第15章 任务切换

从 80286 开始的处理器是面向多任务系统而设计的。在一个多任务的环境中，可以同时存在多个任务，每个任务都有各自的局部描述符表（LDT）和任务状态段（TSS）。在局部描述符表中存放着专属于任务局部空间的段的描述符。可以在多个任务之间切换，使它们轮流执行，从一个任务切换到另一个任务时，具体的切换过程是由处理器固件负责进行的。

所谓多任务系统，是指能够同时执行两个以上任务的系统。即使前一个任务没有执行完，下一个任务也可以开始执行。但是，什么时候切换到另一个任务，以及切换到哪一个任务执行，主要是操作系统的责任，处理器只负责具体的切换过程，包括保护前一个任务的现场。

有两种基本的任务切换方式，一种是协同式的，从一个任务切换到另一个任务，需要当前任务主动地请求暂时放弃执行权，或者在通过调用门请求操作系统服务时，由操作系统“趁机”将控制转移到另一个任务。这种方式依赖于每个任务的“自律”性，当一个任务失控时，其他任务可能得不到执行的机会。

另一种是抢占式的，在这种方式下，可以安装一个定时器中断，并在中断服务程序中实施任务切换。硬件中断信号总会定时出现，不管处理器当时在做什么，中断都会适时地发生，而任务切换也就能够顺利进行。在这种情况下，每个任务都能获得平等的执行机会。而且，即使一个任务失控，也不会导致其他任务没有机会执行。

抢占式多任务将在第 17 章讲解，本章先介绍多任务任务切换的一般工作原理，掌握任务切换的几种方法，以及它们各自的特点。

15.1 本章代码清单

本章有配套的汇编语言源程序，并围绕这些源程序进行讲解，请对照阅读。
本章代码清单：15-1（保护模式微型核心程序），源程序文件：c15_core.asm
本章代码清单：15-2（动态加载的用户程序），源程序文件：c15.asm

15.2 任务切换前的设置

在上一章里，有关特权级间的控制转移落墨较多，容易使读者混淆了它和任务切换之间的区别。如图 15-1 所示，所有任务共享一个全局空间，这是内核或者操作系统提供的，包含了系统服务程序和数据；同时，每个任务还有自己的局部空间，每个任务的功能都不一样，所以，局部空间包含的是一个任务区别于其他任务的私有代码和数据。

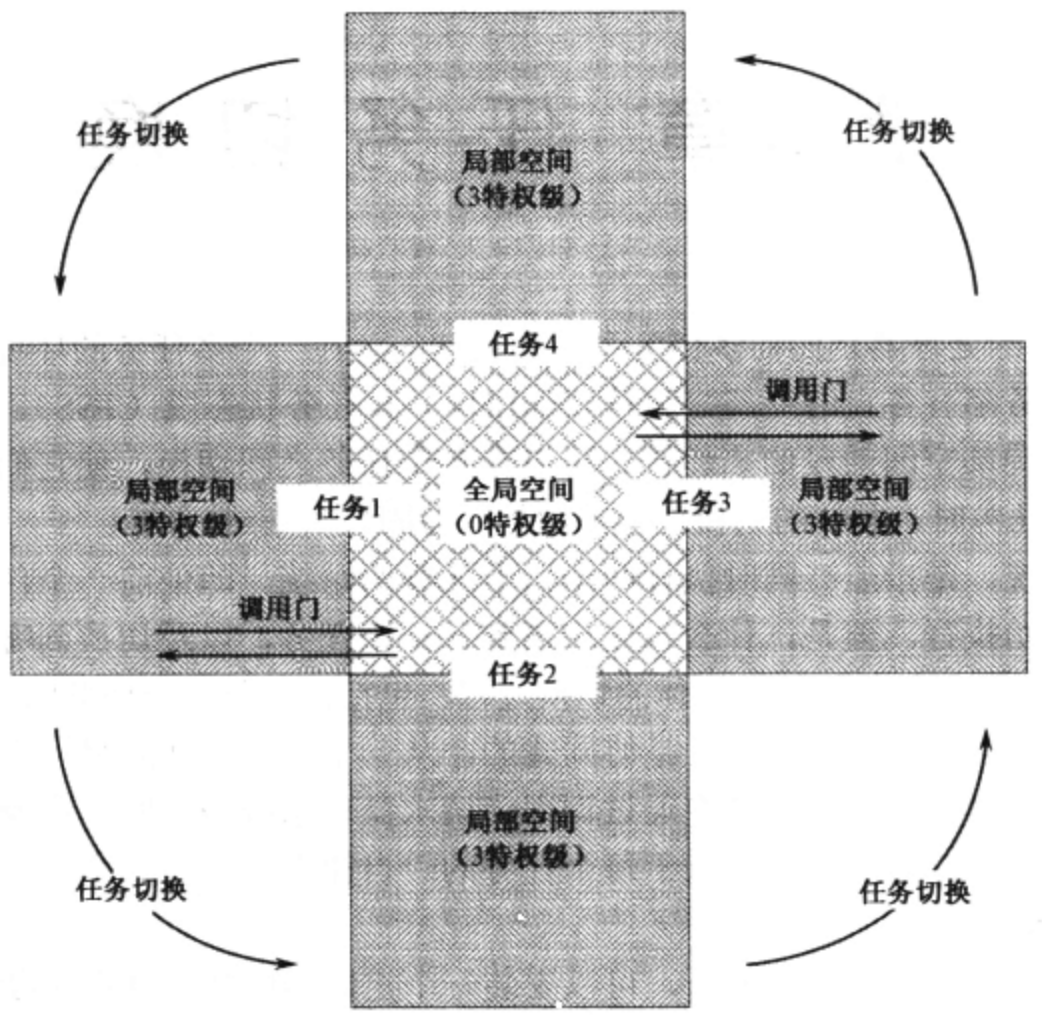

图 15-1　任务切换和任务内特权级间的控制转移

在一个任务内，全局空间和局部空间具有不同的特权级别。使用门，可以在任务内将控制从 3 特权级的局部空间转移到 0 特权级的全局空间，以使用内核或者操作系统提供的服务。

