2021/04/29_Windows32位内核_第7课_获取GDT、LDT、页表

笔记本： Windows32位内核
创建时间： 2021/4/29 星期四 15:00
作者： ileemi

段描述表

描述符表有两种类型：

1. 全局描述符表(GDT)
2. 本地描述符表(LDT)

段基址：32位（分为三段）
段界限：20位（分为两段）
属性：12位

段描述表各个属性的作用：

- G（粒度标志 1bit）：G = 0，单位：1字节；G = 1，单位：4kb
  段大小 = limit * 1;
  段大小 = limit * 4KB + 0xFFF; (2 ^ 20 - 1) * 4096 + 0xFFF, 1 * 0x1000 + 0xFFF = 0x1FFF;
- D/B（1bit）：默认操作数大小，0（16位的段），1（32位的段）
- L（1bit）：段是否为64位，在32位CPU中该字段没有意义
- AVL（1bit）：系统软件可利用位
- P：段是否存在位，解析段描述表是否为有效可首先解析这个字节。1（有效），0（无效），删除表项时，可将该为置0。
- DPL（2bit）：描述符特权级（Ring0~Ring3），描述当前段在哪环可以使用。
  CPL：当前特权级。
- S（1bit）：描述符类型，0（系统段），1（代码或者数据段）
- TYPE（4bit）：段类型，描述内存属性
  0 -- 只读数据段
  1 -- 只读数据段，已访问（CPU访问了这个段的内存，会设置的一个标志）
  2 -- 读/写
  3 -- 读/写，已访问
  ...
  0~7：数据相关

8~15：代码相关

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

设置可读可写可执行：
0~1200 R/W
0~1200 E

段描述表项的解析：

```
kd> dq gdtr
8003f000  00000000`00000000 00cf9b00`0000ffff
8003f010  00cf9300`0000ffff 00cffb00`0000ffff
8003f020  00cff300`0000ffff 80008b04`200020ab
8003f030  ffc093df`f0000001 0040f300`00000fff
8003f040  0000f200`0400ffff 00000000`00000000
8003f050  80008955`22000068 80008955`22680068
8003f060  00009302`2f40ffff 0000920b`80003fff
8003f070  ff0092ff`700003ff 80009a40`0000ffff
```

无效项

解析下标第1项：00cf9b00 - 0000ffff
Base：0000 00 00
Limit：FFFF F
属性：CF9B
1100 FFFF 1001 1011
G = 1，段的范围：0x00000000~0xFFFFFFFF
D/B = 1
L = 0
AVL = 0
P = 1
DPL = 00
S = 1
Type = 1011 -- Execute/Read, accessed // 可以执行一般为代码段

段描述表的地址需要告诉CPU，将其地址写入到GDTR寄存器中。

全局描述符表 GDT

> 在整个系统中，全局描述符表GDT只有一张(一个处理器对应一个GDT)，GDT可以被放在内存的任何位置，但CPU必须知道GDT的入口，也就是基地址放在哪里，Intel的设计者门提供了一个寄存器GDTR用来存放GDT的入口地址，GDT设定在内存中某个位置之后，可以通过LGDT指令将GDT的入口地址装入此寄存器，从此以后，CPU就根据此寄存器中的内容作为GDT的入口来访问GDT了。GDTR中存放的是GDT在内存中的基地址和其表长界限。

GDTR（48位）：全局描述符表，指向共享内存。32Bit Base + 16Limit，最大 2 ^ 16 / 8 = 8192 项。

局部描述符表 LDT

> 局部描述符表可以有若干张，指向不共享的数据内存，每个任务都可以有一张。我们可以这样理解GDT和LDT：GDT为一级描述符表，LDT为二级描述符表。在Windows中用寄存器 "LDTR（48位）" 表示。

内存隔离的实现就是为每个进程申请一个描述符表。

每个系统必须定义一个GDT，可用于系统中的所有程序和任务。也可以选择定义一个或多个LDT。例如，可以为正在运行的每个单独任务定义一个LDT，或者部分或所有任务可以共享同一个LDT。

