保护模式内存管理

2.1 内存管理概览

IA-32 架构的内存管理功能分为两部分：分段和分页。分段提供了一种隔离各个代码、数据和堆栈模块的机制，以便多个程序（或任务）可以在同一处理器上运行而不会互相干扰；分页提供了一种实现传统按需分页虚拟内存系统的机制，其中程序执行环境的各个部分根据需要映射到物理内存中，分页还可用于在多个任务之间提供隔离。在受保护模式下操作时，必须使用某种形式的分段，没有模式位可以禁用分段，但是分页的使用是可选的。

这两种机制（分段和分页）可以配置为支持简单的单程序（或单任务）系统、多任务系统或使用共享内存的多处理器系统。

2.1.1 逻辑地址（Logical Address）

系统中的所有段都包含在处理器的线性地址空间中，要定位特定段中的字节，必须提供逻辑地址（也称为远指针）。逻辑地址由段选择器和偏移量组成。段选择器是段的唯一标识符。此外，它还为描述符表（例如全局描述符表，GDT）中的数据结构（称为段描述符）提供偏移量。每个段都有一个段描述符，它指定段的大小、段的访问权限和特权级别、段类型以及段在线性地址空间中的第一个字节的位置（称为段的基址）。逻辑地址的偏移量部分添加到段的基址，以定位段内的字节，基址加上偏移量就形成了处理器线性地址空间中的线性地址。

2.1.2 线性地址（Linear Address）

如图 3-1 所示，分段提供了一种机制，用于将处理器的可寻址内存空间（称为线性地址空间）划分为较小的受保护地址空间（称为段）。段可用于保存程序的代码、数据和堆栈，或保存系统数据结构（如 TSS 或 LDT）。如果处理器上运行着多个程序（或任务），则可以为每个程序分配一组自己的段，然后处理器会强制执行这些段之间的边界，并确保一个程序不会通过写入另一个程序的段来干扰另一个程序的执行。分段机制还允许对段进行类型化，以便可以限制在特定类型的段上执行的操作。

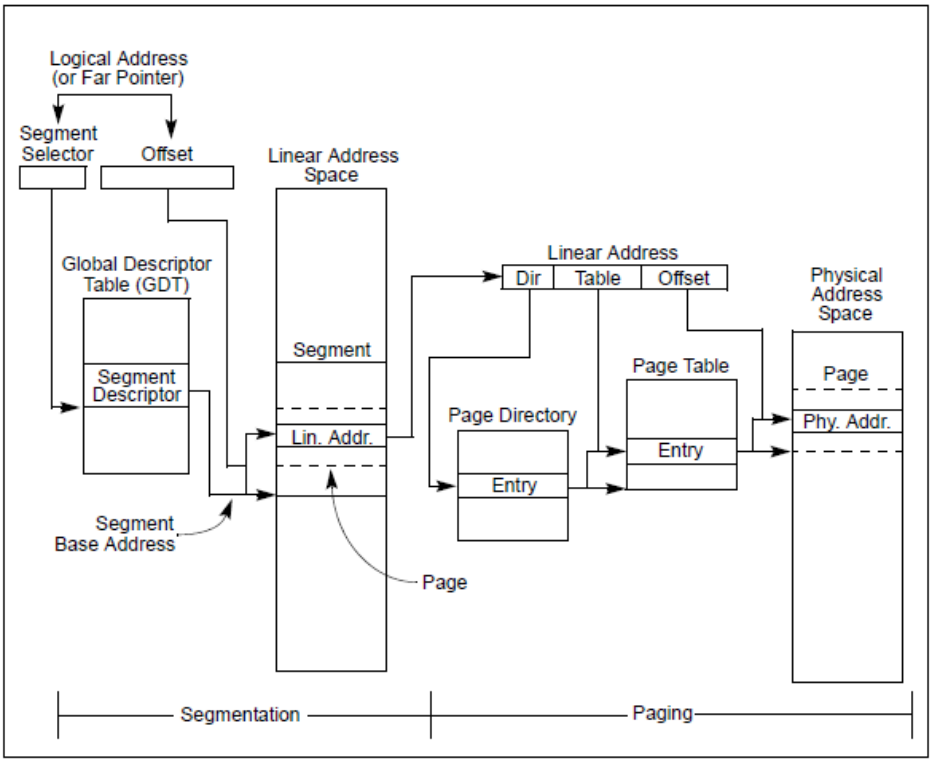

Figure 3-1. Segmentation and Paging

2.1.3 物理地址（Physical Address）

如果不使用分页，处理器的线性地址空间将直接映射到处理器的物理地址空间。物理地址空间定义为处理器可以在其地址总线上生成的地址范围。

由于多任务计算系统通常定义的线性地址空间比一次性将所有地址都包含在物理内存中经济可行的空间大得多，因此需要某种“虚拟化”线性地址空间的方法。这种线性地址空间的虚拟化是通过处理器的分页机制来处理的。

