80386的分段机制、分页机制和物理地址的形成

注：本分类下文章大多整理自《深入分析linux内核源代码》一书，另有参考其他一些资料如《linux内核完全剖析》、《linux c 编程一站式学习》等，只是为了更好地理清系统编程和网络编程中的一些概念性问题，并没有深入地阅读分析源码，我也是草草翻过这本书，请有兴趣的朋友自己参考相关资料。此书出版较早，分析的版本为2.4.16，故出现的一些概念可能跟最新版本内核不同。

此书已经开源，阅读地址 [http://www.kerneltravel.net](http://www.kerneltravel.net/kernel-book/%E6%B7%B1%E5%85%A5%E5%88%86%E6%9E%90Linux%E5%86%85%E6%A0%B8%E6%BA%90%E7%A0%81.html)

MOVE REG，ADDR ;它把地址为ADDR（假设为10000）的内存单元的内容复制到REG 中

在8086 的实模式下，把某一段寄存器（段基址）左移4 位，然后与地址ADDR 相加后被直接送到内存总线上，这个相加后的地址（20位）就是内存单元的物理地址，而程序中的这个地址ADDR就叫逻辑地址（或叫虚地址）。

在80386 的段机制中，逻辑地址由两部分组成，即段部分（选择符）及偏移部分。

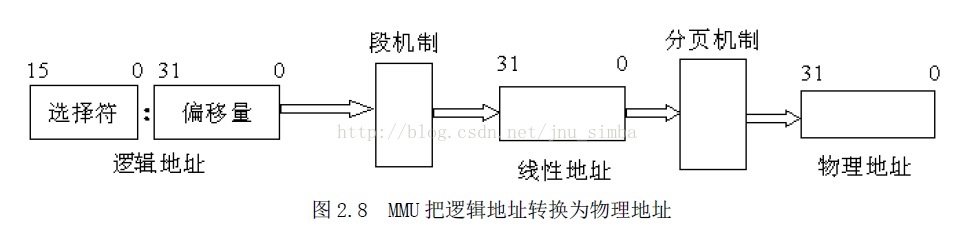
段是形成逻辑地址到线性地址转换的基础。如果我们把段看成一个对象的话，那么对它的描述如下。

（1）段的基地址（Base Address）：在线性地址空间中段的起始地址。

（2）段的界限（Limit）：表示在逻辑地址中，段内可以使用的最大偏移量。

（3）段的属性（Attribute）： 表示段的特性。例如，该段是否可被读出或写入，或者该段是否作为一个程序来执行，以及段的特权级等。

## 1、逻辑地址、线性地址和物理地址



所谓描述符（Descriptor），就是描述段的属性的一个8 字节存储单元。

## 2、用户段描述符（Descriptor）





一个段描述符指出了段的32 位基地址和20 位段界限（即段大小）。第6 个字节的G 位是粒度位，当G=0 时，段长表示段格式的字节长度，即一个段最长可达1M 字节。当G=1 时，段长表示段的以4K 字节为一页的页的数目，即一个段最长可达1M×4K=4G 字节。D 位表示缺省操作数的大小，如果D=0，操作数为16 位，如果D=1，操作数为32 位。

第7 位P 位（Present） 是存在位，表示段描述符描述的这个段是否在内存中，如果在内存中。P=1；如果不在内存中，P=0。

DPL（Descriptor Privilege Level），就是描述符特权级，它占两位，其值为0～3，用来确定这个段的特权级即保护等级。0为内核级别，3为用户级别。

S 位（System）表示这个段是系统段还是用户段。如果S=0，则为系统段，如果S=1，则为用户程序的代码段、数据段或堆栈段。

类型占3 位，第3 位为E 位，表示段是否可执行。当E=0 时，为数据段描述符，这时的第2 位ED 表示地址增长方向。第1 位（W）是可写位。当段为代码段时，第3 位E=1，这时第2 位为一致位（C）。当C=1 时，如果当前特权级低于描述符特权级，并且当前特权级保持不变，那么代码段只能执行。所谓当前特权级CPL（Current Privilege Level），就是当前正在执行的任务的特权级。第1 位为可读位R。

存取权字节的第0 位A 位是访问位，用于请求分段不分页的系统中，每当该段被访问时，将A 置1。对于分页系统，则A 被忽略未用。



## 3、系统段描述符



系统段描述符的第5 个字节的第4 位为0，说明它是系统段描述符，类型占4 位，没有A 位。第6 个字节的第6 位为0，说明系统段的长度是字节粒度，所以，一个系统段的最大长度为1M 字节。

