深入理解 Linux 内核 第二章内存寻址

Contents

第二章 内存寻址

内存地址

引用内存地址 memory address 是访问内存单元内容的一种方式。需要区分以下三种不同的地址：

- 逻辑地址 logical address

包含在机器语言指令中用来指定一个操作数或一条指令的地址。每一个逻辑地址都由一个段 segment 和偏移量 offset 或 displacement 组成，偏移量指明了从段开始的地方到实际地址之间的距离。

- 线性地址 linear address， virtual address

一个 32 位无符号整数，可以用来高达 4GB 的地址。通常使用十六进制表示，范围从 0x00000000 到 0xffffffff。

- 物理地址 physical address

用于内存芯片级内存单元寻址。他们与从微处理器的地址引脚发送到内存总线上的电信号相对应。物理地址由 32 位或 64 位无符号整数表示。

内存管理单元 MMU 通过一种称为分段单元 segmentation unit 的硬件电路把一个逻辑地址转换为现行地址；接着，第二个称为分页单元 paging unit 的硬件电路把线性地址转换为一个物理地址。

逻辑地址–>[分段单元]–>线性地址–>[分页单元]–>物理地址

硬件中的分段

段选择符和段寄存器

一个逻辑地址由两部分组成：一个段标识符和一个指定段内相对地址的偏移量。段标识符是一个 16 位长的字段，称为段选择符 segment selector，偏移量是一个 32 位长的字段。

为了快速方便地找到段选择符，处理器提供段寄存器，段寄存器的唯一目的是存放段选择符。这些段寄存器称为 cs，ss，ds，es，fs 和 gs。程序可把一个段寄存器用于不同的目的，先将值保存到内存中，用完之后再恢复。

6 个寄存器中 3 个有专门的用途：

- cs 代码段寄存器，指向包含程序指定的段。cs 寄存器还有一个很重要的功能，它包含一个两位的字段，用以指明 CPU 的当前特权等级 Current Privilege Level，CPL。0 表示最高优先级，3 表示最低优先级。Linux 使用 0 和 3 来表示内核态和用户态。
- ss 栈段寄存器，指向包含当前程序栈的段。
- ds 数据段寄存器，指向包含静态数据或者全局数据段。

段描述符

每个段有一个 8 字节的段描述符 segment descriptor 表示，描述了段的特征。段描述符放在全局描述符表 Global Descriptor Table，GDT 或局部描述符表 Local Descriptor Table，LDT 中。

通常只定义给一个 GDT，每个进程处理存放在 GDT 中的段之外，如果还需要创建附加的段，就可以有自己的 LDT。GDT 在主存中的地址和大小存放在 gdtr 控制寄存器中，当前正被使用的 LDT 地址和大小放在 ldtr 控制寄存器中。

快速访问段描述符

每当一个段选择符被装入段寄存器时，相应的段描述符就由内存装入到对应的非编程 CPU 寄存器。从那时起，转对那个段的逻辑地址转

换就可以不访问 GDT 或 LDT，处理器只需要直接引用存放段描述符的 CPU 寄存器即可。仅当段寄存器的内容改变时，才有必要访问 GDT 或 LDT。

段选择符字段

字段名	描述
index	指定放在 GDT 或 LDT 中的相应段描述符的入口
TI	Table Indicator，指明段描述符是在 GDT 中 TI=0，或在 LDT 中 TI=1
RPL	请求者特权级：当相应的端选择符装入到 cs 寄存器中时指示出 CPU 当前的特权等级

分段单元

逻辑地址转换成相应的线性地址，分段单元要执行如下操作：

- 先检查段选择符的 TI 字段，以决定段描述符保存在哪一个描述符表中。
- 从段选择符的 index 字段计算出段描述符的地址。
- 把逻辑地址的偏移量与段描述符 Base 字段的值相加就得到了线性地址。

Linux 中的分段

Linux 以非常有限的方式使用分段，实际上，分段和分页在某种程度上有点多余，因为他们都可以划分进程的物理地址空间：分段可以给每一个进程分配不同的线性地址空间，而分页可以把同一线性地址空间映射到不同的物理空间。Linux 更喜欢使用分页的方式，因为

- 当所有进程使用相同的段寄存器值时，内存管理变得更简单，也就是说它们能共享同样的一组线性地址。
- Linux 设计目标之一是可以把它移植到绝大多数流行的处理器平台上。然而 RISC 体系结构对分段的支持很有限。

硬件中的分页

分页单元一个关键任务就是把所请求的访问类型与线性地址的访问权限相比较，如果这次内存访问是无效的，就产生一个缺页异常。

为效率起见，线性地址被分成以固定长度为单位的组，称为页 page。页内部连续的线性地址被映射到连续的物理地址中。

分页单元把所有的 RAM 分成为固定长度的页框 page frame。

把线性地址映射到物理地址的数据结构称为页表 page table。页表存放在主存中，并在启用分页单元之间必须由内核对页表进行适当的初始化。

常规分页

每 4KB 分为一页。

扩展分页

扩展分页用于把大段连续的线性地址转换成相应的物理地址，在这些情况下，内核可以不用中间页表进行地址转换，从而节省内存并保留 TLB 项。

硬件高速缓存

为了缩小 CPU 和 RAM 之间的速度不匹配，引入了硬件高速缓存内存 hardware cache memory。硬件高速缓存基于局部性原理 locality principle，该原理表明由于程序的循环结构及相关数组可以组织称线性数组，最近最常用的相邻地址在最近的将来又被用到的可能性极大。

