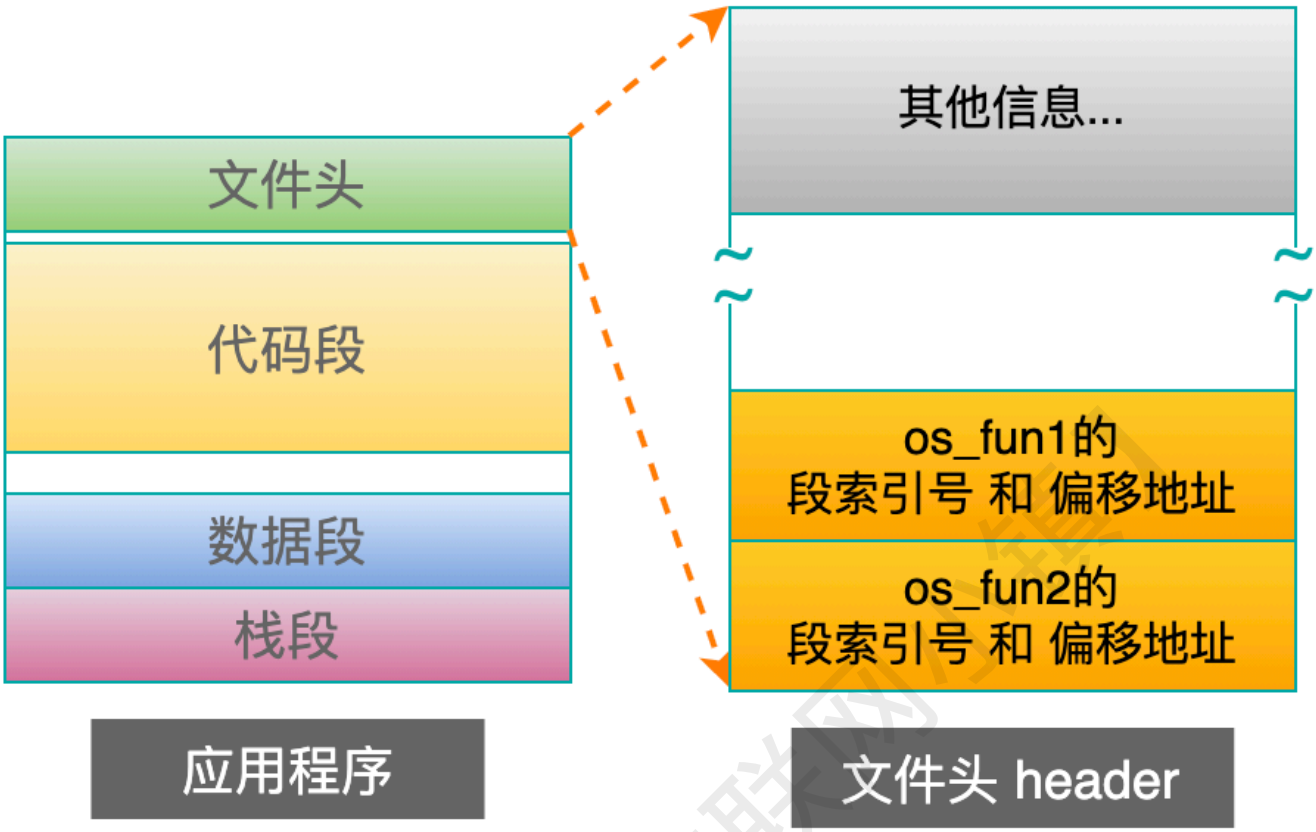
作 者：道哥，10+年嵌入式开发老兵，专注于：C/C++、嵌入式、Linux。

目录

在之前的文章中Linux从头学10：三级跳过程详解-从 bootloader 到 操作系统，再到应用程序，由于当时没有引入特权级的概念，用户程序和操作系统都工作在相同的特权级，因此可以直接通过[段选择子:偏移量] 的方式，来调用属于操作系统代码段中的函数，如下所示：



