

保护模式 简写笔记

CPU权限的作用：

代码段

CS 代码段

数据段

DS 作用于地址上

数据

SS 描述

ES 额外段

movsb

movsw

edi

esi

FS 是 扩展，在windows 的作用中

应用层保存是 TEB地址，内核中保存是KPCR

gs 在 x86没用

每一个线程 由两个结构来描述 内核描述的结构叫 ETHREAD 执行体，微内核

应用层描述的结构叫TEB thread env... block

局部变量：

```
源.cpp  + x
简单测试 (全局范围)

4  int main()
5  {
6      MessageBoxA(0, 0, 0, 0);
7      int a;
8      a = 0;
9      __asm
10     {
11         mov dword ptr[a], eax;
12     }
13
14     return 0;
```

简单测试.exe - [LCG - 主线程, 模块 - 简单测试]

看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 Tools

l e m t w l c p k

地址	汇编指令
F3:AB	rep stos dword ptr es:[edi]
A1 04A01800	mov eax,dword ptr ds:[0x18A004]
33C5	xor eax,ebp
8945 FC	mov dword ptr ss:[ebp-0x4],eax
8BF4	mov esi,esp
6A 00	push 0x0
6A 00	push 0x0
6A 00	push 0x0
6A 00	push 0x0
FF15 98B01800	call dword ptr ds:[<USER32.MessageBoxA>]
3BF4	cmp esi,esp
E8 CEFAFFFF	call 简单测试.00101200
C745 F4 000000	mov dword ptr ss:[ebp-0xC],0x0
8945 F4	mov dword ptr ss:[ebp-0xC],eax

修改段权限，使地址不能访问。只能作用于x86系统：

x64 mov不能修改 ds段权限。

地址	汇编指令
7	int a;
8	a = 0;
9	__asm
10	{
11	mov ax, cs;
12	mov ds, ax;
13	mov eax, 0x100; 已用时间 <= 1ms
14	mov dword ptr ds:[a], eax;
15	mov ax, es;
16	}
17	
18	return 0;
19	
20	}

x64上, mov ds,xxx 没用

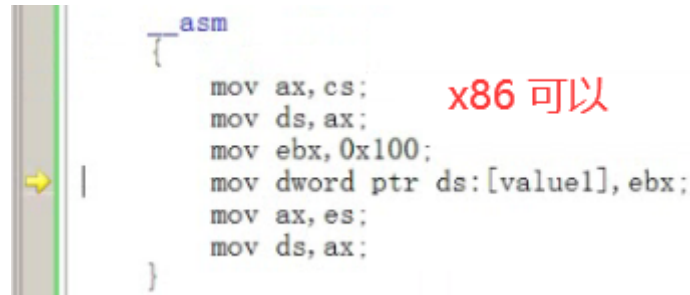
查看选项

地址	汇编指令
00F11725 89 45 FC	mov dword ptr [ebp-4],eax
//MessageBoxA(0, 0, 0, 0);	
int a;	
a = 0;	
00F11728 C7 45 F4 00 00 00 00	mov dword ptr [a],0
__asm	
{	
00F1172F 66 8C C8	mov ax, cs;
00F11732 66 8E D8	mov ds, ax;
00F11735 B8 00 01 00 00	mov eax, 100h;
00F1173A 89 45 F4	mov dword ptr ds:[a], eax;
00F1173D 66 8C C0	mov ax, es;
}	

寄存器

EAX = 43760023 EBX = 007A2000 ECX = 00000000 EDX = 00D4CD13 ESI = 00F11325
EBI = 008FFCA4 EIP = 00F11735 ESP = 008FFBC8 EBP = 008FFCA4 EFL = 00000206
CS = 0023 DS = 002B ES = 002B SS = 002B FS = 0053 GS = 002B

x86 可以用mov 修改ds段权限:



```
asm
{
    mov ax, cs;
    mov ds, ax;
    mov ebx, 0x100;
    mov dword ptr ds:[value1], ebx;
    mov ax, es;
    mov ds, ax;
}
```

x86 可以

X64(VS) 1909 FS的 TEB:

mov eax,dword ptr fs:[0x30 或 0x0], 取不到值。

```
struct _TEB
{
    struct _NT_TIB NtTib; //0x0
    VOID* EnvironmentPointer; //0x38
    struct _CLIENT_ID ClientId; //0x40
    VOID* ActiveRpcHandle; //0x50
    VOID* ThreadLocalStoragePointer; //0x58
    struct _PEB* ProcessEnvironmentBlock; //0x60
    ULONG LastErrorValue; //0x68
    ULONG CountOfOwnedCriticalSections; //0x6c
    VOID* CsrClientThread; //0x70
```

```
//0x38 bytes (sizeof)
struct _NT_TIB
{
    struct _EXCEPTION_REGISTRATION_RECORD* ExceptionList; //0x0
    VOID* StackBase; //0x8
    VOID* StackLimit; //0x10
    VOID* SubSystemTib; //0x18
    union
    {
        VOID* FiberData; //0x20
        ULONG Version; //0x20
    };
    VOID* ArbitraryUserPointer; //0x28
    struct _NT_TIB* Self; //0x30
};
```

其他人的方法: https://www.baidu.com/link?url=-5s75alYe-cfLosYNa-4DyV2_MRkb_YiMU78ZbbBz62ByUC4z0WiKeNZ1xdkvLkITXkX6gffpNZMIxYXSZeI-IHkhGuPGBbxrLyKIKGS3B_&wd=&eqid=e1b0952a00175cb6000000065f8c2e89

x64进程如何获得wow64进程的PEB

原创

nLif

2015-07-23 15:15:46

4532

★ 收藏 2

版权

分类专栏: x64进程如何获得wow64进程的PEB

android 破解 反编译

文章标签:

x64进程如何获得wow64进程的PEB

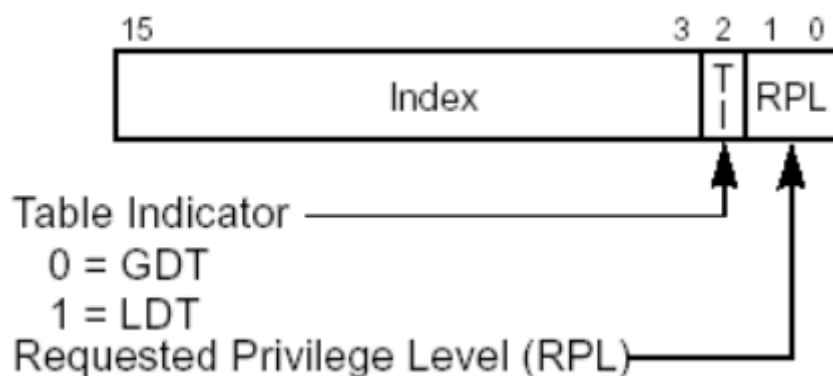
如果通过ZwQueryInformationProcess查询的到的是wow64进程的native层PEB, 有的信息并不是我们想要的

wow64进程的实质是通过2个不同的代码段, 共享一个相同的数据段来确保一个进程内能同时运行32位程序和64位

这2个段的CS分别位23 和33, 不过这个和PEB无关,

PEB的位置: (1) native的gs段 (2) wow64的是fs段

1.段选择符拆分:



例如: FS=003B

拆分后为:

0000 0000 0011 1011 -->取后两位 0000 0000 0011 10-->取GDT/LDT位 0000 0000 0011 1

重新组合0000 0000 0000 0111,代表查 GDT表的 第七项 (从0开始数)。(索引)

```
kd> dq 80b99000 L20
80b99000 00000000`00000000 00cf9b00`0000ffff
80b99010 00cf9300`0000ffff 00cf9b00`0000ffff
80b99020 00cff300`0000ffff 80008b1e`400020ab
80b99030 834093f3`4c003748 0040f300`00000fff
```

或者 把0x3B && 0xFF8 =0x38;

2.段描述符:

段限长和段base:

$(0x000FFFFFF + 1) * 0x1000 / 1024 / 1024 / 1024 = 4G$; //限长

base不用说

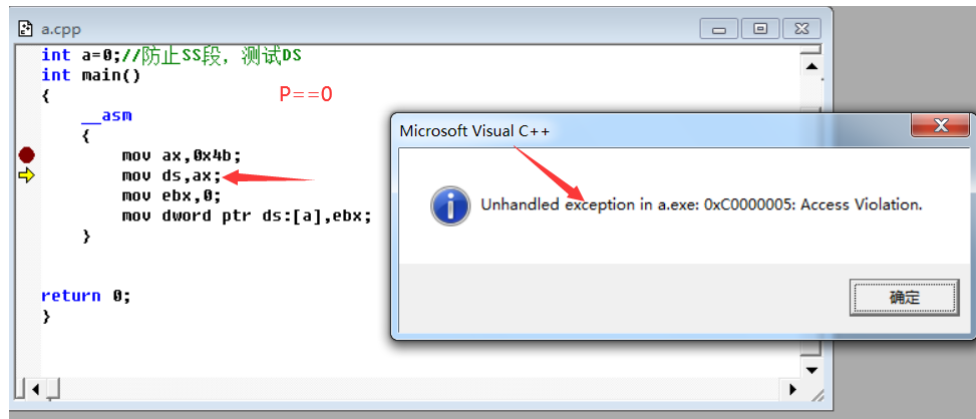
96位 寄存器（表）：

类似于一个缓存，在段选择子不变的情况下，目的是不让CPU每次都去查GDT表，提高效率。

64位段描述符+16位段选择符+12位（G=1 补的3个0）+ 为了对齐的 4位

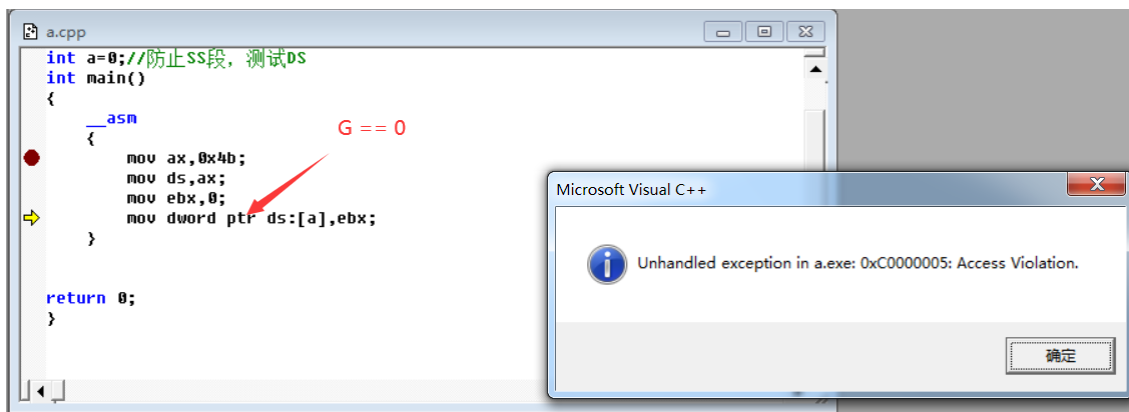
P位（第二天）：

P = 0 为无效段，P=1 代表段描述符有效。



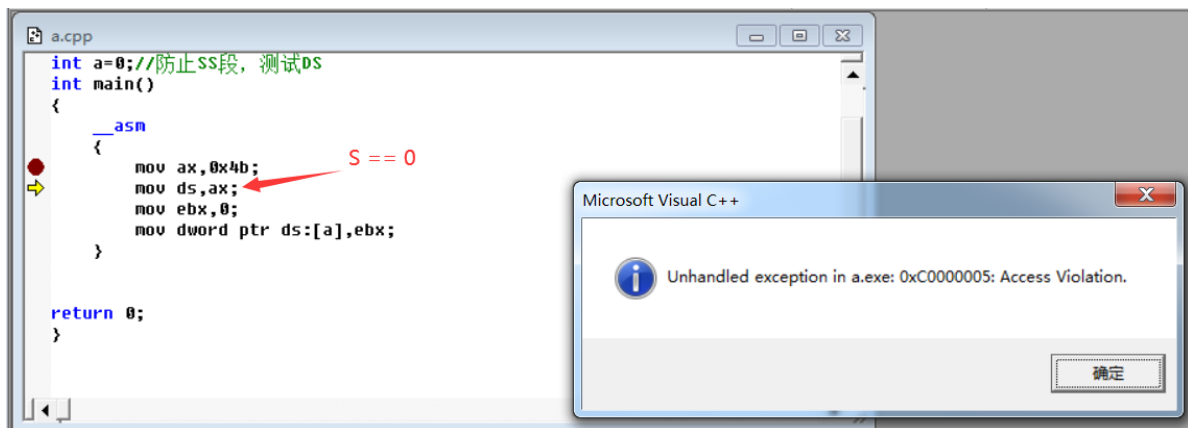
G位（第二天）：

0 = 字节对齐，1 = 页对齐 = 4KB = 0x1000 BYTE



S位（第二天）：

S == 0 == 系统段描述符，S == 1 == 用户段描述符。



TYPE域 (第二天) :

小于 8 数据段，大于 8 代码段。

Table 3-1. Code- and Data-Segment Types

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read-Only, conforming
15	1	1	1	1	Code	Execute/Read-Only, conforming, accessed

DS:

A 位: mov ds,0x48就置为1。 A位是给CPU使用的，微软来统计哪些段常用，现在已经废除A位了。

W 位: 读写位。清0后，不能写入数据。但是，段限制了，页也能用。

CS: 常用 10 和 11

C 位: C==1==conforming为一致代码段，应用层调内核函数（只能R3调R0）。但是仍然有页限制。

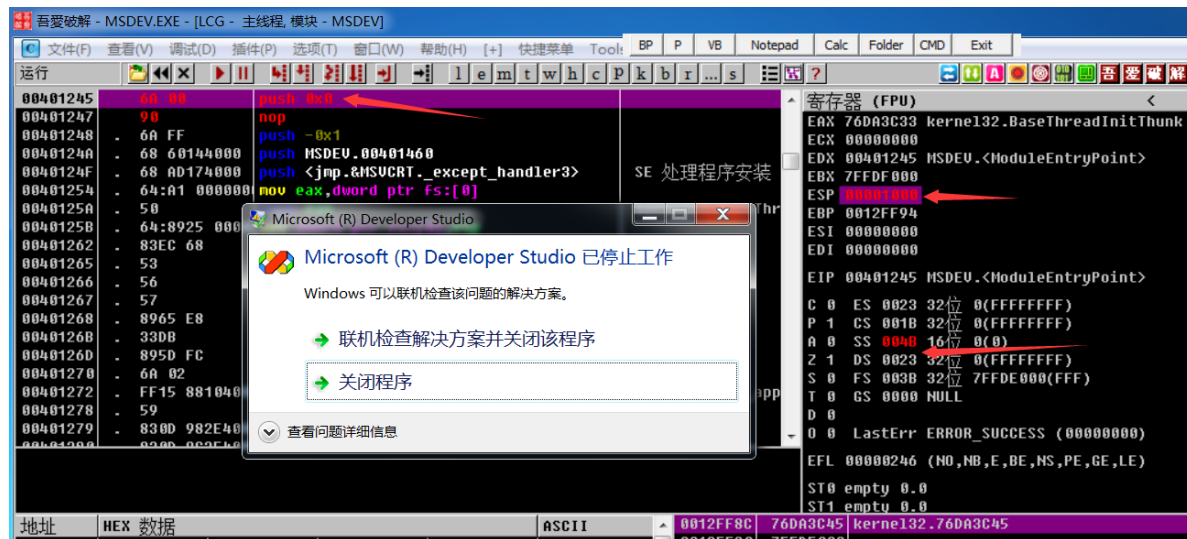
D\B位(第三天):

D\B位 作用于 代码段，目的是描述代码段的默认操作数。 注意：64位下废除。

当D\B==1时，按操作系统的默认操作数。32位系统32位操作数。

当D\B==0时，32位系统16位操作数。

SS段:D\B位改为0,一直挂。



DS段:D\B位改为0，修改为16位操作数。

向上/下拓展(第三天):

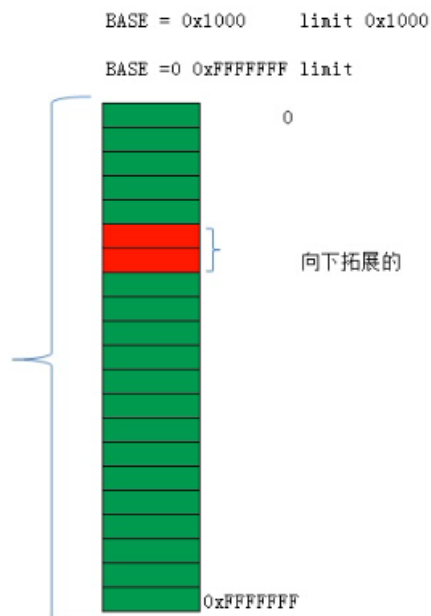
windows 没有采用 向下拓展 (因为难以理解

其次对 SS堆栈段的限制 是 0xFFFF

1.向上拓展: 从 base 到Limit可访问。



2.向下拓展: 从 base 到Limit不可访问。



DPL(第四天):

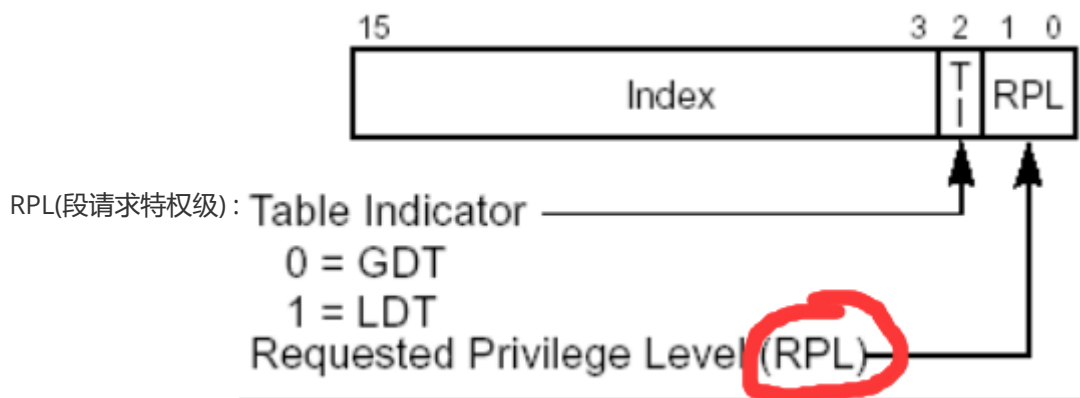


Figure 3-6. Segment Selector

CPL (当前的特权级): 就是CS段的RPL。

DPL (段的描述权限级), 从DPL位可以看出, 操作系统没有使用 1环 2环。

```
mov ax,0x4b //每行代码都会检查当前的CPL,RPL (CPL=RPL),DPL。不卡是因为96位缓存的存在。
mov ds,ax
```

