

任务管理-读书笔记

2021113117-王宇轩

1. 任务管理概述

任务是处理器可以分派、执行和暂停的工作单元，可以用于执行程序、任务或进程、操作系统服务实例程序、中断或异常处理程序，和内核或执行实例程序。

IA-32架构提供了一种保存任务状态、分派任务执行及任务切换的机制。当运行于保护模式，**所有处理器的执行都在任务中发生**，即使简单的系统也要至少定义一个任务，更复杂的系统可以使用处理器的任务管理设备来支持多任务应用。

Q&A

1. **80x86 提供了哪些硬件支持？** 如任务状态段TSS、任务寄存器TR和用于任务切换的指令，如 `LJMP`、`LRET` 等。
2. **描述符表中与任务相关的描述符有哪些？** TSS描述符，描述TSS的属性和地址；LDT描述符，描述任务的局部描述符表LDT。
3. **任务切换与过程调用的区别是什么？** 任务切换是指从一个任务切换到另一个任务的过程，用于实现多任务调度和并发执行；过程调用是指在同一个任务内部执行的过程或函数之间的跳转和返回，用于实现程序的模块化和函数调用。

1.1 任务的结构

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）。

任务执行空间由**代码段、堆栈段和一个或多个数据段**组成。如果**操作系统或执行程序使用了处理器的特权级别保护机制**，那么任务执行空间还将为每个特权级别提供单独的堆栈。

****TSS ****指定组成任务执行空间的段，并为任务状态信息提供存储的位置。在多任务系统中，TSS还提供了一种链接任务的机制。

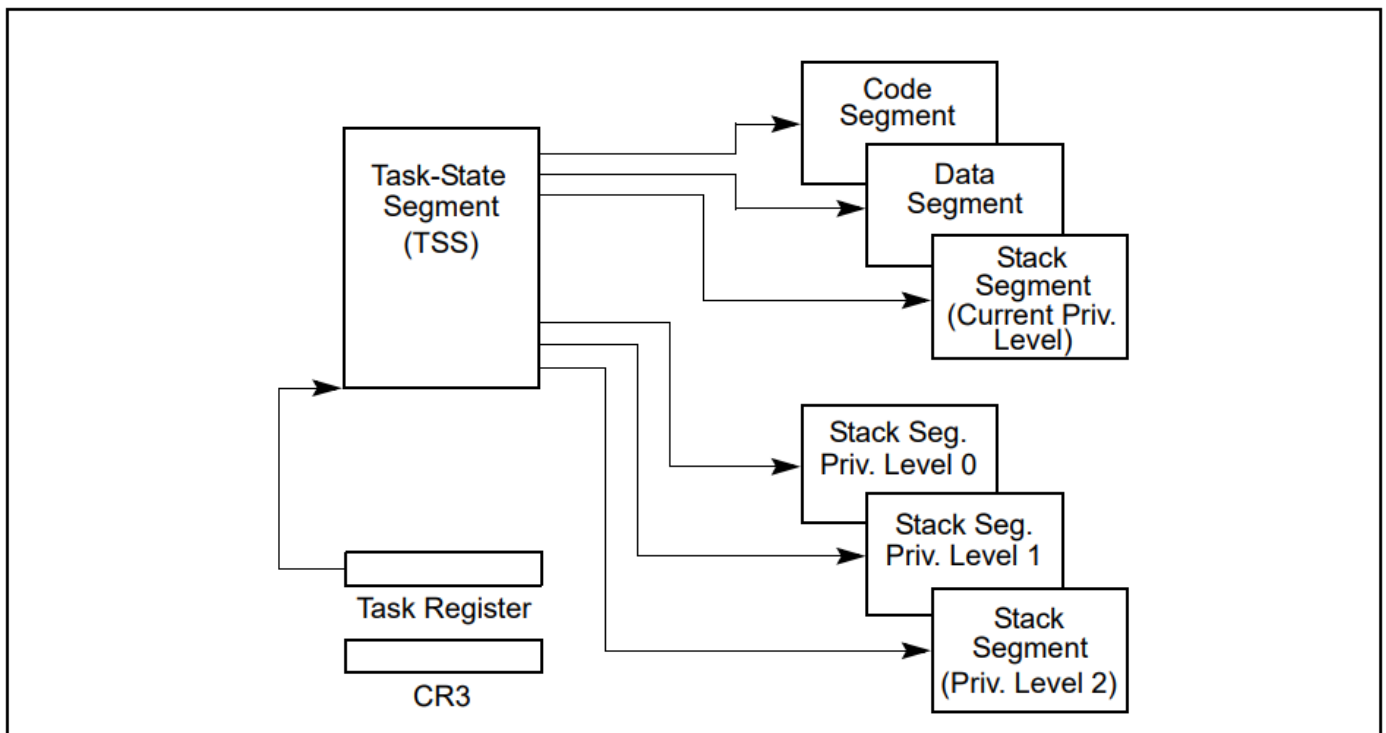


Figure 7-1. Structure of a Task

****任务由其TSS的段选择子标识。****当一个任务载入处理执行时，其TSS的段选择器、基址、界限和段描述符属性被载入任务寄存器TR中。如果为任务实现了分页，则任务使用的页目录的基址被加载到控制寄存器CR3中。

1.2 任务状态

任务状态由以下项目定义：

- 当前任务的执行空间，由段寄存器（CS、DS、SS、ES、FS 和 GS）中的段选择子定义。
- 通用寄存器的状态。
- EFLAGS 寄存器的状态。
- EIP 寄存器的状态。
- 控制寄存器CR3的状态。存储了页表的物理地址。
- 任务寄存器的状态。
- LDTR寄存器的状态。
- I/O 映射基地址和I/O 映射（包含在 TSS 中）。
- 指向特权级为 0、1 和 2 的堆栈的堆栈指针（包含在TSS 中）。
- 到先前执行的任务的链接（包含在TSS 中）。

处理器在分派任务之前，除了任务寄存器的状态，以上其他所有项都包含在任务的 TSS 中。此外，TSS 中并不包含 LDTR 寄存器的完整内容，而仅仅包含 LDT 的段选择器。

这些内容被包含在任务状态中是为了能够准确保存和恢复任务的执行状态。通过保存这些信息，可以在任务切换之后正确地恢复任务的执行位置、数据和状态，实现任务的无缝切换和并发执行。同时，这些内容也提供了对任务的管理和调度所需的信息，如内存管理、地址转换和权限控制等。

1.3 任务的执行

软件或处理器通过以下方式之一来分派任务去执行：

- 使用 `CALL` 指令显式调用任务。
- 使用 `JMP` 指令显式跳转至任务。
- 对中断处理程序任务的隐式调用（由处理器执行）。
- 对异常处理程序任务的隐式调用。
- 当设置 `EFLAGS` 寄存器中的 `NT` 标志时从某处返回（使用 `IRET` 指令启动）。