任务切换是以任务为单位的，是指离开一个任务，转到另一个任务中去执行。任务转移相对来说要复杂得多，当一个任务正在执行时，处理器的各个部分都和该任务息息相关：段寄存器指向该任务所使用的内存段；通用寄存器保存着该任务的中间结果，等等。离开当前任务，转到另一个任务开始执行时，要保存旧任务的各种状态，并恢复新任务的运行环境。

这就是说，要执行任务切换，系统中必须至少有两个任务，而且已经有一个正在执行中。在上一章中，我们已经创建过一个任务，那个任务的特权级别是 3，即最低的特权级别。一开始，处理器是在任务的全局空间执行的，当前特权级别是 0，然后，我们通过一个虚假的调用门返回，使处理器回到任务的局部空间执行，当前特权级别降为 3。

事实上，这是没有必要的，这样做很别扭。首先，处理器在刚进入保护模式时，是以 0 特权级别运行的，而且执行的一般是操作系统代码，也必须是 0 特权级别的，这样才能方便地控制整个计算机。其次，任务并不一定非得是 3 特权级别的，也可以是 0 特权级别的。特别是，操作系统除了为每一个任务提供服务外，也会有一个作为任务而独立存在的部分，而且是 0 特权级别的任务，以完成一些管理和控制功能，比如提供一个界面和用户进行交互。

既然是这样，当计算机加电之后，一旦进入保护模式，就直接创建和执行操作系统的 0 特权级任务，这既自然，也很方便。然后，可以从该任务切换到其他任务，不管它们是哪个特权级别的。

既然如此，我们在这一章里就要首先创建 0 特权级别的操作系统（内核）任务。

本章同样没有主引导程序，还要使用第 13 章的主引导程序，内核部分有一些改动，增加了和任务切换有关的代码。

现在来看代码清单 15-1。

内核的入口点在第 848 行，第 906 行之前的工作都和上一章相同，主要是显示处理器品牌信

息，以及安装供每个任务使用的调用门。

接下来的工作是创建 0 特权级的内核任务，并将当前正在执行的内核代码段划归该任务。当前代码的作用是创建其他任务，管理它们，所以称做任务管理器，或者叫程序管理器。

任务状态段（TSS）是一个任务存在的标志，没有它，就无法执行任务切换，因为任务切换时需要保存旧任务的各种状态数据。第 909～911 行用于申请创建 TSS 所需的内存。为了追踪程序管理器的 TSS，需要保存它的基地址和选择子，保存的位置是内核数据段。第 431 行，声明并初始化了 6 字节的空间，前 32 位用于保存 TSS 的基地址，后 16 位则是它的选择子。

接着，第 914～918 行对 TSS 进行最基本的设置。程序管理器任务没有自己的 LDT，任务可以没有自己的 LDT，这是允许的。程序管理器可以将自己所使用的段描述符安装在 GDT 中。另外，程序管理器任务是运行在 0 特权级别上的，不需要创建额外的栈。因为除了从门返回外，不能将控制从高特权级的代码段转移到低特权级的代码段。

第 923～928 行，在 GDT 中创建 TSS 的描述符。必须创建 TSS 的描述符，而且只能安装在 GDT 中。

为了表明当前正在任务中执行，所要做的最后一个工作是将当前任务的 TSS 选择子传送到任务寄存器 TR 中。第 932 行正是用来完成这个工作的。执行这条指令后，处理器用该选择子访问 GDT，找到相对应的 TSS 描述符，将其 B 位置“1”，表示该任务正在执行中（或者处于挂起状态）。同时，还要将该描述符传送到 TR 寄存器的描述符高速缓存器中。

第 935、936 行，任务管理器显示一条信息：

```
      [PROGRAM MANAGER]: Hello! I am Program Manager,run at CPL=0.Now,create
user task and switch to it by the CALL instruction...
```

信息文本位于内核数据段中，代码清单的第 434 行声明了标号 prgman_msg1，并初始化了以上的字符串。本章后面还有其他一些字符串，也是在内核数据段声明和初始化的，不再赘述。

方括号中显示了信息的来源，是程序管理器。后面那段话的意思是“你好！我是程序管理器，运行在 0 特权级上。现在，我要创建并通过 CALL 指令切换到用户任务……”。

让任务之间对话，这是本章的特点，有助于更好地理解任务切换过程。既然要创建另外的任务，并执行任务切换，我们就来看看实际上是怎么做到的。

15.3 任务切换的方法

对多任务的支持是现代处理器的标志之一。为此，Intel 处理器提供了多种方法，以灵活地在各个任务之间实施切换。

尽管如此，处理器并没有提供额外的指令用于任务切换。事实上，用的都是我们熟悉的老指令和老手段，但是扩展了它们的功能，使之除了能够继续执行原有的功能外，也能用于实施任务切换操作。

第一种任务切换的方法是借助于中断，这也是现代抢占式多任务的基础。原因很简单，只要中断没有被屏蔽，它就能随时发生。特别是定时器中断，能够以准确的时间间隔发生，可以用来强制实施任务切换。毕竟，没有哪个任务愿意交出处理器控制权，也没有哪个任务能精确地把握交出控制权的时机。

我们知道，在实模式下，内存最低地址端的 1KB 是中断向量表，保存着 256 个中断处理过程的段地址和偏移地址。当中断发生时，处理器把中断号乘以 4，作为表内索引号访问中断向量表，

从相应的位置取出中断处理过程的段地址和偏移地址，并转移到那里执行。

在保护模式下，中断向量表不再使用，取而代之的，是中断描述符表。不要害怕，它和 GDT、LDT 是一样的，用于保存描述符。唯一不同的地方是，它保存的是门描述符，包括中断门、陷阱门和任务门。如果你觉得这些术语太过于陌生，那就回忆一下调用门，这些门和调用门是非常类似的。当中断发生时，处理器用中断号乘以 8（因为每个描述符占 8 字节），作为索引访问中断描述符表，取出门描述符。门描述符中有中断处理过程的代码段选择子和段内偏移量，这和调用门是一样的。接着，转移到相应的位置去执行。

