Linux 内存管理

理解程序设计与内存管理

Author	Taosheng Shi
WeChat Contact	data-lake
Mail Contact	tshshi@126.com
Organization	NOKIA
Document category	Distributed System
Document location	https://github.com/stone-note/articles

Version	Status	Date	Author	Description of changes
0.1	Draft	12/7/2017	Taosheng Shi	Initiate
0.2	Draft	DD-MM-YYYY	YourNameHere	TypeYourCommentsHere
1.0	Approved	DD-MM-YYYY	YourNameHere	TypeYourCommentsHere

Contents

自冯诺伊曼已降，计算机都是采用程序存储架构。

什么是程序存储架构？简单来说就是把用来运行的程序也当做数据一样存储在计算机中。现在听起来这么简单的一个思想，首次提出却是了不起的贡献：里面包含了哥德尔精致的集合悖论和图灵美妙的图灵机设想。最早的计算机器仅内含固定用途的程序。现代的某些计算机依然维持这样的设计方式，通常是为了简化或教育目的。例如一个计算器仅有固定的数学计算程序，它不能拿来当作文字处理软件，更不能拿来玩游戏。若想要改变此机器的程序，你必须更改线路、更改结构甚至重新设计此机器。当然最早的计算机并没有设计的那么可编程化。当时所谓的“重写程序”很可能指的是纸笔设计程序步骤，接着制订工程细节，再施工将机器的电路配线或结构改变。而程序存储型电脑的概念改变了这一切。借由创造一组指令集结构，并将所谓的运算转化成一串程序指令的运行细节，让此机器更有弹性。借着将指令当成一种特别类型的静态数据，一台存储程序型电脑可轻易改变其程序，并在程控下改变其运算内容。

讲这一段，就是要明确，在计算机中，准确的说是 CPU 中，程序和数据都是存储在计算机的内存或者硬盘中的，要使用时，都是要经过寻址来找出来加载到 CPU 中的。内存只是更快的硬盘，以下将会使用内存指代存储，忽略内存到硬盘的换入换出。

内存地址本质上对应物理硬件上的地址引脚。使用内存地址访问物理存储天经地义，内存地址和物理地址一一对应，也是天经地义。内存地址就是物理地址，这就是“实模式”。

那是什么时候内存地址有了“逻辑地址”，“线性地址”，“虚拟地址”，“物理地址”等，这些称呼？

这一切都要从 80286 说起，Intel 微处理器从这个版本开始引入了“保护模式”，也是就是内存地址的分段表示。顾名思义，这是为了内存保护的目的（还有就是分离用户空间和内核空间）。这样内存地址不再是物理地址了，而仅仅是一个偏移量了，原来的内存地址变成了逻辑地址，而这个偏移量则是“线性地址”或“虚拟地址”。要获得物理就需要在自己所在的段中去找（下面提到的段描述中有一个基地址）！为此，Intel 引入了段描述符以及保护目的的鉴权属性，结合“程序存储”的概念，至少存在两种段描述符：代码段描述符和数据段描述符。如何找到内存地址所在的段描述符呢？Intel 又引入段选择子，以及鉴权属性。这样使用段选择子就是可以找到段描述符，再使用原来的内存地址作为偏移量来找到真正的物理地址了。但是从实模式到保护模式，只有一个地址序号，哪里去找段选择子和段描述符呢？Intel 引入了一些段寄存器来存储段选择子，当然至少是代码段寄存器和数据段寄存器。而内核在启动时会建立全局段描述符表，这样一切都解决了。为了加快段描述符表的访问（否则又是瓶颈），Intel 又引入了不可编程的段描述符寄存器，随着段寄出器加载而加载。这就是鼎鼎大名的影子寄存器。

现在看来，这次失败的设计还是挺成功的。

Linux 就直接绕过了分段机制。分段机制引入段选择子和段描述符把地址空间转换成偏移量，Linux 通过为所有程序设置相同的段选择子和段描述符，把偏移量又转换成地址空间，一切又回到了原点。

看看分页是多么简单和优雅。以 32 位地址空间为例，分为三段（10，10，12），分别是作为页目录，页表和页框的索引，即可访问 32G 的物理内存。只需要存储页目录一个寄存器（cr3）即可。

在整个寻址的过程中，TLB 缓存是至关主要的。TLB 缓存内存线性地址到物理地址的直接映射，你说重要不重要？

TLB 缓存还间接了影响了地址空间架构。我们知道 Linux 环境下 32 位系统进程地址空间是 4G，这 4G 地址空间用户态占 3G，内核态占 1G。并且用户态各个进程的地址空间是独立的，也就是每个进程都可以访问 0 到 3G 的进程地址空间。而内核态的进程地址空间是所有进程共享的，即所有进程共用 1 个的地址空间。（更准确的说法是把内核地址空间映射到所有进程的地址空间）。内核为什么这么设计呢？为什么要划分用户空间和内核空间呢？答案就在 TLB 缓存：因为在内核空间进入/退出时，刷新整个 TLB 的代价太高了。

In the i386 arch, for example, we choose to map the kernel into every process's VM space so that we don't have to pay the full TLB invalidation costs for kernel entry/exit. This means the available virtual memory space (4GiB on i386) has to be divided between user and kernel space.