实验1: 修改0x4b指向的DPL(9), 执行 `mov ax,0x4b`语句, 结果无影响

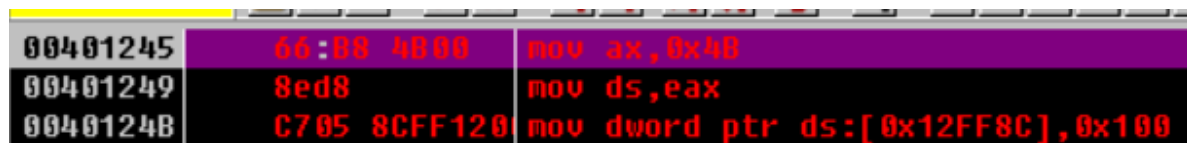
实验2: 修改0x4b指向的DPL(9), 执行 `mov ax,0x4b`; `mov ds,ax`, 结果第二条语句触发异常。



img

数据段权限检查规则:

修改0x4b指向的DPL(9), 执行 `mov ax,0x4b`; `mov ds,ax`; 重点 `mov dword ptr ds:[地址],0x100`;



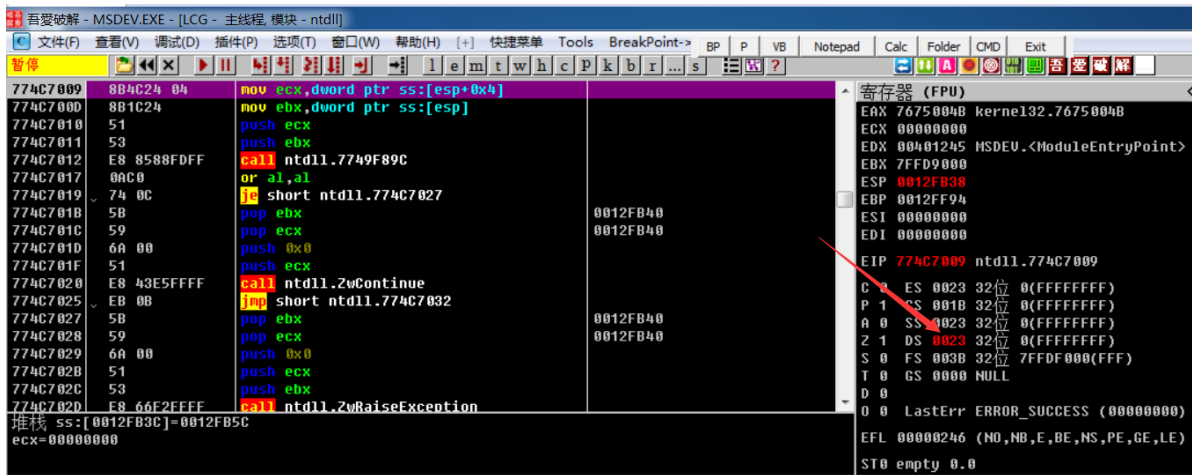
执行到第三条语句时, 修改0x4b的DPL为 00 (9)

```
kd> eq 80b99048 00cf9300`0000ffff
kd> g
```

再执行第三条语句, 程序触发异常崩掉。



触发异常跑到内核空间, 又修改回原来的 ds:



为什么修改GDT表，没有修改缓存，也能影响代码执行？是因为windows是多任务的操作系统，在任务切换的时候会频繁刷新缓存。

总结：DPL的数值不能小于CPL（或者是RPL），也就是DPL的权限不能大于CPL（或RPL）

继续实验探究：

00401245	66:B8 4800	mov ax, 0x48
00401249	8ed8	mov ds, eax

CPL=3,RPL=0,DPL=3 (目前已知DPL>=CPL和DPL)

结果：运行成功。

继续探究，当CPL=3,RPL=0,DPL=0时：

00401245	66:B8 4800	mov ax, 0x48
00401249	8ed8	mov ds, eax
0040124B	C705 8CFF120	mov dword ptr ds:[0x12FF8C], 0x100

结果：触发异常。

得出结论：DPL>=CPL。目前还不知道RPL和DPL的关系（可以默认为 RPL<=DPL）

代码段权限规则检查：

跨段跳转：E9(JMP) CALL(E8)

```
char buf[6]={0,0,0,0,0x4b,0x0};
*(int *)&buf[0]=(int)函数地址;
call fword ptr buf; //实际上是 fword ptr ss:[buf] 因为局部变量的原因
```

```

__declspec(naked) void test()
{
    printf("6666");
}
int main()
{
    char buf[6]={0,0,0,0,0x4b,0};
    *(int *)buf=(int)test;
    __asm
    {
        call fword ptr buf;
    }

    return 0;
}

```

先push (原来的)段选择子 然后push返回地址;

地址:	esp				
0012FEEC	8A D4 40 00	0.00000000			
0012FEF0	1B 00 00 00	0.00000000			
0012FEF4	00 00 00 00				
0012FEF8	00 00 00 00				
0012FEFC	00 40 FD 7F	0.00000000			
0012FF00	CC CC CC CC				
0012FF04	CC CC CC CC				

如何返回?

```

__declspec(naked) void test()
{
    printf("6666");
    retf;
}

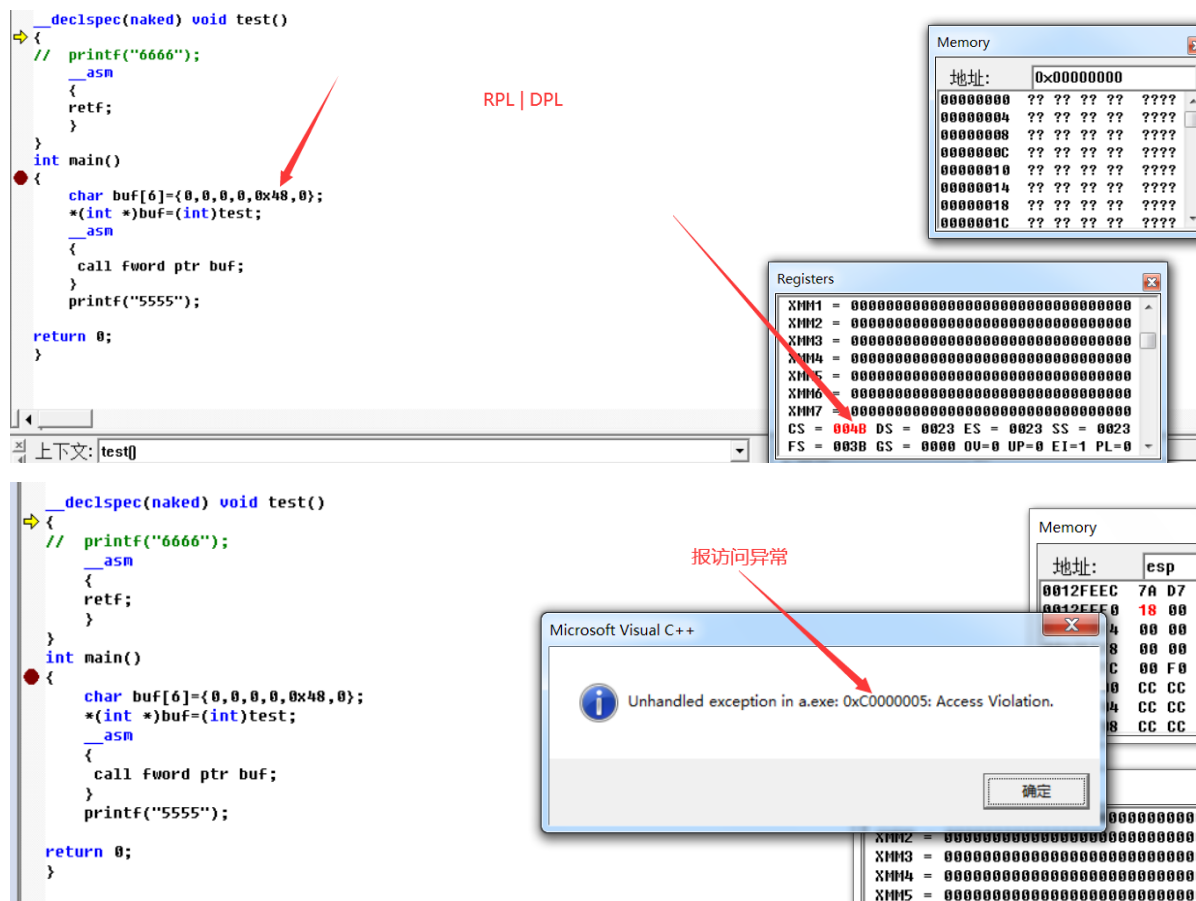
__declspec(naked) void test()
{
    printf("6666");
    __asm
    {
        retf;
    }
}
int main()
{
    char buf[6]={0,0,0,0,0x4b,0};
    *(int *)buf=(int)test;
    __asm
    {
        call fword ptr buf;
    }
    printf("5555");

    return 0;
}

```

C:\Users\TalS
66665555

实验: DPL=3 RPL=0 CPL=3。结论: CPL=DPL

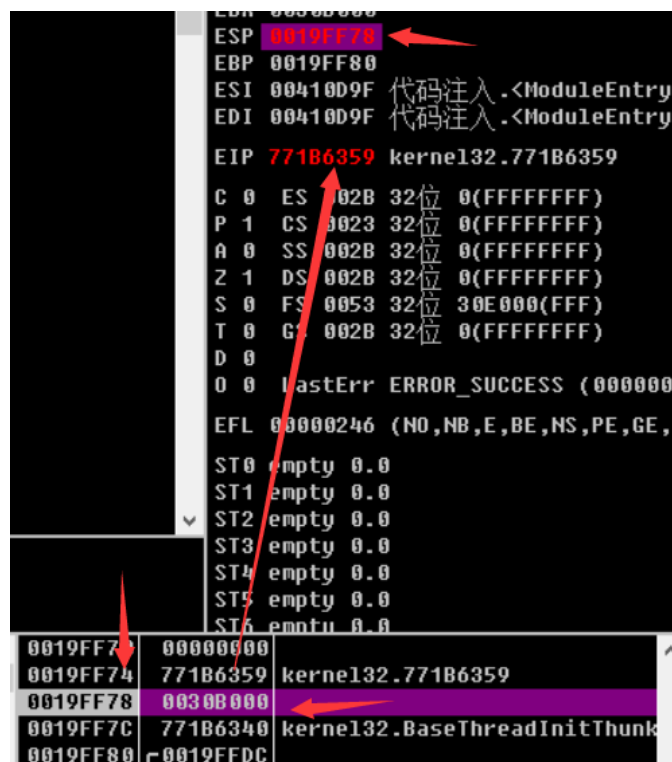


结论：JMP和CALL 只能用于同权限，或者往高权限跳。

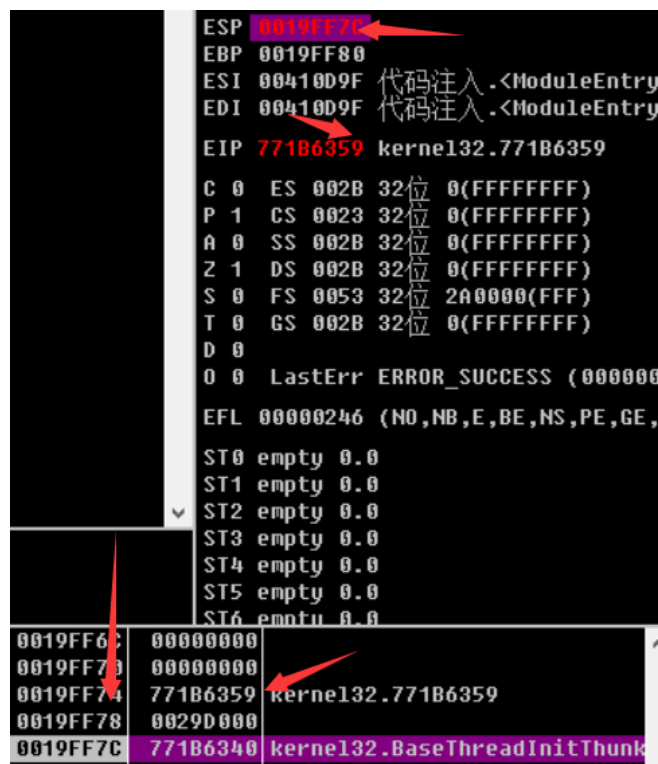
retf 和 iretd 只能用于同权限，或者往低权限返回。

ret和ret 0x4:

ret 默认== retn --> 把esp当前的值赋给eip，然后esp+4;



ret 0x4 == retn 0x4 -->把 esp当前值赋给eip，然后esp+8;



3.系统描述符

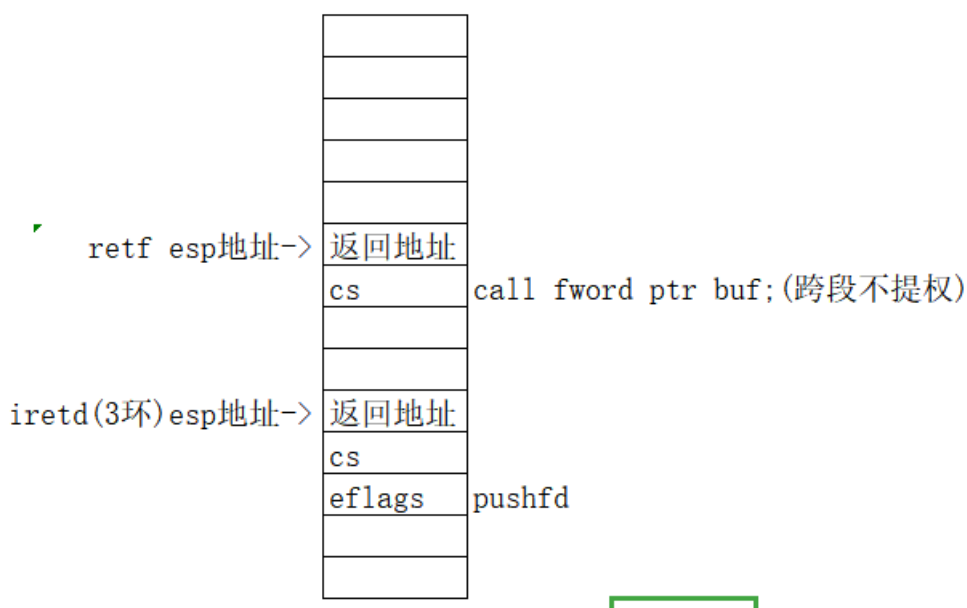
调用门(S==0的情况):

iret和iretd:

iret==iretd (32位下) (dword) iretq (64位) (qword) iretf(fword)

iretd 和 int 0x (异常) 匹配。

iretd 和 iretf 的区别在于, iretd 后面不能加0x字节, iretf 后面可以加 0x字节 == ret 0x;



调用门描述符 (探究S==1的情况下的段描述符):

0x12345678 按34:16 到15:00的顺序依次填入。

Param Count : 参数个数, 一共占5位 $2^5=32$ 位, 最多可有31个参数。

~xxxxEC00`段选择符xxxxx （保证段选择符处于0环，DPL一定是位于3环。P位一定有效。
调用门一定是 call指令来调用，jmp指令不能提权。

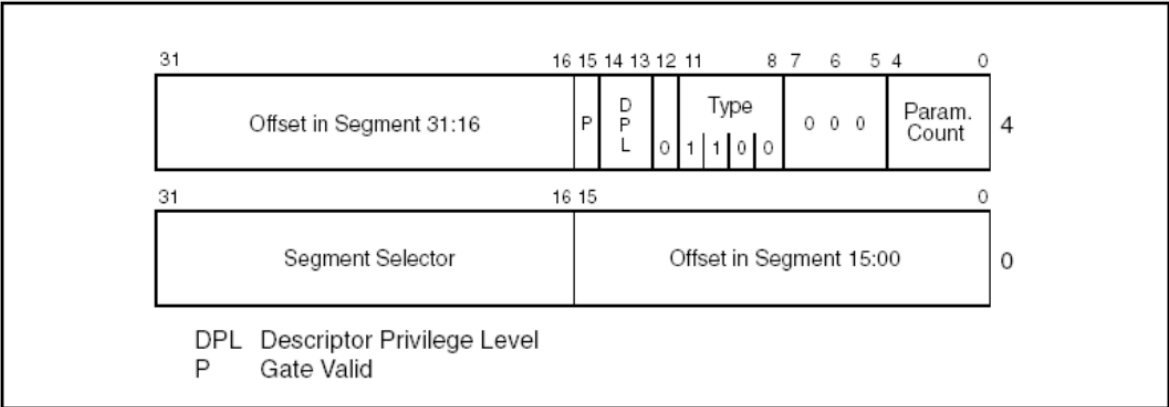


Figure 4-7. Call-Gate Descriptor

普通段描述符：

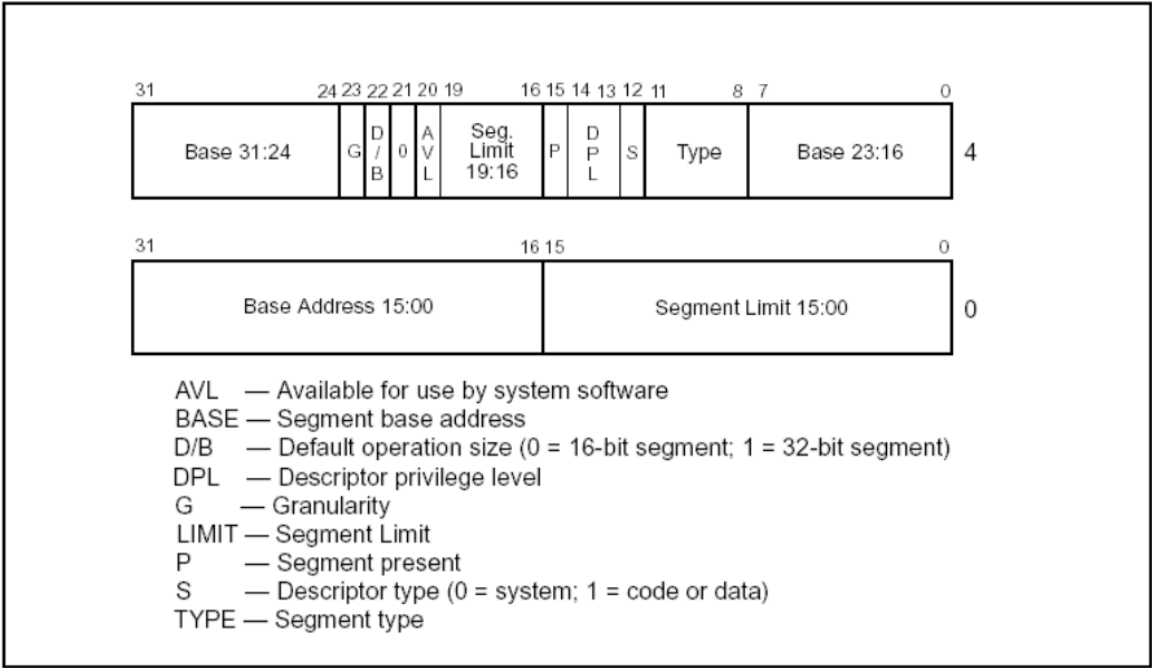


Figure 3-8. Segment Descriptor

注意：此时 (S==0) Tpye位变为 系统描述符。

Type:

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

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

中断门

增量链接和随机地址ASLR:

配置属性
常规
高级
调试
VC++ 目录
C/C++
链接器
常规
输入
清单文件
调试
系统
优化
嵌入的 IDL
Windows 元数据
高级
所有选项
命令行
清单工具
XML 文档生成器