用户程序header中橙色部分的信息，表示操作系统提供的2个系统函数，位于操作系统的哪个段描述符中，偏移地址是多少。

一旦引入了特权级别，上面这样的调用方式就行不通了。

因为用户程序的特权级一定比操作系统的特权级别低，所以即使用户程序能够知道函数的段选择子和偏移地址，操作系统也会禁止用户程序跳转进去。

> 例如：应用程序的 CPL 和 RPL 都为 3，而操作系统中的函数所在的段 DPL = 0，不能通过特权级的检查。

看过上一篇文章的小伙伴一定知道，如果把目标代码段的描述符中，TYPE.C标志设置为1，也就意味着这是一个依从(或者叫一致性)代码段，就允许低特权级的用户程序调用了。

除了这个方法之外，处理器还提供了另外一种更“正规”的方式，来实现低特权级的代码转移到高特权级的代码，这就是：调用门。

这篇文章，我们就一起来学习调用门的机制，顺带着把所有的门描述符也一起介绍下。

门描述符

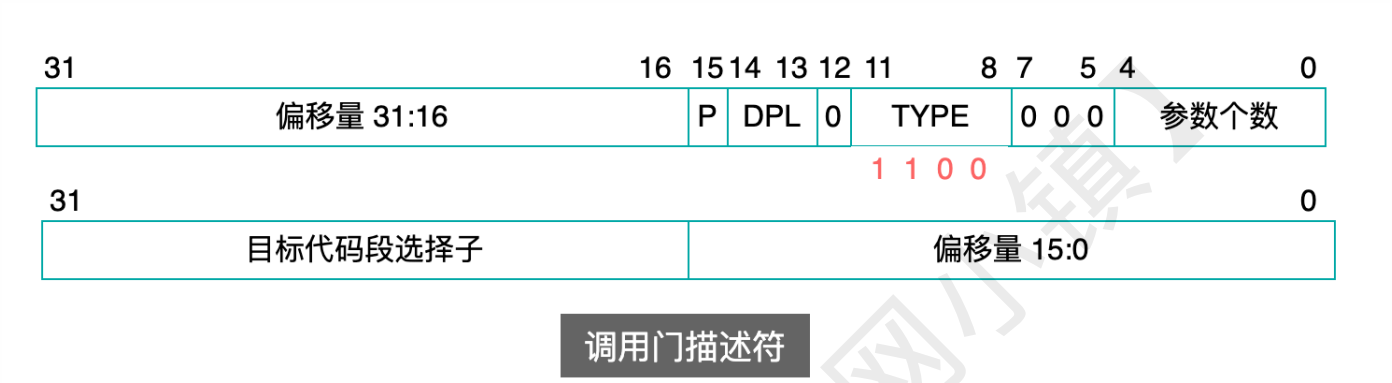
所谓的门，就是一个通道。通过这个通道，可以进入另一个代码段中进行执行。

在x86中，有下面这些门：

> 调用门：用于低特权级代码转移到高特权级代码；
>
> 任务门：用于不同任务之间的调度；
>
> 中断门：用于异步执行中断处理程序；
>
> 陷阱门：也用于执行中断处理程序，不过这里的中断是处理器内部产生的；