段描述符表是一个段描述符数组：

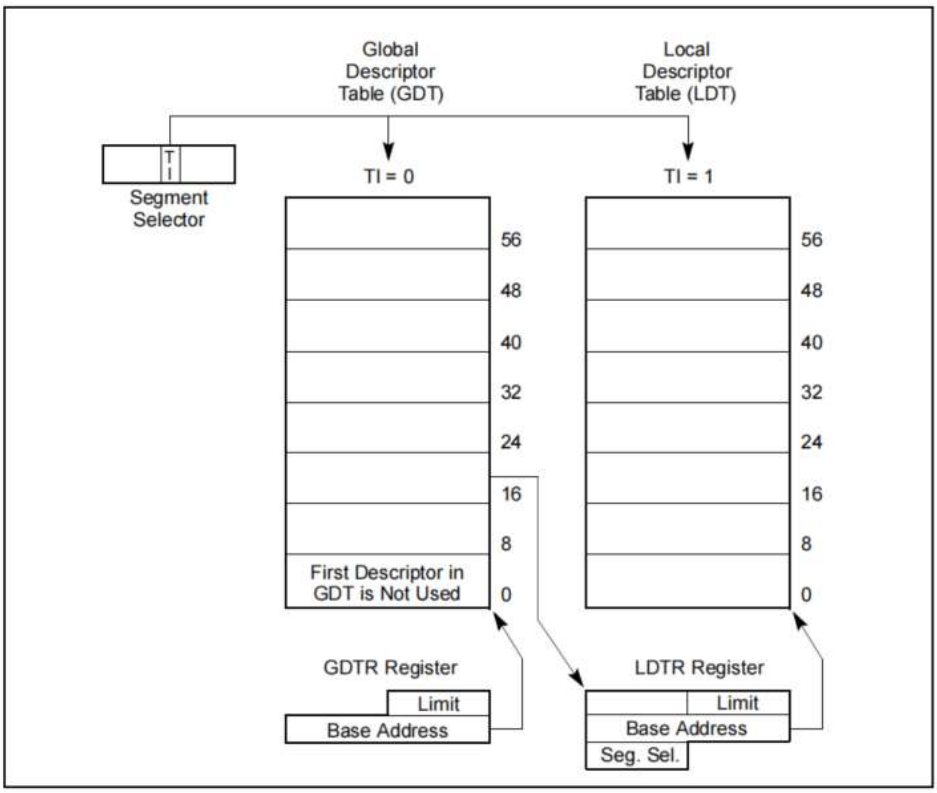


Figure 3-10. Global and Local Descriptor Tables

段选择子

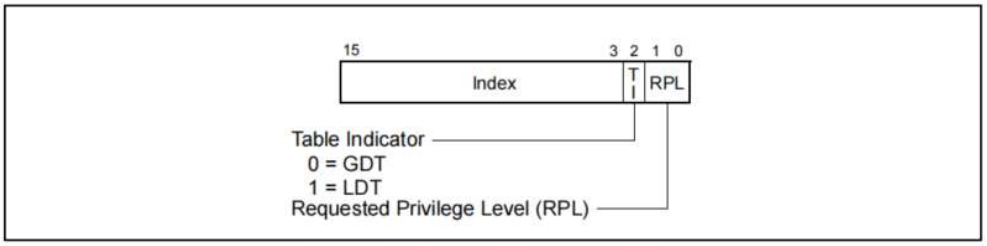


Figure 3-6. Segment Selector

> 由GDTR访问全局描述符表是通过“段选择子”（实模式下的段寄存器）来完成的。段选择子是一个16位的寄存器（同实模式下的段寄存器相同）。

在32位CPU中，所有的段寄存器都是 "段选择子"。
RPL：请求的特权级

FS 0x30

> Index(13bit) TI(1bit) RPL(2bit)
> 0000 0000 0011 0000
> Index：0000 0000 0011 0 -- 6
> TI：0（GDT），1（IDT）
> RPL：00

2 ^ 13 = 8192
对上面的段选择子进行拆分，得出要查询的表在GDTR表的第6项，"ffc093df - f0000001"。

```
kd> dg 30
                                    P Si Gr Pr Lo
Sel    Base     Limit     Type    l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0030 ffdff000 00001fff Data RW Ac 0 Bg Pg P  Nl 00000c93
```

mov eax, fs:[1000]

首先判断请求的特权级是否有执行的权限，接着判断偏移值是否在GDTR表的范围内，在范围中，CPU就检查其内存属性（可读、可写、可执行等）。
offset：0x1000，其在GDTR表范围中（ffdff000 ~ ffdff000 + 00001fff）。

满足上述条件，物理内存地址 = ffdff000 + 0x1000，CPU将计算出的物理内存地址的值赋值给 eax。在GDTR表中，零地址会出现 0xC00000005错误的原因就再于这个表项是无效的，可通过修改标志属性，使CPU承认这个表项。DPL的值从0修改为3，三环的程序就可以访问内核空间。