输出文件 \$(OutDir)\$(TargetName)\$(TargetExt)
显示进度 未设置
版本
启用增量链接 否 (/INCREMENTAL:NO)
取消显示启动版权标志 是 (/NOLOGO)
忽略导入库 否
注册输出 否
逐用户重定向 否
附加库目录
链接库依赖项 是
使用库依赖项输入 否
链接状态
阻止 Dll 绑定
将链接器警告视为错误
强制文件输出
创建可热修补映像
指定节特性

配置属性
常规
高级
调试
VC++ 目录
C/C++
链接器
常规
输入
清单文件
调试
系统
优化
嵌入的 IDL
Windows 元数据
高级
所有选项
命令行
清单工具
XML 文档生成器

入口点
无入口点 否
设置校验和 否
基址
随机基址 否 (/DYNAMICBASE:NO)
固定基址
数据执行保护(DEP) 是 (/NXCOMPAT)
关闭程序集生成 否
卸载延迟加载的 DLL
取消绑定延迟加载的 DLL
导入库
合并节
目标计算机 MachineX86 (/MACHINE:X86)
配置文件 否
CLR 线程特性
CLR 映像类型 默认映像类型
密钥文件
密钥容器
延迟签名

CALL Gate 代码实验:

1.关掉增量链接 ASLR随即地址 开启MT多线程生成。

2.windbg eq xxxxEC00`0008xxxx 然后执行代码

注意: 0008作为新的段选择符, 有些细节。

知识点更新 (S位):

S位==0时, 为系统描述符, 为1时 是描述代码, 数据段的 S==1。这也解释了为什么 系统门的DPL =3。

```
kd> dq gdtr L20
80b99000 00000000`00000000 00cf9b00`0000ffff
80b99010 00cf9300`0000ffff 00cf9b00`0000ffff
80b99020 00cff300`0000ffff 80008b1e`400020ab
80b99030 834093f6`dc003748 0040f300`0000ffff
80b99040 0000f200`0400ffff 0040ec00`00101000
80b99050 830089f6`b0000068 830089f6`b0680068
80b99060 00000000`00000000 00000000`00000000
80b99070 800092b9`900003ff 00000000`00000000
80b99080 00000000`00000000 00000000`00000000
80b99090 00000000`00000000 00000000`00000000
80b990a0 86008940`71c00068 00000000`00000000
80b990b0 00000000`00000000 00000000`00000000
80b990c0 00000000`00000000 00000000`00000000
80b990d0 00000000`00000000 00000000`00000000
80b990e0 00000000`80b99100 00009200`0000ffff
80b990f0 830092e7`a97003b2 00009200`0000ffff
kd> eq 80b99060 00cf9b00`0000ffff
kd> eq 80b99048 0040ec00`00601000
kd> g
Break instruction exception - code 80000003 (first chance)
0060:00401000 cc int 3
```

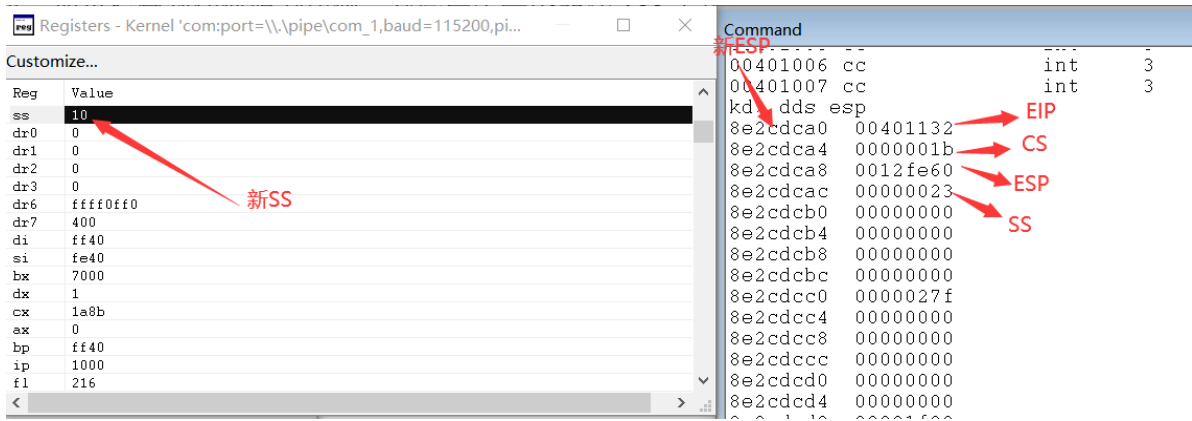
代码段 Exec/Read
1001 用户段描述 权限为0环
当数据段的0环描述符替换时, 报异常错误
复制到60的位置
指向60
成功断下

Registers - Kernel 'com:port=\\.\pipe\com_1,baud=115200,pi...	
Customize...	
Reg	Value
gs	0
fs	3b
es	23
ds	23
edi	12ff40
esi	20fe40
ebx	7ffd7000
edx	1
ecx	d3ce1a8b
eax	0
ebp	12ff40
eip	401000
cs	60
efl	216
esp	8e2cdca0
ss	10

3.进R0时 call fword 和R3时, push的不一样。

kd>dds esp //四字节查看内存

```
kd> dds esp
8e2cdca0 00401132
8e2cdca4 0000001b
8e2cdca8 0012fe60
8e2cdcac 00000023
8e2cdcb0 00000000
8e2cdcb4 00000000
8e2cdcb8 00000000
8e2cdcbc 00000000
8e2cdcc0 00000027f
```



原理:

操作系统要负责创建栈和所有特权级使用的栈段的描述符，还要负责将那些栈的初始指针载入 TSS 中。每个栈都必须是可读写的（在它的段描述符中的类型域定义），而且必须有足够的空间（在限长域中定义）来持有如下这些数据：

- 调用例程的 SS, ESP, CS 和 EIP 等寄存器中的内容
- 被调用例程所需的参数和临时变量
- 当隐含调用异常或中断处理函数时，还要存放 EFLAGS 寄存器和出错码

栈需要有足够的空间来包含许多帧上述所列的内容，因为一个例程经常会调用其他的例程，并且操作系统可能支持多重中断的嵌套。每个栈必须在它的特权级内足够大，这样才能顾及最糟糕的情况。

kd> u (默认是ub) 00401132 //反汇编返回地址（保证是正确的返回地址）

```
kd> u 00401132-8
0040112a 0100 add dword ptr [eax],eax
0040112c 83c404 add esp,4
0040112f ff5df0 call fword ptr [ebp-10h]
00401132 68b8514400 push 4451B8h
00401137 e867d60100 call 0041e7a3
0040113c 83c404 add esp,4
0040113f 33c0 xor eax,eax
00401141 52 push edx
```

代码:

```
#include<stdio.h>
#include<windows.h>
__declspec(naked) void a()
{
    __asm
    {
        int 3;
    }
}
```



```

        retf;
    }
}
int main()
{
    char buf[6]={0,0,0,0,0x48,0};
    printf("%08x",a);
    system("pause");
    __asm
    {
        call fword ptr buf;
    }
    system("pause");
    return 0;
}

```

实验提权调用R0的DbgPrint:

1. Kd>uf nt!DbgPrint //反汇编整个函数

```

kd> uf nt!Dbgprint
nt!DbgPrint:
83e5141f 8bff          mov     edi,edi
83e51421 55           push    ebp
83e51422 8bec          mov     ebp,esp
83e51424 51           push    ecx
83e51425 6a01          push    1
83e51427 8d450c        lea     eax,[ebp+0Ch]
83e5142a 50           push    eax
83e5142b ff7508        push    dword ptr [ebp+8]
83e5142e b93048e983    mov     ecx,offset nt! ?? ::FNODOBFM::`string' (83e94830)
83e51433 6a03          push    3
83e51435 6a65          push    65h
83e51437 e844590b00    call    nt!vDbgPrintExWithPrefixInternal (83f06d80)
83e5143c 59           pop     ecx
83e5143d 5d           pop     ebp
83e5143e c3           ret

```

2. 在 wdk 里面偷定义。

```

ULONG
__cdecl
DbgPrint (
    _In_z_ _Printf_format_string_ PCSTR Format,
    ...
);

```

3. 传参数 (mov和lea关于指针的操作)

```

//注意细节
mov eax,[a] 或者mov eax,a
    eax = 0x12345678;
lea eax,[a]
    eax = 0x指针的地址;

```

4. 编写代码:

```

#include<stdio.h>
#include<windows.h>
typedef ULONG (__cdecl* PDbgPrint)(

```

```

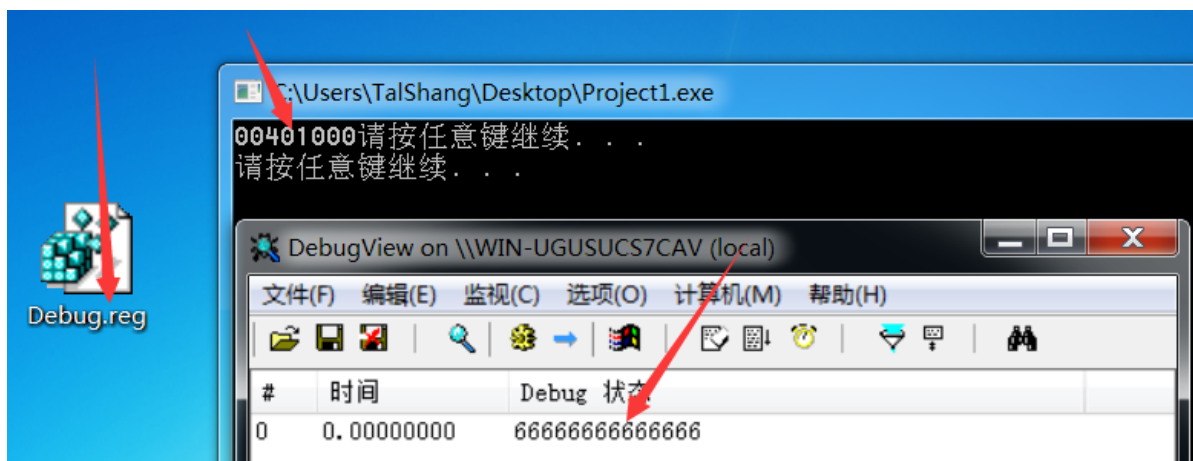
    _In_z_ _Printf_format_string_ PCSTR Format,
    ...);
PDbgPrint DbgPrint= (PDbgPrint)0x83e5041f;
char* str =(char*)"6666666";
__declspec(naked) void a()
{
    __asm
    {
        //int 3;
        pushfd; //一定要在 调用DbgPrint前 保存环境，重点是fs。不然会蓝屏
        pushad;
        push fs;

        mov ax, 0x30; //R0 下 FS 使用KPCR CPU状态控制块
        mov fs, ax;
        mov eax, [str];
        push eax;
        call DbgPrint;
        add esp, 4;

        pop fs;
        popad;
        popfd;
        retf;
    }
}
int main()
{
    char buf[6]={0,0,0,0,0x48,0};
    printf("%08x",a);
    system("pause");
    __asm
    {
        call fword ptr buf;
    }
    system("pause");
    return 0;
}

```

5.结果



6.蓝屏原因

Command

```
kd> uf nt!dbgprint
nt!DbgPrint:
83e5041f 8bff      mov     edi,edi
83e50421 55       push   ebp
83e50422 8bec     mov     ebp,esp
83e50424 51       push   ecx
83e50425 6a01     push   1
83e50427 8d450c   lea     eax,[ebp+0Ch]
83e5042a 50       push   eax
83e5042b ff7508   push   dword ptr [ebp+8]
83e5042e b93038e983 mov     ecx,offset nt! ?? ::FNODOBFM::`string' (83e93830)
83e50433 6a03     push   3
83e50435 6a65     push   65h
83e50437 e844590b00 call    nt!vDbgPrintExWithPrefixInternal (83f05d80)
83e5043c 59       pop     ecx
83e5043d 5d       pop     ebp
83e5043e c3       ret
kd> uf nt!vDbgPrintExWithPrefixInternal
nt!vDbgPrintExWithPrefixInternal:
83f05d80 6834020000 push   234h
83f05d85 68105ee983 push   offset nt! ?? ::FNODOBFM::`string'+0x1290 (83e95e10)
83f05d8a e8c1adfbff call    nt!_SEH_prolog4_GS (83ec0b50)
83f05d8f 8bf9     mov     edi,ecx
83f05d91 8b4510   mov     eax,dword ptr [ebp+10h]
83f05d94 8985dcfdffff mov     dword ptr [ebp-224h],eax
83f05d9a 8b4514   mov     eax,dword ptr [ebp+14h]
83f05d9d 8985d0fdffff mov     dword ptr [ebp-230h],eax
83f05da3 ff750c   push   dword ptr [ebp+0Ch]
83f05da6 ff7508   push   dword ptr [ebp+8]
83f05da9 e886ffff call    nt!NtQueryDebugFilterState (83f05d34)
83f05dae 85c0     test    eax,eax
83f05db0 7507     jne     nt!vDbgPrintExWithPrefixInternal+0x39 (83f05db9) Branch

nt!vDbgPrintExWithPrefixInternal+0x32:
83f05db2 33c0     xor     eax,eax
83f05db4 e9c1020000 jmp     nt!vDbgPrintExWithPrefixInternal+0x2fa (83f0607a) Branch

nt!vDbgPrintExWithPrefixInternal+0x30:
83f05e3e 8d8de0fdffff lea     ecx,[ebp-220h]
83f05e44 898dc8fdffff mov     dword ptr [ebp-238h],ecx
83f05e4a 668985c4fdffff mov     word ptr [ebp-23Ch],ax
83f05e51 f60578a1f78303 test    byte ptr [nt!KiBugCheckActive (83f05e51)],0
83f05e58 0f859ff010000 jne     nt!vDbgPrintExWithPrefixInternal+0x30

nt!vDbgPrintExWithPrefixInternal+0xde:
83f05e5e 803d5badf78301 cmp     byte ptr [nt!RtlpDebugPrintCallbackLock (83f7ad70)],0
83f05e65 0f8592010000 jne     nt!vDbgPrintExWithPrefixInternal+0x30

nt!vDbgPrintExWithPrefixInternal+0xeb:
83f05e6b 83a5dcfdffff00 and     dword ptr [ebp-224h],0
83f05e72 ff156821e483 call    dword ptr [nt!_imp__KeGetCurrentIrql (83e42168)]
83f05e78 8885dbfdffff mov     byte ptr [ebp-225h],al
83f05e7e b11b     mov     cl,1Bh
83f05e80 3ac1     cmp     al,cl
83f05e82 7306     jae     nt!vDbgPrintExWithPrefixInternal+0x10a (83f05e8a) Branch

nt!vDbgPrintExWithPrefixInternal+0x104:
83f05e84 ff155c21e483 call    dword ptr [nt!_imp__KfRaiseIrql (83e4215c)]

nt!vDbgPrintExWithPrefixInternal+0x10a:
83f05e8a 648b3d20000000 mov     edi,dword ptr fs:[20h]
83f05e91 ff8760350000 inc     dword ptr [edi+3560h]
83f05e97 8b0d70adf783 mov     ecx,dword ptr [nt!RtlpDebugPrintCallbackLock (83f7ad70)]
83f05e9d 81e1ffff7f and     ecx,7FFFFFFFh
83f05ea3 8d4101   lea     eax,[ecx+1]
83f05ea6 8bd0     mov     edx,eax
83f05ea8 be70adf783 mov     esi,offset nt!RtlpDebugPrintCallbackLock (83f7ad70)
83f05ead 8bde     mov     ebx,esi
83f05eaf 8bc1     mov     eax,ecx
83f05eb1 f00fb113 lock cmpxchg dword ptr [ebx],edx
83f05eb5 3bc1     cmp     eax,ecx
83f05eb7 7411     je      nt!vDbgPrintExWithPrefixInternal+0x14a (83f05eca) Branch

nt!vDbgPrintExWithPrefixInternal+0x139:
83f05eb9 e8e6d1f6ff call    nt!ExpWaitForSpinLockSharedAndAcquire (83e730a4)
83f05ebe ff8764350000 inc     dword ptr [edi+3564h]
```

☐ 全字匹配(W)

方向

☐ 区分大小写(C)

☐ 向

调用门的参数细节:

返回地址

CS

参数1 //参数是CPU通过调用门的ParamCount 从R3拷贝过来

参数2

参数3

esp

SS

```

__declspec(naked) void a()
{
    __asm
    {
        retf 0x4; //注意，这里的retf 0x4 不仅pop了R0堆栈的参数，也pop了R3堆栈的参数。
    }
}

//R3:
__asm
{
    push 0x12345678; //假设调用门的ParamCount的参数为1
    call fword ptr buf;
}

```

思考:

call 和 retf配套，retf会帮助平掉R3和R0的堆栈。

如何使用 iretd 来返回不报错？

```

#include<stdio.h>
#include<windows.h>
__declspec(naked) void a()
{
    __asm
    {
        iretd; //平掉R0的堆栈
    }
}

int main()
{
    char buf[6] = { 0,0,0,0,0x48,0 };
    printf("%08x", a);
    system("pause");
    __asm
    {
        pushfd;
        call fword ptr buf; //调用门参数赋值为 1
        add esp, 4; //平掉R3的堆栈。
    }
    system("pause");
    return 0;
}

```

R0的esp来自哪里？

4.中断门:

SXS.DLL

解决DbgView 狂刷 Windbg日志。

DbgView调试的输出都在 SysWow64文件夹下的SXS.DLL中。

IDA View-A

输入

十六进制视图-1

地址	序号	名称	库
6E56102C		vDbgPrintEx	ntdll
6E561064		DbgPrintEx	ntdll

```

.text:6E5B0E2D ; int __stdcall sub_6E5B0E2D(ULONG Level, PCCH Format, va_list ap)
.text:6E5B0E2D sub_6E5B0E2D proc near
.text:6E5B0E2D
.text:6E5B0E2D Level = dword ptr 8
.text:6E5B0E2D Format = dword ptr 0Ch
.text:6E5B0E2D ap = dword ptr 10h
.text:6E5B0E2D
.text:6E5B0E2D mov edi, edi
.text:6E5B0E2F push ebp
.text:6E5B0E30 mov ebp, esp
.text:6E5B0E32 push esi
.text:6E5B0E33 push edi
.text:6E5B0E34 mov eax, large fs:34h
.text:6E5B0E3A push [ebp+ap] ; ap
.text:6E5B0E3D mov edi, eax
.text:6E5B0E3F push [ebp+Format] ; Format
.text:6E5B0E42 push [ebp+Level] ; Level
.text:6E5B0E45 push 34h ; ComponentId
.text:6E5B0E47 call ds:vDbgPrintEx
.text:6E5B0E4D mov esi, eax
.text:6E5B0E4F test esi, esi
.text:6E5B0E51 jz short loc_6E5B0E6F
.text:6E5B0E53 call ds:IsDebuggerPresent
.text:6E5B0E59 test eax, eax
.text:6E5B0E5B jz short loc_6E5B0E6F
.text:6E5B0E5D push [ebp+ap]
.text:6E5B0E60 push [ebp+Format]