一般的中断处理可以使用中断门和陷阱门。回忆一下调用门的工作原理，它只是从任务的局部空间转移到更高特权级的全局空间去执行，本质上是一种任务内的控制转移行为。与此相同，中断门和陷阱门允许在任务内实施中断处理，转到全局空间去执行一些系统级的管理工作，本质上，也是任务内的控制转移行为。

但是，在中断发生时，如果该中断号对应的门是任务门，那么，性质就截然不同了，必须进行任务切换。即，要中断当前任务的执行，保护当前任务的现场，并转换到另一个任务去执行。

如图 15-2 所示，这是任务门（Task-Gate）描述符的格式。从图中可见，相对于其他各种描述符，任务门描述符中的多数区域没有使用，所以显得特别简单。

任务门描述符中的主要成份是任务的 TSS 选择子。任务门用于在中断发生时执行任务切换，而执行任务切换时必须找到新任务的任务状态段（TSS）。所以，任务门应当指向任务的 TSS。为了指向任务的 TSS，只需要在任务门描述符中给出任务的 TSS 选择子就可以了。

任务门描述符中的 P 位指示该门是否有效，当 P 位为“0”时，不允许通过此门实施任务切换；DPL 是任务门描述符的特权级，但是对因中断而发起的任务切换不起作用，处理器不按特权级施加任何保护。但是，这并不意味着 DPL 字段没有用处，当以非中断的方式通过任务门实施任务切换时，它就有用了，关于这一点，你马上就会看到。

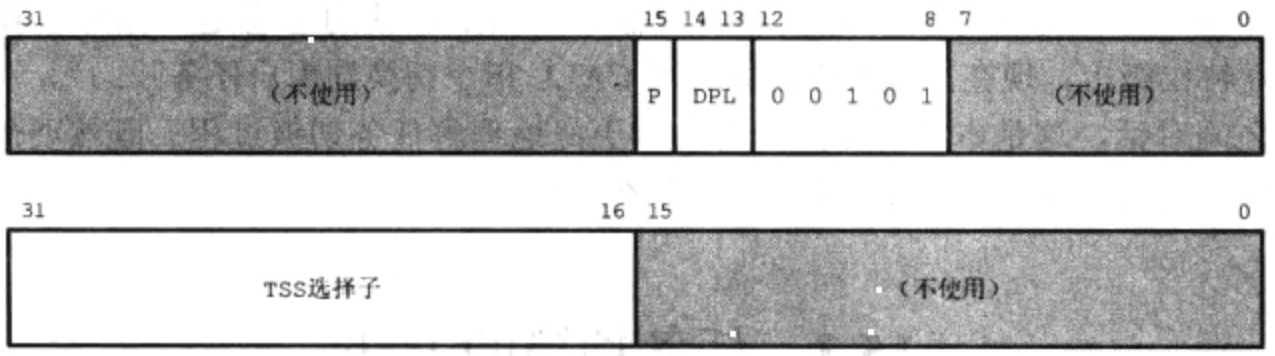

图 15-2　任务门描述符的格式

这样，当中断发生时，处理器用中断号乘以 8 作为索引访问中断描述符表。当它发现这是一个任务门（描述符）时，就知道应当发起任务切换。于是，它取出任务门描述符；再从任务门描述符中取出新任务的 TSS 选择子；接着，再用 TSS 选择子访问 GDT，取出新任务的 TSS 描述符。在转到新任务执行前，处理器要先把当前任务的状态保存起来。当前任务的 TSS 是由任务寄存器 TR 的当前内容指向的，所以，处理器把每个寄存器的“快照”保存到由 TR 指向的 TSS 中。然后，处理器访问新任务的 TSS，从中恢复各个寄存器的内容，包括通用寄存器、标志寄存器 EFLAGS、段寄存器、指令指针寄存器 EIP、栈指针寄存器 ESP，以及局部描述符表寄存器（LDTR）等。最终，任务寄存器 TR 指向新任务的 TSS，而处理器旋即开始执行新的任务。一旦新任务开始执行，处理器固件会自动将其 TSS 描述符的 B 位置“1”，表示该任务的状态为忙。

当中断发生时，可以执行常规的中断处理过程，也可以进行任务切换。尽管性质不同，但它们都要使用 iret 指令返回。前者是返回到同一任务内的不同代码段；后者是返回到被中断的那个

任务。问题是，处理器如何区分这两种截然不同的返回类型呢？

如图 15-3 所示，32 位处理器的 EFLAGS 有 NT 位（位 14），意思是嵌套任务标志（Nested Task Flag）。每个任务的 TSS 中都有一个任务链接域（指向前一个任务的指针，参见上一章 TSS 的结构），可以填写为前一个任务的 TSS 描述符选择子。如果当前任务 EFLAGS 寄存器的 NT 位是“1”，则表示当前正在执行的任务嵌套于其他任务内，并且能够通过 TSS 任务链接域的指针返回到前一个任务。

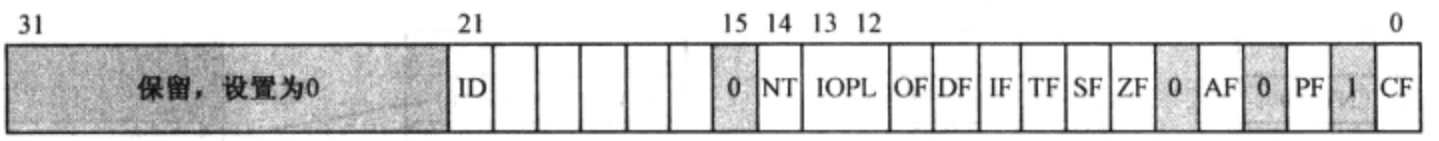

图 15-3 标志寄存器 EFLAGS 的 NT 位

因中断而引发任务切换时，取决于当前任务（旧任务）是否嵌套于其他任务内，其 EFLAGS 寄存器的 NT 位可能是“0”，也可能是“1”。不过这无关紧要，因为处理器不会改变它，而是和其他寄存器一道，写入 TSS 中保护起来。另外，当前任务（旧任务）肯定处于“忙”的状态，其 TSS 描述符的 B 位一定是“1”，在任务切换后同样保持不变。