高速缓存单元插在分页单元和主内存之间。它包含一个硬件高速缓存内存 hardware cache memory 和一个高速缓存控制器 cache controller。高速缓存控制器存放一个表项数组，每个表项对应高速缓存内存中的一个行。每个表项有一个标签 tag 和描述高速缓存行状态的几个标识 flag。这个标签由一些位组成，这些位让高速缓存控制器能够辨别由这个行当前所映射的内存单元。这种内存物理地址通常分为 3 组：最高几位对应标签，中间几位对应高速缓存控制器的子集索引，最低几位对应行内的偏移量。

当访问一个 RAM 存储单元时，CPU 从物理地址中提取出子集的索引并把子集中所有行的标签与物理地址的高几位相比较。如果发现某一行的标签与这个物理地址的高位相同，则 CPU 命中一个高速缓存 cache hit；否则高速缓存没有命中 cache miss。

对于读操作，命中高速缓存后，控制器从高速缓存行中选择数据并送到 CPU 寄存器。对于写操作，控制器可采用以下两种基本策略之一，分别称为通写 write-through 和回写 wirte-back。通写表明控制器即写 RAM 也写高速缓存，回写表示只更新高速缓存，而不改变 RAM 的内容。回写结束后，RAM 最终必须被更新。只有当 CPU 执行一条要求刷新高速缓存的指令时，或者当一个 FLUSH 硬件信号产生时（通常在高速缓存不命中之后），高速缓存控制器才把高速缓存行写回到 RAM 中。

多处理器系统的每一个处理器都有一个单独的硬件高速缓存，因此它们需要额外的硬件电路用于保护高速缓存内容的同步。当一个 CPU 修改了它的硬件高速缓存，它就必须检查同样的数据是否包含在其他的硬件高速缓存中；如果是，它必须通知其他 CPU 用适当的值对其更新。常把这种活动叫做高速缓存侦听 cache snooping。

转换后援缓冲器 TLB

translation lookaside buffer 用于加快线性地址的转换。当一个线性地址第一次使用时，通过慢速访问 RAM 中的页表计算出相应的物理地址。同时，物理地址被存放在一个 TLB 表项 TLB entry 中，以便以后对同一个线性地址的引用可以快速地得到转换。

在多处理器中，每个 CPU 都有自己的 TLB。

Linux 中的分页

Linux 采用了一种同时适用于 32 位和 64 位系统的普通分页模型。从 2.6.11 版本开始，Linux 采用了四级分页模型，四种页表分别为：

- 页全局目录 page global directory
- 页上级目录 page upper directory
- 页中间目录 page middle directory
- 页表 page table

页全局目录包含若干个页上级目录的地址，页上级目录包含若干个页中间目录的地址，页中间目录包含若干个页表的地址。每一个页表指向一个页框。

对于没有启用物理地址扩展的 32 位系统，两级页表已经足够。Linux 通过使页上级目录和页中间目录位全为 0，从根本上取消了页上级目录和页中间目录字段。

Linux 的进程处理很大程度上依赖于分页。事实上，线性地址到物理地址的自动转换使下面的设计目标变得可行：

- 给每一个进程分配一块不同的物理地址空间，这确保了可以有效地防止寻址错误。

- 区别页（一组数据）和页框（即主存中的物理地址）之不同。这就允许存放在某个页框中的一个页，然后保存到磁盘上，以后重新装入这同一页时又可以被装在不同的页框中。这就是虚拟内存机制的基本要素。

物理内存布局

在初始化阶段，内核必须建立一个物理地址映射来指定哪些物理地址范围对内核可用，哪些不可用。

内核将下列页框记为保留 reserved：

- 在不可用的物理地址范围内的页框。
- 含有内核代码和已初始化的数据结构的页框。

保留页框中的页绝不能被动态分配或交换到磁盘上。

一般来说，Linux 内核被安装在 RAM 中从物理地址 0x00100000 开始的地方，为什么？

- 页框 0 由 BIOS 使用，存放上电自检 power-on self-test POST 期间检查到的系统硬件配置。
- 物理地址从 0x000a0000 到 0x000fffff 的范围通常留给 BIOS 例程。
- 第一个 MB 内的其他页框可能由特定计算机模型保留。

进程页表

进程的线性地址空间分为两部分：

- 从 0x00000000 到 0xbfffffff 的线性地址，无论进程运行在用户态还是内核态都可以寻址。
- 从 0xc0000000 到 0xffffffff 的线性地址，只有内核态的进程才能寻址。

内核页表

内核维持着一组自己使用的页表，驻留在所谓的主内核页全局目录 master kernel page global directory 中。系统初始化后，这组页表还未被任何进程或任何内核线程直接使用；更确切地说，主内核也全局目录的最高目录项部分作为参考模型，为系统中每个普通进程对应的页全局目录项提供参考模型。

内核如何初始化自己的页表？分为两个阶段：

- 内核创建一个有限的地址空间，包括内核的代码段和数据段、初始页表和用于存放动态数据结构的共 128KB 大小的空间。这个最小限度的地址空间仅够将内核装入 RAM 和对其他初始化的核心数据结构。
- 内核充分利用剩余的 RAM 并适当地建立分页表。

临时内核页表

临时页全局目录是在内核编译过程中静态地初始化的。

注：硬件高速缓存和 TLB 的关系与区别如下：

- TLB 存放在主存中，硬件高速缓存则是处理器中一个组成部分
- 当 CPU 收到应用程序传递过来的虚拟地址后，先访问 TLB，找到虚拟地址对应的物理地址，如果没找到，就需要访问通用页表来获取物理地址。然后访问硬件高速缓存，看是否有缓存该物理地址的数据，如果没有，就是没有命中。