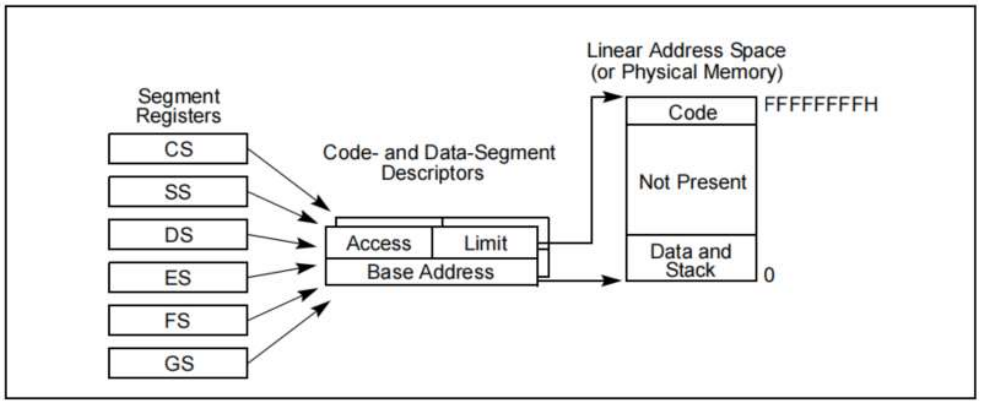
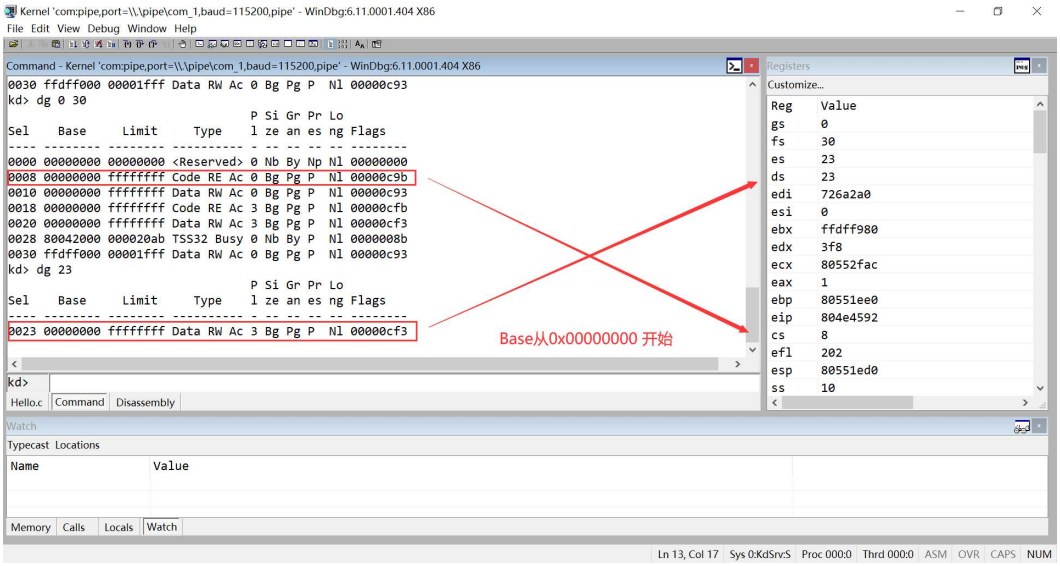


Figure 3-2. Flat Model

GDTR 表项的值应该是物理地址，但是在时机情况中确实虚拟地址。在16位的xp系统中，只使用了GDTR表，LDTR值为空，也就是所有进程内存共享。

Windows的设置中并没有使用 GDTR、LDTR表（不采用分段管理），不使用CPU使不承认的。从下图中可以看出，操作系统给Ring0，Ring3的内存属性都是可读、可写、可执行的。段选择子基址的起始地址大部分都是0x00000000（**FS段选择子正常**），所以，就导致GDTR、LDTR表没有作用，原偏移地址直接当作物理地址使用。

mov eax, CS:[00401000] --> 00401000
mov eax, DS:[00402000] --> 00402000



段选择子的修改只能在Ring0中进行修改，在Ring3中修改没有权限。在Windows中所有进程的段选择子（段寄存器）的值都是一致的。

```
GS 002B   FS 0053
ES 002B   DS 002B
CS 0023   SS 002B
```

> Ring0代码段、堆栈段、FS与Ring3不一致。在 Windows 操作系统中，虚拟地址等价于线性地址（FS、GS除外）。

加载描述表：lgst、lldt
存储描述表：sgdt、sldt

Operation

```
IF instruction is SGDT
        IF OperandSize =16 or OperandSize = 32 (* Legacy or Compatibility Mode *)
            THEN
                DEST[0:15] := GDTR(Limit);
                DEST[16:47] := GDTR(Base); (* Full 32-bit base address stored *)
                FI;
            ELSE (* 64-bit Mode *)
                DEST[0:15] := GDTR(Limit);
                DEST[16:79] := GDTR(Base); (* Full 64-bit base address stored *)
        FI;
FI;
```