对新任务的处理是，要把老任务的 TSS 选择子填写到新任务 TSS 中的任务链接域，同时，将新任务 EFLAGS 寄存器的 NT 位置“1”，以允许返回（转换）到前一个任务（老任务）继续执行。同时，还要把新任务 TSS 描述符的 B 位置“1”（忙）。

可以使用 iret 指令从当前任务返回（转换）到前一个任务，前提是当前任务 EFLAGS 寄存器的 NT 位必须是“1”。无论任何时候处理器碰到 iret 指令，它都要检查 NT 位，如果此位是 0，表明是一般的中断过程，按一般的中断返回处理，即，中断返回是任务内的（中断处理过程虽然属于操作系统，但属于任务的全局空间）；如果此位是 1，则表明当前任务之所以能够正在执行，是因为中断了别的任务。因此，应当返回原先被中断的任务继续执行。此时，由处理器固件把当前任务 EFLAGS 寄存器的 NT 位改成“0”，并把 TSS 描述符的 B 位改成“0”（非忙）。在保存了当前任务的状态之后，接着，用新任务（被中断的任务）的 TSS 恢复现场。

除了因中断引发的任务切换之外，还可以用远过程调用指令 CALL，或者远跳转指令 JMP 直接发起任务切换。在这两种情况下，CALL 和 JMP 指令的操作数是任务的 TSS 描述符选择子或任务门。以下是两个例子：

```
call 0x0010:0x00000000
jmp 0x0010:0x00000000
```

当处理器执行这两条指令时，首先用指令中给出的描述符选择子访问 GDT，分析它的描述符类型。如果是一般的代码段描述符，就按普通的段间转移规则执行；如果是调用门，按调用门的规则执行；如果是 TSS 描述符，或者任务门，则执行任务切换。此时，指令中给出的 32 位偏移量被忽略，原因是执行任务切换时，所有处理器的状态都可以从 TSS 中获得。注意，任务门描述符可以安装在中断描述符表中，也可以安装在全局描述符表（GDT）或者局部描述符表（LDT）中。

如果是用于发起任务切换，call 指令和 jmp 指令也有不同之处。使用 call 指令发起的任务切换类似于因中断发起的任务切换。这就是说，由 call 指令发起的任务切换是嵌套的，当前任务（旧任务）TSS 描述符的 B 位保持原来的“1”不变，EFLAGS 寄存器的 NT 位也不发生变化；新任务 TSS 描述符的 B 位置“1”，EFLAGS 寄存器的 NT 位也置“1”，表示此任务嵌套于其他任务中。同时，TSS 任务链接域的内容改为旧任务的 TSS 描述符选择子。

如图 15-4 所示，假设任务 1 是整个系统中的第一个任务。当任务 1 开始执行时，其 TSS 描述符的 B 位是“1”，EFLAGS 寄存器的 NT 位是“0”，不嵌套于其他任务。

当从任务 1 转换到任务 2 后，任务 1 仍然为“忙”，EFLAGS 寄存器的 NT 位不变（在其 TSS 中）；任务 2 也变为“忙”，EFLAGS 寄存器的 NT 位变为“1”，表示嵌套于任务 1 中。同时，任务 1 的 TSS 描述符选择子也被复制到任务 2 的 TSS 中（任务链接域）。

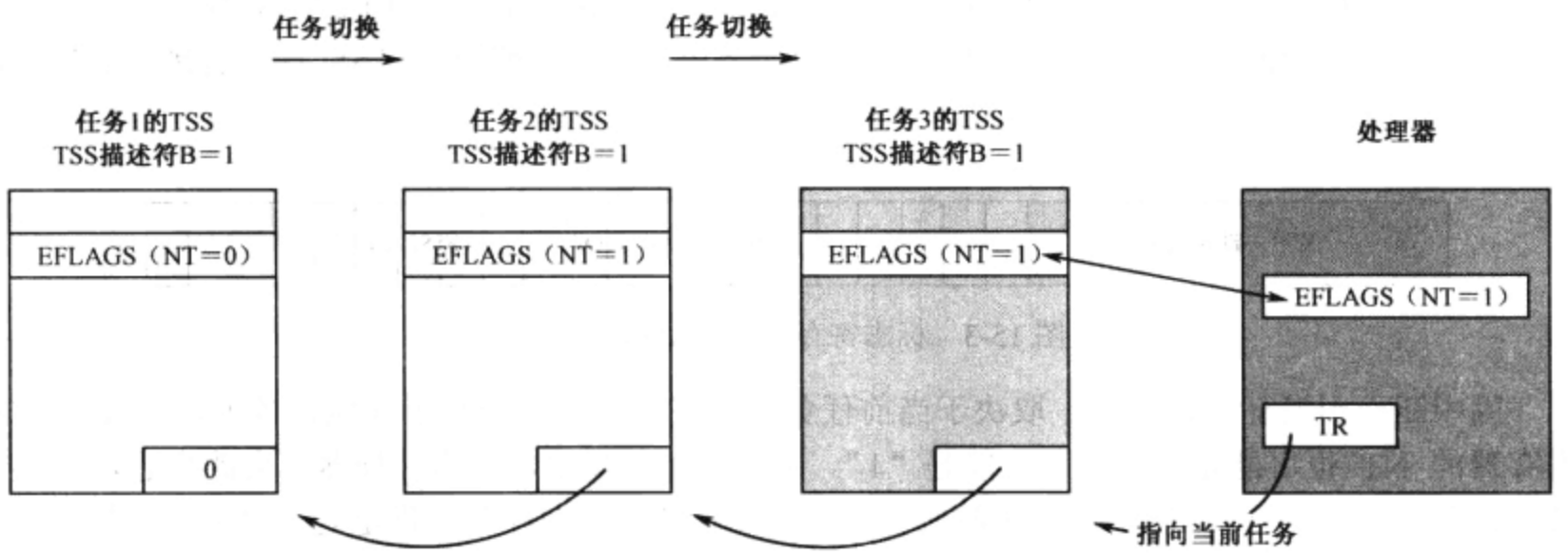

图 15-4　任务嵌套示意图

最后是从任务 2 转换到任务 3 执行。和从前一样，任务 2 保持“忙”的状态，EFLAGS 寄存器的 NT 不变（在其 TSS 中）；任务 3 成为当前任务，其 TSS 描述符的 B 位变成“1”（忙），EFLAGS 寄存器的 NT 位也变成“1”，同时，其 TSS 的任务链接域指向任务 2。

