

OS概念与Linux内核代码分析之二：内存寻址

李中国

苏州大学计算机科学与技术学院

June 5, 2012

内容提要

80x86微处理器的地址格式

实现分段的硬件装置

Linux + Intel平台上的内存分页

内容提要

80x86微处理器的地址格式

实现分段的硬件装置

Linux + Intel平台上的内存分页

内容提要

80x86微处理器的地址格式

实现分段的硬件装置

Linux + Intel平台上的内存分页

三类地址及其相互关系

逻辑地址 根据80x86处理器的设计，二维的逻辑地址格式为(segment ID, offset).

线性地址 一维的32-bit无符号整型数

物理地址 用于对物理内存芯片进行寻址。一般为32-bit或者36-bit.

三类地址的关系



内存寻址: 实模式与保护模式

实模式 寻址采用和8086相同的16位段和偏移量，最大寻址空间1MB($4 \times \text{seg} + \text{off}$)，最大分段64KB。

保护模式 寻址采用32位段和偏移量，最大寻址空间4GB，最大分段4GB (Pentium Pro及以后为64GB)。

两种模式的主要区别

二者的根本区别是进程内存受保护与否。实模式下系统程序和用户程序没有区别对待，每一个指针都是指向“实在”的物理地址。

段选择符及段寄存器

段选择符

逻辑地址由16位的段号(segment ID)和32位段内偏移(offset)组成。其中段号在Intel平台上称为段选择符(segment selector)。其16个二进制位中存放信息如下图。



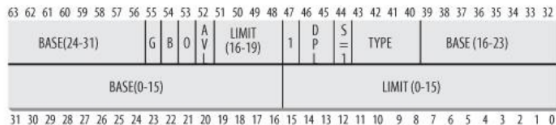
段寄存器

CPU内部专门用来存放段选择符的存储单元。共有六个：**cs**, **ss**, **ds**, **es**, **fs**, **gs**。其中：

- cs** 代码段寄存器(程序指令)
- ss** 栈段寄存器(运行时的栈)
- ds** 数据段寄存器(全局及静态变量)

段描述符

内存分段后，每个段表项存放相应段的性质（如地址、存放内容的类型等）。段表项在Intel平台上称为段描述符(segment descriptor)。有两类段表：全局段表(GDT,系统中仅有一个), 局部段表(LDT,每个进程可以单独有一个)。



几个关键字段

Base 该段起始地址

Limit 该段总长度

G 标记Limit字段的单位是1字节还是4096字节

Type 该段类型(代码、数据、其它)

段选择符的三个字段

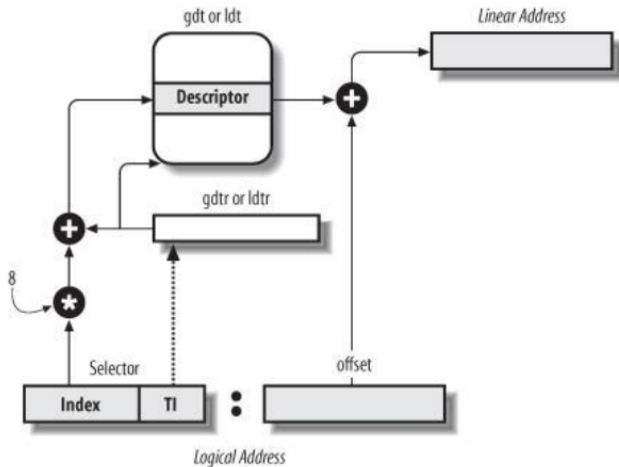
index 用于对段表进行索引

TI 用于指定该选择符对应GDT(TI=0)还是LDT(TI=1)

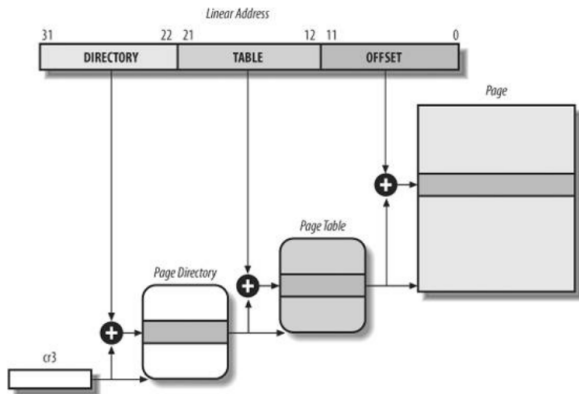
RPL 记录CPU权限状态(核心态/用户态)



