Windows内核学习

作者：Delort

截图来源：

- Intel 白皮书
- 《x86/x64体系探索及编程》

1.保护模式

先上几张图镇镇楼：

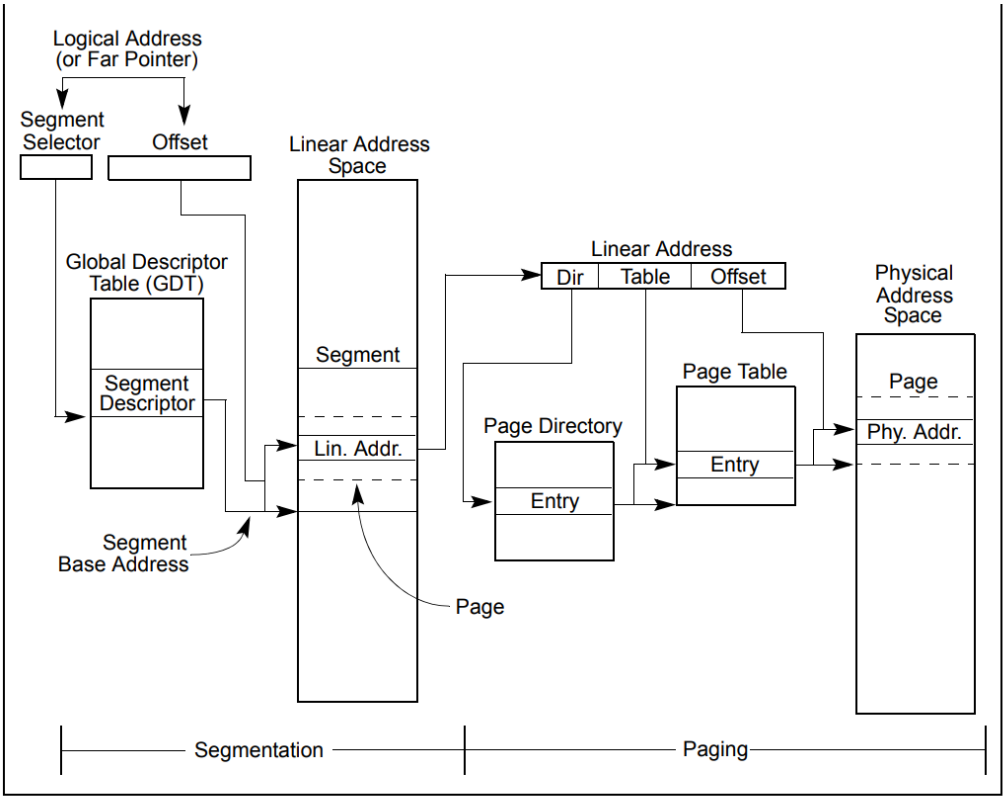


Figure 3-1. Segmentation and Paging

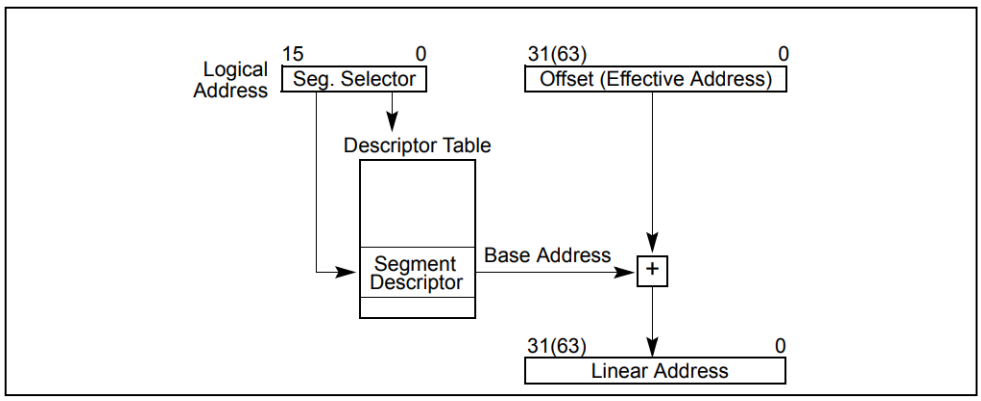


Figure 3-5. Logical Address to Linear Address Translation

1.1 段选择子

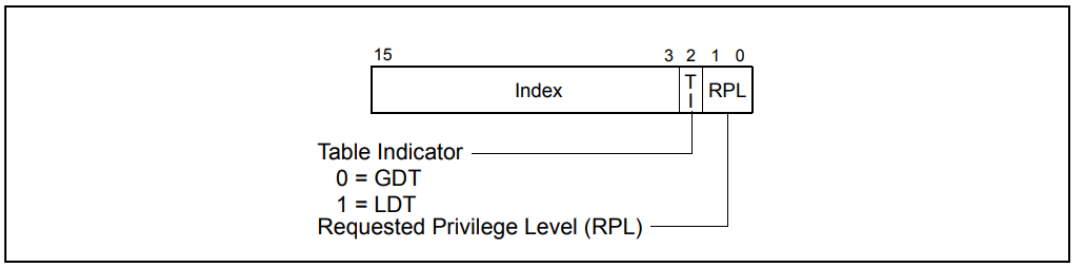
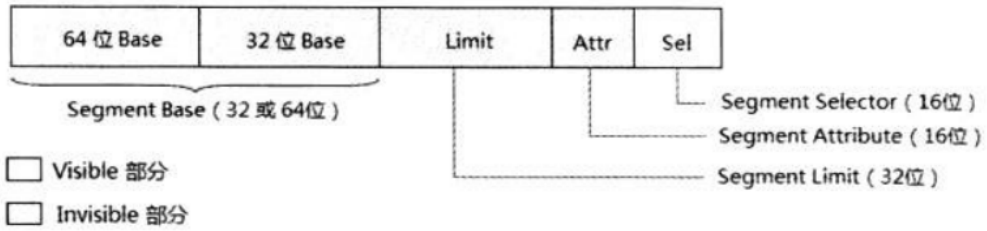


Figure 3-6. Segment Selector



段寄存器存储了段选择子，段选择子有以下三个内容：

- 在段描述表里的索引，具体计算时为index*8+Base为该描述符

  可以使用windbg 命令dg查询该描述符细节。（GDT 的0项为无效被称为NULL Seclector）
- TI位：0标志着使用全局段描述符表（GDT），1描述着使用局部描述符表（LDT）
- RPL：请求权限级别。它表示发起请求的访问者的权限，访问者可以有不同的RPL。

这里提到RPL，有必要说说CPL，RPL，DPL的区别：

RPL上文说了，是请求者的权限。