```

```

.text:6E5B0E45 6A 34 push 34h ; ComponentId
.text:6E5B0E47 FF 15 2C 10 56 6E call ds:vDbgPrintEx
.text:6E5B0E4D 8B F0 mov esi, eax
.text:6E5B0E4F 85 F6 test esi, esi
.text:6E5B0E51 74 1C jz short loc_6E5B0E6F
.text:6E5B0E53 FF 15 C0 10 56 6E call ds:IsDebuggerPresent
.text:6E5B0E59 85 C0 test eax, eax
.text:6E5B0E5B 74 12 jz short loc_6E5B0E6F
.text:6E5B0E5D FF 75 10 push [ebp+ap]
.text:6E5B0E60 FF 75 0C push [ebp+Format]
.text:6E5B0E63 FF 75 08 push [ebp+Level]
.text:6E5B0E66 6A 34 push 34h
.text:6E5B0E68 E8 2F FF FF FF call sub_6E5B0D9C
.text:6E5B0E6D 8B F0 mov esi, eax

```

```

.text:6E5B0E6F
.text:6E5B0E6F loc_6E5B0E6F:
.text:6E5B0E6F ; CODE XREF: sub_6E5B0E2D+24↑j
.text:6E5B0E6F ; sub_6E5B0E2D+2E↑j
.text:6E5B0E6F 64 A1 18 00 00 00 mov eax, large fs:18h
.text:6E5B0E75 89 78 34 mov [eax+34h], edi
.text:6E5B0E78 5F pop edi
.text:6E5B0E79 8B C6 mov eax, esi
.text:6E5B0E7B 5E pop esi
.text:6E5B0E7C 5D pop ebp
.text:6E5B0E7D C2 0C 00 retn 0Ch

```

```

.text:6E5B0E2D ; int __stdcall locret_6E5B0E2D(ULONG Level, PCCH Format, va_list ap)
.text:6E5B0E2D locret_6E5B0E2D:
.text:6E5B0E2D C2 0C 00 retn 0Ch

```

```

.text:6E5B0E30 ;
.text:6E5B0E30 8B EC mov ebp, esp
.text:6E5B0E32 56 push esi
.text:6E5B0E33 57 push edi
.text:6E5B0E34 64 A1 34 00 00 00 mov eax, large fs:34h
.text:6E5B0E3A FF 75 10 push dword ptr [ebp+10h] ; ap
.text:6E5B0E3D 8B F8 mov edi, eax
.text:6E5B0E3F FF 75 0C push dword ptr [ebp+0Ch] ; Format
.text:6E5B0E42 FF 75 08 push dword ptr [ebp+8] ; Level

```



中断:

硬件叫中断, 软件叫异常。

IDT 表: `kd> r idtr L20 (sidt lidt`

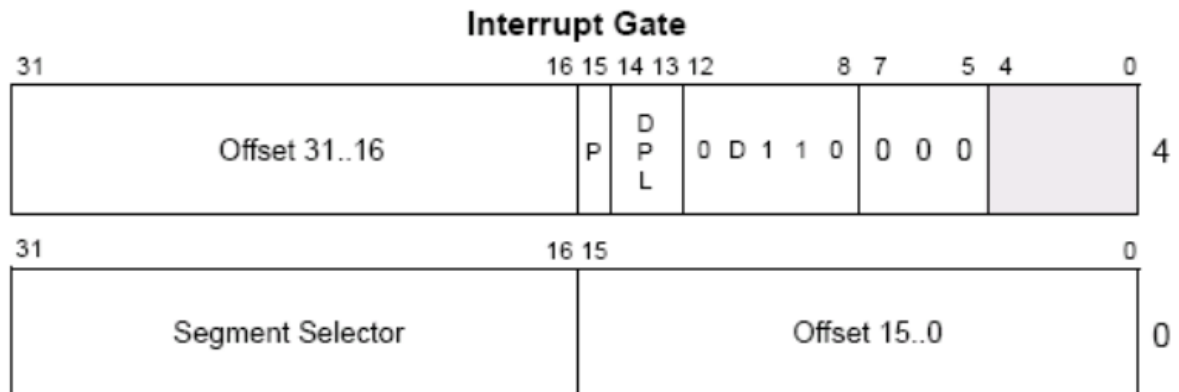


Table 3-2. System-Segment and Gate-Descriptor Types

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

int3与int 3

3 代表了 索引，类似于数组下标。这里的3代表第4项的意思。

775E4D30	CC	← int3	← int 0x3	← 英特尔 特别为int3指令 设计了0xcc 方便地址计算
775E4D31	CD 03	←	←	
775E4D33	8BEC	←	←	
775E4D35	83E4 F8	←	←	
775E4D38	83EC 64	←	←	
775E4D3B	A1 60336C77	←	←	

注意：只有 DPL=3的中断，才能在3环里 按F8跳过，否则会直接报错。（0xee

```
kd> dq idtr L20
80b99400 83e48e00`00081fc0 83e48e00`00082150
80b99410 00008500`00580000 83e4ee00`000825c0
80b99420 83e4ee00`00082748 83e48e00`000828a8

kd> !idt ← 查看IDT所有中断号的全部信息
Dumping IDT: 80b99400
e1687e6400000037: 8422f104 hal!HalInitializeProcessor+0xFC
e1687e6400000051: 865b7a58 pci+0x9254 (KINTERRUPT 865b7a00)
e1687e6400000052: 865c17d8 pci+0x9254 (KINTERRUPT 865c1780)
e1687e6400000053: 865cb558 pci+0x9254 (KINTERRUPT 865cb500)
e1687e6400000054: 865d52d8 pci+0x9254 (KINTERRUPT 865d5280)
e1687e6400000055: 865e8558 vmci!DllUnload+0x17e2 (KINTERRUPT 865e8500)
e1687e6400000060: 865b7cd8 pci+0x9254 (KINTERRUPT 865b7c80)
e1687e6400000061: 869777d8 i8042prt+0x2cf9 (KINTERRUPT 86977780)
e1687e6400000062: 865c1a58 pci+0x9254 (KINTERRUPT 865c1a00)
```

```
kd> ! idt 0
Dumping IDT: 80b99400
e1687e6400000000:      83e41fc0 nt!KiTrap00
```

索引

内核了int3 下断点

```
kd> !idt 3
Dumping IDT: 80b99400
e1687e6400000003:      83e425c0 nt!KiTrap03
kd> bp 83e425c0
kd> g
*** Fatal System Error: 0x0000007f
(0x00000008,0x801E4000,0x00000000,0x00000000)
```

循环触发中断，电脑死机

自建中断门：

```
kd> dq idtr L30
80b99400 83e48e00`00083fc0 83e48e00`00084150
80b99410 00008500`00580000 83e4ee00`000845c0
80b99420 83e4ee00`00084748 83e48e00`000848a8
80b99430 83e48e00`00084a1c 83e48e00`00085018
80b99440 00008500`00500000 83e48e00`00085478
80b99450 83e48e00`0008559c 83e48e00`000856dc
80b99460 83e48e00`0008593c 83e48e00`00085c2c
80b99470 83e48e00`000862fc 83e48e00`000866b0
80b99480 83e48e00`000867d4 83e48e00`00086914
80b99490 00008500`00a00000 83e48e00`00086a80
80b994a0 83e48e00`000866b0 83e48e00`000866b0
80b994b0 83e48e00`000866b0 83e48e00`000866b0
80b994c0 83e48e00`000866b0 83e48e00`000866b0
80b994d0 83e48e00`000866b0 83e48e00`000866b0
80b994e0 83e48e00`000866b0 83e48e00`000866b0
80b994f0 83e48e00`000866b0 84238e00`00081af8
80b99500 00000000`00080000 00000000`00080000
80b99510 00000000`00080000 00000000`00080000
80b99520 00000000`00080000 00000000`00080000
80b99530 00000000`00080000 00000000`00080000
80b99540 00000000`00080000 00000000`00080000
80b99550 83e4ee00`0008363a 83e4ee00`000837c0
80b99560 83e4ee00`000838fc 83e4ee00`00084498
80b99570 83e4ee00`00082fee 83e48e00`000866b0
```

500 - 400

=100

100 /8

=0x20

0x20

```
#include<stdio.h>
#include<windows.h>
__declspec(naked)void a()
{
    __asm
    {
        int 3;
        iretd;
    }
}
int main()
{
    printf("%08x", a);
    system("pause");
    __asm
    {
        int 0x20;
    }
}
```



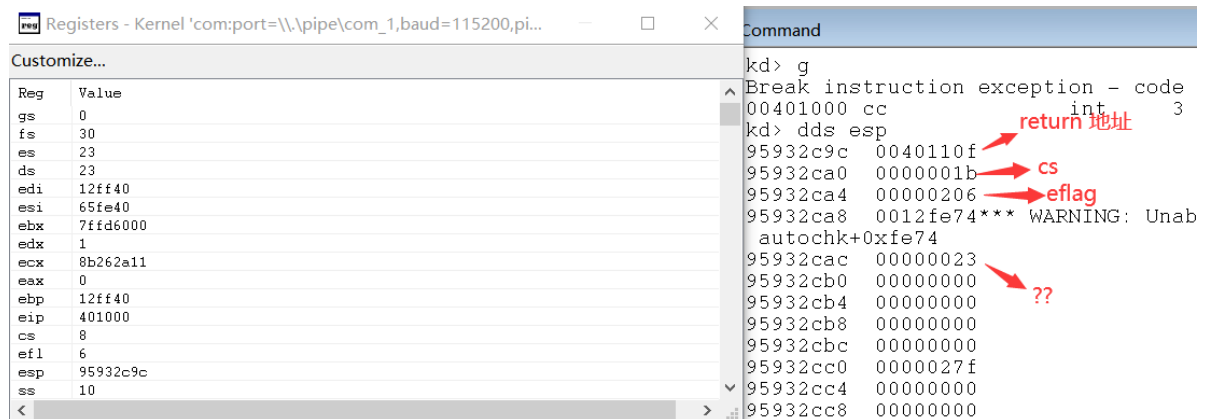
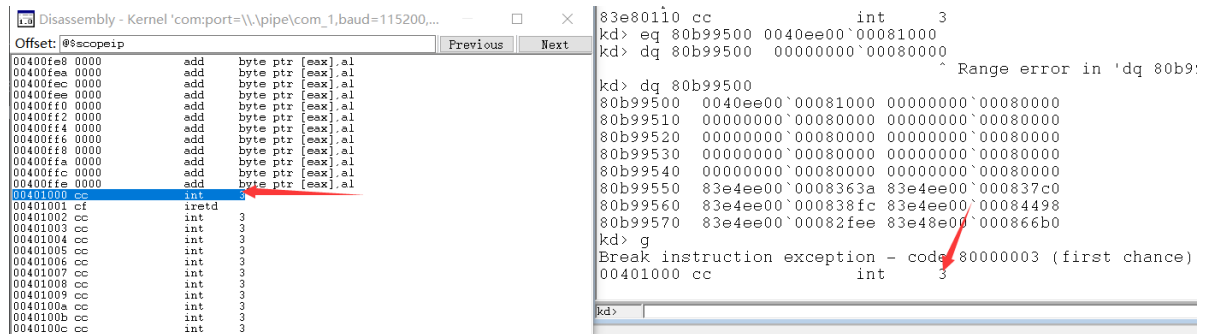
```
return 0;
}
```

kd > eq 80b99500 0040ee00`00081000



实验结果:

