北京科海培训中心

Windows 深入剖析

内核篇

杨 亮
魏晋鹏 万玉丹 编著

清华大学出版社

北京科海培训中心

Windows 深入剖析——内核篇

杨亮　万玉丹　魏晋鹏　编著

清华大学出版社

(京)新登字 158 号

内 容 提 要

本书向读者提示了 Windows 及其应用程序的运行过程，深入剖析了构成 Windows 内核实体的几个最主要的方面，其中包括：内存的分配和管理，模块的装载和卸载，任务的启动和终止，动态链接机制，任务调度机制，消息驱动机制，同时对 Windows 内核的启动和终止过程也进行了比较全面的剖析。

本书与姊妹篇《Windows 深入剖析——初始化篇》构成了一个整体，帮助读者完整透彻地了解 Windows 内部奥秘。

本书面向计算机系统开发人员，也可供高等院校相关专业的师生阅读参考。

书　名：Windows 深入剖析——内核篇
作　者：杨亮　万玉丹　魏晋鹏
出版者：清华大学出版社(北京清华大学校内，邮编 100084)
印刷者：北京门头沟胶印厂
发　行：新华书店总店北京科技发行所
开　本：16　印张：31.75　字数：772 千字
版　次：1997 年 8 月第 1 版　　1997 年 8 月第 1 次印刷
印　数：0001～5000
书　号：ISBN 7-302-02677-7/TP・1382
定　价：45.00 元

绪 论

勿庸置疑，Microsoft 的 Windows 系列是 PC 机上真正的主流操作系统，从现在直到将来，Windows 都将是我们的主要工作和开发平台。虽然随着 Windows 95 的推出，Windows 3.1 在微软公司的眼里已不再是主要的摇钱树，但根据国际数据集团(IDG)1996 年 9 月份的最新报告，Windows 95 的销售情况远远没有预计的那么好，而相反 Windows 3.X 仍然保持着良好的销售势头，尤其是一些大公司仍然对 Windows 95 持观望态度，他们宁愿守着 Windows 3.X 或 OS/2 或 Windows NT 这些他们看起来更放心一点的地盘，而不愿轻易地转移系统平台，软件升级的幌子对他们的吸引力似乎并不大。

更重要的一点是 Windows 3.X 的累计销售量要比 Windows 95 多得多，运行在其上的应用软件数目也比视窗 95 上的应用软件多得多。对于习惯了黑白 DOS 屏幕的广大中国用户来说，虽然内存价格暴跌，但似乎并没有改变电脑公司里所出售的计算机的内存配置(名牌机尤其如此)。8M 内存如果选系统平台的话，Windows 3.X+DOS 6.2 绝对要比 Windows 95 好——“等待的滋味就像喝了一杯冰冷的水”，就像 4M 内存下运行 Windows 应用程序(如 WORD 6.0)，那种感觉相信大家都有过。更何况 Windows 97 已经是呼之欲出了，赶时髦的角色在计算机界也许更应该由一些发烧友之类的人来承担，而不必也不会去拖累那些老老实实搞开发的人。一个成熟的操作系统的寿命是很长的，正如许多人当初预言 DOS 不会随着 Windows 的推出而退出一样，Windows 3.X(包括 Windows 3.1 和 Windows Work Group 3.11)同样会在相当长的时间内继续扮演着一个相当重要的角色。举一个有趣的例子，Windows 95 刚刚上市时，各种计算机专业媒体上介绍如何安装和使用 Windows 95 的文章比比皆是，并且其中不乏好心肠的人告诉读者如何在一台计算机里面同时安装 Windows 95 和 DOS/Windows 3.1。但一段时间过后这种文章似乎被如何彻底删除计算机中的 Windows 95 的经验之谈所取代。作者也有过这种经历，总觉得 Windows 95 用起来没有 DOS+Windows 3.1 方便，看来惰性的力量也是无穷的！说了这么多，权作我们把分析的对象定在 Windows 3.1 的借口吧。