以上所有用于分派任务的方法都使用**指向任务门或TSS的段选择子**来标识要分派的任务。当使用 `CALL` 或 `JMP` 指令调度任务时，指令中的选择子可直接选用TSS或选用保存TSS选择子的任务门。当分派任务来处理中断或异常时，中断或异常的IDT表项必须包含一个任务门，其中存有中断或异常处理程序TSS的选择子。

当任务被分派执行时，一次任务切换在当前运行的任务和被分派任务间发生。任务切换期间，当前执行任务的执行环境（称为任务的状态或**上下文context**）保存在其 TSS 中，并暂停该任务执行。被分派任务的上下文随后载入处理器，并从新加载的 `EIP` 寄存器指向的指令开始执行被分派任务。如果自系统上次初始化以来，任务还没有被运行过，那么 `EIP` 将指向任务代码的第一条指令；否则，`EIP` 将指向任务上次激活时执行的最后一条指令之后的下一条指令。

如果当前执行的任务（发起调用的任务）调用正在被分派的任务（被调用的任务），那么调用任务的 TSS 段选择子存储在调用任务的 TSS 中，以提供返回调用任务的链接。

对于所有 IA-32 处理器，任务都不是递归的，即任务不能调用或跳转到自身。

中断或异常可以通过任务切换到处理程序任务来处理。在这里，处理器执行一次任务切换以处理中断或异常，并在从中断处理程序任务或异常处理程序任务返回时自动切换回被中断的任务。该机制也可以处理中断任务期间发生的中断。

作为任务切换的一部分，处理器还可以切换到另一个 LDT，允许每个任务对基于 LDT 的段具有不同的逻辑到物理地址映射。页目录基址寄存器 (`CR3`) 也会在任务切换时被重新加载，从而允许每个任务都有自己的一组页表。这些保护措施有助于隔离任务，并防止它们相互干扰。

如果不使用保护机制，那么处理器将不提供任务之间的保护，即便对于使用多个特权级别进行保护的操作系统来说也是如此。一个以特权级别为 3 运行的任务如果使用与另外的特权级别为 3 的任务相同的 LDT 和页表，就可以访问它们的代码、损坏它们数据以及堆栈。

可以选择使用任务管理工具来处理多任务应用程序。多任务处理可以在软件中处理，每个软件定义的任务都在单个 IA-32 架构任务的上下文中执行。

Q&A

1. **Linux 0.00 用的是哪种方式？** 使用 `LJMP` 指令显式跳转至任务。

2. 任务的数据结构

处理器定义了以下五种数据结构来处理与任务相关的活动：

- 任务状态段 Task-State Segment (TSS)。
- 任务门描述符。
- TSS描述符。
- 任务寄存器TR。
- ELFLAGS寄存器中的NT标志位。

保护模式下，至少为一个任务创建一个TSS和一个TSS描述符，且TSS的段选择子必须被载入TR（使用 `LTR` 指令）。

2.1 任务状态段 Task-State Segment (TSS)

恢复任务所需的处理器状态信息保存在一个称为任务状态段（task-state segment, TSS）的**系统段**中。下图显示了为32位CPU设计的任务的TSS格式。TSS的字段分为两大类：动态字段和静态字段。

31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0


 Reserved bits. Set to 0.

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

当任务在任务切换期间挂起时，处理器更新动态字段。**动态字段如下：**

- **通用寄存器字段** (General-purpose register fields) : EAX、ECX、EDX、EBX、ESP、EBP、ESI 和 EDI 寄存器在任务切换之前的状态。
- **段选择器字段** (Segment selector fields) : 任务切换之前存储在ES、CS、SS、DS、FS 和 GS 寄存器的段选择子。
- **EFLAGS 寄存器字段** (EFLAGS register field) : EFLAGS 寄存器在任务切换之前的状态。
- **EIP (指令指针) 字段** (EIP (instruction pointer) field) : 任务切换之前 EIP 寄存器的状态。

- **前一个任务链接字段** (Previous task link field) : 包含前一个任务的TSS 的段选择子 (该段选择子在由调用、中断或异常启动的任务切换时更新)。该字段 (有时称为反向链接字段) 允许使用 `IRET` 指令将任务切换回先前的任务。

处理器读取静态字段, 但通常不进行修改, 这些字段在创建任务时建立。静态字段如下:

- **LDT 段选择子字段** (LDT segment selector field) : 包含任务的 LDT 的段选择子。
- **CR3 控制寄存器字段** (CR3 control register field) : 包含任务要使用的页面目录的物理基地址。控制寄存器 CR3 也称为页目录基址寄存器 (PDBR)。
- **特权级别 0、1 和 2 的堆栈指针字段** (Privilege level -0, -1, and -2 stack pointer fields) : 这些堆栈指针由一个逻辑地址组成, 该逻辑地址由堆栈段 (SS0、SS1 和 SS2) 的段选择器和堆栈中的偏移量 (ESP0、ESP1 和 ESP2) 组成。请注意, 这些字段中的值对于特定任务是静态的, 但如果任务中发生了堆栈切换, SS 和 ESP 值将发生变化。
- **T (调试陷阱debug trap) 标志** (debug trap flag, 第 100 字节的第 0 位) : 为1时, T 标志会导致处理器在从其他任务切换到此任务时引发调试异常。
- **I/O 映射基址字段** (I/O map base address field) : 该字段包含从 TSS 基址到 I/O 许可位图和中断重定向位图的 16 位偏移量。如果存在, 这些映射将存储在 TSS 中的更高地址处。I/O 映射基址指向 I/O 权限位图的开始和中断重定向位图的结束位置。

若启用分页:

- 避免在任务切换期间处理器读取的 TSS 部分中 (前 104 个字节) 设置页边界。如果该区域中出现边界, 处理器可能无法正确执行地址转换。在任务切换期间, 处理器读取和写入每个 TSS 的前 104 个字节 (使用从 TSS 第一个字节的物理地址开始的连续物理地址)。因此, 在 TSS 访问开始后, 如果 104 字节中的一部分在物理上不连续, 则处理器将访问不正确的信息, 并且不会产生页错误异常 (page-fault exception)。
- 与前一个任务的TSS、当前任务的TSS 和每一个的描述符表条目相对应的页都应该被标记为读/写。
- 如果包含这些数据结构的页在任务切换开始之前就存在于内存中, 则任务切换的执行速度会更快。

2.2 TSS描述符

与所有其他段一样, TSS 由段描述符定义。下图展示了 TSS 描述符的格式。TSS 描述符只可放在 GDT 中, 而不能放在 LDT 或 IDT 中。

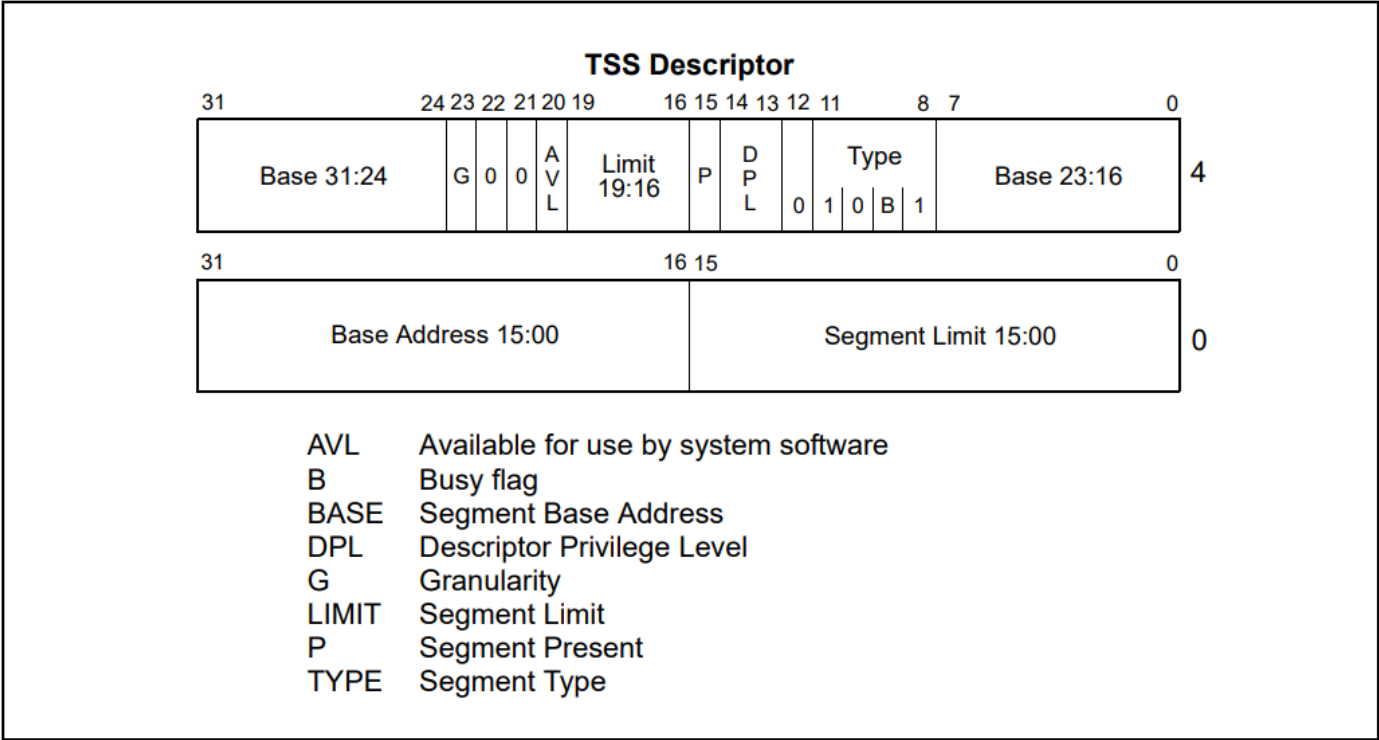


Figure 7-3. TSS Descriptor

如果程序尝试使用 TI 标志为1（表示段选择子所引用的描述符位于LDT中）的段选择子访问 TSS，将会导致在执行 CALL 和 JMP 指令期间生成一般保护异常（#GP）；它会在 IRET 期间导致无效 TSS 异常 (invalid TSS exception, #TS)。如果尝试将 TSS 的段选择器加载到段寄存器中，也会生成一般保护异常（#GP）。

Type字段中的忙标志（busy flag, B）指示任务是否忙：一个忙的任务是指该任务当前正在运行或者被挂起。Type字段的值如果为 1001B，则表示该任务是一个非活动任务；1011B 则表示任务是忙的。任务不是递归的。处理器使用B标志来检测对已被中断执行的任务的调用。为了确保只有一个B标志与任务相关联，每个 TSS 都应该只有一个指向它的 TSS 描述符。

Base、limit 和 DPL 字段以及 granularity 和 present 标志的功能类似于它们在数据段描述符中的作用。当 32 位 TSS 的 TSS 描述符中的 G 标志为 0 时，限制字段的值必须等于或大于67H，比 TSS 的规定最小尺寸小一个字节。如果尝试切换到 TSS 描述符限制小于 67H 的任务，则会生成无效 TSS 异常#TS。如果包含 I/O 权限位图或操作系统存储额外数据，那么需要更大的限制。处理器不会在任务切换时检查大于67H的界限，但它会在访问 I/O 权限位图或中断重定向位图时检查该值。

任何可以访问 TSS 描述符的程序或过程（即CPL在数值上小于等于TSS 描述符的DPL）都可以通过调用或跳转来分派任务。

在大多数系统中，TSS 描述符的 DPL 被设置为小于 3 的值。因此，只有特权级软件（privileged software）才能执行任务切换。但是，在多任务应用程序中，某些 TSS 描述符的 DPL 可能会设置为 3，以允许在应用程序（或用户）特权级别进行任务切换操作。

64位模式下不支持任务切换，但TSS描述符依然存在，TSS的格式如下图。此时，TSS描述符被扩展到16字节，此扩展也用于64位模式的LDT描述符。

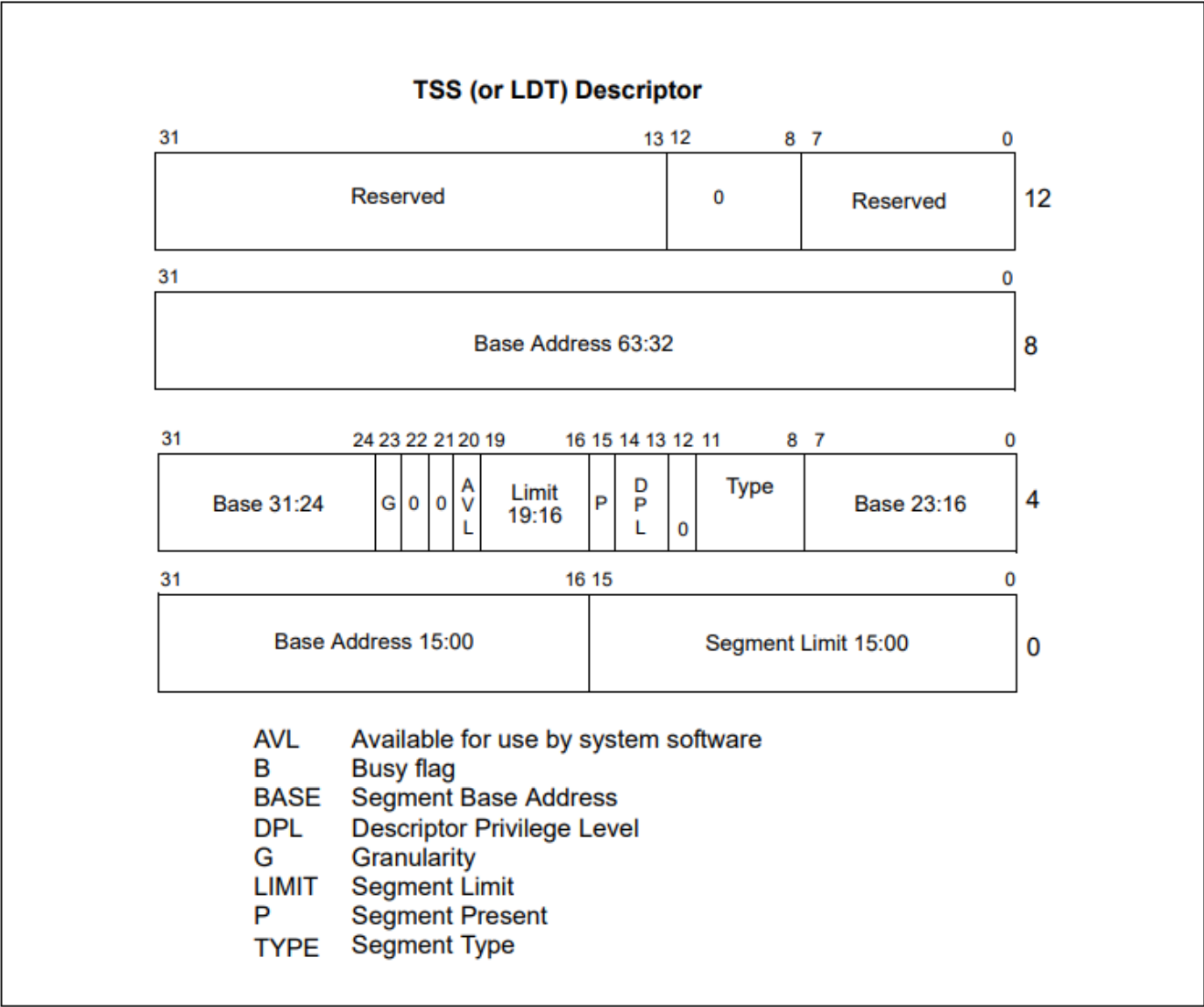


Figure 7-4. Format of TSS and LDT Descriptors in 64-bit Mode

2.3 任务寄存器

任务寄存器保存当前任务的 TSS 的 16 位段选择子和整个段描述符（32 位基地址（IA-32e 模式下基地址为 64 位）、16 位段界限和描述符属性）。这些信息是从当前任务在 GDT 中的 TSS 描述符复制而来的。下图展示了处理器是如何（使用任务寄存器中的信息）访问 TSS 的。

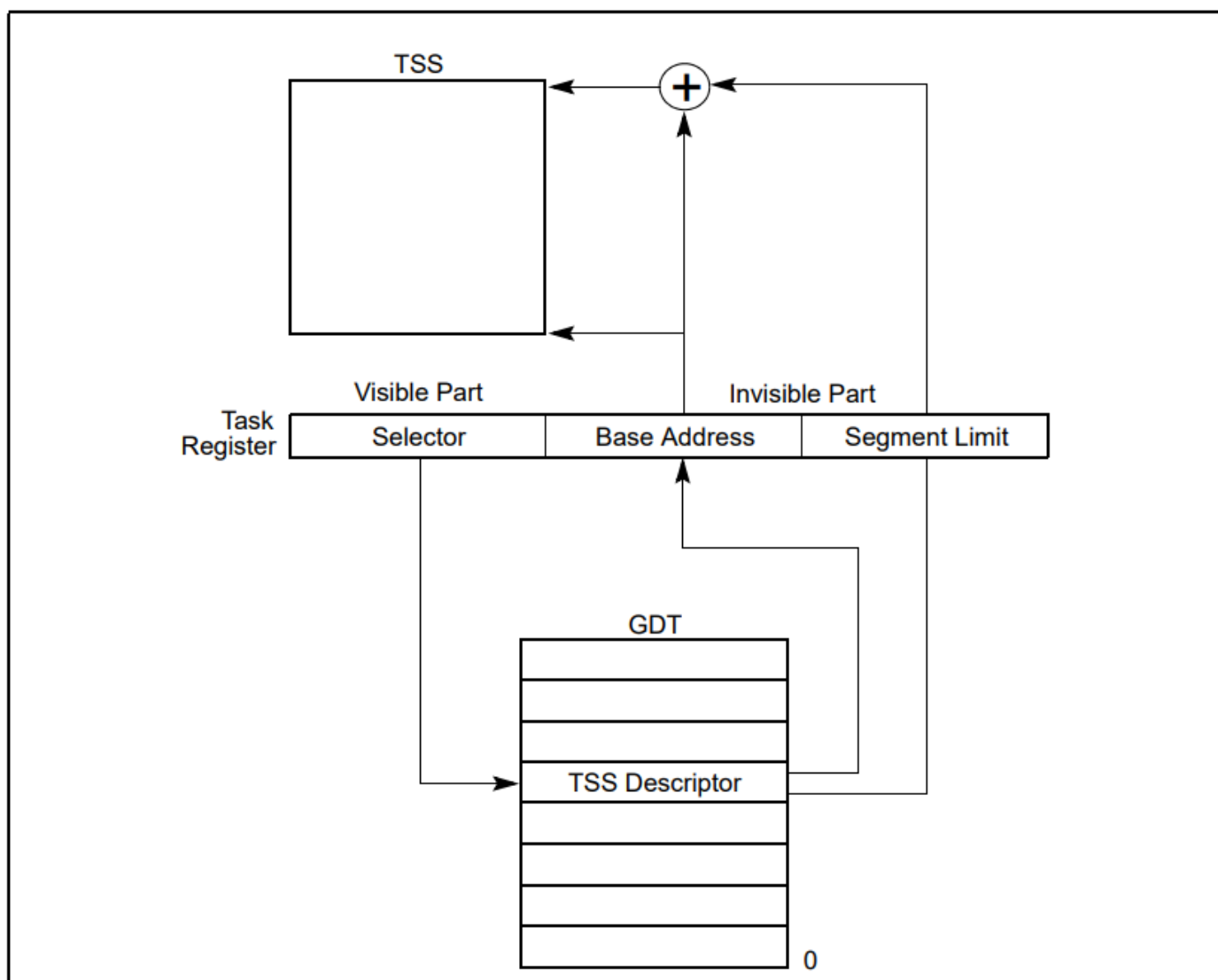


Figure 7-5. Task Register

任务寄存器中存储的信息有**可见**的部分（可以通过软件读取和更改）和**不可见**的部分（由处理器维护，软件无法访问）。可见部分的段选择器指向 GDT 中的一个 TSS 描述符。处理器使用任务寄存器的不可见部分来缓存 TSS 的段描述符，将这些值缓存在寄存器中可以提高任务的执行效率。

LTR (load task register) 和 **STR** (store task register) 指令加载和读取任务寄存器的可见部分：

LTR 指令将段选择器（源操作数）加载到指向 GDT 中的 TSS 描述符的任务寄存器中。然后它使用来自 TSS 描述符的信息加载任务寄存器的不可见部分。LTR 是一条**特权指令**，只有在 CPL 为 0 时才能执行。它用于系统初始化期间，来将初始值放入任务寄存器中。之后，当发生任务切换时，任务寄存器的内容会被隐式更改。

STR 指令将任务寄存器的可见部分存储在通用寄存器或内存中。该指令可以由以任何特权级别运行的代码执行，以识别当前正在运行的任务。但是，它通常仅由操作系统软件使用。

在处理器上电或复位时，TR 中的段选择器和基地址被设置为默认值 0；限制设置为 FFFFH。

2.4 任务门描述符 Task-Gate Descriptor

任务门描述符提供对任务的间接、受保护的引用（见下图）。它可以放在 GDT、LDT 或 ID 中。任务门描述符中的 TSS 段选择子字段指向 GDT 中的一个 TSS 描述符。此段选择子中未使用 RPL。

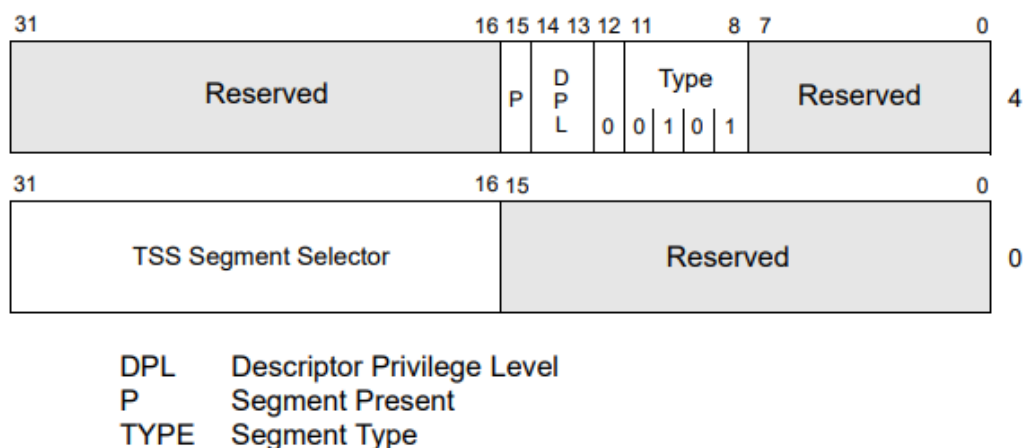


Figure 7-6. Task-Gate Descriptor

任务门描述符的 DPL 在任务切换期间控制对 TSS 描述符的访问。当程序或过程通过任务门调用或跳转到任务时，指向任务门的门选择子的 CPL 和 RPL 字段必须小于或等于任务门描述符的 DPL。值得注意的是，当使用任务门时，不会使用目标 TSS 描述符的 DPL。

可以通过任务门描述符或 TSS 描述符访问任务。这两种结构都满足以下需求：

- **任务只有一个忙标志的需求：**因为任务的B标志存储在 TSS 描述符中，每个任务应该只有一个 TSS 描述符。但是，可能有多个任务门引用相同的 TSS 描述符。
- **提供对任务的选择性访问的需求：**任务门满足了这一需求，因为它们可以驻留在 LDT 中并且可以具有不同于 TSS 描述符的 DPL。如果一个程序或过程的权限等级不够访问某个任务在 GDT 中的 TSS（通常 DPL 为 0），可能通过一个 DPL 更高的任务门来访问任务。任务门为操作系统提供了更大的自由度，来限制对特定任务的访问。
- **中断或异常被独立的程序来处理的需求：**任务门也可能位于 IDT 中，来允许中断和异常由处理程序任务处理。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。

下图展示了 LDT 中的任务门、GDT 中的任务门和 IDT 中的任务门都指向同一个任务的情形。

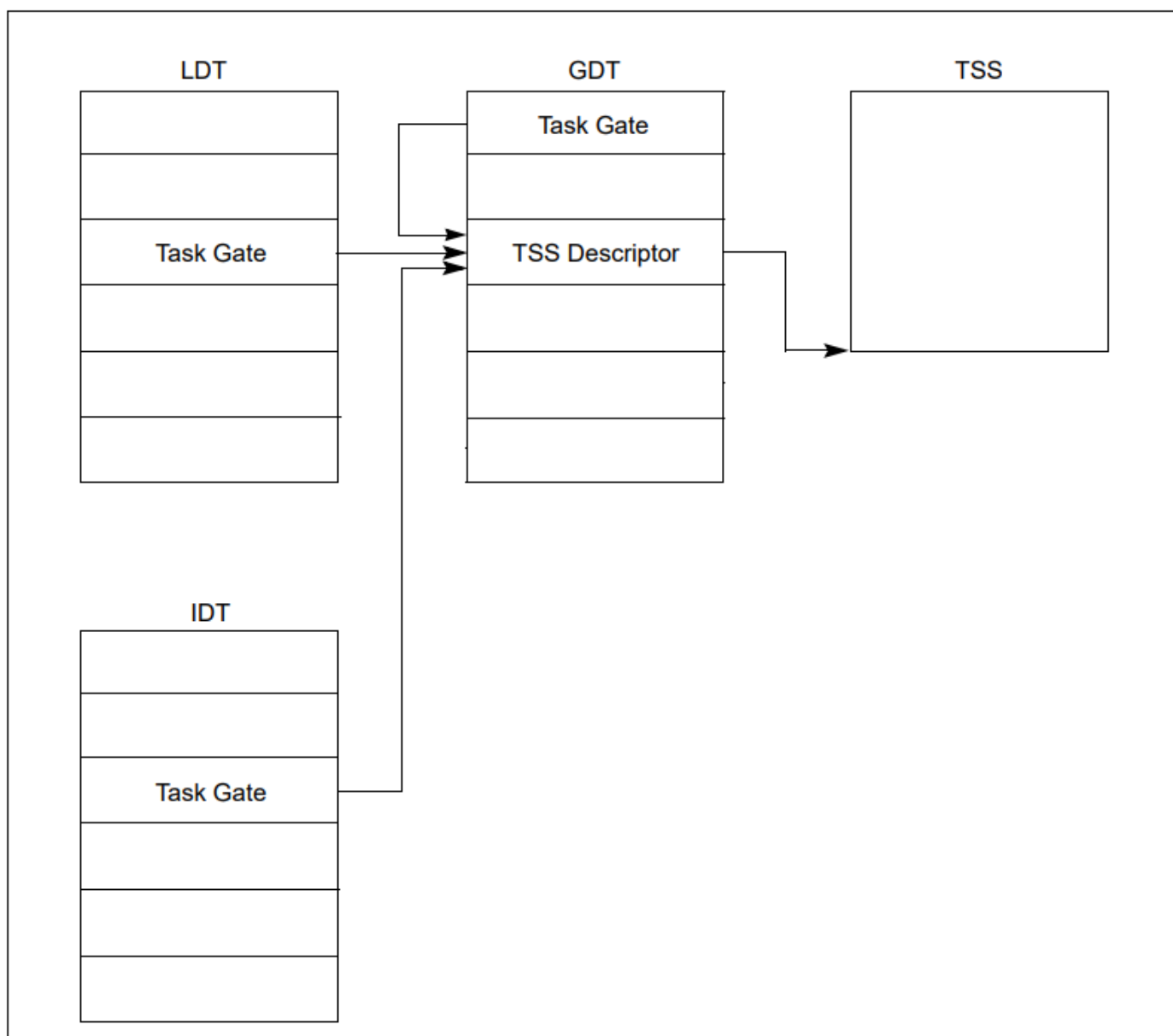


Figure 7-7. Task Gates Referencing the Same Task

3. 任务切换

处理器转移任务的执行于如下四种情形之一：

- 当前程序，任务或过程执行 `JMP` 或 `CALL` 指令至GDT中的一个TSS描述符。
- 当前程序，任务或过程执行 `JMP` 或 `CALL` 指令至GDT或当前LDT中的一个任务门描述符。
- 中断或异常向量指向IDT中的一个**任务门**描述符。**中断或异常向量指向 IDT 表中的中断门或陷阱门，不会发生任务切换。**
- EFLAGS寄存器中的NT标志为1时，当前任务执行 `IRET` 指令。

`JMP`、`CALL` 和 `IRET` 指令以及中断和异常，都是**重定向程序的机制**。TSS 描述符或任务门的引用（调用或跳转到任务时）或 NT 标志的状态（执行 `IRET` 指令时）**决定是否发生任务切换**。

任务切换时，处理器执行如下操作：

1. 从任务门或（对于由 IRET 指令启动的任务切换：）前任务字段获取新任务的 TSS 段选择子，作为 JMP 或 CALL 指令的操作数。
2. 检查是否允许当前（旧）任务切换到新任务。数据访问权限规则适用于 JMP 和 CALL 指令。当前（旧）任务的 CPL 和新任务的段选择子的 RPL 必须小于或等于被引用的 TSS 描述符或任务门的 DPL。异常、中断（除由 INT n 指令生成的中断）和 IRET 指令无视目标任务门或 TSS 描述符的 DPL，允许切换任务。对于 INT n 指令产生的中断，需要检查 DPL。
3. 检查新任务的 TSS 描述符是否标记为“存在”（present）并且具有有效界限值（大于等于 67H）。
4. 检查新任务是否可用（调用、跳转、异常或中断）或忙（IRET 返回）。
5. 检查任务切换中使用的当前（旧）TSS、新 TSS 和所有段描述符是否已分页到系统内存中。
6. 如果任务切换是用 JMP 或 IRET 指令启动的，处理器会清除当前（旧）任务的 TSS 描述符中的 B 标志；如果使用 CALL 指令、异常或中断启动：B 标志保持 1。（见 4. 任务链中的表 7-2）
7. 如果任务切换是用 IRET 指令启动的，处理器会清除 EFLAGS 寄存器临时保存映像中的 NT 标志；如果使用 CALL 或 JMP 指令、异常或中断启动，则保存的 EFLAGS 映像中的 NT 标志将保持不变。
8. 在当前任务的 TSS 中保存当前（旧）任务的状态。处理器在任务寄存器中找到当前 TSS 的基地址，然后将以下寄存器的状态复制到当前 TSS 中：所有通用寄存器，来自段寄存器们的段选择子们，EFLAGS 寄存器的临时保存映像和指令指针寄存器 (EIP)。
9. 如果任务切换是用 CALL 指令、异常或中断启动的，处理器将在从新任务加载的 EFLAGS 中设置 NT 标志为 1。如果用 IRET 指令或 JMP 指令启动，NT 标志将反映从新任务加载的 EFLAGS 中 NT 的状态（见表 7-2）。
10. 如果任务切换是通过 CALL 指令、JMP 指令、异常或中断启动的，则处理器会在新任务的 TSS 描述符中设置 B 标志为 1；如果使用 IRET 指令启动，则 B 标志保持设置状态。
11. 将新任务的 TSS 的段选择子和描述符加载到任务寄存器中。
12. TSS 状态被加载到处理器中。包括 LDTR 寄存器、PDBR（控制寄存器 CR3）、EFLAGS 寄存器、EIP 寄存器、通用寄存器和段选择子们。加载此状态期间的故障可能会破坏体系结构状态。（如果未启用分页，则会从新任务的 TSS 中读取 PDBR 值，但不会将其加载到 CR3 中。）
13. 加载并限定与段选择子相关的描述符。与此加载和资格相关的任何错误都会发生在新任务的上文中，并且可能会破坏体系结构状态。
14. 开始执行新任务。（对于异常处理程序，新任务的第一条指令似乎没有被执行。）

注意：

如果所有检查和保存都已成功执行，则处理器提交任务切换。如果在步骤 1 到 11 中出现不可恢复的错误，则处理器不会完成任务切换，并确保处理器返回到执行启动任务切换的指令之前的状态。

如果在步骤 12 中发生不可恢复的错误，架构状态可能会被破坏，但处理器将尝试在先前的执行环境中处理错误。如果一个不可恢复的错误在提交点之后（在步骤 13 中）发生了，那么处理器将完成任务切换（无需执行额外的访问和段可用性检查），并在开始执行新任务之前生成适当的异常。

如果在提交点之后发生异常，则异常处理程序必须在允许处理器开始执行新任务之前自行完成任务切换。

当任务切换成功时，当前正在执行的任务的状态总是被保存。如果任务被恢复，则从保存的 EIP 值指向的指令开始执行，并且寄存器恢复到任务暂停时的值。

切换任务时，新任务的权限级别不会继承挂起任务的权限级别。新任务开始以从 TSS 加载的 CS 寄存器的 CPL 字段中指定的特权级别执行。因为任务被它们单独的地址空间和 TSS 隔离，并且因为权限规则控制对 TSS 的访问，所以软件不需要在任务切换时执行显式的权限检查。

下表展示了处理器在切换任务时检查的异常条件。它还展示了为每次检查生成的异常（如果检测到错误）以及错误代码引用的段。（表中的检查顺序是 P6 系列处理器中使用的顺序。确切的顺序是特定于型号的，并且对于其他 IA-32 处理器可能有所不同。）设计用于处理这些异常的异常处理程序可能会受到递归调用，如果它们试图重新加载产生异常的段选择器的话。在重新加载选择器之前，应该修复异常的原因（或多个原因中的第一个）。

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
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

注：

1. #NP 是段不存在异常，#GP 是一般保护异常，#TS 是无效-TSS 异常，#SS 是堆栈错误异常。
2. 错误代码包含此列中引用的段描述符的索引。
3. 如果段选择符在兼容类型的表（GDT 或 LDT）中，占用表的段限制内的地址，并且引用兼容类型的描述符（例如，CS 寄存器中的段选择符），则该段选择符的有效仅在指向代

码段描述符时有效)。

****每次发生任务切换时，控制寄存器 CR0 中的 TS（任务切换）标志都会被设置。****系统软件在与处理器的其余部分产生浮点异常时使用 TS 标志来协调浮点单元的操作。TS 标志表示浮点单元的上下文可能与当前任务的上下文不同。

4. 任务链

TSS 的**前一个任务链接字段**（previous task link field，有时称为“反向链接backlink”）和 EFLAGS 寄存器中的 NT 标志用于将处理器的执行返回到前一个任务。**EFLAGS.NT = 1 表示当前执行的任务嵌套（nested）在另一个任务的执行中。**

当 CALL 指令、中断、或异常导致任务切换时：处理器将当前 TSS 的段选择子复制到新任务的 TSS 的前一个任务链接字段；然后设置 EFLAGS.NT = 1。如果软件使用 IRET 指令暂停新任务，处理器****检查 EFLAGS.NT = 1；然后它使用上一个任务链接字段中的值返回到上一个任务。****此过程见下图。

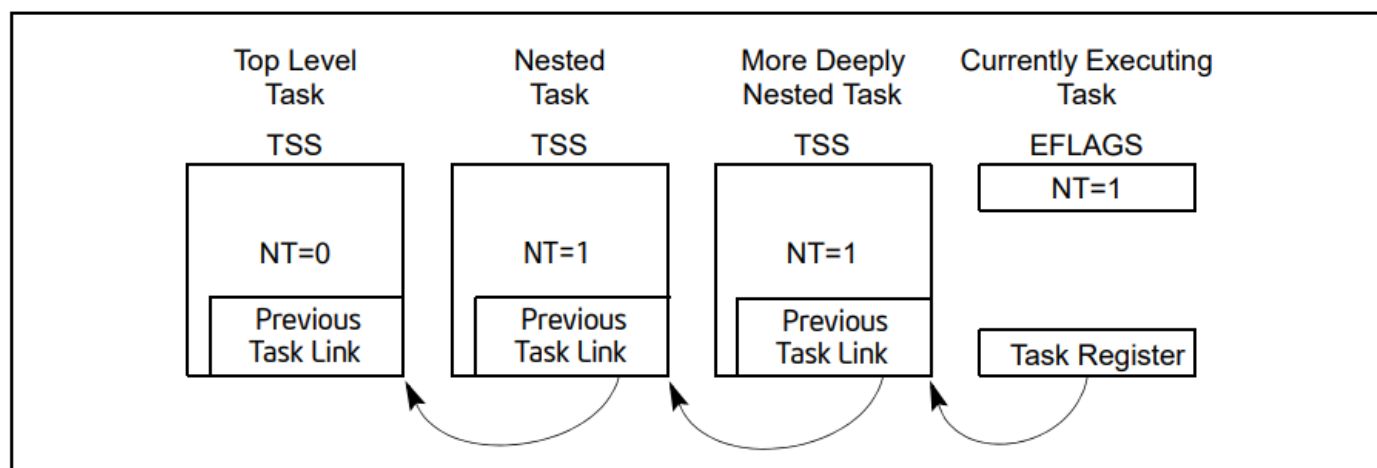


Figure 7-8. Nested Tasks

当 JMP 指令导致任务切换时，新任务不会嵌套，也就是不会使用先前的任务链接字段且 EFLAGS.NT = 0。因此，当任务不需要嵌套时，使用 JMP 指令分派新任务。

下表显示了任务切换期间的忙标志（在 TSS 段描述符中）、NT 标志、前一个任务链接字段和 TS 标志（在控制寄存器 CR0 中）。NT 标志可以被在任何特权级别上执行的软件修改，程序可以设置 NT 标志并执行 IRET 指令。这可能会随机调用当前任务的 TSS 的前链接字段中指定的任务。为了防止这种虚假的任务切换，操作系统应该将它创建的每个 TSS 中的前一个任务链接字段初始化为 0。

Table 7-2. Effect of a Task Switch on Busy Flag, NT Flag, Previous Task Link Field, and TS Flag

Flag or Field	Effect of JMP instruction	Effect of CALL Instruction or Interrupt	Effect of IRET Instruction
Busy (B) flag of new task.	Flag is set. Must have been clear before.	Flag is set. Must have been clear before.	No change. Must have been set.
Busy flag of old task.	Flag is cleared.	No change. Flag is currently set.	Flag is cleared.
NT flag of new task.	Set to value from TSS of new task.	Flag is set.	Set to value from TSS of new task.
NT flag of old task.	No change.	No change.	Flag is cleared.
Previous task link field of new task.	No change.	Loaded with selector for old task's TSS.	No change.
Previous task link field of old task.	No change.	No change.	No change.
TS flag in control register CRO.	Flag is set.	Flag is set.	Flag is set.

4.1 使用 B 标志来防止递归任务切换

TSS 只允许为一项任务保存一个上下文。因此，一旦调用（分派）任务，对该任务的递归（或重入）调用将导致任务的当前状态丢失。TSS 段描述符中的 B 标志用于防止重入任务切换和随后的任务状态信息丢失。处理器按如下方式管理 B 标志：

1. 调度任务时，处理器设置新任务的 B 标志。
2. 如果在任务切换期间，当前任务被放置在嵌套链中（任务切换由 `CALL` 指令、中断，或异常生成），则当前任务的 B 标志保持设置。
3. 当切换到新任务（由 `CALL` 指令、中断、或异常启动）时，如果新任务的 B 标志已设置，则处理器会生成一般保护异常（#GP）。如果使用 `IRET` 指令启动任务切换，则不会引发异常，因为 B 标志本该为1。
4. 当一个任务被跳转到一个新任务终止（由任务代码中的 `JMP` 指令启动），或因任务代码中的 `IRET` 指令终止时，处理器清除 B 标志，将任务返回到“不忙”状态。

处理器通过防止**任务切换到自身或嵌套任务链中的任何任务**来防止递归任务切换。由于多次调用、中断或异常，嵌套的挂起任务链可能会增长到任意长度。B标志会阻止调用已在链中的任务。

B 标志可用于多处理器配置，因为处理器在设置或清除 B 标志时遵循 LOCK 协议（在总线上或缓存中）。这个锁可以防止两个处理器同时调用同一个任务。

4.2 修改任务链接

在单处理器系统中，如果需要**从链接任务链中删除任务**，那么使用以下过程来删除任务：

1. 关中断。
2. 改变抢占任务（使要删除的任务被挂起的任务）TSS中的previous task link字段。假设抢占任务是链中要删除的任务的下一个任务（较新的任务）。将先前的任务链接字段更改为指向链中下一个最旧任务的 TSS，或者指向链中更旧的任务。
3. 清除从链中移除的任务 TSS 段描述符中的 B 标志。如果从链中删除了多个任务，则必须清除每个正在删除的任务的 B 标志。

4. 开中断。

在多处理系统中，必须向此过程添加额外的同步和序列化操作，以确保在更改先前的任务链接字段并清除 B 标志时，TSS 及其段描述符都被锁定。

5. 任务地址空间

****任务的地址空间由任务可以访问的段组成。这些段包括 TSS 中引用的代码、数据、堆栈和系统段以及任务代码访问的任何其他段。这些段被映射到处理器的线性地址空间，而线性地址空间又被映射到处理器的物理地址空间（直接或通过分页）。**

TSS 中的 LDT 段字段可用于为每个任务提供其自己的 LDT。这样，通过将与任务关联的所有段的段描述符放置在任务的 LDT 中，可允许任务地址空间与其他任务隔离。

多个任务也可以使用相同的 LDT。这是一种允许特定任务相互通信或控制的内存高效的方式，而不会降低整个系统的保护屏障。

因为所有任务都可以访问 GDT，所以也可以创建可以通过 GDT 中的段描述符访问的共享段。

如果启用分页，则 **TSS 中的 CR3 寄存器 (PDBR) 字段** 允许每个任务拥有自己的一组页表，用于**将线性地址映射到物理地址。或者，多个任务可以共享同一组页表。

5.1 将任务映射到线性和物理地址空间

可以通过以下两种方式之一将任务映射到线性地址空间和物理地址空间：

- **所有任务共享一个线性到物理地址空间映射：** 未启用分页时，这是唯一的选择。在没有分页的情况下，所有线性地址都映射到相同的物理地址。当启用分页时，这种形式的**线性到物理地址空间映射**是通过对**所有任务使用一个页目录**来获得的。如果支持请求分页的虚拟内存，那么线性地址空间可能会超过可用的物理空间。
- **每个任务都有自己的映射到物理地址空间的线性地址空间。**：这种形式的映射是通过**为每个任务使用不同的页目录**来完成的。因为 PDBR（控制寄存器 CR3）是在任务切换时加载的，每个任务可能有不同的页目录。

不同任务的线性地址空间可能映射到完全不同的物理地址。如果不同页目录的条目指向不同的页表，而页表又指向物理内存的不同页，那么任务不共享物理地址。

无论使用哪种映射任务线性地址空间的方法，所有任务的 TSS 都必须位于物理空间的共享区域中，所有任务都可以访问该区域。此映射是必需的，以便在任务切换期间，处理器在读取和更新 TSS 时，TSS 地址的映射不会更改。GDT 映射的线性地址空间也应该映射到物理空间的共享区域；否则，GDT 的目的就落空了。下图展示了两个任务的线性地址空间如何通过共享页表在物理空间中重叠。

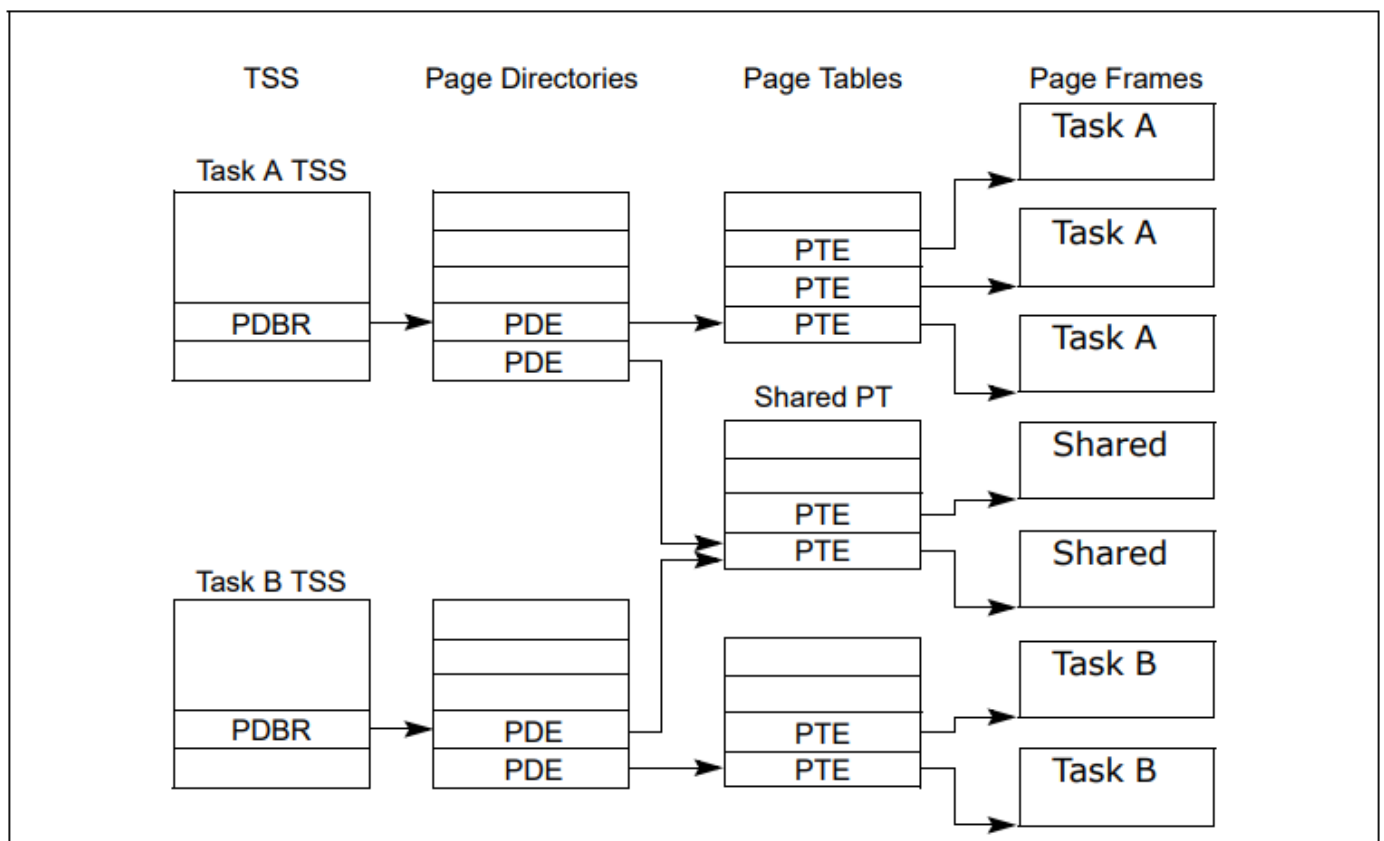


Figure 7-9. Overlapping Linear-to-Physical Mappings

5.2 任务逻辑地址空间

要允许在任务之间共享数据，需要使用以下技术为数据段创建共享的逻辑到物理地址空间映射：

- **通过GDT 中的段描述符：**所有任务都必须能够访问 GDT 中的段描述符。如果 GDT 中的某些段描述符指向线性地址空间中的段，这些段映射到所有任务共有的物理地址空间区域，那么所有任务都可以共享这些段中的数据 and 代码。
- **通过共享LDT：**如果两个或多个任务的TSS 中的 LDT 字段指向同一个LDT，那么两个或多个任务可以使用同一个 LDT。如果共享 LDT 中的某些段描述符指向映射到物理地址空间公共区域的段，则这些段中的数据 and 代码可以在共享 LDT 的任务之间共享。这种共享方法比通过 GDT 共享更加值得选择，因为共享可以限于特定任务。系统中的其他任务可能有不同的 LDT，而这些 LDT 不允许它们访问共享段。
- **通过不同 LDT 中，映射到线性地址空间中的公共地址的段描述符：**如果线性地址空间的这个公共区域映射到每个任务的物理地址空间的相同区域，这些段描述符允许任务共享段。这样的段描述符通常称为别名 (aliases)。这种共享方法比上面列出的方法更加值得选择，因为 LDT 中的其他段描述符可能指向不共享的独立线性地址。