用 CALL 指令发起的任务切换，可以通过 iret 指令返回到前一个任务。此时，旧任务 TSS 描述符的 B 位，以及 EFLAGS 寄存器的 NT 位都恢复到“0”。

和 call 指令不同，使用 jmp 指令发起的任务切换，不会形成任务之间的嵌套关系。执行任务切换时，当前任务（旧任务）TSS 描述符的 B 位清零，变为非忙状态，EFLAGS 寄存器的 NT 位不变；新任务 TSS 描述符的 B 位置“1”，进入忙的状态，EFLAGS 寄存器的 NT 位保持从 TSS 中加载时的状态不变。

任务是不可重入的。

任务不可重入的本质是，执行任务切换时，新任务的状态不能为忙。这里有两个典型的情形：

第一种情形，执行任务切换时，新任务不能是当前任务自己。试想一下，如果允许这种情况发生，处理器该如何执行现场的保护和恢复操作？

第二种情形，如图 15-4 所示，不允许使用 CALL 指令从任务 3 切换到任务 2 和任务 1 上。如果不禁止这种情况的话，任务之间的嵌套关系将会因为 TSS 任务链接域的破坏而错乱。

处理器是通过 TSS 描述符的 B 位来检测重入的。因中断、iret、call 和 jmp 指令发起任务切换时，处理器固件会检测新任务 TSS 描述符的 B 位，如果为“1”，则不允许执行这样的切换。

15.4　用 call/jmp/iret 指令发起任务切换的实例

保护模式下的中断和异常中断处理要在第 17 章才能详细阐述；和中断有关的任务切换也将在第 17 章介绍。在本章，我们重点关注的是用 call、jmp 和 iret 指令发起的任务切换。

前面所讲的一切，都是纸上谈兵，或者说是在纸上谈任务切换。要想加深对任务切换的理解，具体的实例是必不可少的，而正在执行中的代码最能说明问题。为此，让我们回到代码清单 15-1 中。

第 938～945 行是用来加载用户程序的。先分配一个任务控制块（TCB），然后将它挂到 TCB 链上。接着，压入用户程序的起始逻辑扇区号及其 TCB 基地址，作为参数调用过程 load_relocate_program。

过程 load_relocate_program 的工作和上一章相比没有太大变化，仅仅是对 TSS 的填写比较完整。注意，这是任务切换的要求，从一个任务切换到另一个任务时，处理器要从新任务的 TSS 中恢复（加载）各个寄存器的内容。尽管这是任务的第一次执行，但处理器并不知道，这是它的例行工作，你得把任务执行时，各个寄存器的内容放到 TSS 中供处理器加载。

新增加的指令是从第 766 行开始，到 790 行结束的。首先，从栈中取出 TCB 的基地址；然后，通过 4GB 的内存段访问 TCB，取出用户程序加载的起始地址，这也是用户程序头部的起始地址。

接着，依次登记指令指针寄存器 EIP 和各个段寄存器的内容。因为这是用户程序的第一次执行，所以，TSS 中的 EIP 域应该登记用户程序的入口点，CS 域应该登记用户程序入口点所在的代码段选择子。

第 787～790 行，先将 EFLAGS 寄存器的内容压入栈，再将其弹出到 EDX 寄存器，因为不存在将标志寄存器的内容完整地传送到通用寄存器的指令。接着，把 EDX 中的内容写入 TSS 中 EFLAGS 域。注意，这是当前任务（程序管理器）EFLAGS 寄存器的副本，新任务将使用这个副本作为初始的 EFLAGS。一般来说，此时 EFLAGS 寄存器的 IOPL 字段为 00，将来新任务开始执行时，会用这个副本作为处理器 EFLAGS 寄存器的当前值，并因此而没有足够的 I/O 特权。

好，回到第 947 行。

这是一条 32 位间接远调用指令 CALL，操作数是一个内存地址，指向任务控制块（TCB）内的 0x14 单元。这样的指令我们非常熟悉，一般来说，转移到的目标位置可以由 16 位的代码段选择子和 32 位段内偏移量组成，也可以由 16 位的调用门选择子和 32 位偏移量组成。所以，从 TCB 内偏移量为 0x14 的地方，应当先是一个 32 位的段内偏移量，接着是一个 16 位的代码段或者调用门选择子。

但是，回到前一章，看图 14-12，TCB 内偏移为 0x14 的地方，是任务的 TSS 基地址。再往后，是 TSS 选择子。这很奇怪，是吗？但却是合法的。当处理器发现得到的是一个 TSS 选择子，就执行任务切换。和通过调用门的控制转移一样，32 位偏移部分丢弃不用。这就是为什么我们可以把 TSS 基地址作为 32 位偏移量使用的原因。

当执行任务切换时，处理器用得到的选择子访问 GDT，一旦它发现那是一个 TSS 描述符，就知道应该执行任务切换的操作。首先，因为当前正在执行的任务是由任务寄存器 TR 指示的，所以，它要把每个寄存器的“快照”保存到由 TR 指向的 TSS 中。

然后，处理器用指令中给出的 TSS 选择子访问 GDT，取得新任务的 TSS 描述符，并从该 TSS 中恢复各个寄存器的内容，包括通用寄存器、标志寄存器 EFLAGS、段寄存器、指令指针寄存器 EIP、栈指针寄存器 ESP，以及局部描述符表寄存器（LDTR）等。最终，任务寄存器 TR 指向新任务的 TSS，而处理器旋即开始执行新的任务。

