## 5.3 段式内存管理

1. 分段基本方法

采用分页内存管理有一个不可避免的问题：用户视角的内存和实际物理内存的分离。可以看到，用户视角的内存与实际物理内存不一样。用户视角的内存需要映射到实际物理内存，该映射允许区分逻辑内存和物理内存。

用户通常愿意将内存看做是一组不同长度的段的集合，这些段之间并没有一定的顺序(见图 5.21)。人们会认为程序是由主程序加上一组方法、过程、函数所构成的，还有各种数据结构：对象、数组、堆栈、变量等。每个模块或其他数据元素都可通过名称引用。人们会说"堆栈"、"库"、"主程序"，而并不关心这些元素所在内存的位置，不关心栈是放在某个库函数之前还是之后。这些段的长度是不同的，其长度是由这些段在程序中的目的所定的。段内的元素是通过它们距段首的偏移来指定：程序的第一条语句、栈里的第 7 个栈帧、函数 Sqrt 的第 5 条指令等。

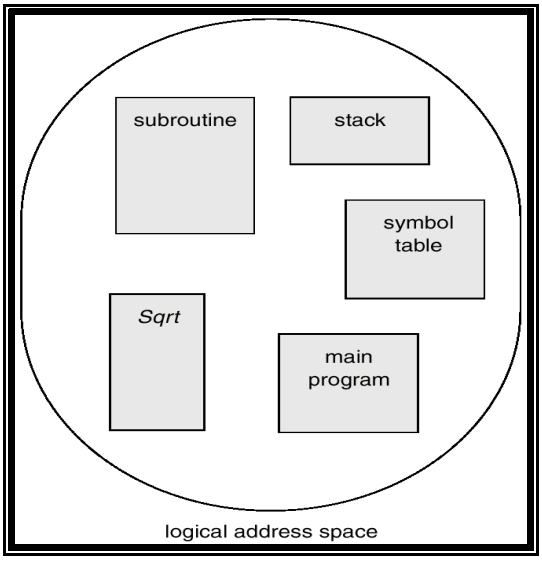


图 5.21 用户眼中的程序

分段 (segmentation)就是支持这种用户视角的内存管理方案。逻辑地址空间是由一组段组成的。每个段都有名称和长度。地址指定了段名称和段内偏移。因此用户通过两个量来指定地址：段名称和偏移(这一方案与分页有所不同：在分页中，用户只指定一个地址，该地址通过硬件分为页码和偏移，对于这些，程序员是看不见的)。

为实现简单起见，段是编号的，是通过段号(segment-number)而不是段名来引用的。因此，逻辑地址由有序对组成:

<segment-number, offset>

| 31 16 | 15 0 |
|---|---|
| 段号 s | 段内偏移 d |

通常，在编译用户程序时，编译器会自动根据输入程序来构造段。

一个 C 编译器可能会创建如下几个段:

(1) 代码。

(2) 全局变量。

(3) 堆(内存从堆上分配)。

(4) 每个线程采用的栈。

(5) 标准的C库函数。

在编译时链接的库可能被分配不同的段。加载程序会装入所有这些段，并为它们分配段号。

2. 段表和段的地址映射

虽然用户现在能够通过二维地址来引用程序中的对象，但是实际物理内存仍然是一维序列的字节。因此，必须走义一个实现方式，以便将二维的用户定义地址映射为一维物理地址。这个地址是通过段表(segment table)来实现的。段表的每个条目都有段基地址和段界限。段基地址包含该段在内存中的开始物理地址，而段界限指定该段的长度。

段表在硬件上有一组段表基地址寄存器(Segment-table base register, STBR)和段表限长寄存器(Segment-table length register, STLR)支持。段表基址寄存器指向段表在内存中的地址，段表限长寄存器表明被一个程序所使用的段的数目。段表的硬件支持和使用如图5.22所示。一个逻辑地址由两部分组成：段号s和段内的偏移d。段号用做段表的索引，逻辑地址的偏移 d应位于0和段界限之间。如果不是这样，会陷入到操作系统中(逻辑地址试图访问段的外面)。如果偏移d合法，那么就与段基地址相加而得到所需字节在物理内存的地址。

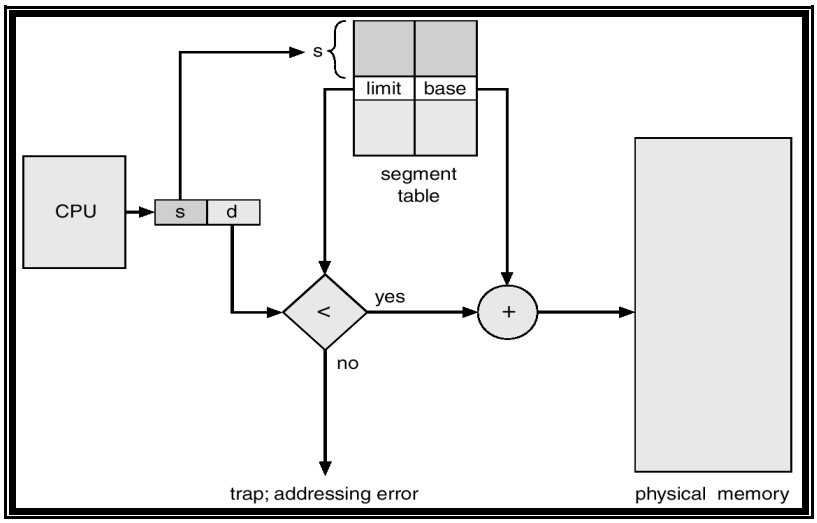


图5.22 段表的硬件支持和使用

如图5.23所示，有5个段，编号为0~4。各段按图中所示存储。每个段都在段表中有一个条日，它包括段在物理内存内的开始地址(或基地址)和该段的长度(或界限)。例如，段2为400B长，开始于位置4300。因此，对段2第53字节的引用映射成位置4300+53=43530。对段3第852字节的引用映射成位置3200+852=4052。段0第1222字节的引用会陷入操作系统，这是由于该段仅为1000 B长。

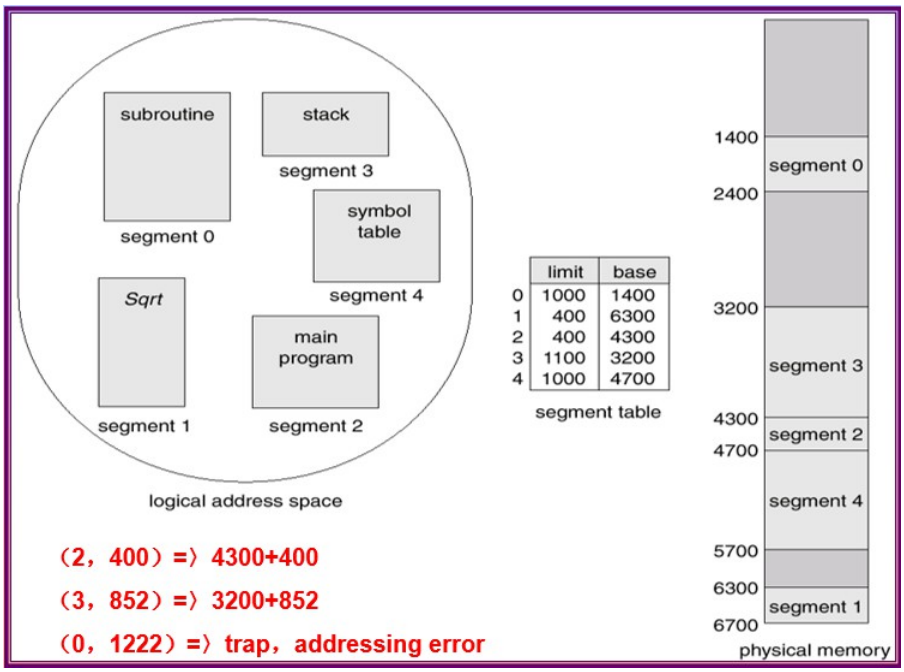


图 5.23 分段的例子

3. 页式管理和段式管理的比较

- 分页是出于系统管理的需要，分段是出于用户应用的需要。因此，一条指令或一个操作数可能会跨越两个页的分界处，而不会跨越两个段的分界处。如图 5.24 所示。
- 页大小是系统固定的，而段大小则通常不固定。
- 通常段比页大，因而段表比页表短。

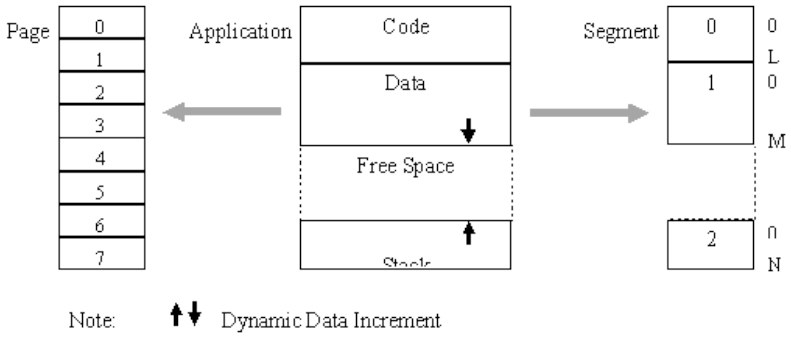


图 5.24 页式管理和段式管理的比较

- 逻辑地址表示：分页是一维的，各个模块在链接时必须组织成同一个地址空间；而分段是二维的，各个模块在链接时可以每个段组织成一个地址空间。
- 分页不会产生外部碎片，但会有内部碎片；分段没有内部碎片，外部碎片也可以通过内存紧缩进行消除。
- Intel 386 使用段页结合来进行二级分页的内存管理。

4. 示例：Intel Pentium

分页或分段都有其优缺点。事实上，有些体系结构两种都提供。这里将讨论支持单纯的分段或分页加分段的 Intel Pentium 体系结构它所基于的主要思想和 Pentium 系统上 Linux 地址转换模型。

在 Pentium 系统中，CPU 产生逻辑地址，它被赋给分段单元。分段单元为每个逻辑地址生成线性地址，然后线性地址交给分页单元，它接下来生成内存中的物理地址。因此，分段单元和分页单元相当于内存管理单元 (MMU) 。这种方案如图 5.25 所示。

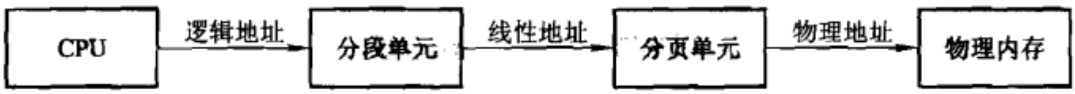


图 5.25 Pentium 中逻辑地址到物理地址的转换

(a) Pentium 分段

Pentium 结构允许一个段的大小最多可达 4GB，每个进程最多的段的数量为 16K 个。进程的逻辑地址空间被分为两部分:第二个部分最多由 8K 个段组成，这部分为私有：第二个部分最多由 8K 个段组成，这部分为所有进程所共享。关于第一个部分的信息保存在本地描述符表(local descriptor table, LDT)中，而关于第二个部分的信息保存在全局描述符表(global descriptor table, GDT)中。LDT 和 GDT 的每个条目为 8B，包括二个段的详细信息，如基位置和段界限等。

逻辑地址是一对(selector, offset)，选择器(selector)是一个 16 位的数：

| s | g | P |
|---|---|---|
| 13 | 1 | 2 |

其中，s 表示段号，g 表示段是在 GDT 还是在 LDT 中，p 表示保护信息。偏移(offset)是一个 32 位的数，用来表示字节(或字)在段内的位置。

机器有 6 个段寄存器，允许一个进程可以同时访问 6 个段。它还存 6 个 8B 的微程序寄存器，用来保存相应的来自于 LDT 或 GDT 的描述符。这一缓冲区允许 Pentium 不必在每次内存引用时都从内存中读取描述符。

Pentium 的线性地址为 32 位长，按如下方式来形成:段寄存器指向 LDT 或 GDT 中的适当条目，段的基地址和界限信息用来产生线性地址。首先，界限用来检查地址的合法性。如果地址无效，就产生内存出错，导致陷入操作系统。如果有效，偏移值就与基地址的值相加，产生 32 位的线性地址，如图 5.26 所示。

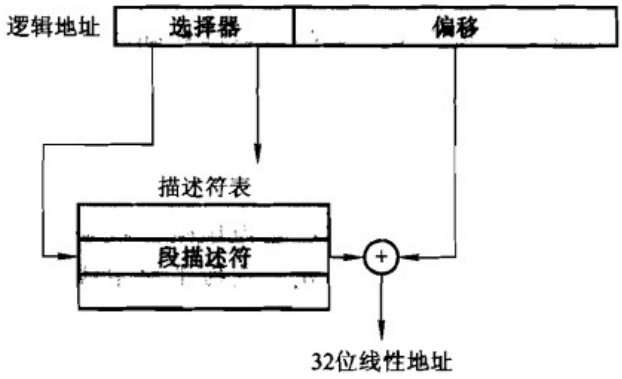


图 5.26 Intel Pentium 分段

(b) Pentium 分页

Pentium 体系结构允许页的大小为 4KB 或 4MB。对于 4KB 的页，Pentium 采用二级分页方案，其中 32 位线性地址如下形式划分：

| 页码 | | 页偏移 |
|---|---|---|
| $p_1$ | $p_2$ | $D$ |
| 10 | 10 | 12 |

Intel Pentium 地址转换的详细情况如图 5.27 所示。最高 10 位引用最外层页表的条目，它被称为页目录(page directory)(CR3 寄存器指向当前进程的页目录)。页目录条目指向由线性地址中最内层的 10 位内容索引的内部页表。最后，最低的 0~11 位是页表项所指向的 4KB 页面内的偏移。

页目录中的一个条目是 Page Size 标志，如果设置了它，表示页帧的大小为 4MB，而不是标准的 4MB。如果设置了该标志，则页目录直接指向 4MB 页帧，而绕过了内层页表，且线性地址的最低 22 位指向 4MB 页帧的偏移。

为了提高物理内存的使用效率，Intel Pentium 的页表可以交换到磁盘上。这时，页目录条目的无效位可用来表示相应的页表是在内存还是在磁盘上。如果在磁盘上，操作系统可以使用其他的 31 位来表示页表在磁盘上的位置，该页表可根据需要调入内存。

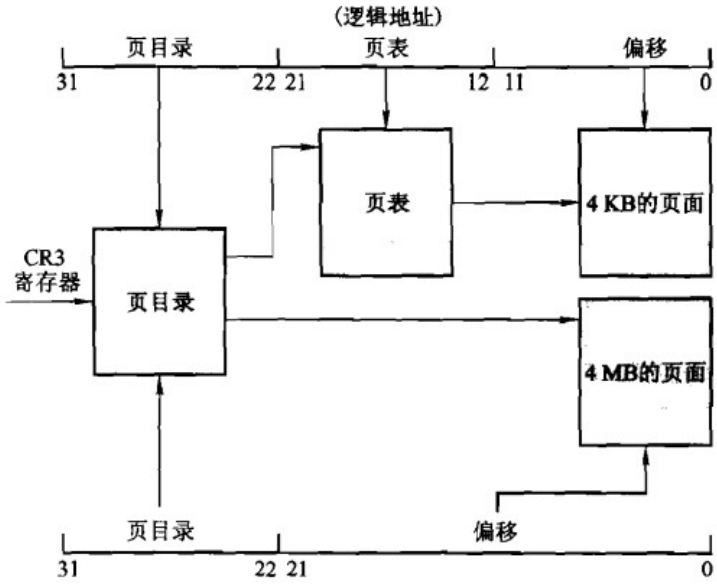


图 5.27 Pentium 体系结构中的分页

(c) Pentium 系统上的 Linux

作为一个例子，考虑运行在 IntelPentium 体系结构上的 Linux 系统。由于 Linux 被设计为在一系列处理器上运行(其中许多可能仅提供对段的有限支持)，Linux 并不依赖段，并最低程度地使用它。在 Pentium 中，Linux 仅使用 6 个段：

