2021/05/05_Windows32位内核_第10课_调用门

笔记本： Windows32位内核
创建时间： 2021/5/5 星期三 15:06
作者： ileemi

.reload /f -- 强制调试器立即加载符号。

Ring0与Ring3的FS值是不一样的。

控件设置值

!idt -a

操作系统如何实现系统调用

系统调用：由操作系统实现提供的所有系统调用所构成的集合即程序接口或应用编程接口(API)。是应用程序同系统之间的接口。
Ring3程序需要操作硬件就需要调用操作系统提供的API，让操作系统帮助Ring3环的程序去完成对应的硬件操作。Ring0环只需要将操作的结果返回给Ring3程序即可。

系统调用会涉及到权限的切换：

Ring3环程序 ==> 系统调用（==> 切换Ring0 ==> 执行系统调用 ==> 切换ring3)这三步由操作系统去完成。

> 权限切换的方法：

- 异常：执行代码时产生异常，会自动进入到Ring0环
- 中断：产生中段信号，会自动进入到Ring0环。iret 切回到 Ring3，中断门：iret 切回到 Ring3 -- 在中断表中（IDT）
- 调用门（Call Gate）：通过 retf 切回到 Ring3 -- 在GDT表中。32位CPU之前主流
- MSR32（32Bit CPU）：模式指令。32位CPU（奔腾）时主流
- MSR64（64Bit CPU）：现在CPU主流

调用门

> 调用门（Call Gate）：属于系统描述符类型。

系统段描述符和门描述符的类型字段及编码

Table 3-2. System-Segment and Gate-Descriptor Types

Type Field					Description	
Decimal	11	10	9	8	32-Bit Mode	IA-32e Mode
0	0	0	0	0	Reserved	Reserved
1	0	0	0	1	16-bit TSS (Available)	Reserved
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-bit TSS (Busy)	Reserved
4	0	1	0	0	16-bit Call Gate	Reserved
5	0	1	0	1	Task Gate	Reserved
6	0	1	1	0	16-bit Interrupt Gate	Reserved
7	0	1	1	1	16-bit Trap Gate	Reserved
8	1	0	0	0	Reserved	Reserved
9	1	0	0	1	32-bit TSS (Available)	64-bit TSS (Available)
10	1	0	1	0	Reserved	Reserved
11	1	0	1	1	32-bit TSS (Busy)	64-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate	64-bit Call Gate
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	32-bit Interrupt Gate	64-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate	64-bit Trap Gate

> 注意，IA-32Bit模式下的系统描述符是16个字节，而不是8个字节。调用门的标志为：12（C）

调用门的调用指令：

Table 5-1. Privilege Check Rules for Call Gates

Instruction	Privilege Check Rules
CALL	CPL ≤ call gate DPL; RPL ≤ call gate DPL Destination conforming code segment DPL ≤ CPL Destination nonconforming code segment DPL ≤ CPL
JMP	CPL ≤ call gate DPL; RPL ≤ call gate DPL Destination conforming code segment DPL ≤ CPL Destination nonconforming code segment DPL = CPL