成功断下。



0x12fe74是esp, 0x23是SS。

fs被 (int3) 更改为0x30。(返回的时候没有改回去, 会崩溃程序)

```
kd> u 83e445c0 L20
nt!KiTrap03:
83e445c0 6a00          push     0
83e445c2 66c74424020000 mov     word ptr [esp+2],0
83e445c9 55            push     ebp
83e445ca 53            push     ebx
83e445cb 56            push     esi
83e445cc 57            push     edi
83e445cd 0fa0          push     fs
83e445cf bb30000000    mov     ebx,30h
83e445d4 668ee3        mov     fs,bx
83e445d7 648b1d00000000 mov     ebx,dword ptr fs:[0]
83e445de 53            push     ebx
83e445df 83ec04        sub     esp,4
83e445e2 50            push     eax
83e445e3 51            push     ecx
83e445e4 52            push     edx
83e445e5 1e            push     ds
83e445e6 5a            pop     ds
83e445e7 59            pop     ecx
83e445e8 58            pop     ebx
83e445e9 5b            pop     esi
83e445ea 5c            pop     edi
83e445eb 5d            pop     ebp
83e445ec 5e            pop     ebx
83e445ed 5f            pop     esi
83e445ee 60            pop     edi
83e445ef 61            pop     ebx
83e445f0 62            pop     esi
83e445f1 63            pop     edi
83e445f2 64            pop     ebx
83e445f3 65            pop     esi
83e445f4 66            pop     edi
83e445f5 67            pop     ebx
83e445f6 68            pop     esi
83e445f7 69            pop     edi
83e445f8 6a            pop     ebx
83e445f9 6b            pop     esi
83e445fa 6c            pop     edi
83e445fb 6d            pop     ebx
83e445fc 6e            pop     esi
83e445fd 6f            pop     edi
83e445fe 70            pop     ebx
83e445ff 71            pop     esi
83e44600 72            pop     edi
83e44601 73            pop     ebx
83e44602 74            pop     esi
83e44603 75            pop     edi
83e44604 76            pop     ebx
83e44605 77            pop     esi
83e44606 78            pop     edi
83e44607 79            pop     ebx
83e44608 7a            pop     esi
83e44609 7b            pop     edi
83e4460a 7c            pop     ebx
83e4460b 7d            pop     esi
83e4460c 7e            pop     edi
83e4460d 7f            pop     ebx
83e4460e 80            pop     esi
83e4460f 81            pop     edi
83e44610 82            pop     ebx
83e44611 83            pop     esi
83e44612 84            pop     edi
83e44613 85            pop     ebx
83e44614 86            pop     esi
83e44615 87            pop     edi
83e44616 88            pop     ebx
83e44617 89            pop     esi
83e44618 8a            pop     edi
83e44619 8b            pop     ebx
83e4461a 8c            pop     esi
83e4461b 8d            pop     edi
83e4461c 8e            pop     ebx
83e4461d 8f            pop     esi
83e4461e 90            pop     edi
83e4461f 91            pop     ebx
83e44620 92            pop     esi
83e44621 93            pop     edi
83e44622 94            pop     ebx
83e44623 95            pop     esi
83e44624 96            pop     edi
83e44625 97            pop     ebx
83e44626 98            pop     esi
83e44627 99            pop     edi
83e44628 9a            pop     ebx
83e44629 9b            pop     esi
83e4462a 9c            pop     edi
83e4462b 9d            pop     ebx
83e4462c 9e            pop     esi
83e4462d 9f            pop     edi
83e4462e a0            pop     ebx
83e4462f a1            pop     esi
83e44630 a2            pop     edi
83e44631 a3            pop     ebx
83e44632 a4            pop     esi
83e44633 a5            pop     edi
83e44634 a6            pop     ebx
83e44635 a7            pop     esi
83e44636 a8            pop     edi
83e44637 a9            pop     ebx
83e44638 aa            pop     esi
83e44639 ab            pop     edi
83e4463a ac            pop     ebx
83e4463b ad            pop     esi
83e4463c ae            pop     edi
83e4463d af            pop     ebx
83e4463e b0            pop     esi
83e4463f b1            pop     edi
83e44640 b2            pop     ebx
83e44641 b3            pop     esi
83e44642 b4            pop     edi
83e44643 b5            pop     ebx
83e44644 b6            pop     esi
83e44645 b7            pop     edi
83e44646 b8            pop     ebx
83e44647 b9            pop     esi
83e44648 ba            pop     edi
83e44649 bb            pop     ebx
83e4464a bc            pop     esi
83e4464b bd            pop     edi
83e4464c be            pop     ebx
83e4464d bf            pop     esi
83e4464e c0            pop     edi
83e4464f c1            pop     ebx
83e44650 c2            pop     esi
83e44651 c3            pop     edi
83e44652 c4            pop     ebx
83e44653 c5            pop     esi
83e44654 c6            pop     edi
83e44655 c7            pop     ebx
83e44656 c8            pop     esi
83e44657 c9            pop     edi
83e44658 ca            pop     ebx
83e44659 cb            pop     esi
83e4465a cc            pop     edi
83e4465b cd            pop     ebx
83e4465c ce            pop     esi
83e4465d cf            pop     edi
83e4465e d0            pop     ebx
83e4465f d1            pop     esi
83e44660 d2            pop     edi
83e44661 d3            pop     ebx
83e44662 d4            pop     esi
83e44663 d5            pop     edi
83e44664 d6            pop     ebx
83e44665 d7            pop     esi
83e44666 d8            pop     edi
83e44667 d9            pop     ebx
83e44668 da            pop     esi
83e44669 db            pop     edi
83e4466a dc            pop     ebx
83e4466b dd            pop     esi
83e4466c de            pop     edi
83e4466d df            pop     ebx
83e4466e e0            pop     esi
83e4466f e1            pop     edi
83e44670 e2            pop     ebx
83e44671 e3            pop     esi
83e44672 e4            pop     edi
83e44673 e5            pop     ebx
83e44674 e6            pop     esi
83e44675 e7            pop     edi
83e44676 e8            pop     ebx
83e44677 e9            pop     esi
83e44678 ea            pop     edi
83e44679 eb            pop     ebx
83e4467a ec            pop     esi
83e4467b ed            pop     edi
83e4467c ee            pop     ebx
83e4467d ef            pop     esi
83e4467e f0            pop     edi
83e4467f f1            pop     ebx
83e44680 f2            pop     esi
83e44681 f3            pop     edi
83e44682 f4            pop     ebx
83e44683 f5            pop     esi
83e44684 f6            pop     edi
83e44685 f7            pop     ebx
83e44686 f8            pop     esi
83e44687 f9            pop     edi
83e44688 fa            pop     ebx
83e44689 fb            pop     esi
83e4468a fc            pop     edi
83e4468b fd            pop     ebx
83e4468c fe            pop     esi
83e4468d ff            pop     edi
83e4468e 00            pop     ebx
83e4468f 01            pop     esi
83e44690 02            pop     edi
83e44691 03            pop     ebx
83e44692 04            pop     esi
83e44693 05            pop     edi
83e44694 06            pop     ebx
83e44695 07            pop     esi
83e44696 08            pop     edi
83e44697 09            pop     ebx
83e44698 0a            pop     esi
83e44699 0b            pop     edi
83e4469a 0c            pop     ebx
83e4469b 0d            pop     esi
83e4469c 0e            pop     edi
83e4469d 0f            pop     ebx
83e4469e 10            pop     esi
83e4469f 11            pop     edi
83e446a0 12            pop     ebx
83e446a1 13            pop     esi
83e446a2 14            pop     edi
83e446a3 15            pop     ebx
83e446a4 16            pop     esi
83e446a5 17            pop     edi
83e446a6 18            pop     ebx
83e446a7 19            pop     esi
83e446a8 1a            pop     edi
83e446a9 1b            pop     ebx
83e446aa 1c            pop     esi
83e446ab 1d            pop     edi
83e446ac 1e            pop     ebx
83e446ad 1f            pop     esi
83e446ae 20            pop     edi
83e446af 21            pop     ebx
83e446b0 22            pop     esi
83e446b1 23            pop     edi
83e446b2 24            pop     ebx
83e446b3 25            pop     esi
83e446b4 26            pop     edi
83e446b5 27            pop     ebx
83e446b6 28            pop     esi
83e446b7 29            pop     edi
83e446b8 2a            pop     ebx
83e446b9 2b            pop     esi
83e446ba 2c            pop     edi
83e446bb 2d            pop     ebx
83e446bc 2e            pop     esi
83e446bd 2f            pop     edi
83e446be 30            pop     ebx
83e446bf 31            pop     esi
83e446c0 32            pop     edi
83e446c1 33            pop     ebx
83e446c2 34            pop     esi
83e446c3 35            pop     edi
83e446c4 36            pop     ebx
83e446c5 37            pop     esi
83e446c6 38            pop     edi
83e446c7 39            pop     ebx
83e446c8 3a            pop     esi
83e446c9 3b            pop     edi
83e446ca 3c            pop     ebx
83e446cb 3d            pop     esi
83e446cc 3e            pop     edi
83e446cd 3f            pop     ebx
83e446ce 40            pop     esi
83e446cf 41            pop     edi
83e446d0 42            pop     ebx
83e446d1 43            pop     esi
83e446d2 44            pop     edi
83e446d3 45            pop     ebx
83e446d4 46            pop     esi
83e446d5 47            pop     edi
83e446d6 48            pop     ebx
83e446d7 49            pop     esi
83e446d8 4a            pop     edi
83e446d9 4b            pop     ebx
83e446da 4c            pop     esi
83e446db 4d            pop     edi
83e446dc 4e            pop     ebx
83e446dd 4f            pop     esi
83e446de 50            pop     edi
83e446df 51            pop     ebx
83e446e0 52            pop     esi
83e446e1 53            pop     edi
83e446e2 54            pop     ebx
83e446e3 55            pop     esi
83e446e4 56            pop     edi
83e446e5 57            pop     ebx
83e446e6 58            pop     esi
83e446e7 59            pop     edi
83e446e8 5a            pop     ebx
83e446e9 5b            pop     esi
83e446ea 5c            pop     edi
83e446eb 5d            pop     ebx
83e446ec 5e            pop     esi
83e446ed 5f            pop     edi
83e446ee 60            pop     ebx
83e446ef 61            pop     esi
83e446f0 62            pop     edi
83e446f1 63            pop     ebx
83e446f2 64            pop     esi
83e446f3 65            pop     edi
83e446f4 66            pop     ebx
83e446f5 67            pop     esi
83e446f6 68            pop     edi
83e446f7 69            pop     ebx
83e446f8 6a            pop     esi
83e446f9 6b            pop     edi
83e446fa 6c            pop     ebx
83e446fb 6d            pop     esi
83e446fc 6e            pop     edi
83e446fd 6f            pop     ebx
83e446fe 70            pop     esi
83e446ff 71            pop     edi
83e44700 72            pop     ebx
83e44701 73            pop     esi
83e44702 74            pop     edi
83e44703 75            pop     ebx
83e44704 76            pop     esi
83e44705 77            pop     edi
83e44706 78            pop     ebx
83e44707 79            pop     esi
83e44708 7a            pop     edi
83e44709 7b            pop     ebx
83e4470a 7c            pop     esi
83e4470b 7d            pop     edi
83e4470c 7e            pop     ebx
83e4470d 7f            pop     esi
83e4470e 80            pop     edi
83e4470f 81            pop     ebx
83e44710 82            pop     esi
83e44711 83            pop     edi
83e44712 84            pop     ebx
83e44713 85            pop     esi
83e44714 86            pop     edi
83e44715 87            pop     ebx
83e44716 88            pop     esi
83e44717 89            pop     edi
83e44718 8a            pop     ebx
83e44719 8b            pop     esi
83e4471a 8c            pop     edi
83e4471b 8d            pop     ebx
83e4471c 8e            pop     esi
83e4471d 8f            pop     edi
83e4471e 90            pop     ebx
83e4471f 91            pop     esi
83e44720 92            pop     edi
83e44721 93            pop     ebx
83e44722 94            pop     esi
83e44723 95            pop     edi
83e44724 96            pop     ebx
83e44725 97            pop     esi
83e44726 98            pop     edi
83e44727 99            pop     ebx
83e44728 9a            pop     esi
83e44729 9b            pop     edi
83e4472a 9c            pop     ebx
83e4472b 9d            pop     esi
83e4472c 9e            pop     edi
83e4472d 9f            pop     ebx
83e4472e a0            pop     esi
83e4472f a1            pop     edi
83e44730 a2            pop     ebx
83e44731 a3            pop     esi
83e44732 a4            pop     edi
83e44733 a5            pop     ebx
83e44734 a6            pop     esi
83e44735 a7            pop     edi
83e44736 a8            pop     ebx
83e44737 a9            pop     esi
83e44738 aa            pop     edi
83e44739 ab            pop     ebx
83e4473a ac            pop     esi
83e4473b ad            pop     edi
83e4473c ae            pop     ebx
83e4473d af            pop     esi
83e4473e b0            pop     edi
83e4473f b1            pop     ebx
83e44740 b2            pop     esi
83e44741 b3            pop     edi
83e44742 b4            pop     ebx
83e44743 b5            pop     esi
83e44744 b6            pop     edi
83e44745 b7            pop     ebx
83e44746 b8            pop     esi
83e44747 b9            pop     edi
83e44748 ba            pop     ebx
83e44749 bb            pop     esi
83e4474a bc            pop     edi
83e4474b bd            pop     ebx
83e4474c be            pop     esi
83e4474d bf            pop     edi
83e4474e c0            pop     ebx
83e4474f c1            pop     esi
83e44750 c2            pop     edi
83e44751 c3            pop     ebx
83e44752 c4            pop     esi
83e44753 c5            pop     edi
83e44754 c6            pop     ebx
83e44755 c7            pop     esi
83e44756 c8            pop     edi
83e44757 c9            pop     ebx
83e44758 ca            pop     esi
83e44759 cb            pop     edi
83e4475a cc            pop     ebx
83e4475b cd            pop     esi
83e4475c ce            pop     edi
83e4475d cf            pop     ebx
83e4475e d0            pop     esi
83e4475f d1            pop     edi
83e44760 d2            pop     ebx
83e44761 d3            pop     esi
83e44762 d4            pop     edi
83e44763 d5            pop     ebx
83e44764 d6            pop     esi
83e44765 d7            pop     edi
83e44766 d8            pop     ebx
83e44767 d9            pop     esi
83e44768 da            pop     edi
83e44769 db            pop     ebx
83e4476a dc            pop     esi
83e4476b dd            pop     edi
83e4476c de            pop     ebx
83e4476d df            pop     esi
83e4476e e0            pop     edi
83e4476f e1            pop     ebx
83e44770 e2            pop     esi
83e44771 e3            pop     edi
83e44772 e4            pop     ebx
83e44773 e5            pop     esi
83e44774 e6            pop     edi
83e44775 e7            pop     ebx
83e44776 e8            pop     esi
83e44777 e9            pop     edi
83e44778 ea            pop     ebx
83e44779 eb            pop     esi
83e4477a ec            pop     edi
83e4477b ed            pop     ebx
83e4477c ee            pop     esi
83e4477d ef            pop     edi
83e4477e f0            pop     ebx
83e4477f f1            pop     esi
83e44780 f2            pop     edi
83e44781 f3            pop     ebx
83e44782 f4            pop     esi
83e44783 f5            pop     edi
83e44784 f6            pop     ebx
83e44785 f7            pop     esi
83e44786 f8            pop     edi
83e44787 f9            pop     ebx
83e44788 fa            pop     esi
83e44789 fb            pop     edi
83e4478a fc            pop     ebx
83e4478b fd            pop     esi
83e4478c fe            pop     edi
83e4478d ff            pop     ebx
83e4478e 00            pop     esi
83e4478f 01            pop     edi
83e44790 02            pop     ebx
83e44791 03            pop     esi
83e44792 04            pop     edi
83e44793 05            pop     ebx
83e44794 06            pop     esi
83e44795 07            pop     edi
83e44796 08            pop     ebx
83e44797 09            pop     esi
83e44798 0a            pop     edi
83e44799 0b            pop     ebx
83e4479a 0c            pop     esi
83e4479b 0d            pop     edi
83e4479c 0e            pop     ebx
83e4479d 0f            pop     esi
83e4479e 10            pop     edi
83e4479f 11            pop     ebx
83e447a0 12            pop     esi
83e447a1 13            pop     edi
83e447a2 14            pop     ebx
83e447a3 15            pop     esi
83e447a4 16            pop     edi
83e447a5 17            pop     ebx
83e447a6 18            pop     esi
83e447a7 19            pop     edi
83e447a8 1a            pop     ebx
83e447a9 1b            pop     esi
83e447aa 1c            pop     edi
83e447ab 1d            pop     ebx
83e447ac 1e            pop     esi
83e447ad 1f            pop     edi
83e447ae 20            pop     ebx
83e447af 21            pop     esi
83e447b0 22            pop     edi
83e447b1 23            pop     ebx
83e447b2 24            pop     esi
83e447b3 25            pop     edi
83e447b4 26            pop     ebx
83e447b5 27            pop     esi
83e447b6 28            pop     edi
83e447b7 29            pop     ebx
83e447b8 2a            pop     esi
83e447b9 2b            pop     edi
83e447ba 2c            pop     ebx
83e447bb 2d            pop     esi
83e447bc 2e            pop     edi
83e447bd 2f            pop     ebx
83e447be 30            pop     esi
83e447bf 31            pop     edi
83e447c0 32            pop     ebx
83e447c1 33            pop     esi
83e447c2 34            pop     edi
83e447c3 35            pop     ebx
83e447c4 36            pop     esi
83e447c5 37            pop     edi
83e447c6 38            pop     ebx
83e447c7 39            pop     esi
83e447c8 3a            pop     edi
83e447c9 3b            pop     ebx
83e447ca 3c            pop     esi
83e447cb 3d            pop     edi
83e447cc 3e            pop     ebx
83e447cd 3f            pop     esi
83e447ce 40            pop     edi
83e447cf 41            pop     ebx
83e447d0 42            pop     esi
83e447d1 43            pop     edi
83e447d2 44            pop     ebx
83e447d3 45            pop     esi
83e447d4 46            pop     edi
83e447d5 47            pop     ebx
83e447d6 48            pop     esi
83e447d7 49            pop     edi
83e447d8 4a            pop     ebx
83e447d9 4b            pop     esi
83e447da 4c            pop     edi
83e447db 4d            pop     ebx
83e447dc 4e            pop     esi
83e447dd 4f            pop     edi
83e447de 50            pop     ebx
83e447df 51            pop     esi
83e447e0 52            pop     edi
83e447e1 53            pop     ebx
83e447e2 54            pop     esi
83e447e3 55            pop     edi
83e447e4 56            pop     ebx
83e447e5 57            pop     esi
83e447e6 58            pop     edi
83e447e7 59            pop     ebx
83e447e8 5a            pop     esi
83e447e9 5b            pop     edi
83e447ea 5c            pop     ebx
83e447eb 5d            pop     esi
83e447ec 5e            pop     edi
83e447ed 5f            pop     ebx
83e447ee 60            pop     esi
83e447ef 61            pop     edi
83e447f0 62            pop     ebx
83e447f1 63            pop     esi
83e447f2 64            pop     edi
83e447f3 65            pop     ebx
83e447f4 66            pop     esi
83e447f5 67            pop     edi
83e447f6 68            pop     ebx
83e447f7 69            pop     esi
83e447f8 6a            pop     edi
83e447f9 6b            pop     ebx
83e447fa 6c            pop     esi
83e447fb 6d            pop     edi
83e447fc 6e            pop     ebx
83e447fd 6f            pop     esi
83e447fe 70            pop     edi
83e447ff 71            pop     ebx
83e44800 72            pop     esi
83e44801 73            pop     edi
83e44802 74            pop     ebx
83e44803 75            pop     esi
83e44804 76            pop     edi
83e44805 77            pop     ebx
83e44806 78            pop     esi
83e44807 79            pop     edi
83e44808 7a            pop     ebx
83e44809 7b            pop     esi
83e4480a 7c            pop     edi
83e4480b 7d            pop     ebx
83e4480c 7e            pop     esi
83e4480d 7f            pop     edi
83e4480e 80            pop     ebx
83e4480f 81            pop     esi
83e44810 82            pop     edi
83e44811 83            pop     ebx
83e44812 84            pop     esi
83e44813 85            pop     edi
83e44814 86            pop     ebx
83e44815 87            pop     esi
83e44816 88            pop     edi
83e44817 89            pop     ebx
83e44818 8a            pop     esi
83e44819 8b            pop     edi
83e4481a 8c            pop     ebx
83e4481b 8d            pop     esi
83e4481c 8e            pop     edi
83e4481d 8f            pop     ebx
83e4481e 90            pop     esi
83e4481f 91            pop     edi
83e44820 92            pop     ebx
83e44821 93            pop     esi
83e44822 94            pop     edi
83e44823 95            pop     ebx
83e44824 96            pop     esi
83e44825 97            pop     edi
83e44826 98            pop     ebx
83e44827 99            pop     esi
83e44828 9a            pop     edi
83e44829 9b            pop     ebx
83e4482a 9c            pop     esi
83e4482b 9d            pop     edi
83e4482c 9e            pop     ebx
83e4482d 9f            pop     esi
83e4482e a0            pop     edi
83e4482f a1            pop     ebx
83e44830 a2            pop     esi
83e44831 a3            pop     edi
83e44832 a4            pop     ebx
83e44833 a5            pop     esi
83e44834 a6            pop     edi
83e44835 a7            pop     ebx
83e44836 a8            pop     esi
83e44837 a9            pop     edi
83e44838 aa            pop     ebx
83e44839 ab            pop     esi
83e4483a ac            pop     edi
83e4483b ad            pop     ebx
83e4483c ae            pop     esi
83e4483d b0            pop     edi
83e4483e b1            pop     ebx
83e4483f b2            pop     esi
83e44840 b3            pop     edi
83e44841 b4            pop     ebx
83e44842 b5            pop     esi
83e44843 b6            pop     edi
83e44844 b7            pop     ebx
83e44845 b8            pop     esi
83e44846 b9            pop     edi
83e44847 ba            pop     ebx
83e44848 bb            pop     esi
83e44849 bc            pop     edi
83e4484a bd            pop     ebx
83e4484b be            pop     esi
83e4484c bf            pop     edi
83e4484d c0            pop     ebx
83e4484e c1            pop     esi
83e4484f c2            pop     edi
83e44850 c3            pop     ebx
83e44851 c4            pop     esi
83e44852 c5            pop     edi
83e44853 c6            pop     ebx
83e44854 c7            pop     esi
83e44855 c8            pop     edi
83e44856 c9            pop     ebx
83e44857 ca            pop     esi
83e44858 cb            pop     edi
83e44859 cc            pop     ebx
83e4485a cd            pop     esi
83e4485b ce            pop     edi
83e4485c cf            pop     ebx
83e4485d d0            pop     esi
83e4485e d1            pop     edi
83e4485f d2            pop     ebx
83e44860 d3            pop     esi
83e44861 d4            pop     edi
83e44862 d5            pop     ebx
83e44863 d6            pop     esi
83e44864 d7            pop     edi
83e44865 d8            pop     ebx
83e44866 d9            pop     esi
83e44867 da            pop     edi
83e44868 db            pop     ebx
83e44869 dc            pop     esi
83e4486a dd            pop     edi
83e4486b de            pop     ebx
83e4486c df            pop     esi
83e4486d e0            pop     edi
83e4486e e1            pop     ebx
83e4486f e2            pop     esi
83e44870 e3            pop     edi
83e44871 e4            pop     ebx
83e44872 e5            pop     esi
83e44873 e6            pop     edi
83e44874 e7            pop     ebx
83e44875 e8            pop     esi
83e44876 e9            pop     edi
83e44877 ea            pop     ebx
83e44878 eb            pop     esi
83e44879 ec            pop     edi
83e4487a ed            pop     ebx
83e4487b ee            pop     esi
83e4487c ef            pop     edi
83e4487d f0            pop     ebx
83e4487e f1            pop     esi
83e4487f f2            pop     edi
83e44880 f3            pop     ebx
83e44881 f4            pop     esi
83e44882 f5            pop     edi
83e44883 f6            pop     ebx
83e44884 f7            pop     esi
83e44885 f8            pop     edi
83e44886 f9            pop     ebx
83e44887 fa            pop     esi
83e44888 fb            pop     edi
83e44889 fc            pop     ebx
83e4488a fd            pop     esi
83e4488b fe            pop     edi
83e4488c ff            pop     ebx
83e4488d 00            pop     esi
83e4488e 01            pop     edi
83e4488f 02            pop     ebx
83e44890 03            pop     esi
83e44891 04            pop     edi
83e44892 05            pop     ebx
83e44893 06            pop     esi
83e44894 07            pop     edi
83e44895 08            pop     ebx
83e44896 09            pop     esi
83e44897 0a            pop     edi
83e44898 0b            pop     ebx
83e44899 0c            pop     esi
83e4489a 0d            pop     edi
83e4489b 0e            pop     ebx
83e4489c 0f            pop     esi
83e4489d 10            pop     edi
83e4489e 11            pop     ebx
83e4489f 12            pop     esi
83e448a0 13            pop     edi
83e448a1 14            pop     ebx
83e448a2 15            pop     esi
83e448a3 16            pop     edi
83e448a4 17            pop     ebx
83e448a5 18            pop     esi
83e448a6 19            pop     edi
83e448a7 1a            pop     ebx
83e448a8 1b            pop     esi
83e448a9 1c            pop     edi
83e448aa 1d            pop     ebx
83e448ab 1e            pop     esi
83e448ac 1f            pop     edi
83e448ad 20            pop     ebx
83e448ae 21            pop     esi
83e448af 22            pop     edi
83e448b0 23            pop     ebx
83e448b1 24            pop     esi
83e448b2 25            pop     edi
83e448b3 26            pop     ebx
83e448b4 27            pop     esi
83e448b
```

```

__asm
{
    push fs;
    int 0x20
    pop fs;
}
//修改后，程序正常运行。