门描述符与之前介绍的段描述符本质是一样的，都是用来描述一个代码段的信息，只不过门描述符增加了一层间接性。

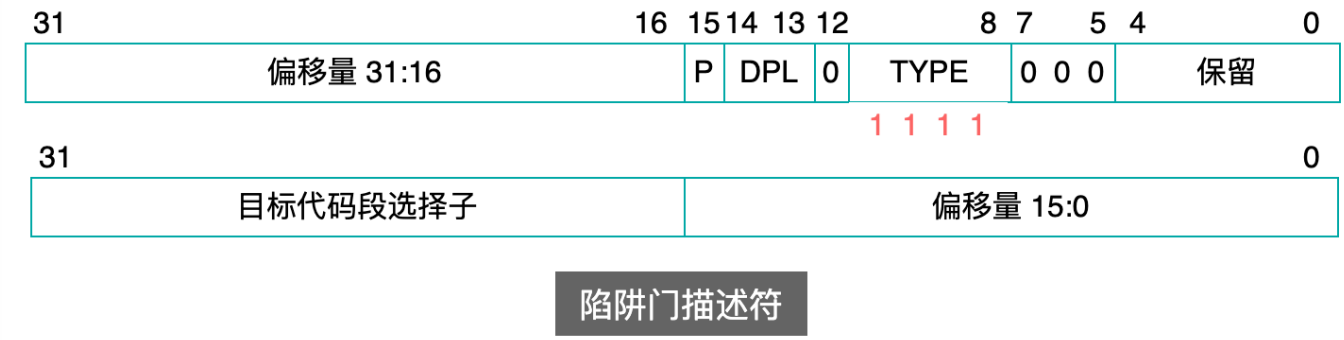
下面是4个门描述符的结构(32位系统)：



调用门描述符

任务门描述符

中断门描述符



陷阱门描述符

从以上这4个门描述符的结构中可以看出: 它们并没有直接记录目标代码段的开始地址和界限，而是记录了目标代码段的选择子。

也就是说：先通过门描述符找到代码段选择子，然后再用这个选择子到 GDT 中去查找真正的目标代码段描述符，最终找到目标代码段的开始地址和界限、属性等信息，也就是下面这个结构:

所以说，这些门就是增加了一层间接性。

这层间接性，为操作系统提供了诸多好处。

首先，对于中断处理来说，把所有的中断描述符放在一个表中，可以对中断处理程序的地址进行解耦。

其次，对于执行代码段的转移来说，可以利用门来提供更灵活的特权级别控制，实现更加复杂的操作。

关于任务门中的TSS选择子:

1. 所谓的任务门可以简单理解为用于任务切换。
2. 因为一个 TSS 段中，保存的就是一个任务的上下文信息快照。
3. 只要处理器发现选择子指向的描述符是一个任务门(通过 TYPE 字段)，它就执行任务切换:

    a. 保存当前 CPU 中的上下文到当前任务的 TSS 段中;

    b. 再把 TSS 选择子中所指向的那个 TSS 段中的上下文内容，加载到 CPU 寄存器中，这样就实现了任务切换。

调用门特权级检查规则

从调用门的名字就可以看出，它是为系统调用服务的。

再来看一下它的描述符结构:

> 参数个数：调用者传递多少个参数给目标代码(是通过栈空间来传参的);
>
> DPL：表示这个调用门本身的特权级;
>
> 目标代码段选择子：最终调用的目标代码段的选择子，需要用这个选择子到 GDT 中寻找目标代码段的基地址;
>
> 偏移量：调用的代码距离目标代码段开始地址的偏移字节数;

从以上这些字段来看，这简直就是为：从低特权级的用户代码，调用高特权级的操作系统代码，量身定做的，只要处理器在特权级上放过用户程序一马就可以了。

事实上也正是如此：当用户请求调用门时，操作系统会进行如下特权级检查：

1. 当前特权级 CPL (用户程序)和请求特权级 RPL，必须 [高于或等于] 调用门中的 DPL;

   > 即在数值上：CPL <= DPL，RPL <= DPL。(注意：这是调用门描述符里的 DPL)

2. 当前特权级 CPL(用户程序)，必须 [低于或等于] 目标代码段中的 DPL;

   > 即在数值上：CPL >= 目标代码段描述符中的 DPL。

从以上规则可以再次看出：即使通过调用门，目标代码段只允许相同或者更低的特权级代码进入，也验证了之前所说的：高特权级代码不会主动转移到低特权级的代码中。

如果特权级检查被通过，进入目标代码段之后，当前特权级CPL是否会改变呢?

这就依赖于目标代码段描述符中的TYPE字段中的 C 标志位的值：

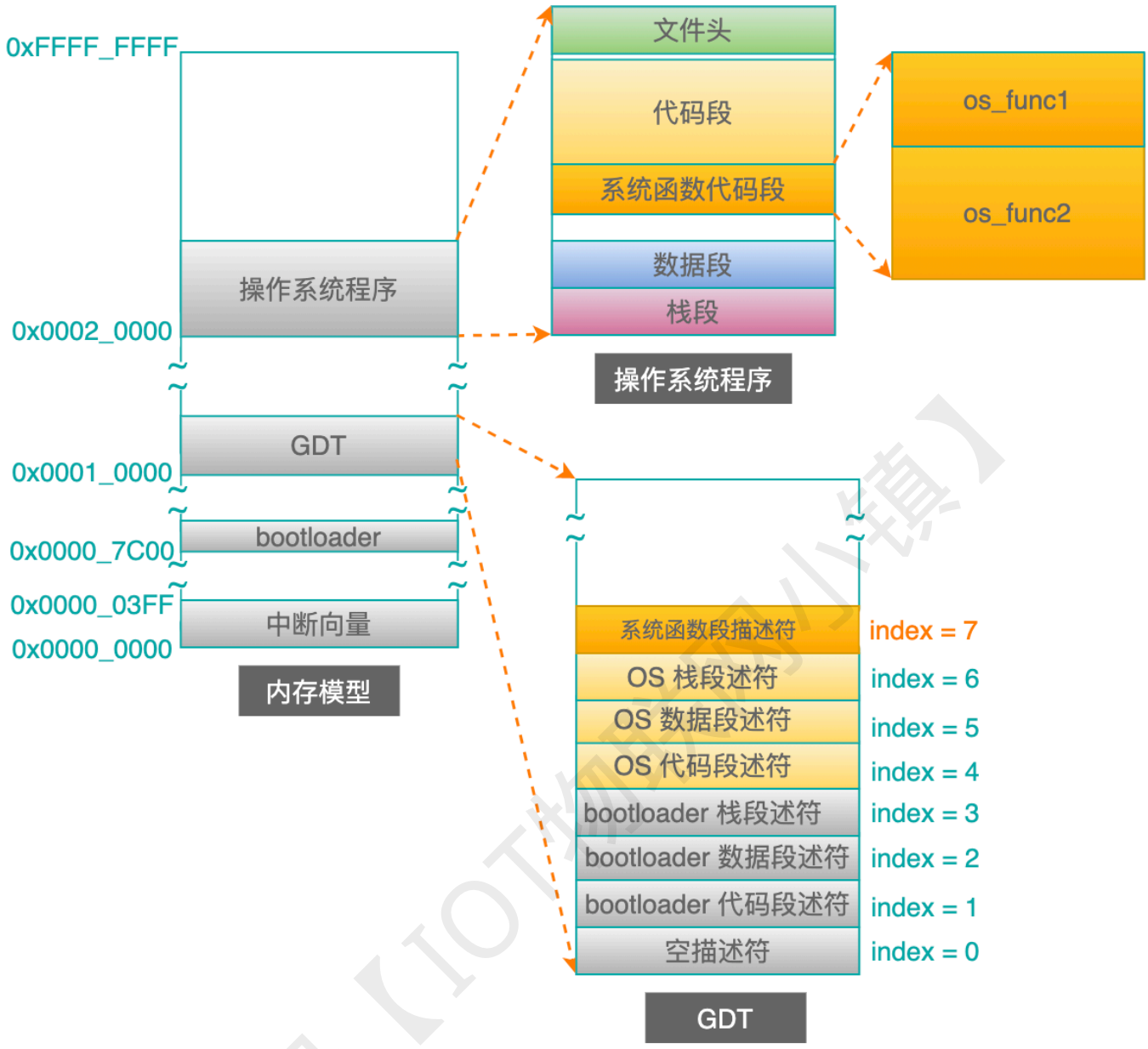
> TYPE.C = 1：CPL 保持不变，仍然为用户程序中的特权级 3;
>
> TYPE.C = 0: CPL 改变，变成目标代码段的特权级;