谢天谢地，幸亏我们已经在 load_relocate_program 过程内完整地设置了新任务的 TSS，尤其是它的 LDT 域、EIP 域、CS 域和 DS 域，LDT 域指向用户程序的局部描述符表，EIP 域指向用户程序的入口点，CS 域指向用户程序的代码段，DS 域指向用户程序头部段。

程序管理器是计算机启动以来的第一个任务，在任务切换前，其 TSS 描述符的 B 位是“1”，EFLAGS 寄存器的 NT 位是“0”。因为本次任务切换是用 CALL 指令发起的，因此，任务切换后，其 TSS 描述符的 B 位仍旧是“1”，EFLAGS 寄存器的 NT 位不变。当任务切换完成，用户任务成为当前任务，其 TSS 描述符的 B 位置“1”，表示该任务的状态为忙；EFLAGS 寄存器的 NT

位置“1”，表示它嵌套于程序管理器任务；TSS 的任务链接域被修改为前一个任务（程序管理器任务）的 TSS 描述符选择子。

现在，用户程序作为任务开始执行了。所以，让我们转到代码清单 15-2。

总体上，用户程序的结构和上一章相比没有变化，而且功能非常简单，大部分工作都是通过调用门来完成的。程序的入口点在第 55 行。

当用户任务开始执行时，段寄存器 DS 指向头部段。第 57、58 行，令段寄存器 FS 指向头部段。其主要目的是保存指向头部段的指针以备后用，同时，腾出段寄存器 DS 来完成后续操作。毕竟 ，访问数据段时，不加段超越前缀会方便很多。

第 60、61 行，令段寄存器 DS 指向当前任务自己的数据段。

接下来的工作是显示问候语，并报告自己的当前特权级别。因为当前特权级别是计算出来的，所以，字符串要分成两个部分显示。第 63～64 行，先显示前一部分：

```
[USER TASK]: Hi! nice to meet you,I am run at CPL=
```

这句的意思是“嗨，很高兴遇到你，我运行的特权级别 CPL=”。话没有说完，因为这个 CPL 还需要经过计算才能知道。

第 66～69 行，计算当前特权级别，转换成 ASCII 码后填写到数据段中，作为第二个字符串的第 1 个字符。当前特权级别是由段寄存器 CS 当前内容的低 2 位指示的，因此，先将 CS 的内容传送到 AX 寄存器；接着，清除 AL 寄存器的高 6 位，只保留低 2 位的原始内容；最后，将这个数字加上 0x30，转换成可显示和打印的 ASCII 码，并填写到数据段中由标号 message_2 所指示的字节单元中。

第 71、72 行，显示包括特权级数值在内的第二个字符串。据我们所知，当前任务的特权级别是 3，因此，在屏幕上显示的完整内容是：

```
[USER TASK]: Hi! nice to meet you,I am run at CPL=3.Now,I must exit...
```

意思是，“嗨，很高兴遇到你，我运行的特权级别 CPL=3。现在，必须退出喽……”。

通过在中断描述符表中安装任务门，可以在中断信号的驱使下周期性地发起任务切换。否则，每个任务都应该在适当的时候主动转换到其他任务，以免计算机的操作者发现别的任务都僵在那里没有任何反应。如果每个任务都能自觉地做到这一点，那么，这种任务切换机制被称为是协同式的。

一般来说，可以在任务内的任何地方设置一条任务切换指令，以发起任务切换。当然，如果你是为某个流行的操作系统写程序，必须听从操作系统设计者的建议，他们的软件开发指南上会告诉你怎么做。

当前任务的做法稍有些特殊，它很简单，在显示了信息之后，第 74 行，通过调用门转到全局空间执行。从该调用门的符号名“TerminateProgram”上看，意图是终止当前任务的执行，而不是临时转换到其他任务。

不管怎样，让我们回到代码清单 15-1 中去，看看该任务在进入全局空间之后都做了些什么。

用户程序通过调用门进入任务的全局空间后，实际的入口点在 354 行，即名字为 terminate_current_task 的过程。该过程用来结束当前任务的执行，并转换到其他任务。

不要忘了，我们现在仍处在用户任务中，要结束当前的用户任务，可以先切换到程序管理器任务，然后回收用户程序所占用的内存空间，并保证不再转换到该任务。为了切换到程序管理器任务，需要根据当前任务 EFLAGS 寄存器的 NT 位决定是采用 iret 指令，还是 jmp 指令。

第 358～360 行，先将 EFLAGS 寄存器的当前内存压栈，然后，用 ESP 寄存器作为地址操作数访问栈，取得 EFLAGS 的压栈值，并传送到 EDX 寄存器。接着，将 ESP 寄存器的内容加上 4，

使栈平衡，保持压入 EFLAGS 寄存器前的状态。

你可能会奇怪，为什么不直接使用下面两条指令来完成以上功能：

```
pushfd
pop edx
```

的确，这两种做法的效果是一样的，之所以采用 3 条指令，是因为想演示如何通过 ESP 寄存器直接访问栈。在 16 位模式下，不能使用 SP 作为基址，所以下面的指令是错误的：

```
mov ax,[sp]                ;错误
```

注意，使用 ESP 寄存器作为指令的地址操作数时，默认使用的段寄存器是 SS，即访问栈段。

第 362、363 行，令段寄存器 DS 指向内核数据段，以方便后面的操作。

DX 寄存器包含了标志寄存器 EFLAGS 的低 16 位，其中，位 14 是 NT 位。第 365、366 行，测试 DX 寄存器的位 14，看 NT 标志位是 0 还是 1，以决定采用哪种方式（iret 或者 call）回到程序管理器任务。因为当前任务是嵌套在程序管理器任务内的，所以 NT 位必然是“1”，应当转到标号.b1 处继续执行。

第 372、373 行，也就是标号.b1 处，先显示字符串

```
        [SYSTEM CORE]: Uh...This task initiated with CALL instruction or an
exeception/ interrupt,should use IRETD instruction to switch back...
```

该字符串位于第 448 行，是在内核数据段，用标号 core_msg0 声明并初始化的。该字符串的内容显示，消息来源同样是系统内核，该消息的意思是“唔……该任务是用 CALL 指令，或者由一个中断/异常发起的，应当使用 IRETD 指令切换回去……”。