```

注意：调用门与中断门的区别是，调用门能被可屏蔽中断打断。

中断影响的EFL

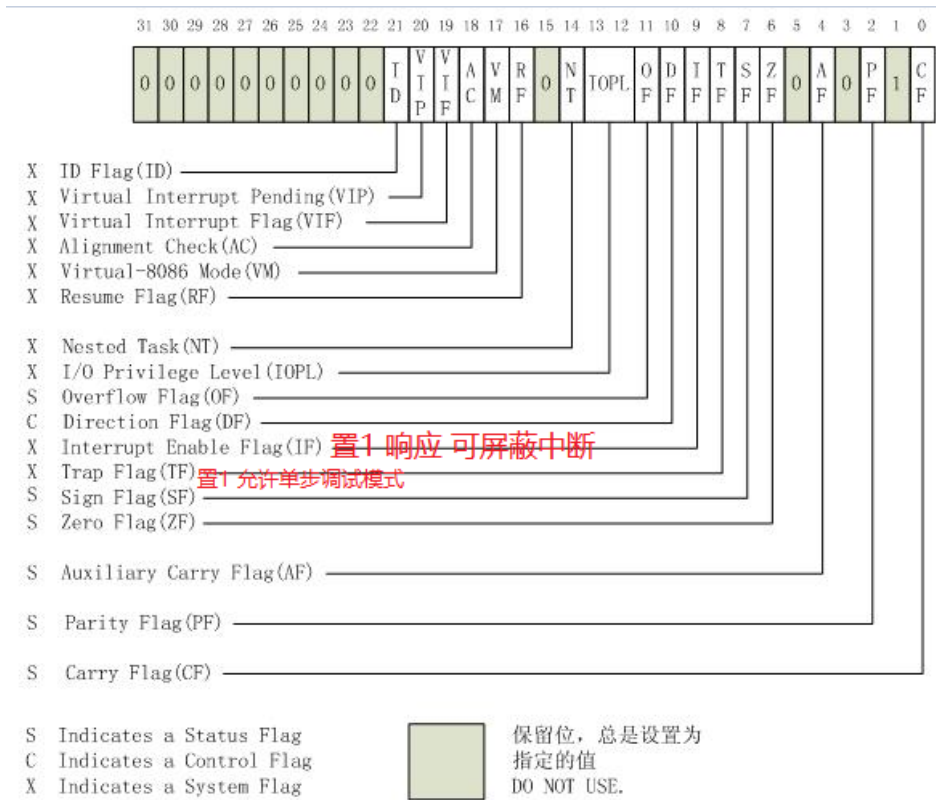
中断 置IF=0，屏蔽掉了可屏蔽中断。

注意：中断门会清空IF TF VM NT位

```

cli; //清除Efl的IF位
sti; //设置Efl的IF位

```



思考：

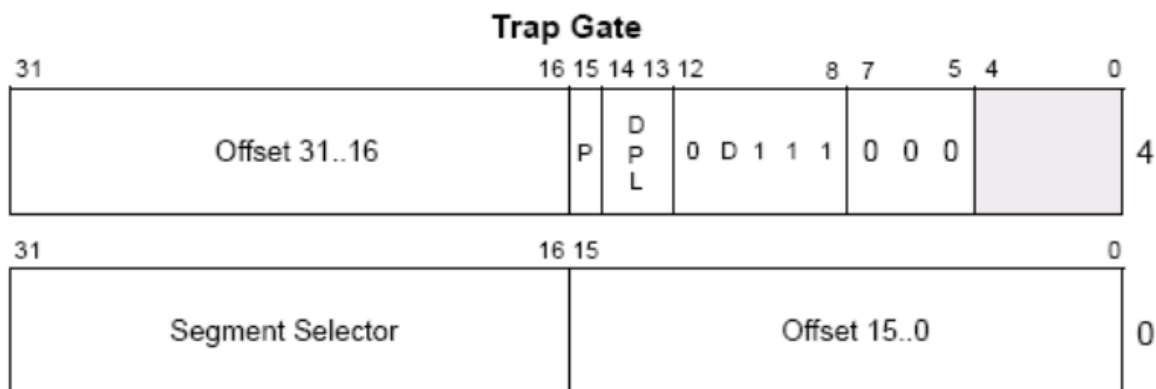
有了中断门为啥需要陷阱门？

陷阱门和中断门的区别？

5.陷阱门

陷阱门清空VM NT TF位，不清空IF位。

注意：IDT表里没有用到陷阱门。



DPL Descriptor Privilege Level
 Offset Offset to procedure entry point
 P Segment Present flag
 Selector Segment Selector for destination code segment
 D Size of gate: 1 = 32 bits; 0 = 16 bits

Reserved

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

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

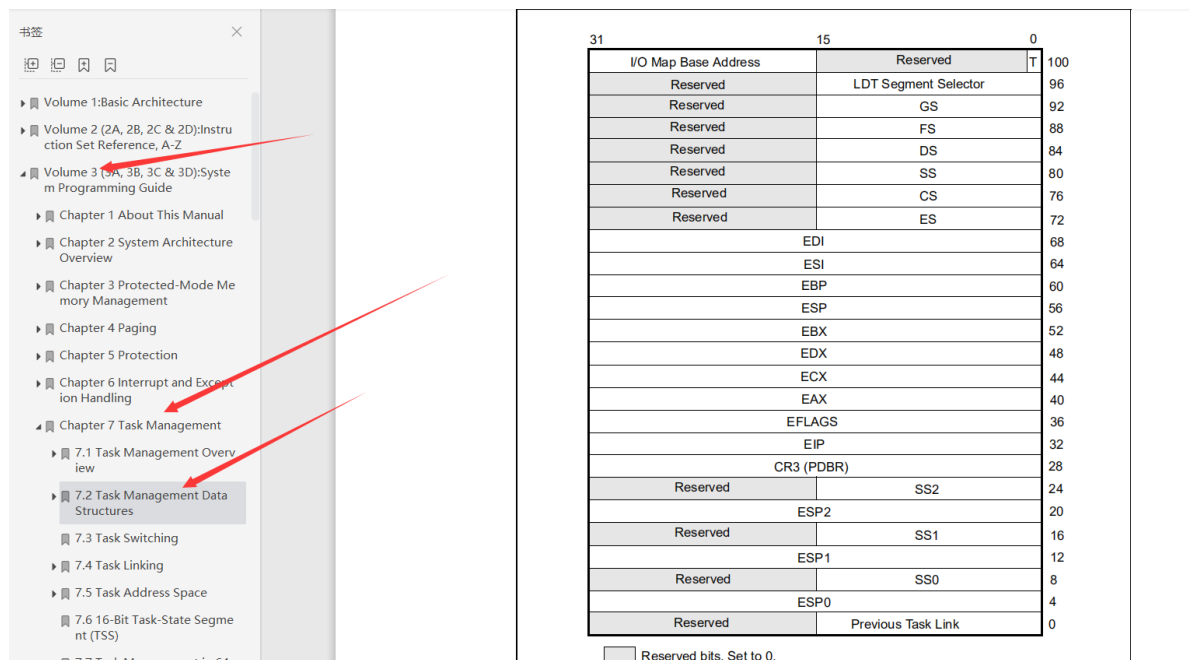
6.任务段

任务是提供一块内存用于 进程和线程的环境的切换。

任务太过依赖于 GDT表中的任务段描述 和 内存块。

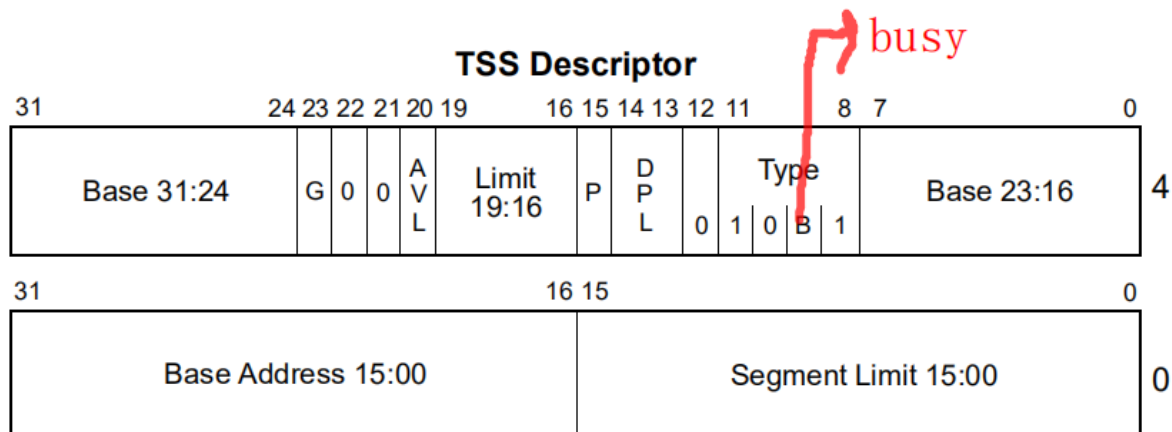
Windows 和Linux 都没有采用任务段。

TSS的内存结构 (Task-State-Segment)



TSS段描述符

0000e90f`71400068 (不忙碌状态)



AVL Available for use by system software
 B Busy flag
 BASE Segment Base Address
 DPL Descriptor Privilege Level
 G Granularity
 LIMIT Segment Limit
 P Segment Present
 TYPE Segment Type

tr寄存器

kd > str && ltr

kd> r tr
 tr=00000028 任务段寄存器

```
kd> dq gdttr L20
80b99000  00000000\00000000  00cf9b00\0000ffff
80b99010  00cf9300\0000ffff  00cffb00\0000ffff
80b99020  00cff300\0000ffff  80008b1e\400020ab
80b99030  834093f3\ec003748  0040f300\0000ffff
80b99040  0000f200\0400ffff  00000000\00000000
80b99050  830089f3\c0000068  830089f3\c0680068
80b99060  00000000\00000000  00000000\00000000
.....
```

任务段描述
tr = 28

```
kd> r cs
cs=00000008
kd> dg cs
```

Sel	Base	Limit	Type	P	Si	Gr	Pr	Lo	Flags
0008	00000000	ffffffff	Code RE Ac	0	Bg	Pg	P	Nl	00000c9b

```
kd> dg tr
```

Sel	Base	Limit	Type	P	Si	Gr	Pr	Lo	Flags
0028	801e4000	000020ab	TSS32 Busy	0	Nb	By	P	Nl	0000008b

查看段寄存器对应的段描述符

dt_KTSS

```
kd> dt _ktss 801e4000
ntdll!_KTSS
+0x000 Backlink          : 0
+0x002 Reserved0         : 0
+0x004 Esp0              : 0x83f67cb0
+0x008 Ss0               : 0x10
+0x00a Reserved1         : 0
+0x00c NotUsed1          : [4] 0
+0x01c CR3               : 0
+0x020 Eip               : 0
+0x024 EFlags            : 0
+0x028 Eax               : 0
+0x02c Ecx               : 0
+0x030 Edx               : 0
+0x034 Ebx               : 0
+0x038 Esp               : 0
+0x03c Ebp               : 0
+0x040 Esi               : 0
+0x044 Edi               : 0
+0x048 Es                : 0
+0x04a Reserved2         : 0
+0x04c Cs                : 0
+0x04e Reserved3         : 0
+0x050 Ss                : 0
+0x052 Reserved4         : 0
+0x054 Ds                : 0
+0x056 Reserved5         : 0
+0x058 Fs                : 0
+0x05a Reserved6         : 0
+0x05c Gs                : 0
+0x05e Reserved7         : 0
+0x060 LDT               : 0
+0x062 Reserved8         : 0
+0x064 Flags             : 0
+0x066 IoMapBase         : 0x20ac
+0x068 IoMaps            : [1] _KiIoAccessMap
+0x208c IntDirectionMap  : [32] "???"
```

切换任务段的指令

通过GDT表(call指令), 通过 iretd返回。 (GDT表保存的是TSS段描述符)

1.windbg修改GDT表, 设置TSS段。

```
kd> eq 80b99048 0000e945`d9f00068
kd> dq gdttr L20
80b99000 00000000`00000000 00cf9b00`0000ffff
80b99010 00cf9300`0000ffff 00cffb00`0000ffff
80b99020 00cff300`0000ffff 80008b1e`400020ab
80b99030 834093f6`ac003748 0040f300`00000fff
80b99040 0000f200`0400ffff 0000e945`d9f00068
```

2.查看程序CR3

```
PROCESS 888e9538 SessionId: 1 Cid: Odd8 Peb: 7ffdb000 ParentCid: 0954
DirBase: 3f32b560 ObjectTable: 909ed7d8 HandleCount: 7.
Image: Project1.exe
```

!process 0 0



3.效果

```
PROCESS 88103708 SessionId: 1 Cid: 06c0 Peb: 7ffde000 ParentCid: 0a18
DirBase: 3f31f5a0 ObjectTable: a0a4d168 HandleCount: 7.
Image: Project1.exe
```

```
PROCESS 881708d0 SessionId: 1 Cid: 0c0c Peb: 7ffdf000 ParentCid: 0188
DirBase: 3f31f3c0 ObjectTable: 96f22290 HandleCount: 57.
Image: conhost.exe
```

```
kd> g
Break instruction exception - code 80000003 (first chance)
00401000 cc int 3
```

成功断下

```
kd> dg tr
Sel Base Limit Type P Si Gr Pr Lo
-----
0048 0045d9f0 00000068 TSS32 Busy 3 Nb By P Nl 000000eb
```

busy标志为置为1

Registers

Customize...

Reg	Value
ds	23
edi	0
esi	0
ebx	0
edx	0
ecx	0
eax	0
ebp	0
eip	401000
cs	8
efl	4002
esp	45fa50
ss	10
dr0	0
dr1	0
dr2	0
dr3	0
dr6	ffff0ff0
dr7	400
di	0
si	0
bx	0
dx	0
cx	0
ax	0
bp	0
ip	1000
fl	4002
sp	fa50
bl	0
dl	0
cl	0
al	0
ah	0

Command

```

0048 0045d9f0 00000068 TSS32 Busy 3 Nb By P Nl 000000eb
kd> dt _ktss 45d9f0
ntdll!_KTSS
+0x000 Backlink           : 0x28
+0x002 Reserved0         : 0
+0x004 Esp0              : 0x461a50
+0x008 Ss0               : 0x10
+0x00a Reserved1        : 0
+0x00c NotUsed1          : [4] 0
+0x01c CR3               : 0x3f31f5a0
+0x020 Eip               : 0x401000
+0x024 EFlags            : 0
+0x028 Eax               : 0
+0x02c Ecx               : 0
+0x030 Edx               : 0
+0x034 Ebx               : 0
+0x038 Esp               : 0x45fa50
+0x03c Ebp               : 0
+0x040 Esi               : 0
+0x044 Edi               : 0
+0x048 Es                : 0
+0x04a Reserved2        : 0
+0x04c Cs                : 8
+0x04e Reserved3        : 0
+0x050 Ss                : 0x10
+0x052 Reserved4        : 0
+0x054 Ds                : 0x23
+0x056 Reserved5        : 0
+0x058 Fs                : 0
+0x05a Reserved6        : 0
+0x05c Gs                : 0

```

任务段切换

调用门 中断门 任务门

代码

```

#include<windows.h>
#include<stdio.h>
struct _KiIoAccessMap
{
    UCHAR DirectionMap[32];
    //0x0
    UCHAR IoMap[8196];
    //0x20
};
typedef struct _KTSS
{
    USHORT Backlink;
    //0x0
    USHORT Reserved0;
    //0x2
    ULONG Esp0;
    //0x4
    USHORT Ss0;
    //0x8
    USHORT Reserved1;
    //0xa
    ULONG NotUsed1[4];
    //0xc
    ULONG CR3;
    //0x1c
    ULONG Eip;
    //0x20
    ULONG EFlags;
    //0x24
    ULONG Eax;
    //0x28

```

```

    ULONG Ecx;
//0x2c
    ULONG Edx;
//0x30
    ULONG Ebx;
//0x34
    ULONG Esp;
//0x38
    ULONG Ebp;
//0x3c
    ULONG Esi;
//0x40
    ULONG Edi;
//0x44
    USHORT Es;
//0x48
    USHORT Reserved2;
//0x4a
    USHORT Cs;
//0x4c
    USHORT Reserved3;
//0x4e
    USHORT Ss;
//0x50
    USHORT Reserved4;
//0x52
    USHORT Ds;
//0x54
    USHORT Reserved5;
//0x56
    USHORT Fs;
//0x58
    USHORT Reserved6;
//0x5a
    USHORT Gs;
//0x5c
    USHORT Reserved7;
//0x5e
    USHORT LDT;
//0x60
    USHORT Reserved8;
//0x62
    USHORT Flags;
//0x64
    USHORT IoMapBase;
//0x66
    //struct _KiIoAccessMap IoMaps[1];
//0x68
    //UCHAR IntDirectionMap[32];
//0x208c
}KTSS,*PKTSS;

KTSS tss = { 0 };
__declspec(naked) void a()
{
    __asm
    {
        int 3;//中断会清空EFL里的任务嵌套NT位，所以要修改回去
    }
}

```



```

        pushfd;
        pop eax;
        or eax, 0x4000;
        push eax;
        popfd;
        iretd;
    }
}
char esp3[0x2000] = { 0 };
char esp0[0x2000] = { 0 };
int main()
{
    char trcode[2] = { 0 };
    __asm
    {
        str trcode;
    }

    memset(esp3,1,0x2000); //挂物理页
    memset(esp0,1,0x2000);
    printf("tss地址: %x\n", &tss);
    tss.Eax = 0;
    tss.Ecx = 0;
    tss.Edx = 0;
    tss.Ebx = 0;

    tss.Ebp = 0;
    tss.Esi = 0;
    tss.Edi = 0;
    tss.Cs = 0x8;
    tss.Ss = 0x10;
    tss.Ds = 0x23;

    tss.Esp = (ULONG)(esp3+0x2000-8);
    tss.Esp0 = (ULONG)(esp0+0x2000-8);
    tss.Ss0 = 0x10;
    tss.Fs = 0x30;
    tss.Eip = (ULONG)a;

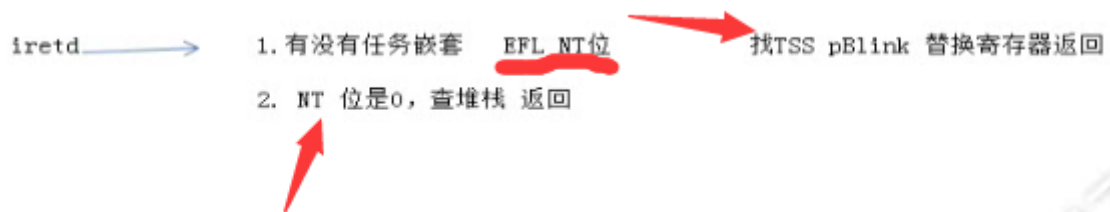
    DWORD dwCr3 = 0;
    printf("请输入CR3:");
    scanf_s("%x", &dwCr3);
    tss.CR3 = dwCr3;
    printf("CR3:%x", tss.CR3);

    printf("func:%x  esp0:%x  esp3:%x", a, tss.Esp0, tss.Esp);

    system("pause");
    // printf("%x\n", sizeof(KTSS));
    char bufcode[6] = { 0,0,0,0,0x48,0 };
    __asm
    {
        call fword ptr bufcode;
    }
    system("pause");
    return 0;
}

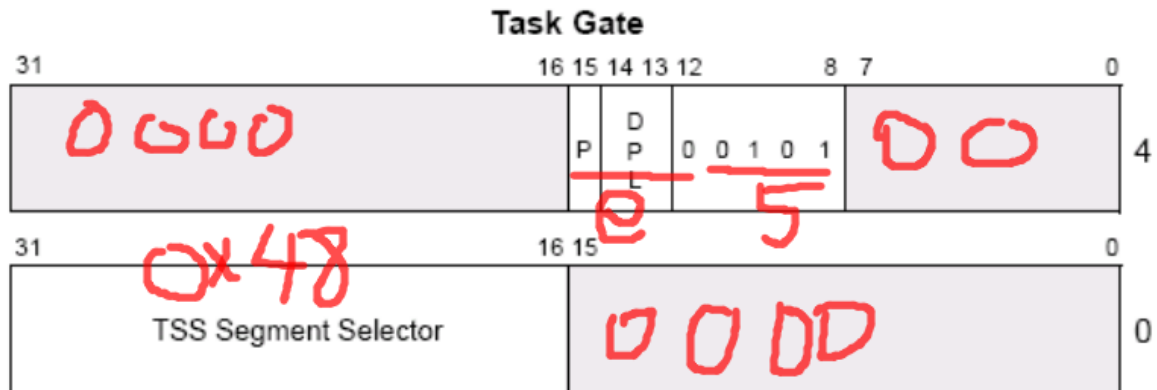
```

iretd指令详解



7.任务门

Task Gate在中断描述表(IDT)里。



1.修改idt表里0x20位置为任务门

```
kd> r idtr
idtr=80b99400
kd> dq 80b99400 L30
80b99400 83e58e00`00086fc0 83e58e00`00087150
80b99410 00008500`00580000 83e5ee00`000875c0
80b99420 83e5ee00`00087748 83e58e00`000878a8
80b99430 83e58e00`00087a1c 83e58e00`00088018
80b99440 00008500`00500000 83e58e00`00088478
80b99450 83e58e00`0008859c 83e58e00`000886dc
80b99460 83e58e00`0008893c 83e58e00`00088c2c
80b99470 83e58e00`000892fc 83e58e00`000896b0
80b99480 83e58e00`000897d4 83e58e00`00089914
80b99490 00008500`00a00000 83e58e00`00089a80
80b994a0 83e58e00`000896b0 83e58e00`000896b0
80b994b0 83e58e00`000896b0 83e58e00`000896b0
80b994c0 83e58e00`000896b0 83e58e00`000896b0
80b994d0 83e58e00`000896b0 83e58e00`000896b0
80b994e0 83e58e00`000896b0 83e58e00`000896b0
80b994f0 83e58e00`000896b0 84248e00`00084af8
80b99500 00000000`00080000 00000000`00080000
80b99510 00000000`00080000 00000000`00080000
80b99520 00000000`00080000 00000000`00080000
80b99530 00000000`00080000 00000000`00080000
80b99540 00000000`00080000 00000000`00080000
80b99550 83e5ee00`0008663a 83e5ee00`000867c0
80b99560 83e5ee00`000868fc 83e5ee00`00087498
80b99570 83e5ee00`00085fee 83e58e00`000896b0
kd> eq 80b99500 0000e500`00480000
```