CPL是当前权限级别，表示运行的代码在哪个权限级别里，CPL存放在CS和SS的RPL中。当段选择子被加载进段寄存器时，CS.RPL和RPL代表当前CPL值。

DPL是描述符权限级别，存放在段描述符里和门(Gate)描述符里面的DPL中。注意：***门描述符中的DPL值只是访问Gate的权限，并不代表Gate所引用的段的权限。***

1.2 执行一条汇编指令的检查过程

```
1 mov eax,es:[0x00004000]
```

- 选择子的检查，查看es寄存器的值是否是NULL，若是发生违例异常。

  在之前es需要加载，es要么是一个最低要求的只读段，要么是个NULL Selector。
- 进行段limit的检查，假如es.limit>=0x00004000+3，检查通过，否则违例异常。

  此处假设es是向上扩展的数据段。即Expand-up。

  若在向下扩展的数据段中，limit+1是最小偏移量，1是一个字节。此时es.limit+1<=0x00004000是合法的。

  这里加上3是因为访问的是DWORD类型的数据。
- 接着访问地址es.Base+0x00004000。

1.3 段式内存管理

线性地址=Base+Offset	Base	Offset
		在段limit范围内（在IA-32 e模式下即64位模式下，不检查limit）
实模式	Base=selector<<4	
保护模式	Base从段描述符的Base域加载	
64位模式	Base为0，所有limit都是0xFFFF_FFFFH	