对于多核CPU，其有对应核心数量的GDT表。在内核中可以通过 "KeGetCurrentProcessNumber" 获取CPU的核心数量。通过 "KeSetSystemAffinityThread" 可以将指定的代码在指定的核心上运行。

可以在Ring3中获取GDT表的首地址，将该地址传递给驱动，让驱动读取对应的数据，返还给Ring3程序。

逻辑地址到线性地址的转换

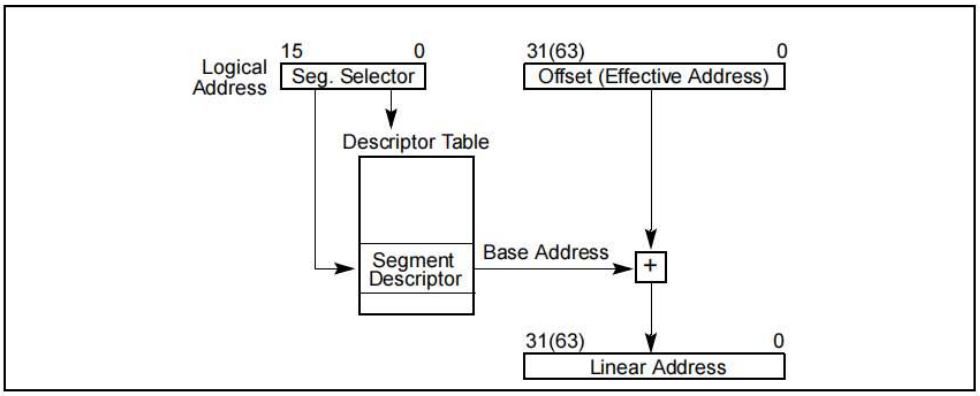


Figure 3-5. Logical Address to Linear Address Translation

1. 使用段选择器中的偏移量来定位GDT或LDT中的段的段描述符，并将其读入处理器。（只有当将新的段选择器加载到段寄存器中时，才需要执行此步骤。）
2. 检查段描述符，检查段的访问权限和范围，确保段可访问，偏移在段的范围内。
3. 将段的基本地址从段描述符添加到偏移量中，以形成一个线性地址。

页表

每个分页模式如何操作的详细信息由以下控制位决定：

- CR0中的WP标志（第16位）
- CR4中的PSE、PGE、PSE、PCIDE、SMEP、SMAP、PKE、CET和CKS标志（分别为位4、位7、位7、位17、位20、位21、位22、位23和位24）
- IA32_EFERMSR中的NXE标志（第11位）

使用32位页表将线性地址转换为4k字节的页表（线性地址到物理地址的转换）：

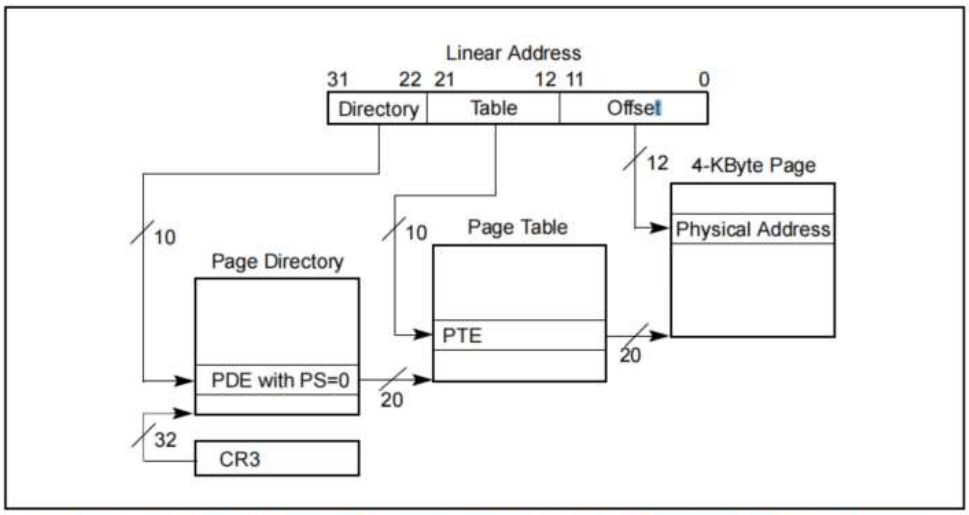


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging