4. 任务管理

4.1. 任务管理概述

> - 段是内存的一个单位。在 x86 体系结构中，内存被划分为多个段，每个段都有不同的属性和用途。每个段可以包含一定范围的内存地址。这种划分内存的方式被称为分段内存管理。
> - 每个段都由一个段描述符来定义，该描述符包括了段的起始地址、大小、特权级别、访问权限等属性。不同的段可以用于存储代码、数据、堆栈以及其他类型的信息。段描述符存储在全局描述符表（Global Descriptor Table，GDT）或局部描述符表（Local Descriptor Table，LDT）中。
> - 选择子是描述符的引用，描述符描述了内存段的属性

1. 什么是任务？
    - 任务是处理器能够调度、执行和挂起的工作单元。它可以是程序、进程、操作系统服务、实用工具、异常或中断处理程序，甚至是内核或系统管理功能。在 IA-32 架构中，任务是执行的基本单元，所有处理都在任务中进行。
2. `80x86` 提供了哪些硬件支持？

    在 IA-32 架构中，`80x86` 提供了以下硬件支持用于任务管理：
    - 任务状态段（TSS）： 存储任务的状态信息，包括寄存器内容、段选择子、堆栈指针和 I/O 许可位图等。
    - 任务寄存器（TR）： 保存当前任务的 TSS 选择子，指向当前任务的 TSS。
    - 任务门描述符： 允许通过任务门进行任务切换，提供对新任务的安全入口。
    - 硬件任务切换机制： 支持通过指令（如 `CALL`、`JMP`）或中断、异常触发任务切换，自动保存和恢复任务状态。
    - 特权级管理： 通过段描述符中的特权级（DPL）和任务状态段中的特权级，控制任务的访问权限和执行权限。
3. 描述符表中与任务相关的描述符有哪些？
    - 任务状态段描述符（TSS Descriptor）： 定义任务状态段的位置和属性，用于任务切换时保存和恢复任务的上下文。
    - 任务门描述符（Task Gate Descriptor）： 提供对任务的安全入口，允许通过任务门进行任务切换。

- 局部描述符表描述符（LDT Descriptor）： 定义任务的局部描述符表的位置和属性，允许每个任务拥有自己的段描述符。

4. 任务切换与过程调用的区别是什么？

- 任务切换：
  - 涉及完整的上下文切换，包括保存当前任务的所有寄存器、段寄存器、控制寄存器（如 CR3）等状态信息，并加载新任务的状态。
  - 可能涉及特权级的变化，需要进行权限检查和堆栈切换。
  - 开销较大，但提供了多任务环境下的任务隔离和保护。
- 过程调用：
  - 仅涉及部分上下文的保存，如返回地址和部分寄存器，通常使用堆栈进行保存和恢复。
  - 不涉及特权级的变化，通常在同一任务的上下文中执行。
  - 开销较小，适用于函数调用等场景。

4.1.1. 任务的结构

4.1.1.1 一个任务由几部分构成 ？

一个任务由两部分组成：任务执行空间（Task Execution Space）和任务状态段（Task-State Segment，TSS）。

4.1.1.2 任务执行空间包括什么 ？

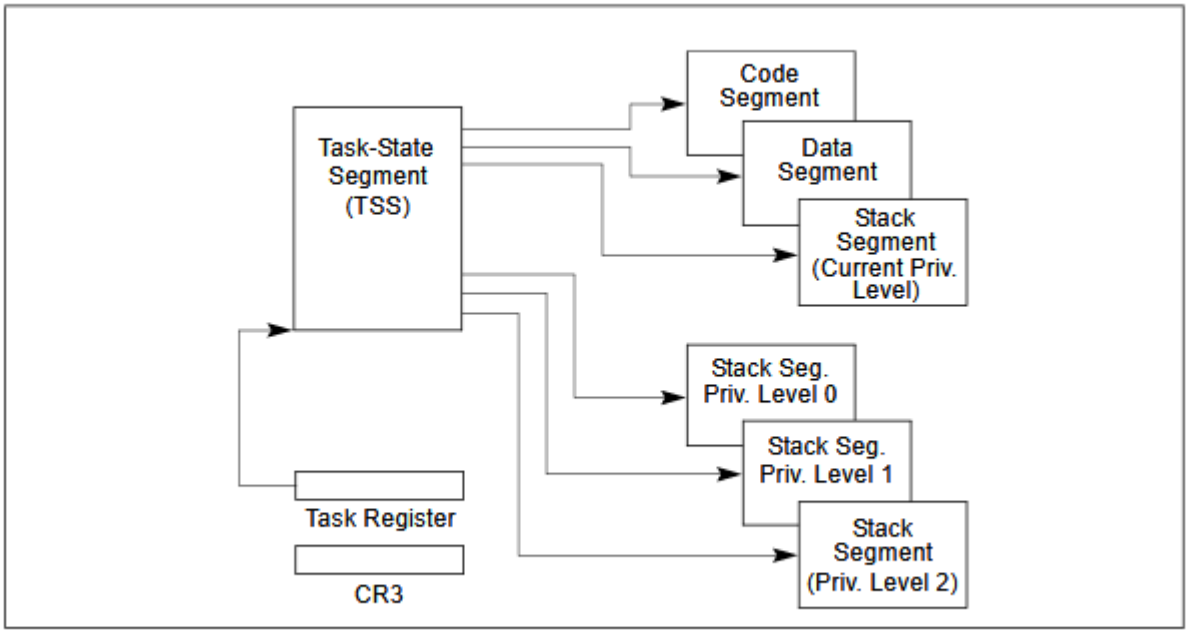

Figure 7-1. Structure of a Task

- 任务执行空间由以下部分组成：

    - 代码段（Code Segment）： 存放任务的可执行指令。
    - 堆栈段（Stack Segment）： 用于存储函数调用、局部变量和中断处理等的堆栈数据。
    - 一个或多个数据段（Data Segments）： 存储任务所需的数据。
- 如果操作系统使用处理器的特权级保护机制，任务执行空间还会为每个特权级别提供单独的堆栈。

4.1.1.3 为什么会有多个特权级栈空间 ？

在支持特权级保护的系统中，不同的特权级别（如 0、1、2、3）有不同的访问权限。为了确保在特权级切换时（例如从用户模式切换到内核模式）堆栈的安全性和稳定性，处理器为每个特权级别提供独立的堆栈空间。这样可以防止低特权级别的代码访问或破坏高特权级别的堆栈数据，增强系统的安全性和稳定性。

4.1.2. 任务状态

- 当前正在执行的任务状态包括哪些内容?
- 掌握每一个被包含内容的含义?
- 为什么要包含这些内容?

当前正在执行的任务状态包括以下内容：

1. 段寄存器的段选择子： 包括 CS、DS、SS、ES、FS 和 GS，定义任务的当前执行空间。
2. 通用寄存器的状态： 如 EAX、EBX、ECX、EDX、ESI、EDI、EBP 和 ESP，保存任务的工作数据。
3. EFLAGS 寄存器的状态： 包含状态标志、控制标志和系统标志，反映任务的当前状态和控制信息。
4. EIP 寄存器的状态： 指示任务下一条将执行的指令地址。
5. 控制寄存器 CR3 的状态： 包含页目录的基地址，用于分页机制下的地址转换。
6. 任务寄存器（TR）的状态： 保存当前任务的 TSS 选择子，指向任务状态段。
7. 局部描述符表寄存器（LDTR）的状态： 包含局部描述符表的段选择子，定义任务的局部段描述符。
8. I/O 位图基地址和 I/O 位图： 位于 TSS 中，定义任务的 I/O 端口访问权限。
9. 特权级 0、1 和 2 的堆栈指针： 位于 TSS 中，指向各特权级的堆栈顶，用于特权级切换时的堆栈管理。

10. **链接到先前执行的任务：** 位于 TSS 中，保存上一个任务的链接信息，支持任务切换。

在任务被调度执行之前，除任务寄存器的状态外，所有这些信息都存储在任务的 TSS 中。LDTR 寄存器的完整内容不在 TSS 中，只有其段选择子被包含。

包含这些内容的原因：

这些信息全面描述了任务的执行环境和状态，确保在任务切换时能够准确保存和恢复任务的上下文。这对于多任务操作系统至关重要，保证各任务独立运行，互不干扰。

4.1.3. 任务的执行

1. **任务的执行方式有几种？**
    - 任务的执行方式有以下五种：
        - 使用 `CALL` 指令显式调用任务。
        - 使用 `JMP` 指令显式跳转到任务。
        - 处理器隐式调用中断处理程序任务。
        - 处理器隐式调用异常处理程序任务。
        - 当 EFLAGS 寄存器中的 NT 标志位设置时，使用 `IRET` 指令返回到任务。
2. **熟悉掌握每一种执行方式的过程**
    - **使用 `CALL` 指令显式调用任务：**
        - `CALL` 指令的操作数是一个段选择子，指向任务门或 TSS 描述符。
        - 执行 `CALL` 指令时，处理器保存当前任务的状态到其 TSS，然后加载目标任务的状态，开始执行目标任务。
    - **使用 `JMP` 指令显式跳转到任务：**
        - `JMP` 指令的操作数是一个段选择子，指向任务门或 TSS 描述符。
        - 执行 `JMP` 指令时，处理器直接切换到目标任务的上下文，开始执行目标任务。
    - **处理器隐式调用中断处理程序任务：**
        - 当中断发生时，处理器在中断描述符表（IDT）中查找对应的中断向量。
        - 如果对应的描述符是任务门，处理器将进行任务切换，执行中断处理任务。
    - **处理器隐式调用异常处理程序任务：**
        - 当异常发生时，处理器在 IDT 中查找对应的异常向量。

    - 如果对应的描述符是任务门，处理器将进行任务切换，执行异常处理任务。
  - 使用 `IRET` 指令返回到任务：
    - 当 EFLAGS 寄存器中的 NT（Nested Task）标志位设置时，执行 `IRET` 指令会导致任务切换。
    - 处理器将切换到前一个任务的上下文，继续执行。
3. `Linux 0.00` 用的是哪种方式？
   - 在 `Linux 0.00` 中，任务切换是通过时钟中断触发的。
   - 时钟中断发生时，处理器进入中断处理程序。
   - 在中断处理程序中，使用 `ljmp` 指令（长跳转）进行任务切换。
   - `ljmp` 指令的操作数是一个段选择子，指向目标任务的 TSS 描述符。
   - 执行 `ljmp` 指令时，处理器保存当前任务的状态到其 TSS，然后加载目标任务的状态，开始执行目标任务。
   - 这种方式属于处理器隐式调用中断处理程序任务。
4. 任务可以递归调用吗？为什么？

- 任务不能递归调用。
  - 在 IA-32 架构中，任务是非递归的。
  - 处理器不允许任务调用或跳转到自身，以*防止无限递归，导致堆栈溢出或系统崩溃*。
  - 这种设计确保了系统的稳定性和可靠性。

4.2. 任务的数据结构

在 IA-32 架构的保护模式下，任务管理涉及以下关键数据结构：

1. 任务状态段（TSS，Task-State Segment）：
   - 功能： 存储任务的上下文信息，包括通用寄存器、段寄存器、EFLAGS、EIP、CR3、I/O 许可位图等。
   - 作用： 在任务切换时，处理器将当前任务的状态保存到其 TSS 中，并从新任务的 TSS 中恢复状态，以实现任务的暂停和恢复。
2. 任务门描述符（Task-Gate Descriptor）：
   - 功能： 提供对任务的受控入口，允许通过任务门进行任务切换。
   - 作用： 在中断描述符表（IDT）或全局描述符表（GDT）中定义，指向特定任务的 TSS 描述符，支持通过中断、异常或显式调用进行任务切换。
3. TSS 描述符（TSS Descriptor）：
   - 功能： 定义 TSS 的段选择子、基地址、段限长和属性等信息。

- 作用： 位于 GDT 中，描述特定任务的 TSS 位置和属性，供任务寄存器（TR）加载和任务切换时使用。

4. 任务寄存器（TR，Task Register）：
   - 功能： 保存当前任务的 TSS 选择子和相关属性。
   - 作用： 指向当前正在执行的任务的 TSS，处理器通过 TR 访问当前任务的上下文信息。
5. EFLAGS 寄存器中的 NT 标志位（Nested Task Flag）：
   - 功能： 指示当前任务是否是嵌套任务。
   - 作用： 在任务切换时，处理器根据 NT 标志位决定是否在 TSS 中设置链接字段，以支持任务的嵌套调用和返回。

在保护模式下，至少需要为一个任务创建 TSS 和对应的 TSS 描述符，并使用 `LTR`（Load Task Register）指令将 TSS 的段选择子加载到任务寄存器中，以初始化任务管理机制。

4.2.1 任务状态段 Task-State Segment (TSS)

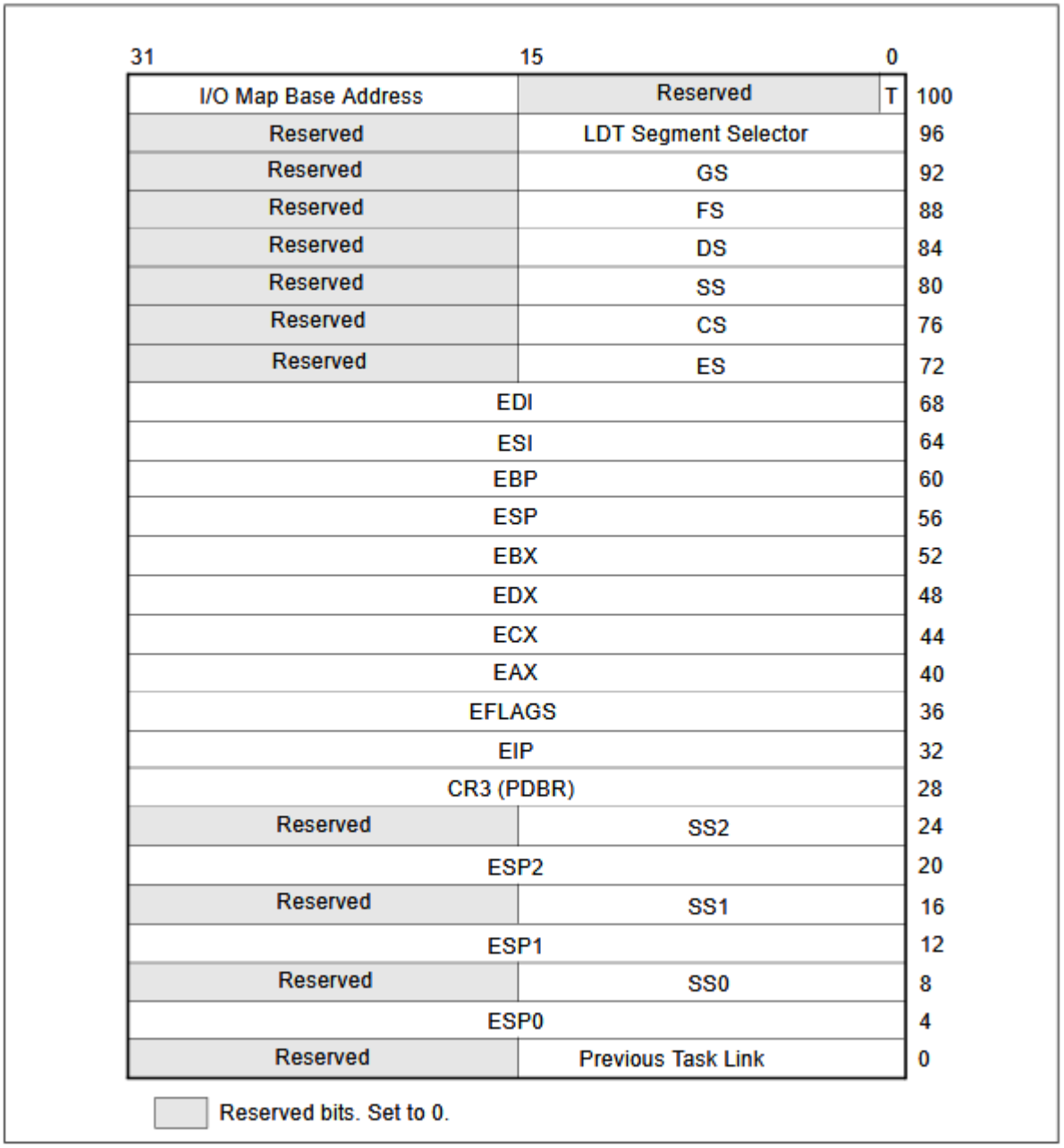

Figure 7-2. 32-Bit Task-State Segment (TSS)

任务状态段（TSS，Task-State Segment）是 IA-32 架构中用于保存任务上下文信息的系统段。在任务切换时，处理器将当前任务的状态保存到其 TSS 中，并从新任务的 TSS 中恢复状态。TSS 的结构包括动态字段和静态字段：

动态字段：

- 通用寄存器字段： 保存 EAX、ECX、EDX、EBX、ESP、EBP、ESI 和 EDI 寄存器的状态。
- 段选择子字段： 保存 ES、CS、SS、DS、FS 和 GS 寄存器的段选择子。
- EFLAGS 寄存器字段： 保存 EFLAGS 寄存器的状态。
- EIP（指令指针）字段： 保存 EIP 寄存器的状态。
- 前任务链接字段： 包含前一个任务的 TSS 段选择子，用于支持通过 `IRET` 指令返回到前一个任务。

静态字段：

- LDT 段选择子字段： 包含任务的局部描述符表（LDT）的段选择子。
- CR3 控制寄存器字段： 包含页目录的基地址，用于分页机制下的地址转换。
- 特权级 0、1 和 2 的堆栈指针字段： 每个特权级的堆栈指针由段选择子（SS0、SS1、SS2）和偏移量（ESP0、ESP1、ESP2）组成。
- T（调试陷阱）标志： 位于 TSS 的字节 100 的第 0 位，设置时在任务切换到该任务时引发调试异常。
- I/O 位图基地址字段： 包含从 TSS 基址到 I/O 许可位图和中断重定向位图的偏移。

在使用分页机制时，应注意以下事项：

- 避免在 TSS 的前 104 字节中跨越页边界，以确保处理器正确执行地址转换。
- 前一个任务的 TSS、当前任务的 TSS 以及描述符表的页应标记为可读/可写。
- 为提高任务切换速度，确保包含这些结构的页在任务切换前已加载到内存中。

4.2.2 TSS 描述符

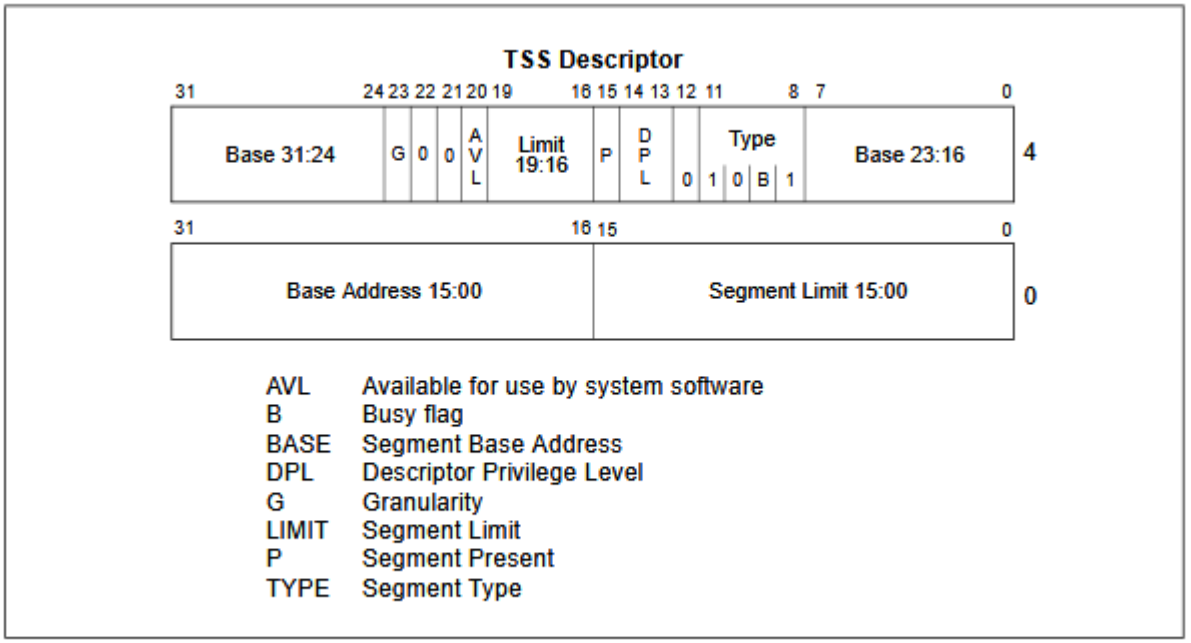

Figure 7-3. TSS Descriptor

在 IA-32 架构中，任务状态段（TSS）由位于全局描述符表（GDT）中的 TSS 描述符定义。TSS 描述符的格式如下：

- 基地址（Base Address）： 指定 TSS 在内存中的起始地址。
- 段限长（Limit）： 定义 TSS 的大小。
- 类型（Type）： 指示 TSS 的状态：

    - `1001B`：非忙（Inactive）任务。
    - `1011B`：忙（Busy）任务。
- DPL（Descriptor Privilege Level）：描述符的特权级，控制对该 TSS 的访问权限。
- G（Granularity）：粒度标志，指示段限长的单位：
    - `0`：段限长以字节为单位。
    - `1`：段限长以 4KB 为单位。
- P（Present）：存在标志，指示 TSS 是否存在于内存中。

需要注意的是，TSS 描述符只能位于 GDT 中，不能放置于局部描述符表（LDT）或中断描述符表（IDT）中。尝试使用段选择子访问 LDT 中的 TSS 会导致一般保护异常（#GP），而在 IRET 指令期间会导致无效 TSS 异常（#TS）。此外，尝试将 TSS 的段选择子加载到段寄存器中也会引发一般保护异常。

忙标志（B）用于指示任务的状态。处理器使用该标志检测对已中断任务的调用尝试。为了确保每个任务只有一个忙标志，建议每个 TSS 仅有一个对应的 TSS 描述符。

当 G 标志为 0 时，32 位 TSS 描述符的段限长必须大于或等于 0x67（即 103 字节），这是 TSS 的最小大小。如果段限长小于 0x67，尝试切换到该任务会引发无效 TSS 异常（#TS）。如果 TSS 包含 I/O 许可位图或操作系统存储了额外的数据，则需要更大的段限长。处理器在任务切换时不会检查段限长是否大于 0x67，但在访问 I/O 许可位图或中断重定向位图时会进行检查。

任何特权级（CPL）小于或等于 TSS 描述符的 DPL 的程序或过程都可以通过调用或跳转来调度任务。在大多数系统中，TSS 描述符的 DPL 设置为小于 3 的值，以限制任务切换仅由特权软件执行。然而，在多任务应用中，某些 TSS 描述符的 DPL 可能设置为 3，以允许在应用程序（或用户）特权级别进行任务切换。

4.2.3 任务寄存器

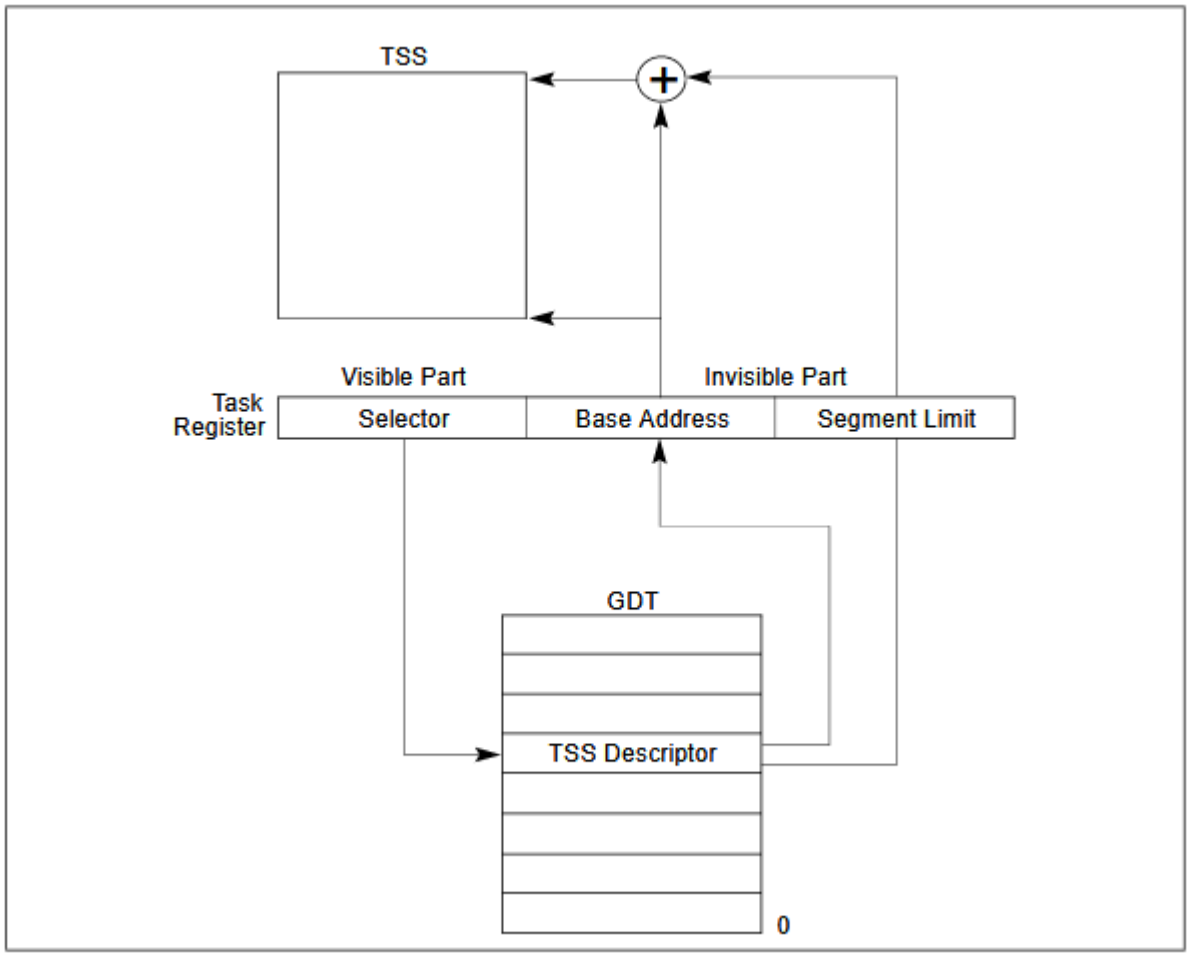

Figure 7-5. Task Register

任务寄存器（Task Register，TR）是 IA-32 架构中用于指示当前任务的关键寄存器。它包含当前任务的任务状态段（TSS）的段选择子和完整的段描述符信息，包括基地址、段限长和描述符属性。这些信息从全局描述符表（GDT）中的 TSS 描述符复制而来。

任务寄存器由可见部分和不可见部分组成：

- 可见部分：
  - 段选择子： 指向 GDT 中的 TSS 描述符。
- 不可见部分：
  - 段描述符缓存： 包含 TSS 的基地址、段限长和描述符属性。

处理器使用不可见部分来缓存 TSS 的段描述符信息，以提高任务执行效率。

与任务寄存器相关的指令有：

- LTR（Load Task Register）：
  - 功能： 将指定的段选择子加载到任务寄存器的可见部分，并使用对应的 TSS 描述符信息初始化不可见部分。
  - 特权级别： 仅在当前特权级（CPL）为 0 时可执行，是一条特权指令。

    - 用途： 通常在系统初始化期间使用，以设置任务寄存器的初始值。
- STR（Store Task Register）：
    - 功能： 将任务寄存器的可见部分存储到通用寄存器或内存中。
    - 特权级别： 可在任何特权级别执行，通常由操作系统软件使用，以标识当前正在运行的任务。

在处理器上电或复位时，任务寄存器的段选择子和基地址被设置为默认值 0，段限长被设置为 0xFFFF。

4.2.4 任务门描述符 Task-Gate Descriptor

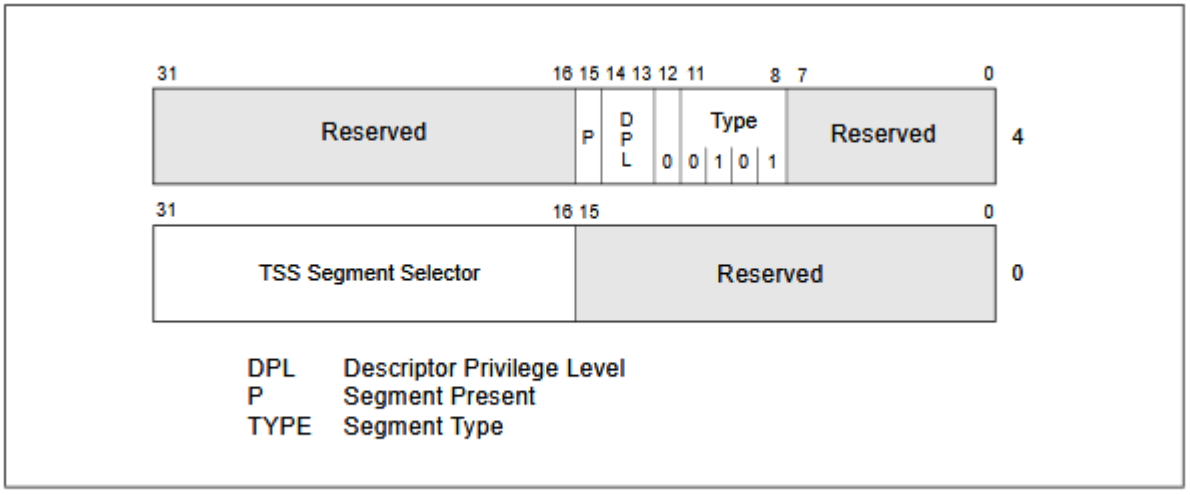

Figure 7-6. Task-Gate Descriptor

任务门描述符（Task-Gate Descriptor）是 IA-32 架构中用于间接、安全地引用任务的结构。它可以位于全局描述符表（GDT）、局部描述符表（LDT）或中断描述符表（IDT）中。任务门描述符的主要功能是提供对任务的受控访问，允许通过任务门进行任务切换。

任务门描述符包含一个指向 GDT 中 TSS 描述符的段选择子。该段选择子的请求特权级（RPL）字段在任务切换过程中不被使用。任务门描述符的描述符特权级（DPL）控制对任务的访问权限。当程序或过程通过任务门进行调用或跳转时，调用者的当前特权级（CPL）和指向任务门的选择子的 RPL 必须小于或等于任务门描述符的 DPL。需要注意的是，当使用任务门时，目标 TSS 描述符的 DPL 不被使用。

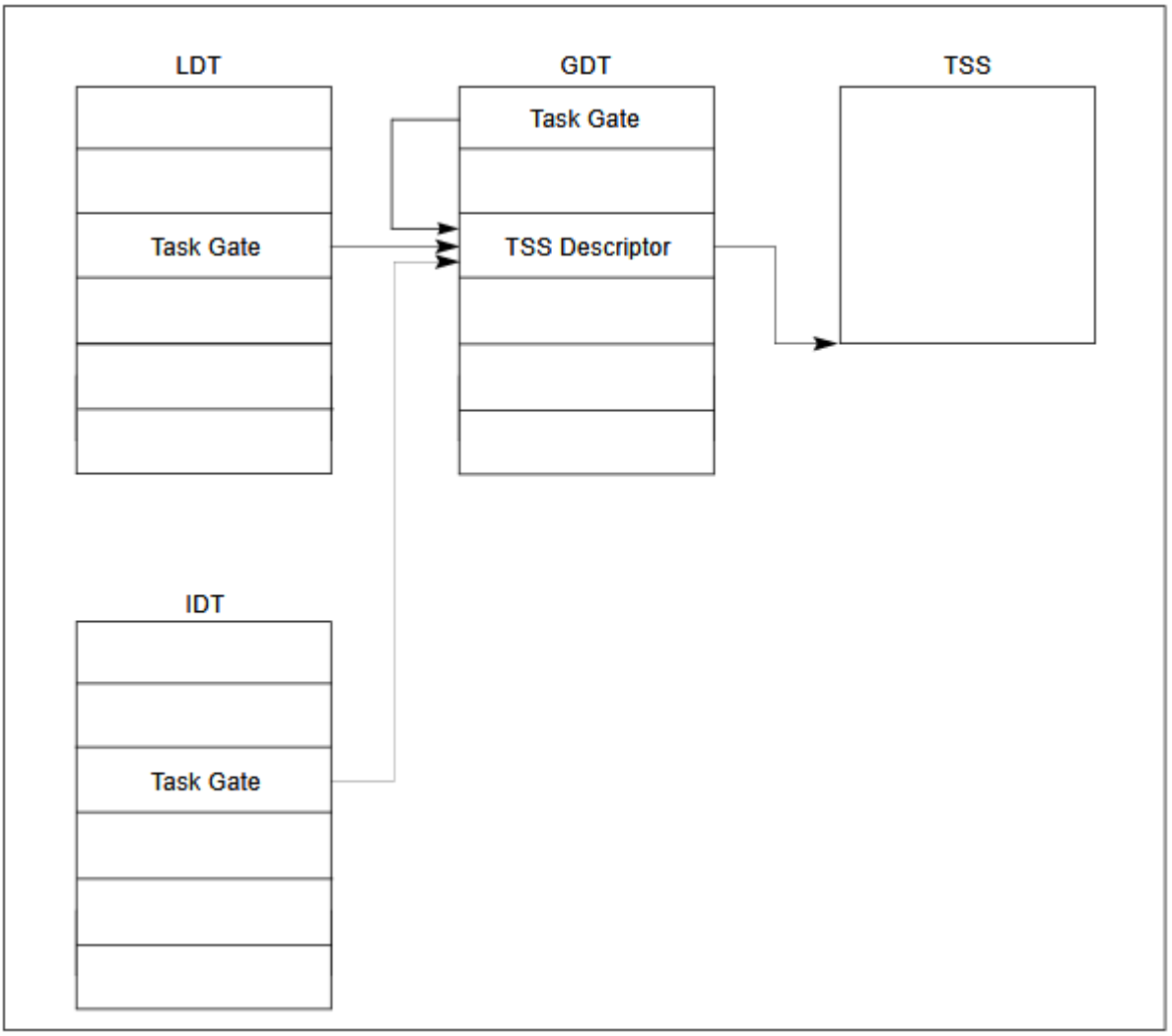

Figure 7-7. Task Gates Referencing the Same Task

任务门描述符满足以下需求：

- 唯一的忙标志： 由于任务的忙标志存储在 TSS 描述符中，每个任务应只有一个 TSS 描述符。然而，可以有多个任务门引用同一个 TSS 描述符。
- 选择性访问任务： 任务门可以位于 LDT 中，并具有不同于 TSS 描述符的 DPL。这样，特权级不足以直接访问 GDT 中 TSS 描述符的程序或过程（通常 TSS 描述符的 DPL 为 0）可以通过具有更高 DPL 的任务门访问任务。
- 中断或异常由独立任务处理： 任务门也可以位于 IDT 中，允许中断和异常由特定的任务处理。当中断或异常向量指向任务门时，处理器切换到指定的任务。

通过任务门，操作系统可以更灵活地控制对任务的访问权限，确保系统的安全性和稳定性。

4.3. 任务切换

4.3.1 什么时候发生任务切换 ？

任务切换是指处理器将执行从一个任务转移到另一个任务的过程。在以下情况下，处理器会进行任务切换：

1. 执行 JMP 或 CALL 指令：当前程序、任务或过程执行跳转（JMP）或调用（CALL）指令，目标是全局描述符表（GDT）中的任务状态段（TSS）描述符。
2. 通过任务门进行跳转或调用：当前程序、任务或过程执行跳转或调用指令，目标是 GDT 或当前局部描述符表（LDT）中的任务门描述符。
3. 中断或异常处理：中断或异常向量指向中断描述符表（IDT）中的任务门描述符。
4. 执行 IRET 指令：当前任务执行中断返回（IRET）指令，且 EFLAGS 寄存器中的 NT（嵌套任务）标志被设置。

4.3.2 发生任务切换时，处理器会执行哪些操作？

当发生任务切换时，处理器会执行以下操作：

1. 获取新任务的 TSS 段选择子：从 JMP 或 CALL 指令的操作数、任务门或先前任务链接字段中获取新任务的 TSS 段选择子。
2. 权限检查：验证当前任务是否被允许切换到新任务。对于 JMP 和 CALL 指令，当前任务的特权级（CPL）和新任务段选择子的请求特权级（RPL）必须小于或等于被引用的 TSS 描述符或任务门的描述符特权级（DPL）。对于异常、中断和 IRET 指令，任务切换不受目标任务门或 TSS 描述符的 DPL 限制。
3. 验证新任务的 TSS 描述符：检查新任务的 TSS 描述符是否存在且具有有效的段限长（至少为 0x67）。
4. 检查新任务的状态：确认新任务是否可用（对于 CALL、JMP、异常或中断）或忙碌（对于 IRET 返回）。
5. 内存分页检查：确保当前 TSS、新 TSS 和任务切换中使用的所有段描述符已加载到系统内存中。
6. 更新当前任务的 TSS 描述符：如果任务切换由 JMP 或 IRET 指令发起，处理器清除当前任务 TSS 描述符中的忙标志（B）；如果由 CALL 指令、异常或中断发起，保持忙标志设置。
7. 更新 EFLAGS 寄存器中的 NT 标志：如果任务切换由 IRET 指令发起，处理器清除暂存的 EFLAGS 寄存器映像中的 NT 标志；如果由 CALL 或 JMP 指令、异常或中断发起，保持 NT 标志不变。
8. 保存当前任务的状态：将当前任务的状态信息（包括通用寄存器、段选择子、EFLAGS 和 EIP 寄存器）保存到当前任务的 TSS 中。
9. 加载新任务的状态：将新任务的状态信息（包括 LDTR、CR3、EFLAGS、EIP、通用寄存器和段选择子）从新任务的 TSS 加载到处理器中。

10. 更新新任务的 TSS 描述符：如果任务切换由 CALL 指令、JMP 指令、异常或中断发起，处理器设置新任务 TSS 描述符中的忙标志（B）；如果由 IRET 指令发起，保持忙标志设置。
11. 加载任务寄存器：将新任务的 TSS 段选择子和描述符加载到任务寄存器中。
12. 开始执行新任务：处理器开始执行新任务，从新任务的 EIP 寄存器指向的指令开始。

4.3.3 中断或异常向量指向 IDT 表中的中断门或陷阱门，会发生任务切换吗？

当中断或异常向量指向 IDT 中的中断门或陷阱门时，处理器不会进行任务切换。中断门和陷阱门用于调用中断或异常处理程序，而不涉及任务切换。只有当中断或异常向量指向任务门时，处理器才会进行任务切换。

4.4. 任务链

Table 7-1. Exception Conditions Checked During a Task Switch

Condition Checked	Exception[1]	Error Code Reference[2]
Segment selector for a TSS descriptor references the GDT and is within the limits of the table.	#GP #TS (for IRET)	New Task's TSS
TSS descriptor is present in memory.	#NP	New Task's TSS
TSS descriptor is not busy (for task switch initiated by a call, interrupt, or exception).	#GP (for JMP, CALL, INT)	Task's back-link TSS
TSS descriptor is not busy (for task switch initiated by an IRET instruction).	#TS (for IRET)	New Task's TSS
TSS segment limit greater than or equal to 108 (for 32-bit TSS) or 44 (for 16-bit TSS).	#TS	New Task's TSS
Registers are loaded from the values in the TSS.		
LDT segment selector of new task is valid [3].	#TS	New Task's LDT
Code segment DPL matches segment selector RPL.	#TS	New Code Segment
SS segment selector is valid [2].	#TS	New Stack Segment
Stack segment is present in memory.	#SS	New Stack Segment
Stack segment DPL matches CPL.	#TS	New stack segment
LDT of new task is present in memory.	#TS	New Task's LDT
CS segment selector is valid [3].	#TS	New Code Segment
Code segment is present in memory.	#NP	New Code Segment
Stack segment DPL matches selector RPL.	#TS	New Stack Segment
DS, ES, FS, and GS segment selectors are valid [3].	#TS	New Data Segment
DS, ES, FS, and GS segments are readable.	#TS	New Data Segment

Table 7-1. Exception Conditions Checked During a Task Switch (Contd.)

Condition Checked	Exception[1]	Error Code Reference[2]
DS, ES, FS, and GS segments are present in memory.	#NP	New Data Segment
DS, ES, FS, and GS segment DPL greater than or equal to CPL (unless these are conforming segments).	#TS	New Data Segment

NOTES:

1. #NP is segment-not-present exception, #GP is general-protection exception, #TS is invalid-TSS exception, and #SS is stack-fault exception.
2. The error code contains an index to the segment descriptor referenced in this column.
3. A segment selector is valid if it is in a compatible type of table (GDT or LDT), occupies an address within the table's segment limit, and refers to a compatible type of descriptor (for example, a segment selector in the CS register only is valid when it points to a code-segment descriptor).

- 任务切换（Task Switching）
  - 任务切换是指处理器从一个任务的执行切换到另一个任务。每个任务有一个独立的任务状态段（TSS），其中保存了该任务的寄存器状态和相关信息。任务切换通常由指令（如 `CALL`、`JMP`）、中断或异常触发。
- 任务嵌套（Nested Tasks）
  - 当一个任务在执行过程中触发了另一个任务，并需要在完成后返回到前一个任务时，就形成了任务嵌套。嵌套任务通过 `NT` 标志和 `Previous Task Link` 字段实现。
  - 嵌套任务的特性：
    - 当前任务的 TSS 被保存到新任务的 TSS 中。
    - `EFLAGS.NT` 被设置为 1，用于标记嵌套。
- 关键标志位
  - NT 标志（Nested Task Flag）
    - 位于 `EFLAGS` 寄存器中，用于标记任务是否嵌套。
  - Previous Task Link
    - 位于 TSS 中，用于存储前一任务的 TSS 段选择符，指向被嵌套的任务。
  - Busy 标志
    - 位于任务段描述符中，标记任务是否正在执行，防止递归调用或任务的重复调度。
- 任务切换指令
  - CALL：导致任务嵌套。
  - JMP：不会导致任务嵌套。
  - IRET：用于返回前一任务。

4.4.1 如何判断任务是否嵌套 ？

判断任务是否嵌套主要通过 `EFLAGS.NT` 标志位：

- 如果 `EFLAGS.NT=1`，则表示当前任务是嵌套任务。
- 如果 `EFLAGS.NT=0`，则当前任务为非嵌套任务。

4.4.2 什么情况会发生任务嵌套 ？

任务嵌套发生在以下几种情况下：

1. CALL 指令：
    - 当通过 `CALL` 指令触发任务切换时，当前任务被标记为嵌套任务。
2. 中断或异常：
    - 当中断或异常导致任务切换时，当前任务上下文被保存，新任务被标记为嵌套任务。

嵌套任务切换的过程：

- 当前任务的段选择符被保存到新任务 TSS 的 `Previous Task Link` 字段。
- 新任务的 `EFLAGS.NT` 标志被设置为 1。

注意：通过 `JMP` 指令进行的任务切换不会导致嵌套，`Previous Task Link` 字段不会更新，`EFLAGS.NT` 也保持为 0。

4.4.3 任务嵌套时修改了哪些标志位?

任务嵌套会修改以下标志位：

1. NT 标志 (EFLAGS.NT) ：
    - 设置为 1，标记当前任务是嵌套任务。
2. Previous Task Link 字段：
    - 更新为当前任务的段选择符，用于记录嵌套链中的上一个任务。
3. Busy 标志：
    - 新任务的 `Busy` 标志被设置为 1，表示任务正在执行。
    - 当前任务的 `Busy` 标志保持不变。

4.4.4 任务嵌套时，如何返回前一任务?

返回前一任务的过程通过 `IRET` 指令实现，具体步骤如下：

1. 检查 `EFLAGS.NT`：
    - 如果 `EFLAGS.NT=1`，处理器认为当前任务是嵌套任务。
2. 读取 `Previous Task Link`：

- 处理器从当前任务 TSS 的 `Previous Task Link` 字段中读取前一任务的 TSS 段选择符。

3. 切换到前一任务：
   - 使用 `Previous Task Link` 字段中的段选择符加载前一任务的 TSS。
   - 恢复前一任务的上下文（包括寄存器、EFLAGS 等）。
4. 清除 `NT` 标志：
   - 在返回到前一任务后，处理器会将 `EFLAGS.NT` 标志清除，表示任务已退出嵌套状态。

4.5. 任务地址空间

4.5.1 什么是任务地址空间 ？

任务地址空间是指一个任务（或进程）在执行过程中可以访问的内存区域，包括代码段、数据段、堆栈段和系统段等。

4.5.2 任务地址空间包括什么？

任务地址空间由以下部分组成：

- 代码段：存放可执行指令的区域。
- 数据段：存放全局变量和静态数据的区域。
- 堆栈段：用于函数调用和局部变量的存储。
- 系统段：包含任务状态段（TSS）等系统相关信息。

4.5.3 了解把任务映射到线性和物理地址空间的方

在将任务映射到线性地址空间和物理地址空间时，有两种主要方法：

1. 共享线性到物理地址空间映射：所有任务共享一个线性地址空间到物理地址空间的映射。这种方式在未启用分页机制时是唯一选择，因为所有线性地址直接映射到相同的物理地址。启用分页后，可以通过为所有任务使用相同的页目录来实现共享映射。
2. 独立的线性地址空间映射：每个任务拥有独立的线性地址空间，并映射到物理地址空间。这通过为每个任务分配不同的页目录实现。由于在任务切换时会加载控制寄存器 CR3（即页目录基址寄存器 PDBR），因此每个任务可以有不同的页目录，从而实现独立的线性地址空间映射。

4.5.4 了解任务逻辑地址空间，及如何在任务之间共享数据的方法？

任务的逻辑地址空间是指任务在执行过程中使用的逻辑地址集合。为了在任务之间共享数据，可以采用以下方法：

- 通过全局描述符表（GDT）中的段描述符：所有任务都可以访问 GDT 中的段描述符。如果某些段描述符指向的线性地址空间被映射到所有任务共享的物理地址空间区域，那么这些段中的数据和代码就可以被所有任务共享。
- 通过共享的局部描述符表（LDT）：多个任务可以共享同一个 LDT。如果共享的 LDT 中的段描述符指向的线性地址空间被映射到共享的物理地址空间区域，那么这些段中的数据和代码就可以在共享该 LDT 的任务之间共享。这种方法比通过 GDT 共享更具选择性，因为共享可以限定在特定的任务之间。
- 通过不同 LDT 中的段描述符映射到相同的线性地址空间：如果不同任务的 LDT 中的段描述符指向相同的线性地址空间，并且这些线性地址空间被映射到相同的物理地址空间区域，那么这些段中的数据和代码就可以在这些任务之间共享。这种方法被称为别名（alias），比前两种方法更具选择性，因为共享可以限定在特定的段和任务之间。