从逻辑地址到线性地址的映射

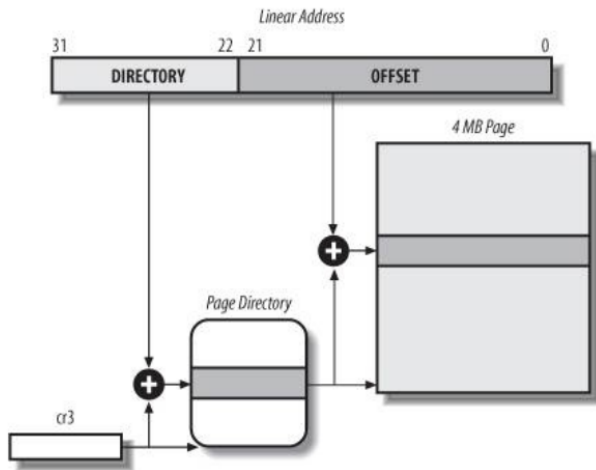


Intel平台上的分页技术



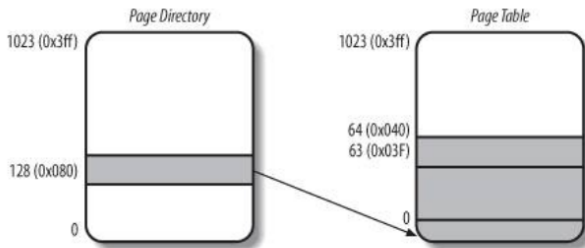
cr3寄存器用于存放当前进程的page directory(第一级页表).

Intel平台上的扩展分页模式



扩展分页模式下，页面大小为**4M**，用于处理大块连续地址空间，以节省页表所占空间及**TLB**空间。

分页的实际例子

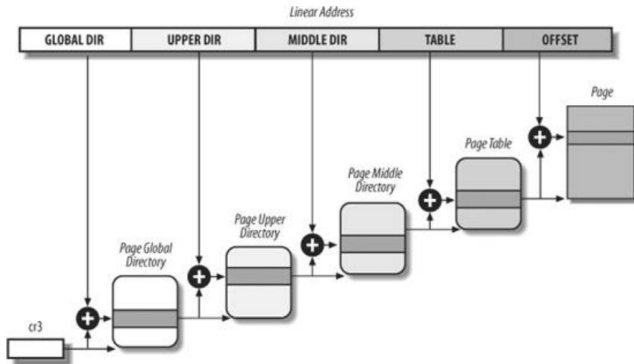


假设内核把从0x20000000到0x2003ffff之间的线性地址(共64页)分配给某进程。

- ▶ 该段地址最高10位均为0010000000, 即0x080(十进制128), 因此第一级页表只有一个页表项有效。
- ▶ 该段地址中间10位范围为0 - 0x03f(即0 - 63). 所以第二级页表中开头64个页表项有效。(图示灰色区域)
- ▶ 给定线性地址0x20021406, 如何确定其对应的物理地址?

Linux的分页技术实现方式

为了能够同时处理32位和64位机器的分页方式, Linux创造性地使用4级分页来同时适应硬件1–4级分页需求。



Linux的分页技术实现方式

上图中共有四种类型的页表:

Page Global Directory

Page Upper Directory

Page Middle Directory

Page Table

对于32位机器，2级页表就够了，此时，Linux把Page Upper Directory和Page Middle Directory两字段对应的地址长度设置为0.

有关线性地址各个字段及页表项的一些宏

`include/asm-x86_64/{page.h, pgtable.h}`

- ▶ `PAGE_SHIFT`, `PAGE_SIZE`, `PAGE_MASK`
- ▶ `PMD_SHIFT`, `PMD_SIZE`, `PMD_MASK`
- ▶ `PUD_SHIFT`, `PUD_SIZE`, `PUD_MASK`
- ▶ `PGDIR_SHIFT`, `PGDIR_SIZE`, `PGDIR_MASK`

如何根据SHIFT, SIZE, MASK的关系定义上述宏?

`include/asm-x86_64/pgtable.h`

- ▶ `PTRS_PER_PTE`
- ▶ `PTRS_PER_PMD`
- ▶ `PTRS_PER_PUD`
- ▶ `PTRS_PER_PGDIR`

需要注意**32**位和**64**位机器(页表级数不同)上述值的差别。

处理页表的函数

各类页表项的定义: `include/asm-x86_64/page.h`

```
typedef struct { unsigned long pte; } pte_t;  
typedef struct { unsigned long pmd; } pmd_t;  
typedef struct { unsigned long pud; } pud_t;  
typedef struct { unsigned long pgd; } pgd_t;
```

注: Linux内核以后缀_t表示系统自定义类型。

思考

为什么不直接用**unsigned long**表示各类页表项?

处理页表的函数

include/asm-x86_64/page.h

```
#define pte_val(x)    ((x).pte)
#define pmd_val(x)    ((x).pmd)
#define pud_val(x)    ((x).pud)
#define pgd_val(x)    ((x).pgd)
#define pgprot_val(x) ((x).pgprot)

#define __pte(x) ((pte_t) { (x) })
#define __pmd(x) ((pmd_t) { (x) })
#define __pud(x) ((pud_t) { (x) })
#define __pgd(x) ((pgd_t) { (x) })
#define __pgprot(x) ((pgprot_t) { (x) })
```