分页支持“虚拟内存”环境，在该环境中，使用少量物理内存（RAM 和 ROM）和一些磁盘存储来模拟较大的线性地址空间。使用分页时，每个段被分成页面（通常每个页面大小为 4 KB），这些页面存储在物理内存或磁盘上。操作系统或执行程序维护一个页面目录和一组页面表来跟踪页面，当程序（或任务）尝试访问线性地址空间中的地址位置时，处理器使用页面目录和页面表将线性地址转换为物理地址，然后在内存位置执行请求的操作（读取或写入）。

2.2 分段机制

IA-32 架构支持的分段机制可用于实现各种系统设计。这些设计包括仅使用最低限度的分段来保护程序的平面模型，以及使用分段来创建可可靠执行多个程序和任务的稳健操作环境的多分段模型。

2.2.1 Basic Flat Model

系统最简单的内存模型是基本的“平面模型”，其中操作系统和应用程序可以访问连续、未分段的地址空间。这种基本的平面模型在最大程度上向系统设计人员和应用程序程序员隐藏了体系结构的分段机制。

要使用 IA-32 体系结构实现基本的平面内存模型，必须创建至少两个段描述符，一个用于引用代码段，一个用于引用数据段（参见图 3-2）。这两个段都映射到整个线性地址空间，即两个段描述符具有相同的基址值 0 和相同的段限制 4 GB。通过将段限制设置为 4 GB，即使没有物理内存驻留在特定地址，分段机制也不会因超出限制的内存引用而生成异常。ROM（EPROM）通常位于物理地址空间的顶部，因为处理器从 FFFF_FFF0H 开始执行。 RAM（DRAM）位于地址空间的底部，因为复位初始化后，DS 数据段的初始基地址为 0。

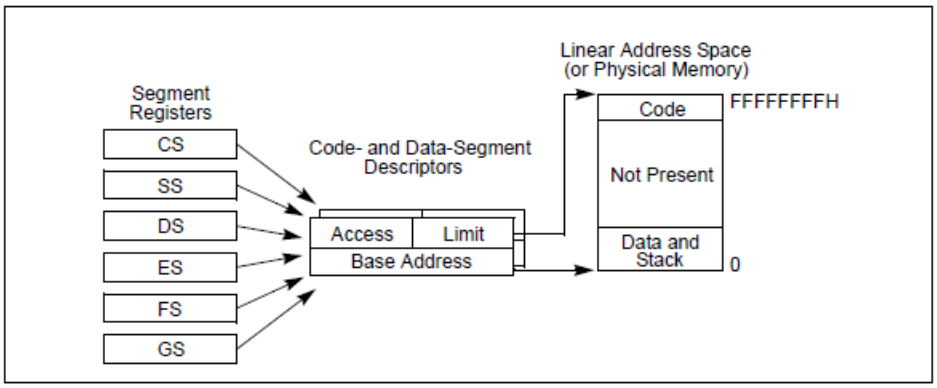

Figure 3-2. Flat Model

2.2.2 Protected Flat Model

Protected Flat Model 与 Basic Flat Model 相似，不同之处在于段限制设置为仅包含物理内存实际存在的地址范围（参见图 3-3），任何试图访问不存在的内存的行为都会生成一般保护异常 (#GP)。此模型针对某些类型的程序错误提供了最低级别的硬件保护。