系统段的类型为16 种，如图2.15 所示。在这16 种类型中，保留类型和有关286 的类型不予考虑。门也是一种描述符,有调用门、任务门、中断门和陷阱门4 种门描述符。



## 4、选择符、描述符表和描述符表寄存器

描述符表（即段表）定义了386 系统的所有段的情况。所有的描述符表本身都占据一个字节为8 的倍数的存储器空间，空间大小在8 个字节（至少含一个描述符）到64K 字节（至多含8K=8192）个描述符之间。

1．全局描述符表（GDT）

全局描述符表GDT（Global Descriptor Table），除了任务门，中断门和陷阱门描述符外，包含着系统中所有任务都共用的那些段的描述符。它的第一个8 字节位置没有使用。

2．中断描述符表（IDT）

中断描述符表IDT（Interrupt Descriptor Table），包含256 个门描述符。IDT 中只能包含任务门、中断门和陷阱门描述符，虽然IDT 表最长也可以为64K 字节，但只能存取2K字节以内的描述符，即256 个描述符，这个数字是为了和8086 保持兼容。

3．局部描述符表（LDT）

局部描述符表LDT（Local Descriptor Table），包含了与一个给定任务有关的描述符，每个任务各自有一个的LDT。有了LDT，就可以使给定任务的代码、数据与别的任务相隔离。每一个任务的局部描述符表LDT 本身也用一个描述符来表示，称为LDT 描述符，它包含了有关局部描述符表的信息，被放在全局描述符表GDT 中，使用LDTR进行索引。

在实模式下，段寄存器存储的是真实的段基址，在保护模式下，16 位的段寄存器无法放下32 位的段基址，因此，它们被称为选择符，即段寄存器的作用是用来选择描述符。选择符的结构如图2.16 所示。



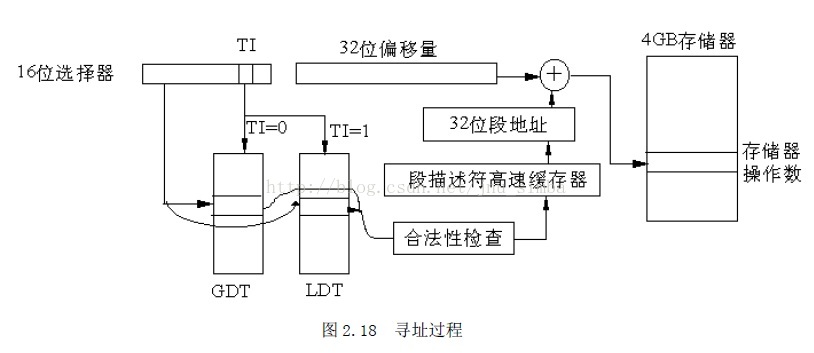
可以看出，选择符有3 个域：第15~3 位这13 位是索引域，表示的数据为0~8129，用于指向全局描述符表中相应的描述符。第2 位为选择域，如果TI=1，就从局部描述符表中选择相应的描述符，如果TI=0，就从全局描述符表中选择描述符。第1、0 位是特权级，表示选择符的特权级，被称为请求者特权级RPL（Requestor Privilege Level）。只有请求者特权级RPL 高于（数字低于）或等于相应的描述符特权级DPL，描述符才能被存取，这就可以实现一定程度的保护。

下面讲一下在没有分页操作时，寻址一个存储器操作数的步骤。

（1）在段选择符中装入16 位数，同时给出32 位地址偏移量（比如在ESI、EDI 中等）。

（2）先根据相应描述符表寄存器中的段地址（确定描述符表的地址）和段界限（确定描述符表的大小），根据段选择符的TI决定从哪种描述符表中取，再根据段选择符的索引找到相应段描述符的位置，比较RPL与DPL，若该段无问题，就取出相应的段描述符放入段描述符高速缓冲寄存器中。