和 TLB 缓存相关还有一个重要的概念是 hugepage。hugepage 支持建立在大多数现代处理器架构提供的多页大小支持之上。例如，x86 CPU 通常支持 4K 和 2M（1G，如果架构支持）页面大小，ia64 架构支持多页大小 4K，8K，64K，256K，1M，4M，16M，256M 以及 ppc64 支持 4K 和 16M。随着越来越大的物理存储器（几 GB）更容易获得，TLB 的优化更为关键。

TLB 驻留在 CPU 的 1 级 cache 里，是芯片访问最快的缓存，一般只能容纳 100 多条页表项，如果采用 hugepage，则可以极大减少 TLB cache miss 导致的开销：TLB 命中，立即就获取到物理地址，如果不命中，需要查 rc3->进程页目录表 pgd->进程页中间表 pmd->进程页框->物理内存，如果这中间 pmd 或者页框被虚拟内存系统替换到交互区，则还需要交互区 load 回内存。。总之，TLB cache miss 是性能大杀手，而采用 hugepage 可以有效降低 TLB cache miss。

一旦大量的页面被预分配给内核作为 hugepage 的页面池，这些页面将在内核中保留，不能用于其他目的。内核使用名字为“hugetlbfs” 的文件系统管理这些页面池。当支持多个 hugepage 大小时，/proc/sys/vm/nr_hugepages 指示预先分配的大量页面的默认大小的当前数量。因此，可以使用以下命令来动态分配/取消分配默认大小的持续 hugepage：

echo 20 > /proc/sys/vm/nr_hugepages

该命令将尝试将 hugepage 页面池中的默认大小的 hugepage 的数量调整为 20 ，根据需要分配或释放 hugepage。

/proc/meminfo 文件提供有关内核 hugepage 池中持久 hugetlb 页面总数的信息。它还显示有关免费，预留和剩余 hugepage 数量以及默认页面大小的信息。“cat /proc/meminfo”的输出将包括以下行：

.....

HugePages_Total: vvv

HugePages_Free: www

HugePages_Rsvd: xxx

HugePages_Surp: yyy

Hugepagesize: zzz kB

其中： HugePages_Total 是 hugepage 页面池的大小。 HugePages_Free 是池中尚未分配的 hugepage 数。 HugePages_Rsvd 是“保留” 的缩写，是从池中分配的承诺的 hugepage 的数量，但尚未分配。保留 hugepage 保证应用程序能够在故障时间从 hugepage 页面池中分配一个 hugepage。 HugePages_Surp 是“剩余”的缩写，是 /proc/sys/vm/nr_hugepages 中的值之上的 hugepage 数。剩余 hugepage 的最大数量由/proc/sys/vm/nr_overcommit_hugepages 控制。

由于内核 1G 地址空间的限制，对于高端内存（物理地址空间大于虚拟地址空间的情况），内核无法同时映射所有物理内存，这意味着当使用这些内存时，内核使用临时映射。

说到高端内存，不禁想起了物理地址扩展。处理器所支持的 RAM 容量受链接到地址总线上的地址管脚数限制。早起 Intel 处理器从 80386 到 Pentium 使用 32 位物理地址。从理论上讲，这样的系统上可以安装高达 4GB 的 RAM，而实际上，由于用户进程线性地址空间的需要，内核不能直接对 1GB 以上的 RAM 进行寻址。然而，大型服务器需要大于 4GB 的 RAM 来同时运行上千的进程，实际上我们现在的很多计算机的 RAM 都可能超过这个量级。Intel 通过在它的处理器上把管脚数从 32 增加到 36 已经满足了这些需求。从 Pentium Pro 开始，Intel 所有的处理器现在的寻址能力达 2^36=64GB.不过，只有引入一种新的分页机制把 32 位线性地址转换为 36 位物理地址才能使用所增加的物理地址。显然，PAE 并没有扩大进程的线性地址空间，因为它只能处理物理地址，此外，只有内核能够修改进程的页表，所以用户态下运行的进程不能使用大于 4GB 的物理地址空间。另一方面，PAE 允许内核使用高达 64GB 的 RAM，从而显著增加了系统中的进程数量。

就写到这儿吧。

知行时刻：Intel 从 Pentium 开始将 Cache 分开，通常分为一级高速缓存 L1 和二级高速缓存 L2。在以往的观念中，L1Cache 是集成在 CPU 中的，被称为片内 Cache。在 L1 中还分数据 Cache(I-Cache)和指令 Cache(D-Cache)。它们分别用来存放数据和执行这些数据的指令，而且两个 Cache 可以同时被 CPU 访问，减少了争用 Cache 所造成的冲突，提高了处理器效能。

在高性能软件开发的过程中，if 语句的使用都要小心。想想程序员随意的深度函数调用和 if 语句使用，CPU 要受到几万点伤害（I-Cache 抖动和 I-Cache miss）？