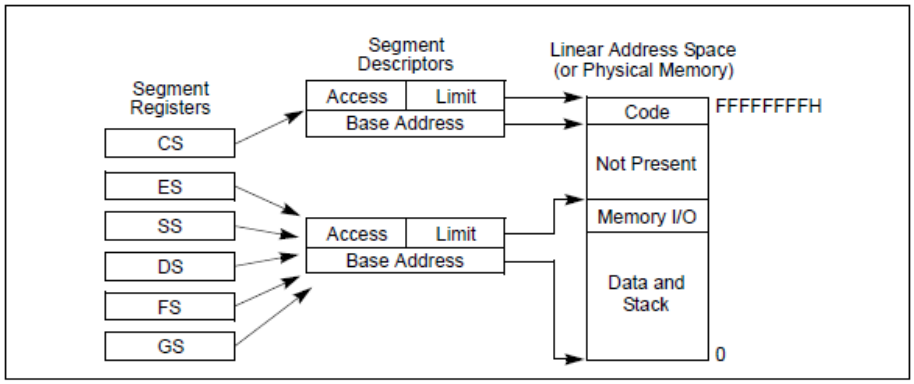

Figure 3-3. Protected Flat Model

这种受保护的平面模型可以添加更多元素，以提供更多保护。例如，为了使分页机制在用户和管理员代码与数据之间提供隔离，需要定义四个段：特权级别 3 上的用户代码和数据段，特权级别 0 上的管理员代码和数据段，通常这些段都相互重叠，并从线性地址空间中的地址 0 开始。这种平面分段模型以及简单的分页结构可以保护操作系统免受应用程序的影响，并且通过为每个任务或进程添加单独的分页结构，它还可以保护应用程序免受彼此的影响。几种流行的多任务操作系统都使用了类似的设计。

2.2.3 Multi-Segment Model

Multi-Segment Model（如图 3-4 所示）充分利用分段机制的功能，为代码、数据结构以及程序和任务提供硬件强制保护。在此，每个程序（或任务）都有自己的段描述符表和自己的段，这些段可以完全专属于其指定程序，也可以在程序之间共享。所有段的访问以及系统上运行的各个程序的执行环境的访问均由硬件控制。

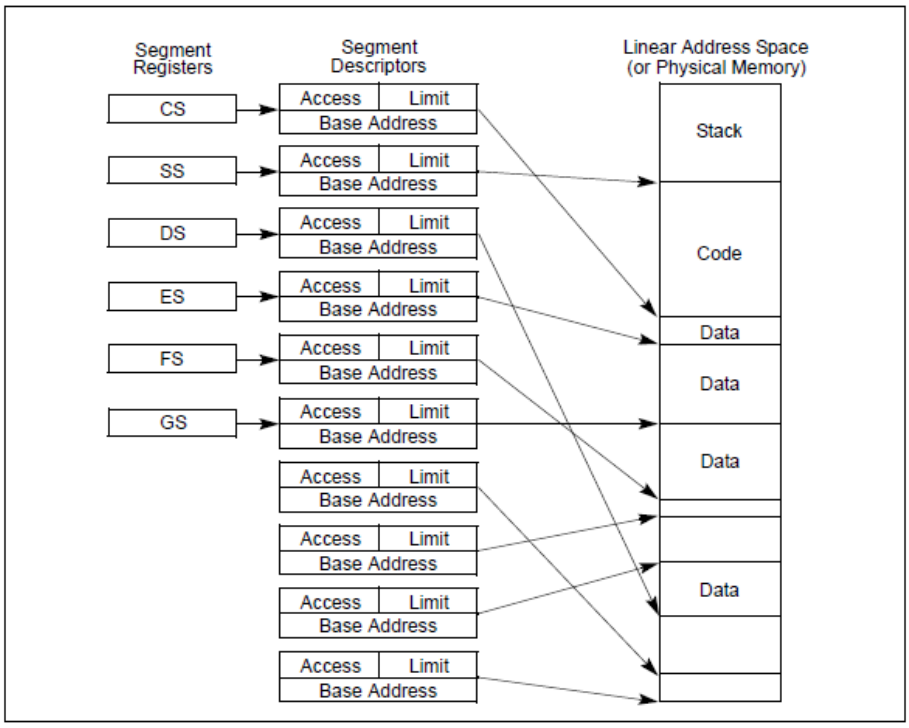

Figure 3-4. Multi-Segment Model

访问检查不仅可用于防止引用超出段范围的地址，还可防止在某些段中执行不允许的操作。例如，由于代码段被指定为只读段，因此可以使用硬件来防止写入代码段；为段创建的访问权限信息还可用于设置保护环或保护级别；保护级别可用于保护操作系统程序免受应用