第 374 行，通过 iretd 指令转换到前一个任务，即程序管理器任务。执行任务切换时，当前用户任务的 TSS 描述符的 B 位被清零，EFLAGS 寄存器的 NT 位也被清零，并被保存到它的 TSS 中。

注意，在此处，我们用的是 iretd，而不是 iret。实际上，这是同一条指令，机器码都是 CF。在 16 位模式下，iret 指令的操作数默认是 16 位的，要按 32 位操作数执行，须加指令前缀 0x66，即 66 CF。为了方便，编译器创造了 iretd。当在 16 位模式下使用 iretd 时，编译器就知道，应当加上指令前缀 0x66。在 32 位模式下，iret 和 iretd 是相同的，下面的示例展示了它们之间的区别：

```
[bits 16]
iret                ;编译后的机器码为 CF
iretd               ;编译后的机器码为 66 CF

[bits 32]
iret                ;编译后的机器码为 CF
iretd               ;编译后的机器码为 CF
```

当程序管理器任务恢复执行时，它的所有原始状态都从 TSS 中加载到处理器，包括指令指针寄存器 EIP，它指向第 952 行的那条指令，紧接着当初发起任务切换的那条指令。

对于刚刚被挂起的那个旧任务，如果它没有被终止执行，则可以不予理会，并在下一个适当的时机再次切换到它那里执行。不过，现在的情况是它希望自己被终止。所以，理论上，接下来的工作是回收它所占用的内存空间，并从任务控制块 TCB 链上去掉，以确保不会再切换到该任务执行（当然，现在 TCB 链还没有体现出自己的用处）。遗憾的是，我们并没有提供这样的代码。所以，这个任务将一直存在，一直有效，不会消失，在整个系统的运行期间可以随时切换过去。

接下来，我们再创建一个新任务，并转移到该任务执行。

第 952、953 行，程序管理器先显示一条消息。标号 prgman_msg2 的位置是在第 439 行，位于

内核数据段，在那里初始化了字符串

```
        [PROGRAM MANAGER]: I am glad to regain control.Now,create another user
task and switch to it by the JMP instruction...
```

这是程序管理器在说话，方括号中的文字显示了消息的来源。该消息的大意是“我很高兴又获得了控制，现在，创建其他用户任务，并使用 JMP 指令切换到它那里”。

第 955～964 行，创建新的用户任务并发起任务切换。与上次相比，这次的任务切换有几个值得注意的特点。首先，可以看出，该任务也是从硬盘的 50 号逻辑扇区开始加载的，就是说，它和上一个用户任务一样，来自同一个程序。这就很清楚地说明了，一个程序可以对应着多个运行中的副本，或者说多个任务。尽管如此，它们彼此却没有任何关系，在内存中的位置不同，运行状态也不一样。

其次，这次是用 JMP 指令发起的任务切换，新任务不会嵌套于旧任务中。任务切换之后，程序管理器任务 TSS 描述符的 B 位被清零，EFLAGS 寄存器的 NT 位不变；新任务 TSS 描述符的 B 位置位，EFLAGS 寄存器的 NT 位不变，保持它从 TSS 加载时的状态；任务链接域的内容不变。

由于两个任务来自于同一个程序，故完成相同的工作，最终都会通过调用门进入任务的全局空间执行。而且，在执行到第 365、366 行时，EFLAGS 寄存器 NT 位的测试结果必定是零，即 NT＝0，当前任务并未嵌套于其他任务中，于是执行第 367～369 行，首先显示字符串：

```
        [SYSTEM CORE]: Uh...This task initiated with JMP instruction,  should
switch to Program Manager directly by the JMP instruction...
```

方括号内显示了消息的来源，即系统内核。该消息的意思是，“唔……该任务是用 JMP 指令发起的，应当直接用 JMP 指令转换到程序管理器……”。

然后，使用 32 位间接远转移指令 JMP 转换到程序管理器任务。指令中的标号 prgman_tss 位于内核数据段（第 431 行），在那里初始化了 6 字节，即 16 位的 TSS 描述符选择子和 32 位的 TSS 基地址。按道理，这里不应该是 TSS 基地址，而应当是一个 32 位偏移量。不过，这是无所谓的，当处理器看到选择子部分是一个 TSS 描述符选择子时，它将偏移量丢弃不用。

从第二个任务返回程序管理器任务时，执行点在第 966 行。从这一行开始，一直到第 969 行，用于显示一条消息，然后停机。消息的内容是：

```
        [PROGRAM MANAGER]: I am gain control again, HALT...
```

消息的来源是程序管理器任务，它说：“我又获得了控制，停机……”

最后，处理器执行 halt 指令，终于变消停了。

15.5 处理器在实施任务切换时的操作

处理器用以下四种方法将控制转换到其他任务：

- 当前程序、任务或者过程执行一个将控制转移到 GDT 内某个 TSS 描述符的 jmp 或者 call 指令；
- 当前程序、任务或者过程执行一个将控制转移到 GDT 或者当前 LDT 内某个任务门描述符的 jmp 或者 call 指令；
- 一个异常或者中断发生时，中断号指向中断描述表内的任务门；
- 在 EFLAGS 寄存器的 NT 位置位的情况下，当前任务执行了一个 iret 指令。

jmp、call、iret 指令或者异常和中断，是程序重定向的机制，它们所引用的 TSS 描述符或者任务门，以及 EFLAGS 寄存器 NT 标志的状态，决定了任务切换是否，以及如何发生。

在任务切换时，处理器执行以下操作：

① 从 JMP 或者 CALL 指令的操作数、任务门或者当前任务的 TSS 任务链接域取得新任务的 TSS 描述符选择子。最后一种方法适用于以 iret 发起的任务切换。

② 检查是否允许从当前任务（旧任务）切换到新任务。数据访问的特权级检查规则适用于 jmp 和 call 指令，当前（旧）任务的 CPL 和新任务段选择子的 RPL 必须在数值上小于或者等于目标 TSS 或者任务门的 DPL。异常、中断（除了以 int n 指令引发的中断）和 iret 指令引起的任务切换忽略目标任务门或者 TSS 描述符的 DPL。对于以 int n 指令产生的中断，要检查 DPL。