(1) 内核代码段。

(2) 内核数据段。

(3) 用户代码段。

(4) 用户数据段。

(5) 任务状态段(TSS)。

(6) 默认的 LDT 段。

用户代码段和用户数据段为所有以用户模式运行的进程所共享。由于所有进程使用相同的逻辑地址空间且所有段描述符保存在全局描述符表(GDT)内，因此这是可能的。进一步讲，每个进程都有它自己的任务状态段(TSS)，该段的描述符被保存在 GDT 中。该 TSS 被用来保存在上下文切换中每个进程的硬件上下文。通常，默认的 LDT 段被所有进程所共享，且通常并不被使用。不过，如果一个进程需要它自己的 LDT，则它可以生成一个 LDT，并用它代替默认的 LDT。

正如所指出的，每个段选择器包括一个 2 位保护域，因此 Pentium 允许四级保护。在此四级保护中，Linux 仅识别用户模式与内核模式。

尽管 Pentium 采用二级分页模式，Linux 被设计为能运行多种硬件平台，其中许多平台是 64 位的，而二级分页对此并不合适。因此，Linux 采用适合 32 位和 64 位体系结构的三级分页方案。

Linux 中的线性地址分成如下 4 个部分：

| 全局目录 | 中间目录 | 页表 | 偏移 |
|---|---|---|---|

图 5.28 中明确了 Linux 中的三级分页模式。

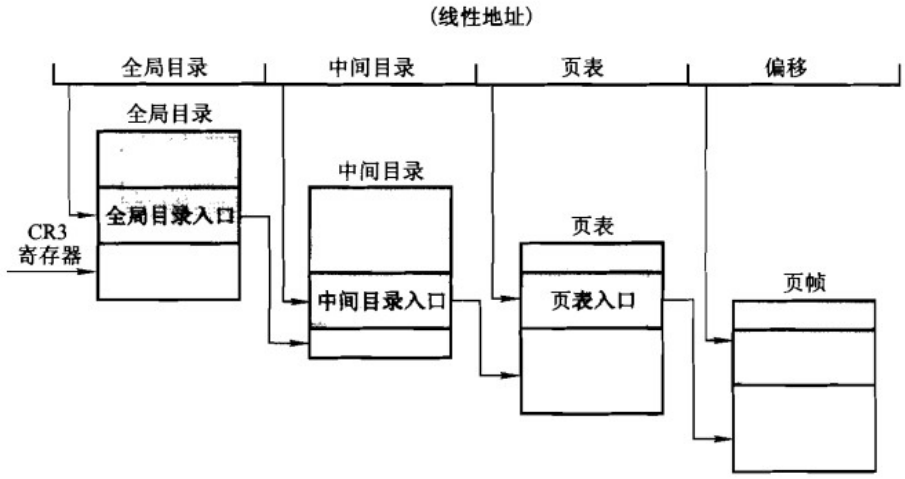


图 5.28 Linux 中的三级分页模式

根据体系结构的不同，线性地址的每一部分的位的多少也有变化。但是，正如前面所讲的，Pentium 体系结构仅采用二级分页模式。那 Linux 是如何在 Pentium 上使用三级模式的呢?此时，中间目录的大小为零位，实际上绕开了中间目录。

Linux 中的每个任务都有它自己的页表，如图 5.x2 所示，CR3 寄存器指向了当前执行的任务的全局目录。在上下文切换中，CR3 寄存器的值被保存和恢复是在与上下文切换相关任务的 TSS 段中进行的。