Windows 的风靡是从 3.0 版开始的，更加成熟和稳定的 3.1 版不过是锦上添花而已。凡是使用过 Windows 的人，无不被她的漂亮外表——图形用户界面(GUI)所吸引。不过，Windows 倒绝不是个“金玉其外，败絮其内”的骗子。当你揭开蒙在 Windows 内核上的面纱时，相信你也会由衷地发出“里面的世界同样很精彩”的感叹。也许有人会问，在 Windows 下进行编程时，有必要对 Windows 内核进行这么深入的了解吗？知道 Windows 的内部秘密是否能够减轻一些调试时的痛苦呢？还有人会说，“知其然而不知其所以然”的黑箱理论以及所谓封装、信息隐蔽的观点不是一向被奉为经典吗？是的，我们应该承认这些都是很正确的。如果没有隐蔽，任何事情都是公开的话，那么我们将变得手足无措。但是，正如 Jeff Duntemamn 在他的一篇题为“The Tragedy Of The Black Box”的文章中所谈到的，隐蔽系统内部工作需要付出代价，“由于对系统隐蔽部分缺乏了解，因此妨碍了对系统可见部分的理解”。Jeff 认为，当黑箱不完全封闭时，更是这样，你不得不同系统内部打交道。Windows 正是如此，其实

在Windows编程时涉及未公开的地方彼彼皆是。例如:HWND指向什么,WinExec()返回什么,当输入一个汉字时,Windows将会发送什么消息等等。尽管我们可以对此视而不见,但还是应该承认,不弄清楚,总觉得有点不舒服。如果我们一开始就对Windows的底层有比较透彻的了解,那么对于开发Windows下的优质无错程式将有很大裨益。因为毕竟,"知道,然后忘却"与"根本不知道"有天壤之别。

此外,正如在DOS下仅满足于INT 21调用是难以编写出精湛的程序来的一样,在Windows下也需要对其内部原理有深刻的认识。如同DOS下需要有TSR,有时候,利用未公开的Windows调用可以完成一些常规方法难以做到的工作(如实时多任务等)。实际上,只要我们长期植根于一个操作系统,几乎可以肯定会产生深入了解其原理的需求;在UNIX和DOS的时代我们就已经看到了这一规律,在今天Windows的时代也不例外。

由于这种需要,也为了促进我国的操作系统研究与国产化,我们以Windows 3.1为蓝本,进行了深入的、然而也是异常艰辛的剖析,本书及其姊妹篇可以说是我们长期工作的总结。

本书的姊妹篇:《Windows的深入剖析——初始化篇》探讨了WIN386.EXE(普通版)的初始化程序,包括实模式下的初始化和保护模式下的初始化,到加载执行KRNL386.EXE(即内核)为止(在本书中,将接着讨论内核部分的初始化)。此外,在姊妹篇中还着重研究了VMM(虚拟机管理器)的工作过程和有关问题。

本书与姊妹篇及笔者以后的著作构成了一个整体,帮助读者完整、透彻地了解Windows的内部奥秘。同时,本书在内容上又可以独立成篇。

本书介绍了构成Windows内核运行实体的几个最主要的方面,如内存的分配和管理、模块的装载和卸载、任务的启动和终止、动态链接机制、任务调度机制、消息驱动机制,同时对Windows内核的启动和终止过程也进行了比较全面的剖析,本书的重点是分析Windows内核(KERNEL,对应的Windows文件为KRNL386.EXE),至于USER.EXE和GDI.EXE则较少涉及(唯一的例外是消息驱动这一章,混合在KERNEL和USER之中,因此也牵涉到了少量USER的知识,如窗口句柄、屏幕重绘等等)。这是因为虽然USER和GDI与Windows编程关系密切,但它们规模庞大、内容复杂,受篇幅所限,我们将在以后的著作中专门论述。

值得指出的是,本书的目的不在于直接告诉读者如何去写Windows程序,而在于向读者揭示Windows及其应用程序的运行过程,从而帮助读者在头脑中形成Windows系统的一幅完整的立体图景。相信如果把这些问题都弄清楚了,读者对于编程的理解也将会达到前所未有的深度,编写高质量的程序也一定不在话下。因此,本书的风格可以说完全不同于Windows编程参考书,那些希望在本书中发现一些编程秘诀的读者也许会失望的。不过,本书是一本帮助读者打基础的书,随着时间的流逝,将会潜移默化地起作用。此外,本书还为那些特殊编程(如底层编程,以及那些对时间要求较高的编程等)提供了开启大门的钥匙。

需说明的是,本书选取蓝本的方式与姊妹篇稍微不同:本书中研究的是KRNL386.EXE的调试版,这可以在Windows SDK中找到:安装好Windows 3.1 SDK后,会创建一个"WINDEV"目录,在它之下有一个"DEBUG"子目录,其中包含一个名为KRNL386.EXE的文件,它就是调试版。