> 一般使用call，调用门对应的函数跑完后，需要从Ring0切回到Ring3，同时要恢复寄存器环境。

表的格式

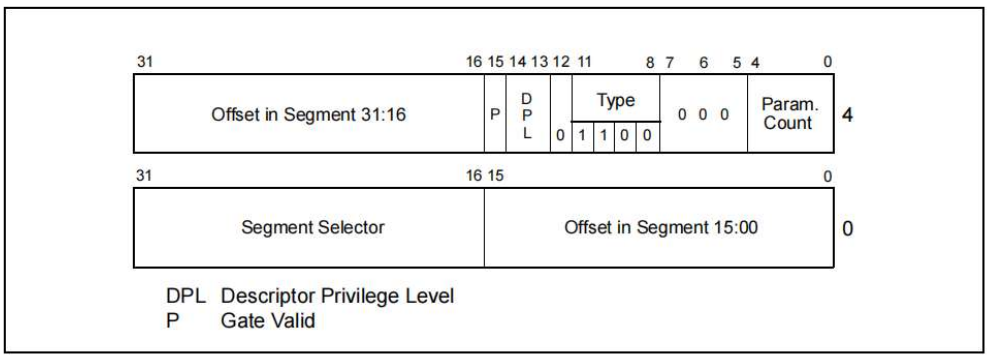

Figure 5-8. Call-Gate Descriptor

跨权限级别调用过程中的堆栈交换

Ring3环程序的堆栈结构以及Ring0环的堆栈结构如下：

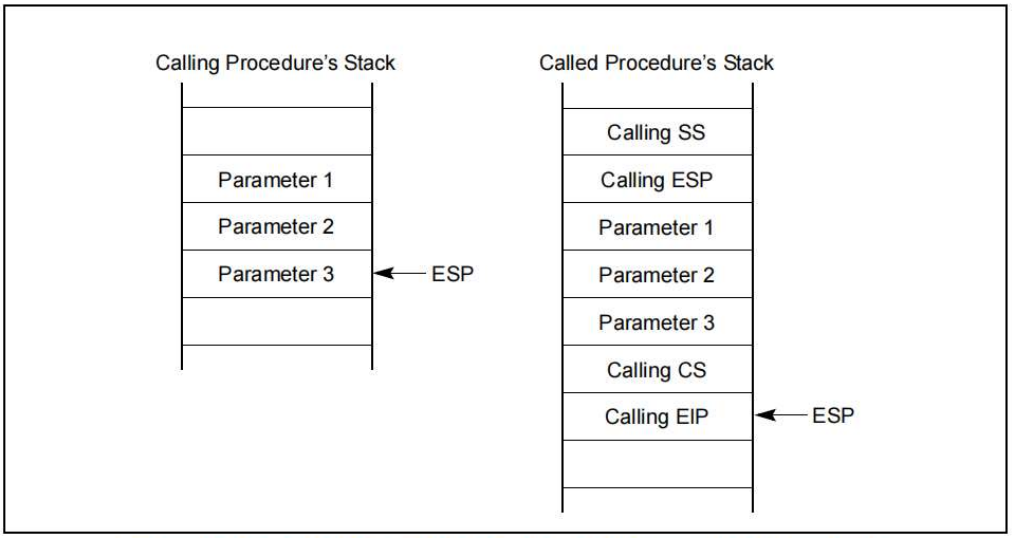

Figure 5-13. Stack Switching During an Interprivilege-Level Call

> Ring0处理完在返回到Ring3时，原Ring3的栈结构（寄存器环境）不能被破坏。为了在返回Ring3程序前（retf）正确恢复寄存器环境，堆栈不受破坏，可以将Ring0对应的函数声明为裸函数（ `__declspec(naked)` ）。也就是API的实现代码需要使用汇编代码编写。

当函数为裸函数时，由于编译器不会对函数添加额外的汇编代码，所以函数内需要自己保存和恢复环境。

假设内核中有一个漏洞可以写任意的地址，可以将该地址写入到GDT表的第9项，高16位低16位填写目标函数（Ring3）的函数首地址。当 call 调用门的是否，就会调用Ring3环的函数，此时该函数就有了内核权限。这种做法就是 "无驱动进内核"。

通过 "dg 0 200" 查看操作系统内部的调用门函数。发现内部并没有调用调用门。Ring0环寄存器CS的值为：0x8，寄存器SS的值为：0x10；而Ring3环寄存器CS的值固定为：0x18，寄存器SS的值为：0x23；

切换权限时，由于EIP、ESP需要进行切换，其对应的段寄存器CS、SS也需要切换。

由于Ring0和Ring3的FS寄存器的值不一样，在调用调用门时需要手动切换（Ring3程序使用 "system("pause"); 时就会影响到FS的值"）。所以在Ring0环的API实现函数中需要首先将寄存器FS的值切换为Ring0的值（0x30），在 retf 之前恢复Ring3寄存器FS的值（0x3b）。操作系统不被调试的前提下且Ring0API函数的实现中没有修改寄存器FS的值（Ring0：0x30），就会导致操作系统蓝屏。

> 当使用Windbg调式Ring3时（内部有断点会权限切换到Ring3），由于Windbg有"dt _KPCR" 这条指令，该指令会检测当前寄存器FS的值是否有效，如果值不是Ring0的值（0x30），Windbg会自动对其进行修改。同时Windbg从Ring0回到Ring3时会将寄存器FS的值修改为0x30，在执行retf时会将其值置为0x0。这就导致Ring3程序会出现崩溃情况。

但是不调试Ring3程序（Ring3程序中没有断点），正常修改FS的值，从Ring0切回到Ring3时，其对应的FS就不会受到影响。

```
// Ring0 API实现代码
__declspec(naked) void SysCall() {
  __asm {
    mov ax, 0x30
    mov fs, ax // 切换Ring0的FS
  }

  //...

  __asm {
    mov ax, 0x3b
    mov fs, ax // 切换Ring3的FS
    retf
  }
}
```

当驱动程序卸载后，Ring3程序在调用调用门，操作系统就会出现蓝屏（GDT表中对应项是有效的，但是其对应的回调函数不存在）。在卸载驱动时，需要将GDT表中对应项设置为无效（更改存在位即可）。

当操作系统API较多时，对应的GDT表项数量就不够用（一个函数一个调用门的前提下）。为此操作操作系统就需要设计API较多，如何存在的问题。

> Ring3环 API 由 ntdll.dll 负责管理，也就是说任何一个程序调用操作系统提供的API都会加载 "ntdll.dll"。