2.修改gdt表里的对应位置为任务段

```
kd> dq gdtr L20
80b99000 00000000`00000000 00cf9b00`0000ffff
80b99010 00cf9300`0000ffff 00cf9b00`0000ffff
80b99020 00cf9300`0000ffff 80008b1e`400020ab
80b99030 834093f4`3c003748 0040f300`0000ffff
80b99040 0000f200`0400ffff 00000000`00000000
80b99050 830089f4`10000068 830089f4`10680068
80b99060 00000000`00000000 00000000`00000000
80b99070 800092b9`900003ff 00000000`00000000
80b99080 00000000`00000000 00000000`00000000
80b99090 00000000`00000000 00000000`00000000
80b990a0 86008940`71c00068 00000000`00000000
80b990b0 00000000`00000000 00000000`00000000
80b990c0 00000000`00000000 00000000`00000000
80b990d0 00000000`00000000 00000000`00000000
80b990e0 00000000`80b99100 00009200`0000ffff
80b990f0 830098e5`097003b2 00009200`0000ffff
kd> eq 80b99048 0000e945`d9f00068
```

3.成功断下

```
Disassembly - Kernel 'com:port=\\.\pipe\com_1,baud=115200,...
Offset: @$scopeip
Previous Next
00400fe8 0000 add byte ptr [eax],al
00400fea 0000 add byte ptr [eax],al
00400fec 0000 add byte ptr [eax],al
00400fee 0000 add byte ptr [eax],al
00400ff0 0000 add byte ptr [eax],al
00400ff2 0000 add byte ptr [eax],al
00400ff4 0000 add byte ptr [eax],al
00400ff6 0000 add byte ptr [eax],al
00400ff8 0000 add byte ptr [eax],al
00400ffa 0000 add byte ptr [eax],al
00400ffc 0000 add byte ptr [eax],al
00400ffe 0000 add byte ptr [eax],al
00401000 cc int 3
00401001 9c pushfd
00401002 58 pop eax
00401003 0d00400000 or eax,4000h
00401008 50 push eax
00401009 9d popfd
0040100a cf iretd
0040100b cc int 3
0040100c cc int 3
0040100d cc int 3
0040100e cc int 3
0040100f cc int 3
00401010 55 push ebp
```

8.101012分页 (1)

win7 默认29912分页。

101012分页环境配置

1.bcdedit /?

```
C:\Windows\system32>bcdedit /?
```

2.bcdedit /set

```
/set 设置存储中的项选项值。
```

3.bcdedit /? TYPES

```
NTLDR 以前的 Windows OS 版本附带的 Windows 启动加载器
OSLOADER Windows Vista OS 加载器
SYSTEM 系统应用程序
```

4.bcdedit /? TYPES OSLOADER

内存	
=====	
INCREASEUSERVA (整数)	增加用户模式进程可以使用的虚拟地址空间量。
NOLOWMEM (布尔)	禁用低内存。
NX	可以为 OptIn、OptOut、AlwaysOn 或 AlwaysOff。
PAE	可以为 Default、ForceEnable、ForceDisable。
REMOVEMEMORY (整数)	将内存从操作系统可以使用的可用总内存中删除。

NX位关闭 无代码段数据段的区别

5.

10/10/12

```
//bcdedit /set {current} NX OptIn AlwaysOff
```

```
//bcdedit /set {current} NX OptOut AlwaysOff
```

```
bcdedit /set {current} NX AlwaysOff //bcdedit /set {current} NX AlwaysOn
```

```
bcdedit /set {current} PAE ForceDisable //bcdedit /set {current} PAE ForceEnable
```

```
C:\Windows\system32>bcdedit /set {current} NX OptIn AlwaysOff
操作成功完成。

C:\Windows\system32>bcdedit /set {current} NX OptOut AlwaysOff
操作成功完成。

C:\Windows\system32>bcdedit /set {current} PAE ForceDisable
操作成功完成。

C:\Windows\system32>
```

101012分页基础知识

```
mov eax,dword ptr ds:[0x12345678];
```

0x12345678+ds.base = 线性地址。

0001 0010 00 11 0100 0101 (0x678) 0110 0111 1000

从右到左拆分：

11 0100 0101 =0x345

0001 0010 00=0x48

寻址

0x48 *4

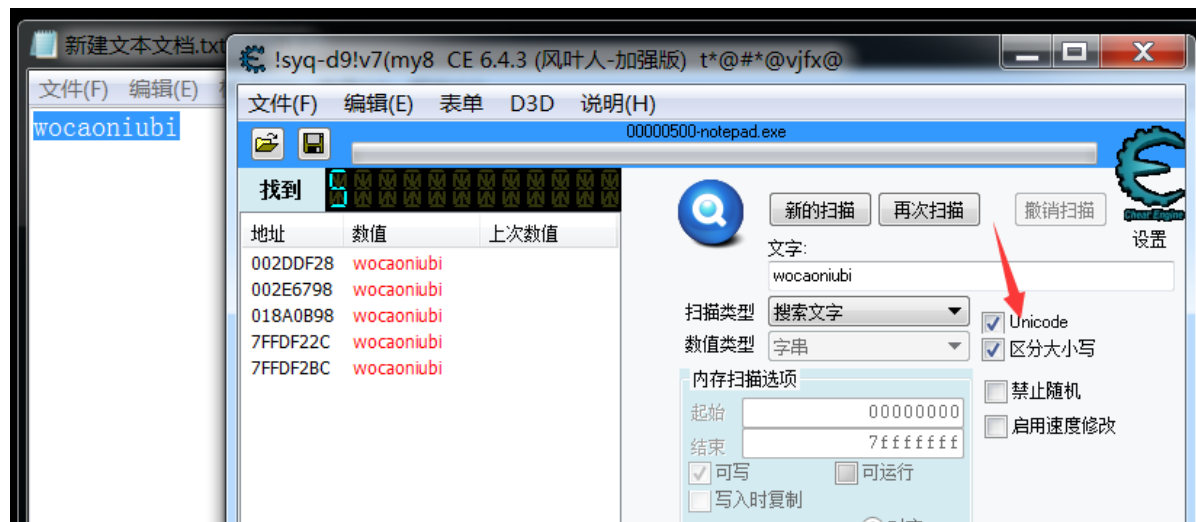
0x345 *4

0x678(页内偏移)

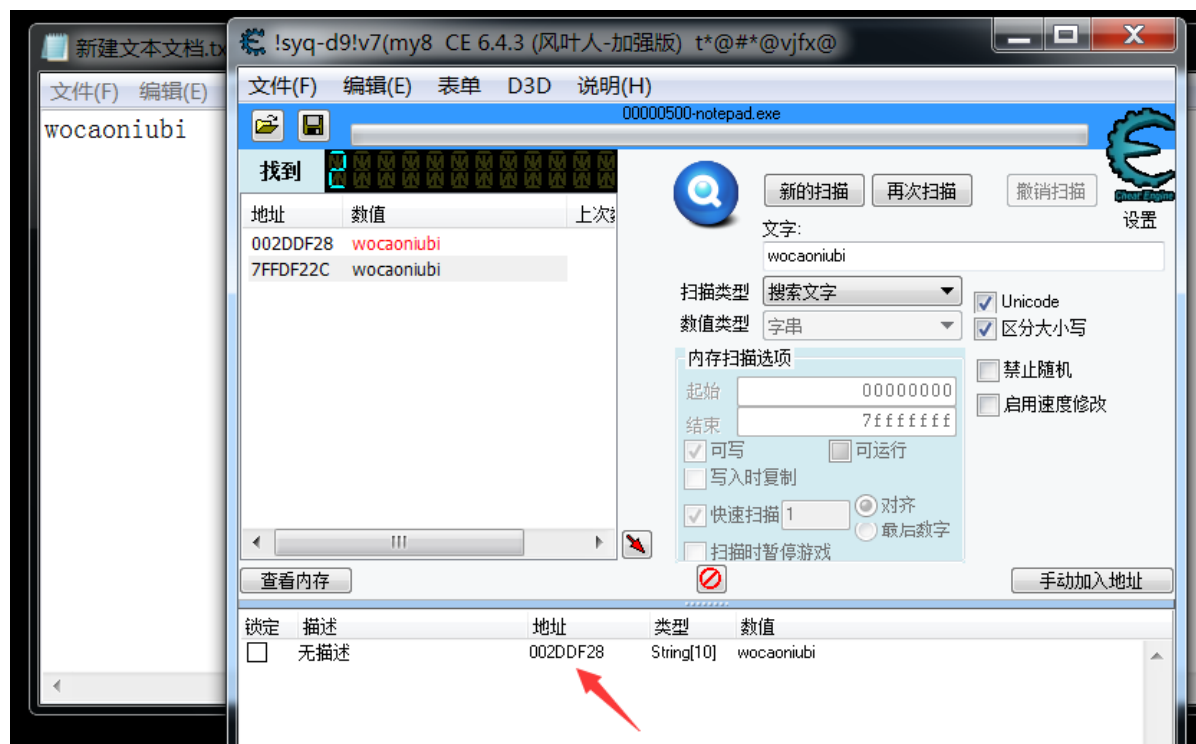
后12位 代表了一个页的大小 0xFFF =4096=0x1000。

寻址实验:

1.



2.过滤地址



3.拆分地址

002DDF28

0000 0000 00

10 1101 1101

0x0 *4

0x2dd *4

0xF28

4.dbg寻址

PROCESS 87c1d9b8 SessionId: 1 Cid: 0500 Peb: 7ffd3000 ParentCid: 0a58
DirBase: 2225d000 ObjectTable: 9f7bcca0 HandleCount: 79.
Image: notepad.exe

2225d000

```
kd> dd 2225d000
2225d000  ?????????? ?????????? ?????????? ??????????
2225d010  ?????????? ?????????? ?????????? ??????????
2225d020  ?????????? ?????????? ?????????? ??????????
2225d030  ?????????? ?????????? ?????????? ??????????
2225d040  ?????????? ?????????? ?????????? ??????????
2225d050  ?????????? ?????????? ?????????? ??????????
2225d060  ?????????? ?????????? ?????????? ??????????
2225d070  ?????????? ?????????? ?????????? ??????????
```

在物理地址寻址的时候，要用kd>!dd

```
kd> !dd 2225d000
#2225d000 1eb36867 1bc04867 00000000 00000000
#2225d010 1ee87867 30c22867 1ed0d867 1abd4867
#2225d020 1d015867 30b2a867 1abac867 00000000
#2225d030 00000000 00000000 00000000 00000000
#2225d040 00000000 00000000 00000000 00000000
#2225d050 00000000 00000000 00000000 00000000
#2225d060 00000000 00000000 00000000 00000000
#2225d070 00000000 00000000 00000000 00000000
kd> !dd 2225d000+0*4 867是属性位 应去掉
#2225d000 1eb36867 1bc04867 00000000 00000000
#2225d010 1ee87867 30c22867 1ed0d867 1abd4867
#2225d020 1d015867 30b2a867 1abac867 00000000
#2225d030 00000000 00000000 00000000 00000000
#2225d040 00000000 00000000 00000000 00000000
#2225d050 00000000 00000000 00000000 00000000
#2225d060 00000000 00000000 00000000 00000000
#2225d070 00000000 00000000 00000000 00000000
```

```
kd> !dd 1eb36000+0x2dd*4
#1eb36b74 03a01867 16fdf867 31238867 2f269867
#1eb36b84 1bf1a867 3131b867 19f9c867 314ad867
#1eb36b94 1ea81867 1707f867 1a964867 19fe3867
#1eb36ba4 311f8867 3112b867 17d2c867 1c362867
#1eb36bb4 02711867 00000000 00000000 00000000
#1eb36bc4 00000000 00000000 00000000 00000000
#1eb36bd4 00000000 00000000 00000000 00000000
#1eb36be4 00000000 00000000 00000000 00000000
```

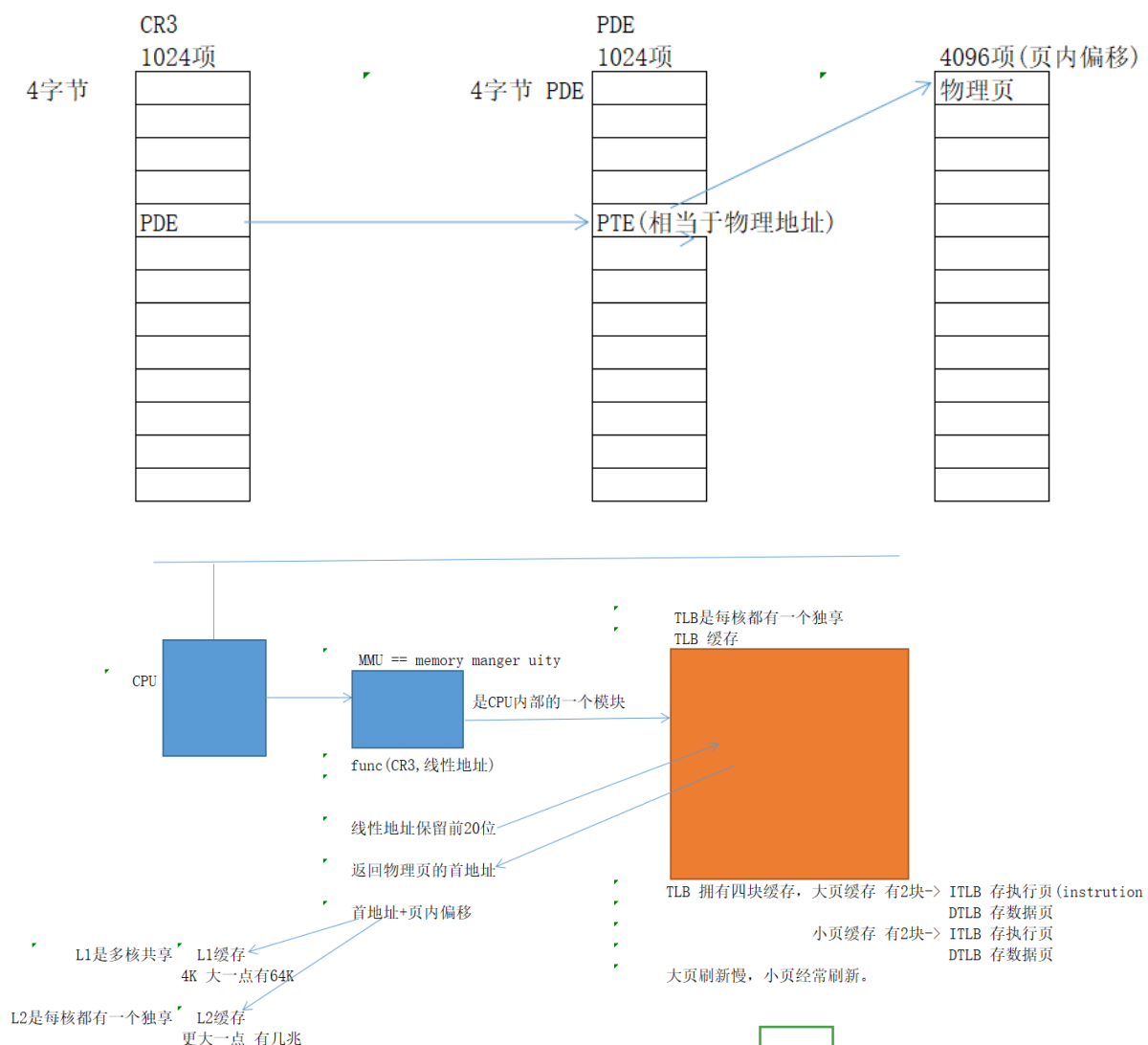
```

kd> !dd 03a01000+0xF28
# 3a01f28 006f0077 00610063 006e006f 00750069
# 3a01f38 00690062 00360036 00360036 002e0348
# 3a01f48 765f3450 00000000 002e492c 00000005
# 3a01f58 002e00a0 00000000 00000001 00000000
# 3a01f68 00000000 014d0004 24178f6d 08007409
# 3a01f78 002e1f28 002e20a8 21178f68 08007401
# 3a01f88 00000409 0012f2e0 0012f220 0012f2e0
# 3a01f98 001165cc 00000003 0012ee7c 00000000
kd> !db 03a01000+0xF28
# 3a01f28 77 00 6f 00 63 00 61 00-6f 00 6e 00 69 00 75 00 w.o.c.a.o.n.i.u.
# 3a01f38 62 00 69 00 36 00 36 00-36 00 36 00 48 03 2e 00 b.i.6.6.6.6.H...
# 3a01f48 50 34 5f 76 00 00 00 00-2c 49 2e 00 05 00 00 00 P4_v.....I.....
# 3a01f58 a0 00 2e 00 00 00 00 00-01 00 00 00 00 00 00 00 .....
# 3a01f68 00 00 00 00 04 00 4d 01-6d 8f 17 24 09 74 00 08 .....M.m..$.t..
# 3a01f78 28 1f 2e 00 a8 20 2e 00-68 8f 17 21 01 74 00 08 (... ..h..!.t..
# 3a01f88 09 04 00 00 e0 f2 12 00-20 f2 12 00 e0 f2 12 00 .....
# 3a01f98 cc 65 11 00 03 00 00 00-7c ee 12 00 00 00 00 00 .e.....|.....