（3）将段描述符中的32 位段基地址和放在ESI、EDI 等中的32 位有效地址相加，就形成了32 位物理地址。



## 5、linux中的段机制

从2.2 版开始，Linux 让所有的进程（或叫任务）都使用相同的逻辑地址空间，因此就没有必要使用局部描述符表LDT。

Linux 在启动的过程中设置了段寄存器的值和全局描述符表GDT 的内容，段寄存器的定义在include/asm-i386/segment.h 中：

C++ Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 |  | #define \_\_KERNEL\_CS 0x10 //内核代码段，index=2,TI=0,RPL=0  #define \_\_KERNEL\_DS 0x18 //内核数据段, index=3,TI=0,RPL=0 #define \_\_USER\_CS   0x23 //用户代码段, index=4,TI=0,RPL=3 #define \_\_USER\_DS   0x2B //用户数据段, index=5,TI=0,RPL=3 |

从定义看出，没有定义堆栈段，实际上，Linux 内核不区分数据段和堆栈段，这也体现了Linux 内核尽量减少段的使用。因为没有使用LDT，因此，TI=0，并把这4 个段描述符都放在GDT中, index 就是某个段描述符在GDT 表中的下标。内核代码段和数据段具有最高特权，因此其RPL为0，而用户代码段和数据段具有最低特权，因此其RPL 为3。

全局描述符表的定义在arch/i386/kernel/head.S 中：

C++ Code

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 |  | ENTRY（gdt\_table） .quad 0x0000000000000000 /\* NULL descriptor \*/ .quad 0x0000000000000000 /\* not used \*/ .quad 0x00cf9a000000ffff /\* 0x10 kernel 4GB code at 0x00000000 \*/ .quad 0x00cf92000000ffff /\* 0x18 kernel 4GB data at 0x00000000 \*/ .quad 0x00cffa000000ffff /\* 0x23 user 4GB code at 0x00000000 \*/ .quad 0x00cff2000000ffff /\* 0x2b user 4GB data at 0x00000000 \*/ .quad 0x0000000000000000 /\* not used \*/ .quad 0x0000000000000000 /\* not used \*/ |

从代码可以看出，GDT 放在数组变量gdt\_table 中。按Intel 规定，GDT 中的第一项为空，这是为了防止加电后段寄存器未经初始化就进入保护模式而使用GDT 的。第二项也没用。从下标2～5 共4 项对应于前面的4 种段描述符值。对照图2.10，从描述符的数值可以得出：

• 段的基地址全部为0x00000000；

• 段的上限全部为0xffff；

• 段的粒度G 为1，即段长单位为4KB；

• 段的D 位为1，即对这4 个段的访问都为32 位指令；

• 段的P 位为1，即4 个段都在内存。

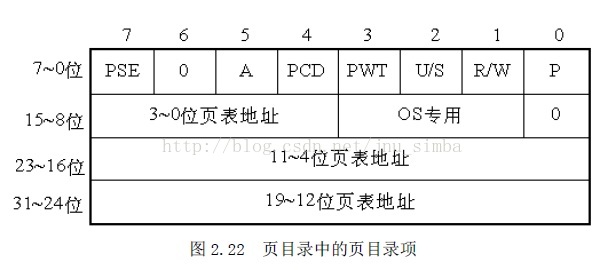
由此可以得出，每个段的逻辑地址空间范围为0～4GB。每个段的基地址为0，因此，逻辑地址到线性地址映射保持不变，也就是说，偏移量就是线性地址，我们以后所提到的逻辑地址（或虚拟地址）和线性地址指的也就是同一地址。看来，Linux 巧妙地把段机制给绕过去了，它只把段分为两种：用户态（RPL＝3）的段和内核态（RPL=0）的段，而完全利用了分页机制。

按Intel 的规定，每个进程有一个任务状态段（TSS）和局部描述符表LDT，但Linux 也没有完全遵循Intel 的设计思路。如前所述，Linux 的进程没有使用LDT，而对TSS 的使用也非常有限，每个CPU 仅使用一个TSS。TSS 有它自己 8 字节的任务段描述符（Task State Segment Descriptor ，简称TSSD）。这个描述符包括指向TSS 起始地址的32 位基地址域，20 位界限域，界限域值不能小于十进制104（由TSS 段的最小长度决定）。TSS 描述符存放在GDT 中，它是GDT 中的一个表项，由中断描述符表（IDT）中的任务门（存放TSS段的选择符）装入TR来进行索引。