调用门的使用过程

安装调用门

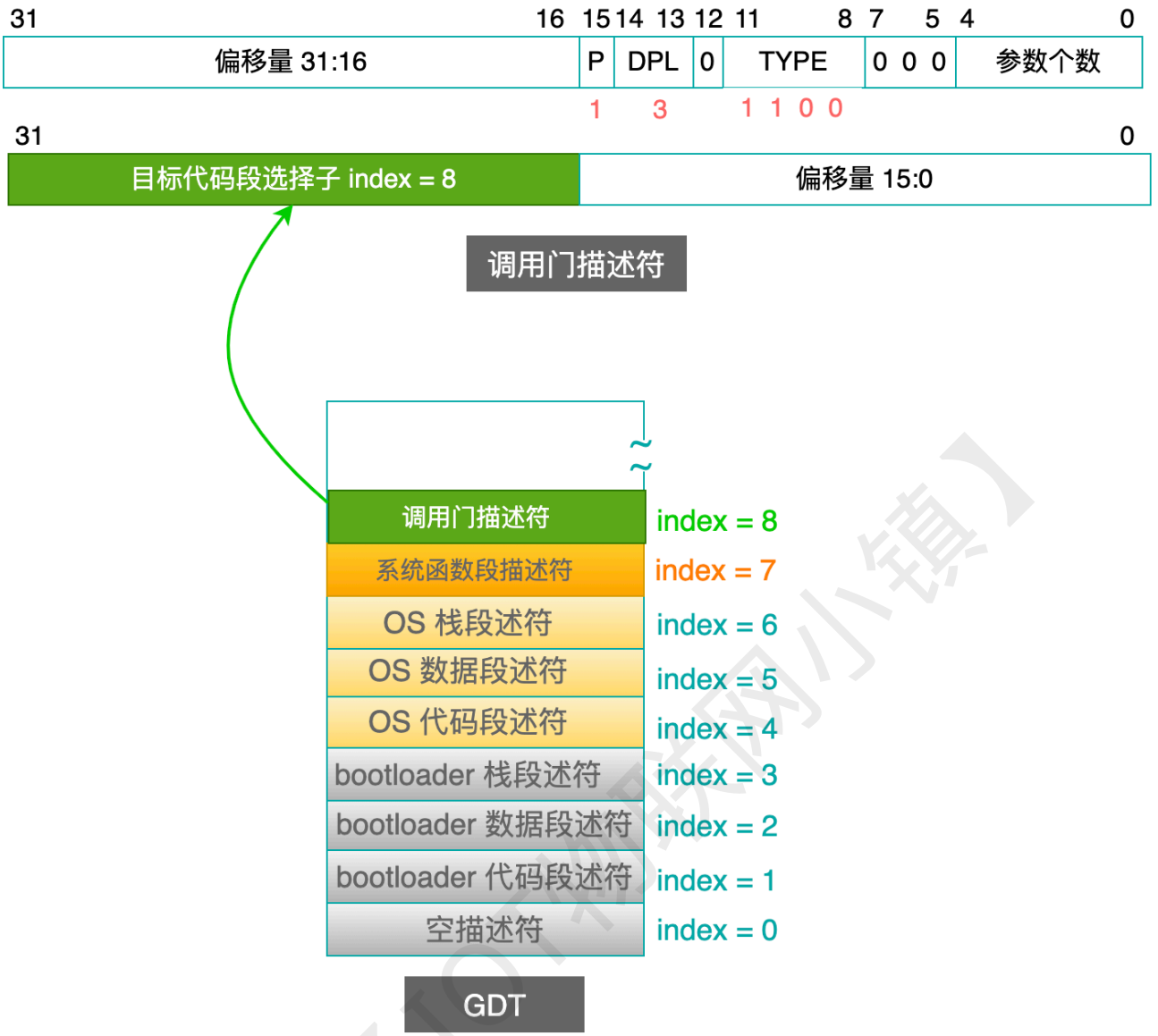
所谓的安装，就是在GDT中构造一个调用门描述符，让它的目标代码段选择子指向真正的代码段。

假设：下面这张图是安装调用门之前的状态：



操作系统提供2个系统函数给用户程序调用，它们的代码位于独立的一个代码段中(在GDT中有一个代码段描述符)。

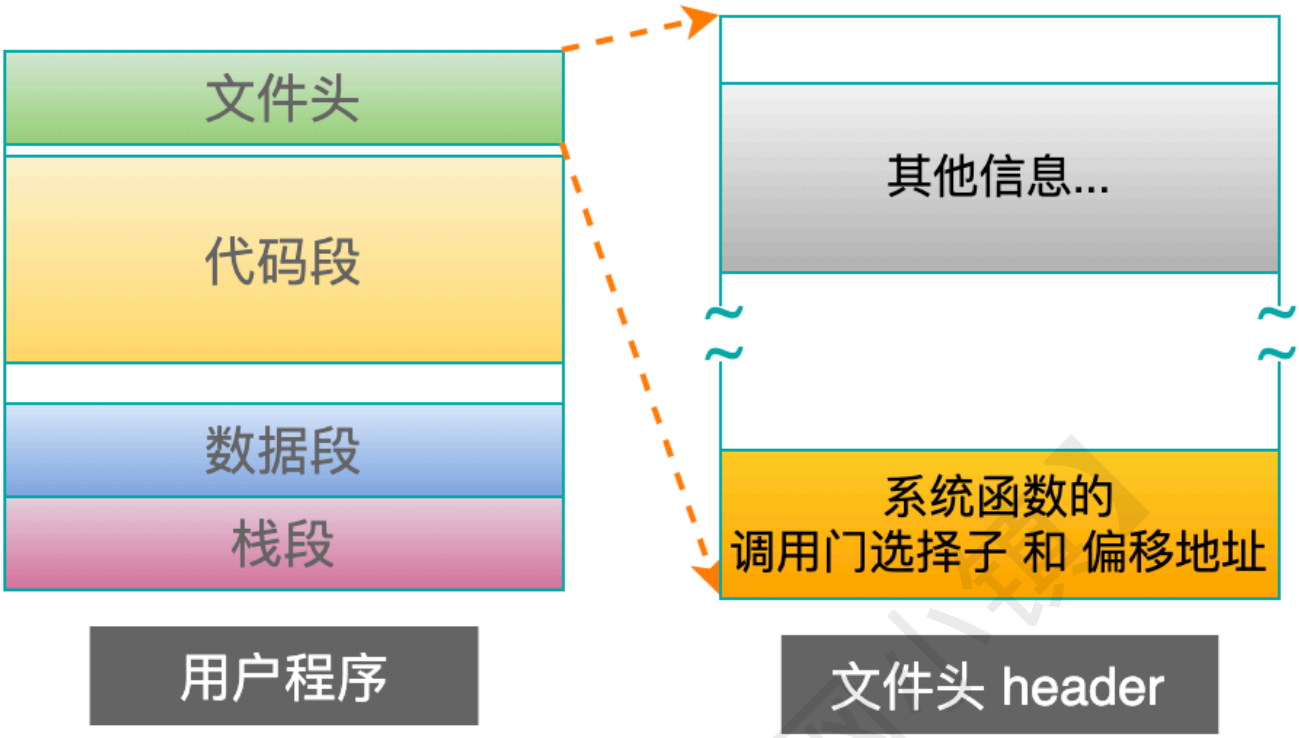
然后在GDT中，新增一个门描述符(index = 8)，描述符中的“目标代码段选择子”中的索引号，就等于 8：



注意：根据前文提到到特权级检查规则，为了让用户程序能正确进入调用门，需要把调用门描述符的DPL设置为3才可以(与用户程序的CPL相同)。

把调用门的选择子告诉用户程序

按照之前的惯例，操作系统可以在用户程序的头部header中的约定位置处，填写调用们的选择子以及函数偏移地址：



选择子的数值为：0x0043(二进制：0000_0000_0100_0011)：

> RPL = 3;
>
> 到 GDT 中去查找;
>
> 索引号 index = 8;

