

请求调页的核心思想就是推迟分配物理页框，直至发生缺页异常，也就是真正需要的时候。 写时复制是允许在只读的情况下，多个进程共享一个物理页框， 并设置共享位。 零页就是提前准备好的一个例，所有未经初始化的，但是需要读页都可以用零页来映射，一旦发生了写入，缺页异常处理程序会手为新的程序分配一个页框。 一个很典型的应用就是bss段。

伙伴系统，SLAB分配器，Kmalloc，Vmalloc分别是什么?

伙伴系统用于管理物理内存的分配，一般是分配一个或几个连续的物理页框，使用伙伴算法以避免外片。 那么问题又来了，为什么需要连续的物理内存？ 直接分页映射不好吗?

这里有几点，

1. DMA之类的连续物理内存刚需。

2. 内核提前做好了物理内存的映射，分配连续的物理内存可以直接使用内核页表的映射而不用修改内页表。 从而避免了tlb的刷新。

3. 。。

SLAB分配器用于分配对象，包括特定对象和通用对象，面向小内存。 好处在复用对象，避免频繁分释放，提高cache的命中率。

kmalloc构建在slab的通用对象分配上，而vmalloc就是上面提到的连续物理页分配的反面典型。 不很精确，但是我这里不是很想继续考虑高端内存映射之类的了。

地址空间部分

什么是线性区?

地址空间中的互相不重叠一块内存区域，代表一块线性空间。 存储在mm_struct的链表和红黑树中，者用于全量遍历，后者用于增删改查。基本上对于内存区域的修改都是对线性区的修改，增大线性区考虑合并等问题。 线性区的页框基本按需分配，也会出现被内核回收交换到磁盘上的情况。 对于初化，无写的情况，会映射一个单例的零页，当真正写的时候才进行cow。

Linux进程的地址空间布局如何?

从低到高，为代码区，数据区，堆区，内存映射区（有共享库的映射，动态内存的匿名映射，真文件射等），栈区（向下扩展），参数区。 之后就到了内核空间区。

Linux给用户分配内存的时候都经历了什么?

http://edsionte.com/techblog/archives/4174

顺便讲一个关于over_commited的技术。 当你malloc申请内存的时候，有两种情况。 当over_commied开启的时候，malloc 保证会返回正确的结果，也就是一个有效的地址。 当内存不够的时候呢， 操系统会选择一个victim杀掉，发送sigterm信号，然后给你分配内存。

如果关闭over_commited，那么malloc可能会返回失败。 注意到可用的总空间受物理内存大小+Swa大小的限制， 所有虚拟内存空间也并不是无限的。

注意到，malloc各有实现，我们关注的是操作系统提供了什么样的接口? 具体来看，大概就是分配释放了，上面大抵都有提到了一些。

内核有内存描述符吗?

内核也是需要页表的， 但可以通过active_mm的设置复用上一个进程的内存描述符，而将mm设置为ULL，避免了tlb和cache的刷新。

为什么用户空间也会出现内存碎片?

主要是线性空间的碎片。 用户态获取内存的方式是先确定一个堆空间，然后在堆空间里做一个利用。么堆有一个特点，就是如果上面的地址空间没有回收，下面的就没有办法回收利用（一般情况下这是用brk分配的）。 栈也是类似的，而且栈一般也不会考虑收缩。 无法回收利用，那就会占资源。而且断地申请线性空间（而不是复用已有的空间），也会造成过量的系统调用。同时也会污染cache和tlb以及swap等。 所以用户态的内存分配可以采用预分配大块内存的方式来提高效率。

Linux系统如何查看内存占用

这里简单提几个方法，可以多角度看看物理内存和虚拟内存占用。 pmap还蛮好玩的：)

1. top grep |grep pid查看虚拟内存和物理内存
2. pmap pid 查看地址空间布局
3. ps -aux | grep (pid or process_name)

Reference

https://www.cnblogs.com/xelatex/p/3491305.html