在同一个时刻最多只有6个段处于active状态，即为可用状态。被段选择子寄存器所使用。当所有段的Base=0，limit=0xFFFF_FFFFH时被称为flat mode(平坦的模式)的内存管理模式。

在访问一个段时保护模式会进行以下段检查：

- 段limit检查
- 段Type检查
- 段privilege检查。即权限检查。

其中，在64位模式下，不进行limit检查。权限检查最为复杂。

1.4 NULL Selector

以下类型代码发生段选择子加载：

```
mov ax,0x8
mov ds,ax
```

当描述符的权限允许以及类型相符时，处理器加载selector到ds寄存器，同时那个段描述符也会加载到ds寄存器的Hidden部分也就是Cache部分。（这就是为啥那么多人说段选择子寄存器是96位）。

备注：***NULL Selector不被允许加载到CS或者SS中，若尝试产生#GP异常，但是可以加载到ES，DS，FS，以及GS中。但是使用时同样产生#GP异常。***

加载NULL Selector也并不是加载GDT的第0项，而是加载一个除S标志为1外全是0的段描述符。

64位模式下：

处理器并不对NULL Selector进行检查，允许加载其到除去CS寄存器外任何一个段寄存器（SS有限制），***以及可以使用NULL Selector进行访问***。

SS的限制：

在R0-R2允许，R3不允许，会发生#GP异常。

有时候会隐式加载NULL Selector：

- 从高权限转到低权限（RETF和IRET时）：ES，DS，FS，GS的DPL<CPL，此时这些段寄存器会加载NULL Selector。

  剩下俩种是Long Mode模式下（64位模式和compatibility模式）

- 使用call gate，从低权限切换到高权限时，SS加载NULL Selector，其RPL会被设为新CPL
- INT中断调用（或异常、中断）从低权限到高权限，SS加载NULL Selector，其RPL被设为新CPL值