用它覆盖Windows的KRNL386.EXE,即可得到与本书相同的研究环境。

调试版比普通版包含有更多的符号信息和提示信息，因而更适合于研究 Windows 。由于调试版只是在普通版的基础上插入了一些调试指令(普通版中的各条指令在调试版中全有，只是位置靠后一些，并间杂一些与调试有关的指令，如 INT 3 或 CALL OutDebugInfo 等)，因此如果您手头只有普通版，也不难与本书中的源码相对照。

本书在注释源程序的过程中，为了帮助读者阅读，用到了一些自创的符号，这里作一个简要的说明。下面就是在本书的源程序中经常会看到的情况：

```
3D95→
0117:00003D92  CALL     MoveBlockNearBy(3606)
```

其中，“3D95→”表示从 0117:00003D95 处转到这里执行，这样便于读者追溯各条指令、各个分支的“源头”，不致在指令的海洋中迷失方向。有时还有像“3D42，3D5A，3DA6，3DB3→”的情况，这表示可以从 3D42 等多处转移至此，因此这里相当于是多个分支的汇合部。

而“CALL MoveBlockNearBy”则表示调用了一个有符号名字的函数或过程。带括号的部份“(3606)”指明了该函数或过程的入口地址，而且还附了其源程序，读者如果想了解其工作过程的细节，可以按地址顺序查找到这段程序(本书各章所附的源程序都是按地址顺序编排的)。如果没有带括号的部份，则意味着本书未附录该函数或过程的源程序，读者只需知道其作用就行了。

本书中对非 Windows 标准函数的命名规则是尽可能根据 Windows 内部提供的调试或帮助信息来确定的，对于一些函数的内部过程则是作者根据该过程的作用而构造的，Windows 并没有给它们取名。

在本书的写作过程中，得到了许多老师和朋友的关心和帮助，在此作者谨向他们致以深深的谢意，尤其感谢黄俊杰教授以及杨旺、杨盛、廖宁、林汉生等诸多朋友的爱护与鼓励，并将此书献给我们的父母，他们永远是天底下最值得尊敬和尊重的人！

当写完这最后几行字的时候，我不禁回想起两年前开始分析 Windows 的那些日子。七百多个日日夜夜，其中所忍受的艰辛与苦难现在似乎都已成过眼烟云。在这段艰难的日子里，我们不仅学到了很多，也懂得了很多。这本书记载着逝去的岁月，记载着我们的欢乐与苦闷，记载着我们几个人的深厚友情，记载着我们在一起互相切磋讨论的一幕一幕。她所带给我们的不仅仅是收获，还有许多许多美好的回忆。“一蓑烟雨任平生”，愿这种回忆永存心底！

目 录

第1章 基础知识

Windows下涉及的保护模式知识很多也很杂，考虑到本书的重点是讲述Windows内部的具体运行机制，因此本章不打算花费太多的篇幅来阐述保护模式的各个细节，而只是将本书中用到的一些保护模式基础知识作一个扼要的说明。如果读者对保护模式没有太深的理解，也没有关系，因为Windows本身就是观摹保护模式的最好范例。读者应先对本章所列的几个保护模式要点有一个基本的了解，然后再在阅读程序的过程中逐步加深这种认识。透彻地掌握一种操作系统运行机制毕竟非一朝一夕之功，多积累一些实践经验以后，自然就会水到渠成的。

1.1 保护模式概述

1.1.1 保护模式的渊源

对于熟悉了DOS编程的计算机工作者来说，保护模式多少显得有些神秘、难以把握。虽然有许多资料(包括Intel的原版资料)可供参考，但仍不容易理解保护模式的精髓。

“保护模式”一词可说是由来已久，早远于Windows。事实上，从Intel 80286开始就有了实模式和保护模式之分，基于PC/AT的ROM-BIOS中还专门提供了INT 15h/AH=89h功能，使得DOS应用程序可以切换到保护模式下。

但保护模式并未真正派上多少用场，因而人们也就很少去研究它；直到Windows流行起来后，这种情况才完全改观。Windows系列(包括Windows 3.1、Windows 95及Windows NT)主要是工作在保护模式下，从而真正发挥了高档CPU的潜力，相对于DOS，在多任务、寻址能力和图形界面等方面都取得了空前的进步。