程序的未经授权的访问。

2.3 逻辑地址和线性地址的转换

要将逻辑地址转换为线性地址，处理器会执行以下操作：

（1）使用段选择器中的偏移量在 GDT 或 LDT 中定位该段的段描述符，并将其读入处理器。（仅当将新的段选择器加载到段寄存器中时才需要此步骤。）

（2）检查段描述符来检查段的访问权限和范围，确保该段可访问并且偏移量在段的限制范围内。

（3）将段描述符中的段基地址与偏移量相加，形成线性地址。

2.3.1 段选择子（Segment Selectors）

段选择器是段的 16 位标识符（见图 3-6），它不直接指向段，而是指向定义该段的段描述符。

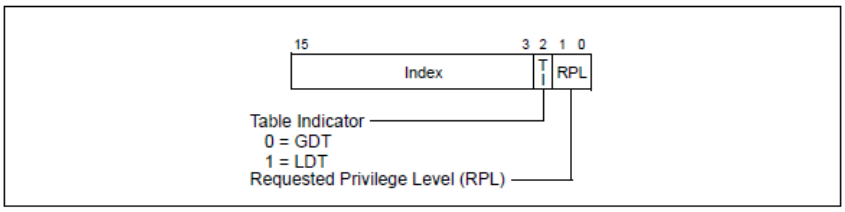

Figure 3-6. Segment Selector

段选择器包含以下项：

（1）索引（位 3 至 15）— 选择 GDT 或 LDT 中的 8192 个描述符之一。处理器将索引值乘以 8（段描述符中的字节数），并将结果添加到 GDT 或 LDT 的基地址（分别来自 GDTR 或 LDTR 寄存器）。

（2）TI（表指示器）标志（位 2）——指定要使用的描述符表：清除此标志表示选择 GDT；设置此标志表示选择当前 LDT。

（3）请求特权级别 (RPL)（位 0 和 1）— 指定选择器的特权级别。特权级别范围为 0 到 3，其中 0 为最高特权级别。

处理器不使用 GDT 的第一个条目。指向 GDT 此条目的段选择器（即索引为 0 且 TI 标志设置为 0 的段选择器）用作“空段选择器”。当段寄存器（CS 或 SS 寄存器除外）加载空选择器时，处理器不会生成异常，但当使用保存空选择器的段寄存器访问内存时，处理器会产生异常。空选择器可用于初始化未使用的段寄存器，使用空段选择器加载 CS 或 SS 寄存器会导致生成通用保护异常 (#GP)。

段选择器作为指针变量的一部分对应用程序可见，但选择器的值通常由链接编辑器或链接加载器（而不是应用程序）分配或修改。

2.3.2 段寄存器（Segment Registers）

为了减少地址转换时间和编码复杂性，处理器提供了最多可容纳 6 个段选择器的寄存器（见图 3-7），每个段寄存器都支持特定类型的内存引用（代码、堆栈或数据）。对于几乎任何类型的程序执行，须至少将有效的段选择器加载到代码段 (CS)、数据段 (DS) 和堆栈段 (SS) 寄存器中。处理器还提供了三个额外的数据段寄存器（ES、FS 和 GS），可用于为当前正在执行的程序（或任务）提供额外的数据段。

要让程序访问某个段，必须先将该段的段选择器加载到其中一个段寄存器中。因此，尽管系统可以定义数千个段，但只有 6 个段可以立即使用，其他段可以通过在程序执行期间将其段选择器加载到这些寄存器中来提供。

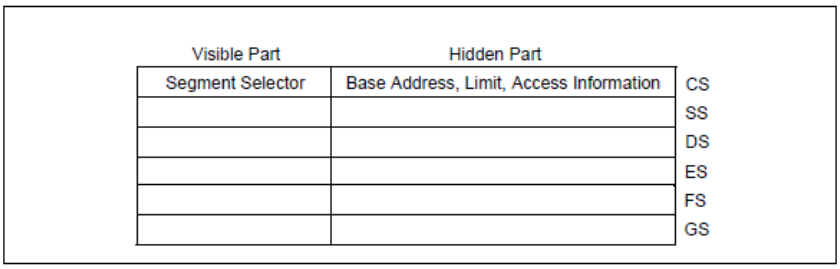

Figure 3-7. Segment Registers

每个段寄存器都有一个“可见”部分和一个“隐藏”部分。（隐藏部分有时称为“描述符缓存”或“影子寄存器”。）当段选择器加载到段寄存器的可见部分时，处理器还会将段选择器指向的段描述符中的基址、段限制和访问控制信息加载到段寄存器的隐藏部分。缓存在段寄存器中的信息（可见和隐藏）允许处理器转换地址，而无需花费额外的总线周期来从段描述符中读取基址和限制。在多个处理器可以访问相同描述符表的系统中，当描述符表被修改时，软件有责任重新加载段寄存器。如果不这样做，缓存在段寄存器中的旧段描述符可能会在其内存驻留版本被修改后被使用。

2.3.3 段描述子（Segment Descriptors）

段描述符是 GDT 或 LDT 中的一种数据结构，它为处理器提供段的大小和位置，以及访问控制和状态信息。段描述符通常由编译器、链接器、加载器或操作系统或执行程序创建，但不是由应用程序创建，图 3-8 说明了所有类型的段描述符的通用描述符格式。

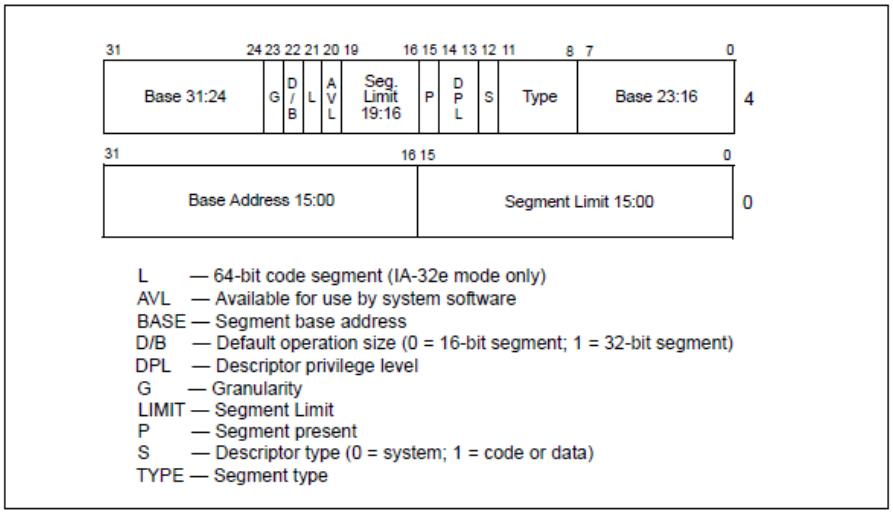

Figure 3-8. Segment Descriptor

2.4 描述符的分类

描述符分为两类：系统段描述符和门描述符。系统段描述符指向系统段（LDT 和 TSS 段）；门描述符本身就是“门”，它保存指向代码段（调用、中断和陷阱门）中过程入口点的指针，或保存 TSS（任务门）的段选择器。

2.4.1 数据段描述符（Data segment Descriptor）

数据段描述符用于定义内存中的数据段，其中存储着程序的数据。包含了数据段的属性，如基址、大小、访问权限等信息。

2.4.2 代码段描述符（Code segment Descriptor）

代码段描述符用于定义内存中的代码段，其中存储着程序的可执行代码。包含了代码段的属性，如基址、大小、访问权限等信息。

2.4.3 局部描述符表描述符（Local descriptor-table (LDT) segment descriptor）

局部描述符表描述符用于定义局部描述符表，允许每个任务或进程具有自己的局部描述符表。它允许不同的任务或进程拥有独立的内存段定义，以提供额外的隔离和安全性。

每个系统可以定义一个或多个 LDT。例如，可以为每个正在运行的单独任务定义一个 LDT，或者部分或所有任务可以共享同一个 LDT。

2.4.4 任务状态段描述符（Task-state segment (TSS) descriptor）

任务状态段描述符用于定义任务状态段，它包含了与任务切换和处理器状态相关的信息。TSS 描述符用于多任务操作系统中，允许处理器在不同的任务之间进行快速切换。

2.4.5 调用门描述符（Call-gate descriptor）

调用门描述符用于支持过程（函数）之间的跳转和调用。它允许程序在不同的特权级别之间进行跳转和调用，提供了一种安全的方式来执行代码。

2.4.6 中断门描述符（Interrupt-gate descriptor）

中断门描述符用于定义中断门，允许处理器在响应中断时跳转到指定的中断服务程序。中断门描述符包含了中断服务程序的地址和特权级别等信息。

2.4.7 陷阱门描述符（Trap-gate descriptor）

陷阱门描述符与中断门描述符类似，但用于定义陷阱门，用于捕获和处理陷阱（异常）。与中断不同，陷阱不会中断当前程序的执行，而是在当前程序执行后再执行陷阱服务程序。

2.4.8 任务门描述符（Task-gate descriptor）

任务门描述符用于定义任务门，允许程序跳转到特定任务的 TSS。任务门描述符通常用于实现任务切换和多任务操作系统的功能。