③ 检查新任务的 TSS 描述符是否已经标记为有效（P=1），并且界限也有效（大于或者等于 0x67，即十进制的 103）。

④ 检查新任务是否可用，不忙（B=0，对于以 CALL、JMP、异常或者中断发起的任务切换）或者忙（B=1，对于以 iret 发起的任务切换）。

⑤ 检查当前任务（旧任务）和新任务的 TSS，以及所有在任务切换时用到的段描述符已经安排到系统内存中。

⑥ 如果任务切换是由 jmp 或者 iret 发起的，处理器清除当前（旧）任务的忙（B）标志；如果是由 call 指令、异常或者中断发起的，忙（B）标志保持原来的置位状态。

⑦ 如果任务切换是由 iret 指令发起的，处理器建立 EFLAGS 寄存器的一个临时副本并清除其 NT 标志；如果是由 call 指令、jmp 指令、异常或者中发起的，副本中的 NT 标志不变。

⑧ 保存当前（旧）任务的状态到它的 TSS 中。处理器从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、段寄存器中的段选择子、刚才那个 EFLAGS 寄存器的副本，以及指令指针寄存器 EIP。

⑨ 如果任务切换是由 call 指令、异常或者中断发起的，处理器把从新任务加载的 EFLAGS 寄存器的 NT 标志置位；如果是由 iret 或者 jmp 指令发起的，NT 标志位的状态对应着从新任务加载的 EFLAGS 寄存器的 NT 位。

⑩ 如果任务切换是由 call 指令、jmp 指令、异常或者中断发起的，处理器将新任务 TSS 描述符中的 B 位置位；如果是由 iret 指令发起的，B 位保持原先的置位状态不变。

⑪ 用新任务的 TSS 选择子和 TSS 描述符加载任务寄存器 TR。

⑫ 新任务的 TSS 状态数据被加载到处理器。这包括 LDTR 寄存器、PDBR（控制寄存器 CR3）、EFLAGS 寄存器、EIP 寄存器、通用寄存器，以及段选择子。载入状态期间只要发生一个故障，架构状态就会被破坏（因为有些寄存器的内容已被改变，而且无法撤销和回退）。所谓架构，是指处理器对外公开的那一部分的规格和构造；所谓架构状态，是指处理器内部的各种构件，在不同的条件下，所建立起来的确定状态。当处理器处于某种状态时，再施加另一种确定的条件，可以进入另一种确定的状态，这应当是严格的、众所周知的、可预见的。否则，就意味着架构状态遭到了破坏。

⑬ 与段选择子相对应的描述符在经过验证后也被加载。与加载和验证新任务环境有关的任何错误都将破坏架构状态。注意，如果所有的检查和保护工作都已经成功实施，处理器提交任务切换。如果在从第 1 步到第 11 步的过程中发生了不可恢复性的错误，处理器不能完成任务切换，并确保处理器返回到执行发起任务切换的那条指令前的状态。如果在第 12 步发生了不可恢复性的错误，架构状态将被破坏；如果在提交点（第 13 步）之后发生了不可恢复性的错误，处理器完成任

务切换并在开始执行新任务之前产生一个相应的异常。

⑭ 开始执行新任务。

在任务切换时，当前任务的状态总要被保存起来。在恢复执行时，处理器从 EIP 寄存器的保存值所指向的那条指令开始执行，这个寄存器的值是在当初任务被挂起时保存的。

任务切换时，新任务的特权级别并不是从那个被挂起的任务继承来的。新任务的特权级别是由其段寄存器 CS 的低 2 位决定的，而该寄存器的内容取自新任务的 TSS。因为每个任务都有自己独立的地址空间和任务状态段 TSS，所以任务之间是彼此隔离的，只需要用特权级规则控制对 TSS 的访问就行，软件不需要在任务切换时进行显式的特权级检查。

任务状态段 TSS 的任务链接域和 EFLAGS 寄存器的 NT 位用于返回前一个任务执行，当前 EFLAGS 寄存器的 NT 位是“1”表明当前任务嵌套于其他任务中。无论如何，新任务的 TSS 描述符的 B 位都会被置位，旧任务的 B 位取决于任务切换的方法。表 15-1 给出了不同条件下，B 位、NT 位和任务链接域的变化情况。

表 15-1　不同任务切换方法对 B 位、NT 位和任务链接域的影响

标志或 TSS 任务链接域	jmp 指令的影响	call 指令或中断的影响	iret 指令的影响
新任务的 B 位	置位。原先必须为零	置位。原先必须为零	不变。原先必须被置位
旧任务的 B 位	清零	不变。原先必须是置位的	清零
新任务的 NT 标志	设置为新任务 TSS 中的对应值	置位	设置为新任务 TSS 中的对应值
旧任务的 NT 标志	不变	不变	清零
新任务的任务链接域	不变	用旧任务的 TSS 描述符选择子加载	不变
旧任务的任务链接域	不变	不变	不变

15.6　程序的编译和运行

首先，虚拟硬盘主引导扇区依然保留和上一章相同的内容。然后，编译本章中提供的两个源程序并写入虚拟硬盘。按要求，从逻辑扇区 1 开始写入内核程序，从逻辑扇区 50 写入用户程序。

完成后，启动虚拟机，应该可以看到图 15-5 所示的画面。

图 15-5　本章程序运行结果

本章习题

1．修改本章的源程序，使之能够顺序完成以下工作：

① 从程序管理器任务切换到任务 A，显示消息后返回程序管理器；

② 从程序管理器任务切换到任务 B，显示消息后返回程序管理器；

③ 再从程序管理器任务切换到任务 A，显示另一条消息，然后返回程序管理器；

④ 再从程序管理器任务切换到任务 B，显示另一条消息，再返回程序管理器。

2．修改本章源程序，使之能够顺序完成以下工作：

① 从程序管理器任务切换到任务 A，显示一条消息；

② 再从任务 A 转换到任务 B，显示一条消息；

③ 从 B 直接返回到程序管理器任务。