1982年就已经随着80286而面世的保护模式，直到1990年随着Windows 3.0的推出才真正大行其道。由此可以看出软件技术相对于硬件技术具有一定的滞后性；另一方面也说明，虽然硬件技术的进步来之不易，但要利用好新的硬件特性则更难；尤其是像操作系统这样的系统软件，由于是一切软件的基础，它的成熟可靠比技术上先进更重要，因此向新技术的迁移也就显得更漫长一些，必须经过长时间的检验，臻于完善之后，才能为人们所接受。

1.1.2 什么是保护模式

关于保护模式的论述已经很多了，翻开一本讲保护模式或Windows的参考书，将会看到许多概念和图表，从分段、段描述符到分页、页表，乃至任务状态段(TSS)等，可谓是内容繁多。要真正深入了解保护模式，就必须先弄清这些基本概念。但对于不熟悉保护模式的人们来说，可能更需要对保护模式的概括性论述，让他们能一下子把握住保护模式的关键。那么，什么是保护模式呢？作者经过长期的深入研究，对于保护模式的理解是：

保护模式的核心是通过段页式寻址机制来实现多个任务之间的有效隔离。

关于保护模式的上述论述有两个要点：一是在段页式寻址机制下CPU能访问多达4GB的寻址空间，突破了实模式下640KB的限制，使得同时装入执行多个任务成为可能；二是分段机制使得段具有访问权限和特权级，操作系统的代码和数据不允许被应用程序访问，使得操作系统与应用程序有效隔离；不同的应用程序访问各自的段，应用程序之间也被隔离开来。因此，在Windows下操作系统和应用程序的执行都是受到有效保护的，保护模式的名称即由此而来。

1.2 分段机制和段描述符表

保护模式与实模式之间的最显著的区别也许应算是寻址方式上的不同了。虽然保护模式下也采用“段：偏移”的地址格式，但其中的“段”的含义却完全不同于实模式下的段；同样还是CS、DS、ES、SS寄存器，在实模式下是这些寄存器的值左移4位，再加上偏移，即得到物理地址；而在保护模式下，这些寄存器中的值为“段选择符”，需要查全局描述符表(GDT)或局部描述符表(LDT)，获得段的基址，再加上偏移，才能得到线性地址，如图1.1所示。

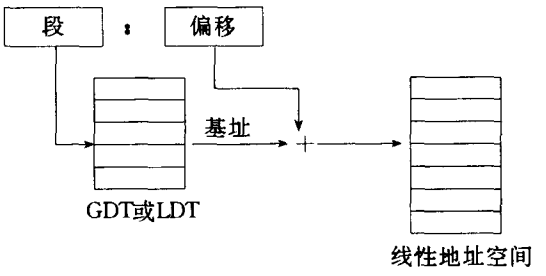

图1.1 保护模式的分段机制

由图1.1可见，在保护模式下，段选择符相当于查找GDT或LDT时的索引值。在Soft-ICE for Windows下，可以直接观察到GDT和LDT中存放的各个段描述符，下面即为在Windows处于稳定状态下时进入Soft-ICE所观察到的GDT和LDT(这里是从全部内容中摘取的少量片段)：

```
Break Due to Hot Key
:gdt
GDTbase=8007655C  Limit=010F
0008  Code16  Base=0000D470  Lim=0000FFFF  DPL=0  P  RE
0010  Data16  Base=0000D470  Lim=0000FFFF  DPL=0  P  RW
0018  TSS32   Base=8000DD74  Lim=00002069  DPL=0  P  B
0020  Data16  Base=8007655C  Lim=0000FFFF  DPL=0  P  RW
0028  Code32  Base=00000000  Lim=FFFFFFFF  DPL=0  P  RE
. . . . . .
007B  Data16  Base=00000522  Lim=00000100  DPL=3  P  RW
0083  Data32  Base=000D33A0  Lim=0000FFFF  DPL=3  P  RW
0088  LDT     Base=80640000  Lim=00001FFF  DPL=0  P
0090  Reserved Base=00000000  Lim=00000000  DPL=0  NP
. . . . . .
0100  Reserved Base=00000000  Lim=00000000  DPL=0  NP
0108  Reserved Base=00000000  Lim=00000000  DPL=0  NP
```

```
:
:
:ldt
LDTbase=80640000  Limit=1FFF
0004  Reserved Base=00000000  Lim=00000000  DPL=0  NP
000C  Reserved Base=00000000  Lim=00000000  DPL=0  NP
. . . . . .
00E7  Data16   Base=000869A0  Lim=FFFFFFFF  DPL=3  P   RW
00EC  TaskG32  Sel=0000       Off=000015E0  DPL=0  NP
00F7  Data16   Base=00083DF0  Lim=0000FFFE  DPL=3  P   RW
. . . . . .
1FF4  Reserved Base=00000000  Lim=00000000  DPL=0  NP
1FFC  Reserved Base=00000000  Lim=00000000  DPL=0  NP
```

在上面 Soft-ICE 所给出的信息中，每行对应一个段描述符，其中第一栏是其段选择符，如 GDT 中的 0018、007B，LDT 中的 0004、000C 等，Windows 及其应用程序正是通过将这些段选择符装入到 CS、DS 等寄存器来访问这些段的；第二栏是段描述符的类型，如 Data16 表示 16 位的数据段，Code32 表示 32 位的代码段等；第三栏是段的基址(32 位的线性地址)，第四栏是段的限长；第五栏是段描述符的特权级，虽然保护模式下特权级取值可以为 0 到 3，但 Windows 只用到了特权级 0(供操作系统使用)和特权级 3(供应用程序使用)，因此在这一栏中读者会看到只有 0 和 3 两种取值；第六栏的标志指示段是否存在于内存中，P 代表存在，NP 表示不存在，如果访问 NP 段，则会产生异常，导致从磁盘上调入段的内容；第七栏表示段的访问权限，比如是可读、可写的还是可执行的等。

段描述符表分为全局描述符表(GDT)和局部描述符表(LDT)，它们是包含段描述符项的两个特殊的内存段，由 WIN386 在 Windows 启动时创建。Soft-ICE 所报告的上述信息实际上都包括在段描述符中。每个段描述符占 8 字节，分为两大类：

- 存储段描述符(存储段是存放可由程序直接访问的代码和数据的段)；
- 系统段描述符或门描述符(系统段是 80386 分段机制所使用的特别段，门与中断、异常及陷阱对应)；

在上述信息中，Data16、Code32 等属于存储段，而 TSS32、TaskG32 等则属于系统段。那么这两种段如何区分开呢？Intel 用段描述符中的一个称为“DT”的位来标识，如果 DT=1，则表示是存储段描述符；如果 DT=0，则表示是系统段描述符或门描述符。

图 1.2 给出了存储段描述符的结构。其中 m 表示段描述符存放的起始位置，从 m 到 m+7 共 8 个字节；显然，m 应该为 8 的倍数。注意 DT 位(m+5 字节的 bit4)=1 表示这是一个存储段描述符。

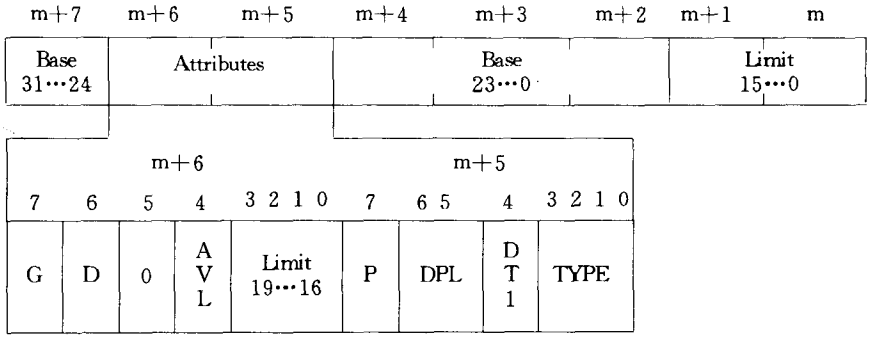

图 1.2 存储段描述符格式

图 1.2 中各个位的含义是：

Limit 存储段的限长，占 20 位。实际上，它与 G 位(下面将要讲述)一起决定了存储段中的偏移地址的范围。如果偏移地址超出该限长，则视为非法。

通常 G=0，这时段限长的最大值为(2^ 20-1)，即 FFFFFh。如果 G=1，则本字段的值以 4KB 为单位，这时实际的段限长的值为本字段的值 * 4KB，这样段限长的取值范围可以达到 2^ 20 * 4KB=4GB，即 32 位地址的寻址范围，从而弥补了 20 位表数范围的不足。注意 20 位的限长字段并不是连续存放的，其高 4 位颇令人费解地放到了 m+6 字节，这使得软件编程有些不便。