用户程序通过调用门进入系统函数

当用户程序请求调用系统函数时，处理器就开始对这3 方的特权级展开检查：

1. 用户程序的 CPL = 3, RPL = 3;
2. 调用门自身的 DPL = 3;
3. 调用门中的目标代码段选择子所指向的描述符(index = 7)中 DPL = 0;

以上这些特权级的数值满足调用门的特权级规则要求，于是就进入系统函数所在的代码中执行了。

栈的切换

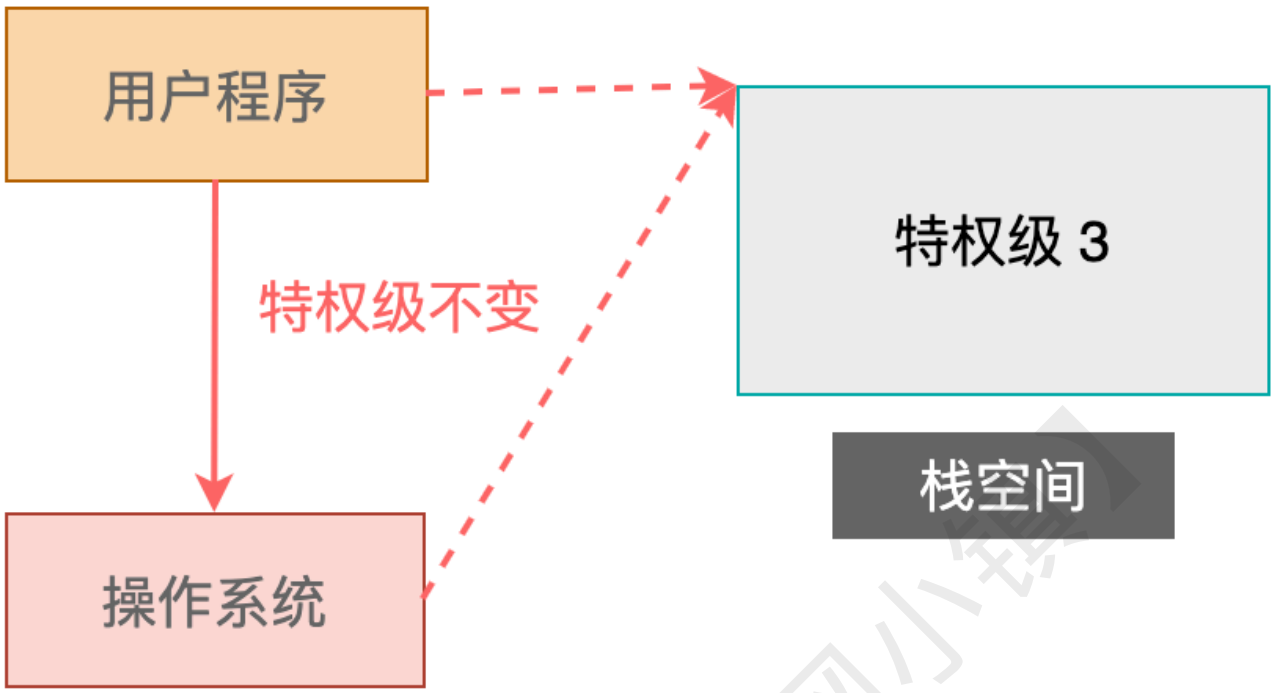
x86处理器要求：当前特权级 CPL 必须与目标栈段的 DPL 相同。

因此，用户程序在进入操作系统中的系统函数之后：

1. 如果特权级 CPL 没有变化

那么在系统函数执行的时候，使用的栈仍然是用户程序之前所使用的那个栈空间。

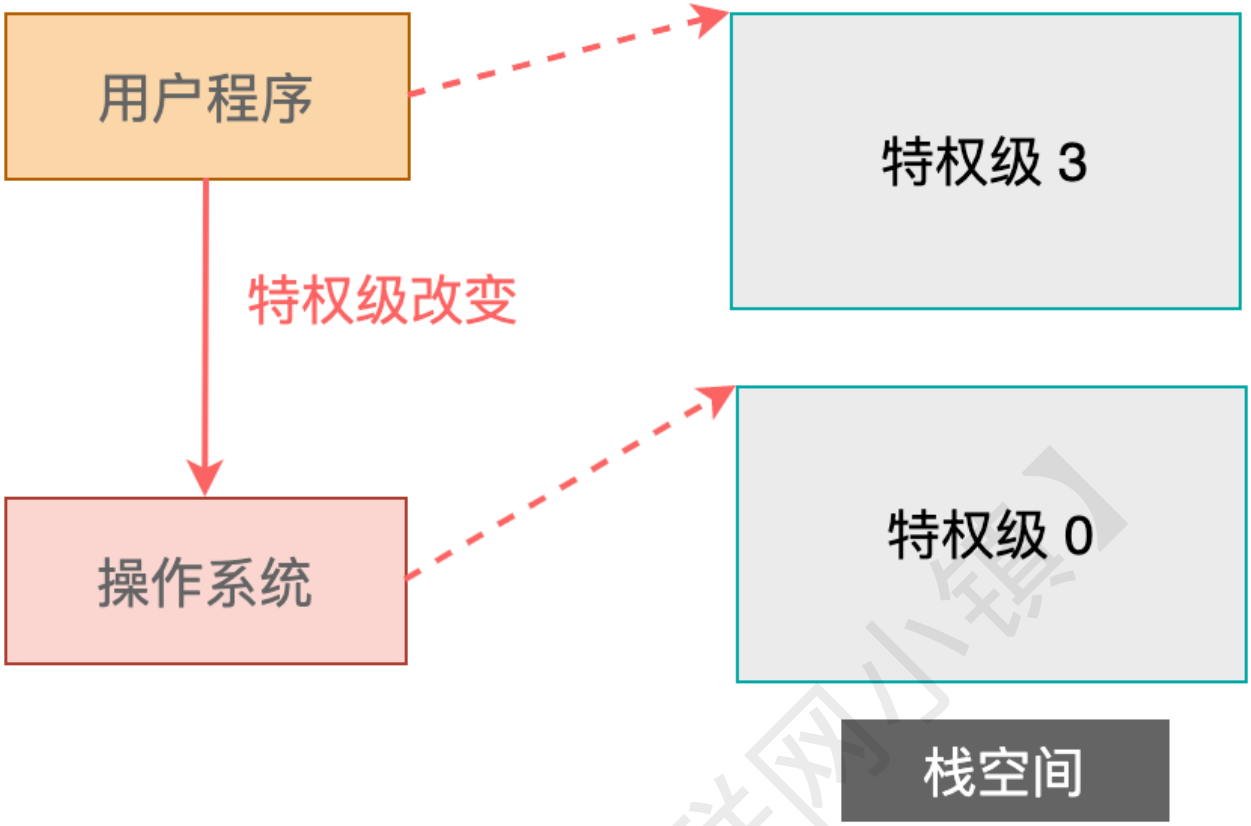
如果用户程序通过栈传递了参数，系统函数可以直接在同一个栈空间中获取到这些参数。



2. 如果特权级 CPL 发生了变化

那么在系统函数执行的时候，就需要切换到用户程序在0 特权级下的栈空间(操作系统在加载用户程序的时候，就提前准备好了)。

同时，处理器会把用户程序在3 特权级下使用的栈空间中的参数，全部复制到0 特权级下的栈空间中，这样的话，系统函数就可以正确获取到这些参数了。



------ End ------

打完收功！

推荐阅读

【1】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
【2】一步步分析-如何用C实现面向对象编程
【3】原来gdb的底层调试原理这么简单
【4】内联汇编很可怕吗？看完这篇文章，终结它！

其他系列专辑：精选文章、C语言、Linux操作系统、应用程序设计、物联网