## 6、页目录项、页表项、页面项

80386 使用4K 字节大小的页。每一页都有4K 字节长，并在4K 字节的边界上对齐，即每一页的起始地址都能被4K 整除。因此，80386 把4G 字节的线性地址空间，划分为1G 个页面，每页有4K 字节大小。分页机制通过把线性地址空间中的页，重新定位到物理地址空间来进行管理，因为每个页面的整个4K 字节作为一个单位进行映射，并且每个页面都对齐4K 字节的边界，因此，线性地址的低12 位经过分页机制直接地作为物理地址的低12 位使用。

页目录表，存储在一个4K 字节的页面中，最多可包含1024 个页目录项，每个页目录项为4 个字节，结构如图2.22 所示。





• 第31~12 位是20 位页表地址，由于页表地址的低12 位总为0，所以用高20 位指出32 位页表地址就可以了。

• 第0 位是存在位，如果P=1，表示页表地址指向的该页在内存中，如果P=0，表示不在内存中。

• 第1 位是读/写位，第2 位是用户/管理员位，这两位为页目录项提供硬件保护。当特权级为3 的进程要想访问页面时，需要通过页保护检查，而特权级为0 的进程就可以绕过页保护，如图2.23 所示。

• 第3 位是PWT（Page Write-Through）位，表示是否采用写透方式，写透方式就是既写内存（RAM）也写高速缓存,该位为1 表示采用写透方式。第4 位是PCD（Page Cache Disable）位，表示是否启用高速缓存，该位为1 表示启用高速缓存。

• 第5 位是访问位，当对页目录项进行访问时，A 位=1。

• 第7 位是Page Size 标志，只适用于页目录项。如果置为1，页目录项指的是4MB 的页面，即扩展分页。

80386 的每个页目录项指向一个页表，存储在一个4K 字节的页面中，页表最多含有1024 个页面项，每项4 个字节，包含页面的起始地址和有关该页面的信息。页面的起始地址也是4K 的整数倍，所以页面的低12 位也留作它用，如图2.24 所示。



第31～12 位是20 位物理页面地址，除第6 位外第0～5 位及9～11 位的用途和页目录项一样，第6 位是页面项独有的，当对涉及的页面进行写操作时，D 位被置1。

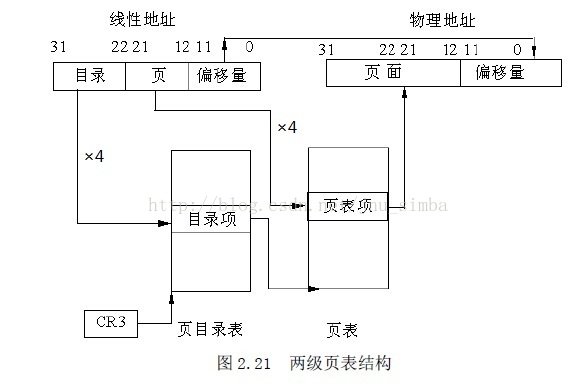
4GB 的存储器只有一个页目录，它最多有1024 个页目录项，每个页目录项又含有1024个页面项，因此，存储器一共可以分成1024×1024=1M 个页面。由于每个页面为4K 个字节，所以，存储器的大小正好最多为4GB。

当访问一个操作单元时，如何由分段结构确定的32 位线性地址通过分页操作转化成32位物理地址呢？

第一步，CR3 包含着页目录的起始地址，用32 位线性地址的最高10 位A31~A22 作为页目录表的页目录项的索引，将它乘以4，与CR3 中的页目录表的起始地址相加，形成相应页目录项的地址。

第二步，从指定的地址中取出32 位页目录项，它的低12 位为0，这32 位是页表的起始地址。用32 位线性地址中的A21~A12 位作为页表中的页表项的索引，将它乘以4，与页表的起始地址相加，形成相应页表项的地址。

第三步，从指定地址中取出32位页表项，它的低12位为0，这32位是页面地址，将A11~A0 作为相对于页面地址的偏移量，与32 位页面地址相加，形成32 位物理地址。



## 7、linux 中的分页机制

Linux 的分段机制使得所有的进程都使用相同的段寄存器值，这就使得内存管理变得简单，也就是说，所有的进程都使用同样的线性地址空间（0～4GB）。Linux 采用三级分页模式而不是两级。如图2.28 所示为三级分页模式，为此，Linux定义了3 种类型的表。

• 总目录PGD（Page Global Directory）

• 中间目录PMD（Page Middle Derectory）

• 页表PT（Page Table）