Base 存储段的基址，占 32 位，也就是说段的基址可以为 4GB(即 2 的 32 次方字节)寻址范围内的任何一个地址。比如在上面我们看到 GDT 中的 0020 段的基址为 8007655C。

须注意占 4 字节的 Base 字段并不是连续存放的，其最高字节被分隔开了，这是由 Intel 当初设计时决定的。

G G 位是限长字段的粒度属性位。G=0 表示限长字段的粒度为字节，G=1 表示限长字段的粒度为 4KB。应注意的是 G 位只对段限长的粒度起作用，而对段基址字段没有影响，即段基址总以字节为粒度。

D ① 可执行段使用 D 位，为段中的指令访问的地址或操作数设置默认值。

D=1 表示指令使用的默认值是 32 位地址及 32 位或 8 位操作数，这是 80386 以上机器的正常使用情况；

D=0 表示使用的默认值是 16 位的地址及 16 位或 8 位的操作数，与 286 系统兼容。

使用指令前缀，可把指令的地址或操作数的长度改变为与默认值相反的值。

② 向下扩展的段使用 D 位，决定段的上部边界。D=1 表示段的上部界限为 4GB；D=0 表示段的上部界限为 64K，这是为了与 286 兼容。

③ 由 SS 寄存器寻址的段，使用 D 位确定隐式的堆栈访问指令(如 PUSH 及 POP 指令)要使用 32 位的 ESP 寄存器(此时 D=1)，还是使用 16 位的 SP 寄存器(此时 D=0)。后者是考虑与 80286 兼容。

AVL　AVL位是软件可利用位。Microsoft在Windows中对该位进行了定义，AVL=0表示该描述符所对应的全局内存段是不可丢弃段，AVL=1表示是可丢弃段。

P　指示段描述符是否有效。P=1表示段描述符有效，可以用来实现对地址的转换；若P=0则表示段描述符无效，使用该描述符会引起异常。

DPL　描述符特权级，共2位，用于定义与段相联系的特权级。

DT　描述符类型位。区分描述符所指的段是存储段(DT=1)，还是系统段及门(DT=0)。

Type　类型字段，占4位，定义存储段描述符的类型，对应代码见表1.1。该表规定了段的访问权限，包括是否允许读、写、执行等；向下扩展表明是从后向前占用内存空间(就象堆栈段那样)，这时段限长规定的是段的前界(即偏移不能小于该值)而不是后界，后界为0FFFFFFFFh；“已访问”特性是用来跟踪存储段是否被用到过。

表1.1 存储段描述符类型

类型	说明	类型	说明
0	只读	8	只执行
1	只读，已访问	9	只执行，已访问
2	读/写	10	执行/读
3	读/写，已访问	11	执行/读，已访问
4	只读，向下扩展	12	只执行，一致码段
5	只读，向下扩展，已访问	13	只执行，一致码段，已访问
6	读/写，向下扩展	14	执行/读，一致码段
7	读/写，向下扩展，已访问	15	执行/读，一致码段，已访问

图1.3给出了系统段描述符的格式。其中m表示段描述符存放的起始位置，从m到m+7共8个字节。注意DT位(m+5字节的bit4)=0表示这是一个系统段描述符。

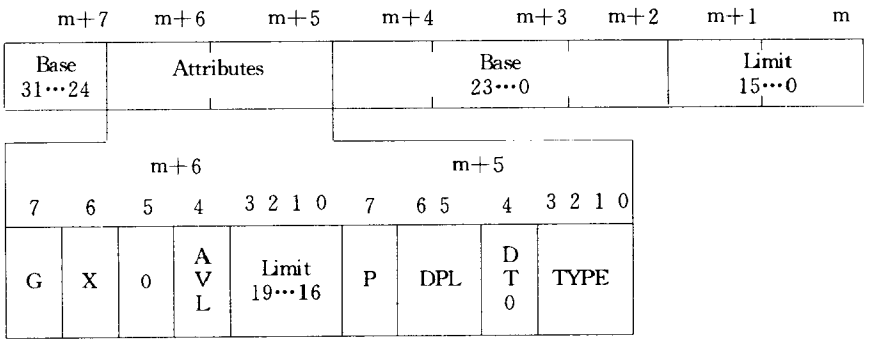

图1.3 系统段描述符格式

在图1.3中，G、P、DPL位的含义与存储段描述符相同，AVL仍为软件可利用位，X表示该位在系统段描述符中未用到。只有TYPE字段的含义不同，见表1.2。注意在该表中，系