1.5 段描述符表

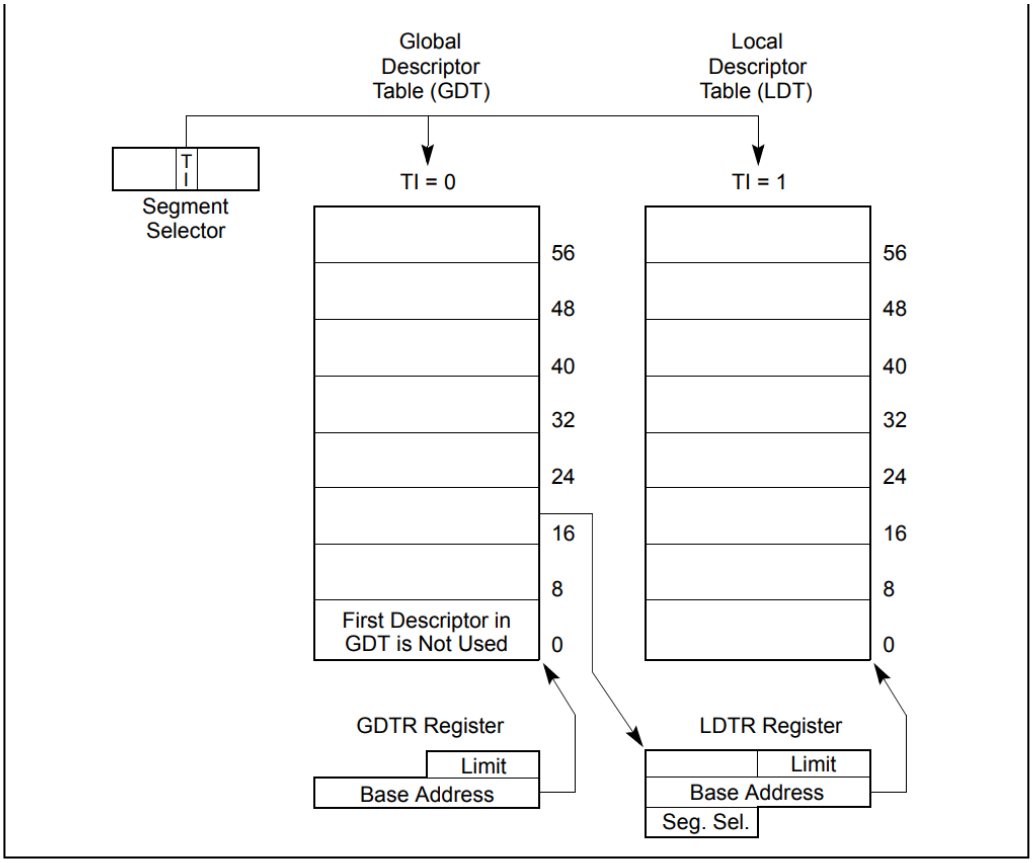


Figure 3-10. Global and Local Descriptor Tables

在x86有三种描述符表：

- GDT——全局描述符表
- LDT——局部描述符表
- IDT——中断描述符表

分别三个寄存器与之对应：

- GDTR

  lgdt加载内存操作数到GDTR
- LDTR
- IDTR

分别有limit和Base。

limit提供表限，Base提供基地址。

举例说明limit的作用：若GDTR.limit=0x3FFH，则GDT的有效范围是0-0x3FFH。

limit是16位，Base为32位，Base在long mode模式下为64位

1.6 段描述符

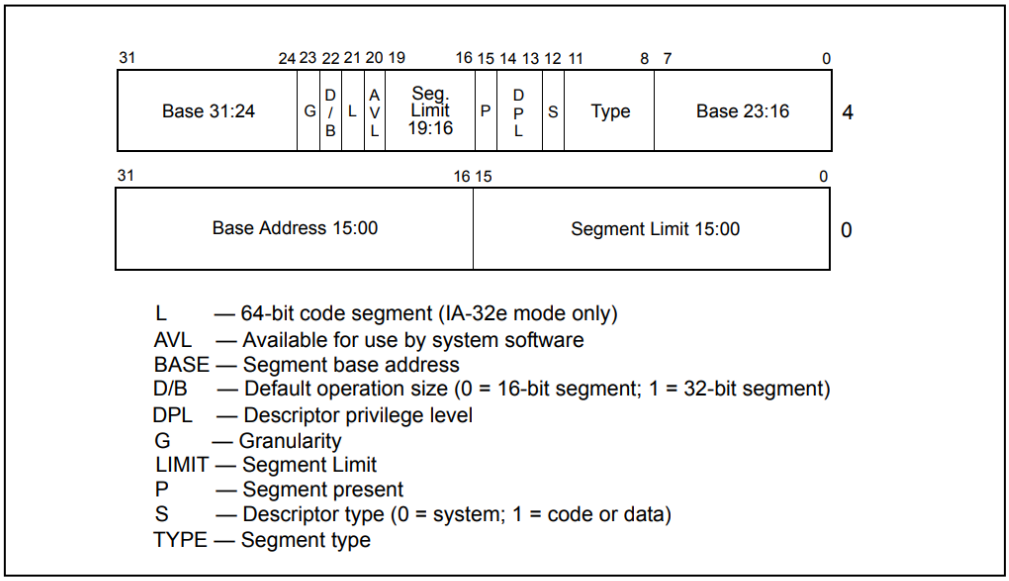


Figure 3-8. Segment Descriptor

S位表示该描述符的类型。

在 legacy 模式下，每个描述符是 8 字节 64 位宽，在 long mode（包括 compatibility 模式）下，所有的 gate 描述符是 16 字节 128 位宽，而 Code/Data 段描述符依然是 8 字节宽。

LDT/TSS 描述符在 64 位模式下是 16 字节 128 位宽，而在 compatibility 模式下依然是 8 字节 64 位宽。

Table 3-1. Code- and Data-Segment Types

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

描述符离不开段寄存器，段寄存器离不开描述符。

TSS 描述符比较特殊，被加载时会变成BUSY状态，不能在已加载状态下再加载。

一个段描述符被加载进段寄存器后，变成active状态。

种类：

- 段描述符

- 门描述符

按照系统性质来分可分为：

- 系统描述符：
  - 系统段描述符：LDT的描述符，TSS描述符
  - 门描述符：call gate，interrupt gate，trap gate，task-gate 描述符

3.5 SYSTEM DESCRIPTOR TYPES

When the S (descriptor type) flag in a segment descriptor is clear, the descriptor type is a system descriptor. The processor recognizes the following types of system descriptors:

- Local descriptor-table (LDT) segment descriptor.
- Task-state segment (TSS) descriptor.
- Call-gate descriptor.
- Interrupt-gate descriptor.
- Trap-gate descriptor.
- Task-gate descriptor.

- Code/Data描述符

Table 3-2. System-Segment and Gate-Descriptor Types

Type Field					Description	
Decimal	11	10	9	8	32-Bit Mode	IA-32e Mode
0	0	0	0	0	Reserved	Upper 8 bytes of an 16-byte descriptor
1	0	0	0	1	16-bit TSS (Available)	Reserved
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-bit TSS (Busy)	Reserved
4	0	1	0	0	16-bit Call Gate	Reserved
5	0	1	0	1	Task Gate	Reserved
6	0	1	1	0	16-bit Interrupt Gate	Reserved
7	0	1	1	1	16-bit Trap Gate	Reserved
8	1	0	0	0	Reserved	Reserved
9	1	0	0	1	32-bit TSS (Available)	64-bit TSS (Available)
10	1	0	1	0	Reserved	Reserved
11	1	0	1	1	32-bit TSS (Busy)	64-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate	64-bit Call Gate
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	32-bit Interrupt Gate	64-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate	64-bit Trap Gate

1.6.1 代码段描述符

代码段描述符一致性和非一致性解释：

基于这种要求，我们来对比一下 conforming 段与 non-conforming 段。

① 使用 conforming 段，并将 DPL 设为 0 级权限，在 3 级权限下可以直接调用（CPL > DPL），在 0 级权限下，依然可以使用直接调用（CPL == DPL）。

② 使用 non-conforming 段，并将 DPL 设为 3 级权限，在 3 级权限下可以直接调用（CPL == DPL），而在其他级别无法直接调用，例如在 0 级不能直接调用 3 级权限的代码（CPL != DPL），那么在 0 级权限使用 gate 符进行调用呢？同样做不到（条件是：CPL >= DPL of Code segment）。如果将 non-conforming 段的 DPL 设为 0 级权限，在 3 级权限下可以使用 gate 符进行调用，在 0 级权限下也可以使用 gate 符进行调用。

相比之下，non-conforming 段的执行权限需要被定义为 0 级，通过 gate 符进行调用，显得不如 conforming 段灵活，并且 conforming 段定义在 3 级权限，不会改变调用者的 CPL 值。对于不重要的库 routine 来说，使用 conforming 段会更适合些。

代码段accessed解释：

Accessed 访问标志

在 type 域里的 A 标志（accessed）指示段是否被访问过，A=1 表示已经被访问过（被加载到段寄存器中），A=0 表示未访问。

当段描述符被加载到段寄存器时，只有当 A 标志位为 0 时，处理器才会对在 GDT/LDT 中的 segment descriptor 中的 A 标志进行置位，这种行为可以让系统管理软件（典型的是内存管理软件）知道哪个段已经被访问过。

可是一旦置位，处理器从不会对 A 标志位进行清位。系统软件在对 descriptor 进行重新设置的时候，可以对 A 标志位进行清位。在处理器再次加载 descriptor 的时候对 A 标志位重新置位，在这种情况下，A 标志往往配合 P 标志位使用。系统软件在对 A 标志和 P 标志位进行修改的时候应当使用 LOCK 指令前缀锁 bus cycle。

当处理器加载 descriptor 到段寄存器时，处理器会对 descriptor 执行自动加 lock 的行为，处理器在访问这个 descriptor 期间，其他处理器不能修改这个 descriptor。

这个加载 descriptor 期间，应当包括从对 descriptor 检查到最后的使用 descriptor 更新段寄存器内的 Cache 部分。