```

理论：

CR3：一个控制寄存器，里面存放 页基址。CR3一共有4096个字节的大小。



9.101012分页 (2) 0地址

0地址挂物理页

```

int p = 0x100;
int main()
{
    int* xxx = (int*)0;
    printf("%x\n", &p);
    system("pause");
    printf("%x", *xxx);
}

```

1.



0044F 000

0000 0000 01 00 0100 1111

1*4

0x4F*4

0

```

kd> !dd 09a03000+1*4
# 9a03004 336e0867 00000000 00000000
# 9a03014 00000000 00000000 00000000
# 9a03024 00000000 00000000 00000000
# 9a03034 00000000 00000000 00000000
# 9a03044 00000000 00000000 00000000
# 9a03054 00000000 00000000 00000000
# 9a03064 00000000 00000000 00000000
# 9a03074 00000000 00000000 00000000
kd> !dd 336e0000+4f*4
#336e013c 16aed867 1284e86 2ee9b225 295d7025
#336e014c 00000000 00000000 00000000 00000000
#336e015c 00000000 00000000 00000000 00000000
#336e016c 00000000 00000000 00000000 00000000
#336e017c 00000000 00000000 00000000 00000000
#336e018c 00000000 00000000 00000000 00000000
#336e019c 00000000 00000000 00000000 00000000
#336e01ac 00000000 00000000 00000000 00000000
kd> !dd 16aed000+0
#16aed000 00000100 00000001 00000001 00000001
#16aed010 00000000 00000001 ffffffff 00000001
#16aed020 894fb234 76b04dcb 0000002f 00000000
#16aed030 00000001 0044639c 0000001c 00000000
#16aed040 19930520 00000000 00000000 00000000
#16aed050 00000000 00000000 00000000 00002001
#16aed060 00000000 00000000 00000000 00000000
#16aed070 001cf7e0 ffffffff 00000000 00000000

```

2.查看0地址


```
kd> !dd 09a03000+0*4
# 9a03000 38da1867 173e6867 00000000 00000000
# 9a03010 00000000 00000000 00000000 00000000
# 9a03020 00000000 00000000 00000000 00000000
# 9a03030 00000000 00000000 00000000 00000000
# 9a03040 00000000 00000000 00000000 00000000
# 9a03050 00000000 00000000 00000000 00000000
# 9a03060 00000000 00000000 00000000 00000000
# 9a03070 00000000 00000000 00000000 00000000
kd> !dd 38da1000+0*0
#38da1000 00000000 00000000 00000000 00000000
#38da1010 00000000 00000000 00000000 00000000
#38da1020 00000000 00000000 00000000 00000000
#38da1030 00000000 00000000 00000000 00000000
#38da1040 173e6867 00000000 00000000 00000000
#38da1050 00000000 00000000 00000000 00000000
#38da1060 00000000 00000000 00000000 00000000
#38da1070 00000000 00000000 00000000 00000000
```

PTT

天 PTT

3. 因为44F 000的后12位是0，0地址的后12位也是0，所以只需要把PTT 赋给0地址即可。

```
kd> !ed 38da1000 16aed867
kd> !dd 38da1000+0*0
#38da1000 16aed867 00000000 00000000 00000000
#38da1010 00000000 00000000 00000000 00000000
#38da1020 00000000 00000000 00000000 00000000
#38da1030 00000000 00000000 00000000 00000000
#38da1040 173e6867 00000000 00000000 00000000
#38da1050 00000000 00000000 00000000 00000000
#38da1060 00000000 00000000 00000000 00000000
#38da1070 00000000 00000000 00000000 00000000
```

4. 如果，当线性地址的页内偏移 != 0的时候，windbg修改时，仍要填入带属性的PTT，而不是带页内偏移的PTT。

访问的时候，再添加偏移。

```
int p = 0x100;
int main()
{
    int* xxx = (int*)0;
    printf("%x\n", &p);
    system("pause");
    printf("%x", *(int *)((ULONG)xxx + (ULONG)&p & 0xFFF));
}
```

分页常识

CPU的分页单位 是页 ==4K

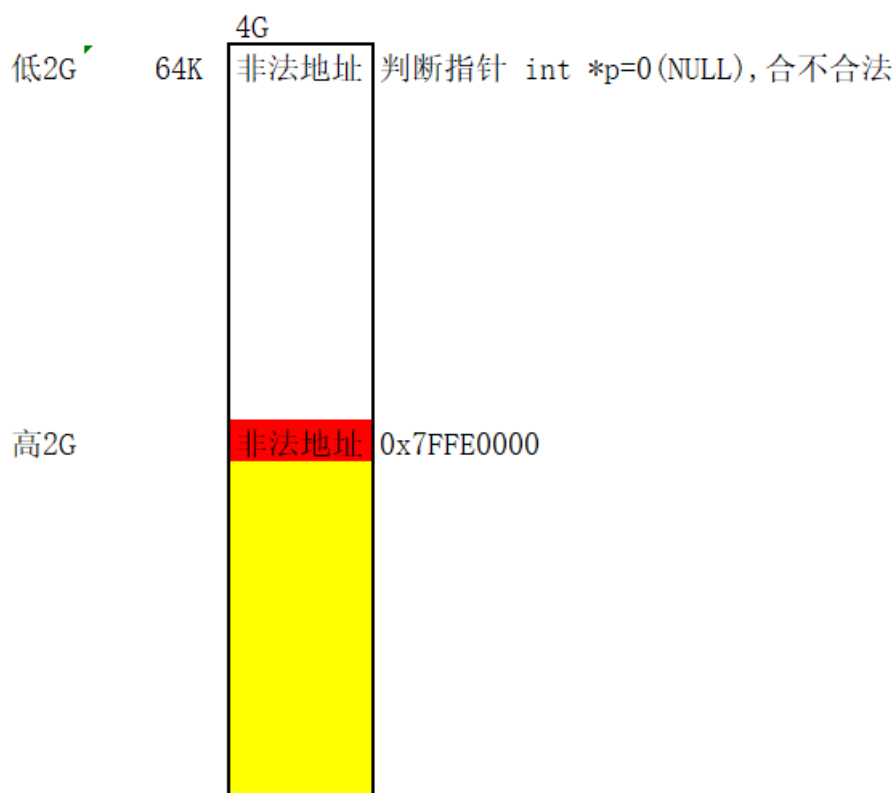
操作系统的分页密度 是64K。

当我们用申请内存函数时，返回的其实是64K (0x10000) 的大小，虽然操作系统的密度比较大，但并不会浪费内存。

当我们再次申请内存时，还会从这64K的页里的某个地址返回给我们。

注意：返回的是64K的线性地址，而不是64K的物理页，当需要用到物理内存时，才会挂上一个物理页。

地址	Allocation Protect	State	保护	类型	长度	其它
00000000		空闲	无法前往	-	10000	
00010000	读+写	提交	读+写	映射	10000	
00020000	读+写	提交	读+写	映射	10000	
00030000	读+写	保留		私有	FD000	
0012D000	读+写	提交	读+写+保护	私有	1000	
0012E000	读+写	提交	读+写	私有	2000	
00130000	读	提交	读	映射	4000	
00134000		空闲	无法前往	-	C000	
00140000	读	提交	读	映射	1000	
00141000		空闲	无法前往	-	F000	
00150000	读+写	提交	读+写	私有	1000	
00151000		空闲	无法前往	-	F000	
00160000	读	提交	读	映射	67000	\\Device\\...
001C7000		空闲	无法前往	-	19000	
001E0000	读+写	提交	读+写	私有	14000	
001F4000	读+写	保留		私有	EC000	
002E0000		空闲	无法前往	-	120000	



10.101012分页 (3) 页属性

1.常量区数据区分

名称	虚拟大小	虚拟地址	Raw 大小	Raw 地址	重定位地址	行号	重定位数值	行号个数	字符特征
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00007FCF	00001000	00008000	00000400	00000000	00000000	0000	0000	60000020
.rdata	00004334	00009000	00004400	00008400	00000000	00000000	0000	0000	40000040
.data	00000544	0000E000	00000200	0000C800	00000000	00000000	0000	0000	C0000040
.rsrc	0000EDF0	0000F000	0000EE00	0000CA00	00000000	00000000	0000	0000	40000040
.reloc	000013FA	0001E000	00001400	0001B800	00000000	00000000	0000	0000	42000040

修改可读属性为可读可写。

名称	虚拟大小	虚拟地址	Raw 大小	Raw 地址	重定位地址	行号	重定位数值	行号个数	字符特征
000002A0	000002A8	000002AC	000002B0	000002B4	000002B8	000002BC	000002C0	000002C2	000002C4
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00043611	00001000	00043800	00000400	00000000	00000000	0000	0000	60000020
.idata	00009BE6	00045000	00009C00	00043C00	00000000	00000000	0000	0000	C0000040
.data	00001CA0	0004F000	00000A00	0004D800	00000000	00000000	0000	0000	C0000040
.msvcjmc	00000016	00051000	00000200	0004E200	00000000	00000000	0000	0000	C0000040
.rsrc	000001E0	00052000	00000200	0004E400	00000000	00000000	0000	0000	C0000040

40000000可读
80000000 可写
4+8=c
C0000000 可读可写

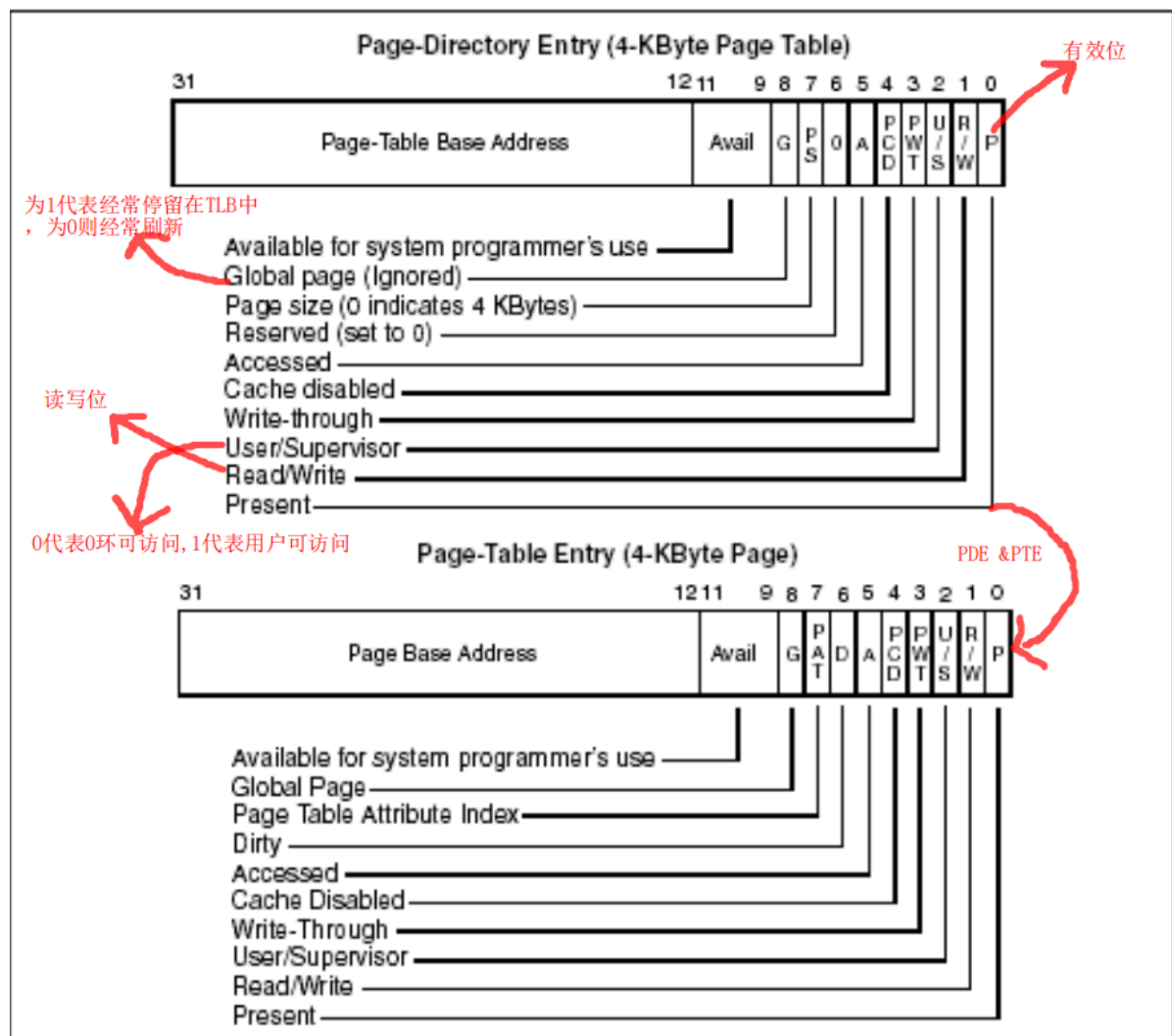
效果：

C:\Users\TalShang\Desktop\Project1.exe
523456请按任意键继续. . .

```
#include<windows.h>
#include<stdio.h>
int main()
{
    char* p = (char*)"123456";
    p[0] = '5';
    printf("%s",p);
    system("pause");
}
```

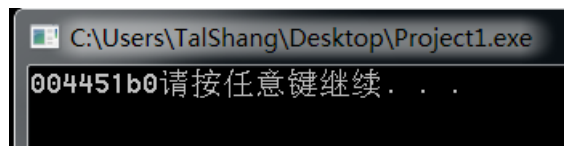
2.PDT&PTT的R\W位

属性图



```
#include<windows.h>
#include<stdio.h>
int main()
{
    char* p = (char*)"123456";
    printf("%s", p);
    system("pause");
    p[0] = '9';
    printf("%s", p);
    system("pause");
    return 0;
}
```

1.拆分线性地址



004451b0

0000 0000 0100 0100 0101

1*4

45*4

1b0

```
kd> !dd 10d65000+1*4
#10d65004 10829867 00000000 00000000 00000000
#10d65014 00000000 00000000 00000000 00000000
#10d65024 00000000 00000000 00000000 00000000
#10d65034 00000000 00000000 00000000 00000000
#10d65044 00000000 00000000 00000000 00000000
#10d65054 00000000 00000000 00000000 00000000
#10d65064 00000000 00000000 00000000 00000000
#10d65074 00000000 00000000 00000000 00000000
kd> !dd 10829000+45*4
#10829114 15149025 107d0025 18ad4025 131bf025
#10829124 00000000 00000000 00000000 00000000
#10829134 25bca025 13577025 192fc867 13d25867
#10829144 3d228225 3d05d025 00000000 00000000
#10829154 00000000 00000000 00000000 00000000
#10829164 00000000 00000000 00000000 00000000
#10829174 00000000 00000000 00000000 00000000
#10829184 00000000 00000000 00000000 00000000
kd> !db 15149000 +1b0
#151491b0 31 32 33 34 35 36 00 00-25 30 38 78 00 00 00 00 123456...%08x....
#151491c0 70 61 75 73 65 00 00 00-25 73 00 00 68 52 44 00 pause...%s...hRD.
#151491d0 48 53 44 00 68 54 44 00-88 54 44 00 c0 54 44 00 HSD.hTD..TD..TD.
#151491e0 ec 54 44 00 01 00 00 00-00 00 00 00 01 00 00 00 .TD.....
#151491f0 01 00 00 00 01 00 00 00-01 00 00 00 53 74 61 63 .....Stac
#15149200 6b 20 61 72 6f 75 6e 64-20 74 68 65 20 76 61 72 k around the var
#15149210 69 61 62 6c 65 20 27 00-27 20 77 61 73 20 63 6f iable '.' was co
#15149220 72 72 75 70 74 65 64 2e-00 00 00 00 54 68 65 20 rrupted.....The
```

2.查看PDE和PTE的属性

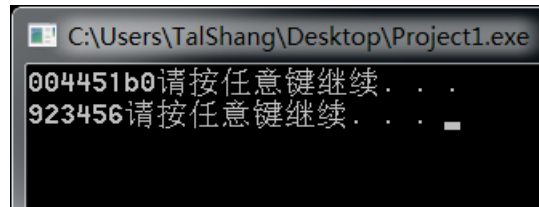
PDE:867==0111==可读可写

PTE:025=0101==可读不可写

修改0101为0111 ==7

```
kd> !ed 10829114 15149027
kd> !dd 10829000+45*4
#10829114 15149027 107d0025 18ad4025 131bf025
#10829124 00000000 00000000 00000000 00000000
#10829134 25bca025 13577025 192fc867 13d25867
#10829144 3d228225 3d05d025 00000000 00000000
#10829154 00000000 00000000 00000000 00000000
#10829164 00000000 00000000 00000000 00000000
#10829174 00000000 00000000 00000000 00000000
#10829184 00000000 00000000 00000000 00000000
```

3.效果



3.PDT&PTT的U\S位与G位 (G位在缓存)

U==普通用户, 3环权限

S==超级用户, 系统权限

```
#include<windows.h>
#include<stdio.h>
int main()
{
    int* p = (int*)0x80b99010;
    system("pause");
    printf("%x", *p);
    system("pause");
    return 0;
}
```

1.寻找高位地址 0x80b99010

```
kd> r gdtr
gdtr=80b99000
kd> dq 80b99000
80b99000 00000000`00000000 00cf9b00`0000ffff
80b99010 00cf9300`0000ffff 00cf9b00`0000ffff
80b99020 00cf9300`0000ffff 80008b1c`a00020ab 这里应该是 0x0000FFFF 不是
80b99030 84409313`7c003748 0040f300`0000ffff
80b99040 0000f200`0400ffff 00000000`00000000
80b99050 84008913`50000068 84008913`50680068
80b99060 00000000`00000000 00000000`00000000
80b99070 800092b9`900003ff 00000000`00000000

0x00cf9300。
```

2.拆分高位地址

80b99010

1000 0000 1011 1001 1001

0010 0000 0010==202

0011 1001 1001=399

010

```

kd> !dd 17ed5000+202*4
#17ed5808 0018a063 3d3f6863 001b7063 001b8063
#17ed5818 001b9063 001ba063 001bb063 001bc063
#17ed5828 001bd063 001be063 001bf063 03d45863
#17ed5838 0fa9a863 001c2063 001c3063 001c4063
#17ed5848 00000000 00000000 001c7063 05400863
#17ed5858 05801863 00000000 00000000 05a02863
#17ed5868 3fed0863 3fec1863 3fec2863 3fec3863
#17ed5878 3fec4863 3fec5863 3fec6863 3fec7863
kd> !dd 0018a000+399*4
# 18ae64 00b99163 00000000 00000000 00000000
# 18ae74 00000000 00000000 00b9f963 00ba0121
# 18ae84 00ba1121 00ba2963 00ba3860 00000000
# 18ae94 00000000 00000000 00000000 00000000
# 18aea4 00000000 00000000 00000000 00000000
# 18aeb4 00000000 00000000 00000000 00000000
# 18aec4 00000000 00000000 00000000 00000000
# 18aed4 00000000 00000000 00000000 00000000
kd> !dd 00b99000+010
# b99010 0000ffff 00cf9300 0000ffff 00cffb00
# b99020 0000ffff 00cff300 a00020ab 80008b1c
# b99030 7c003748 84409313 00000fff 0040f300
# b99040 0400ffff 0000f200 00000000 00000000
# b99050 50000068 84008913 50680068 84008913
# b99060 00000000 00000000 00000000 00000000
# b99070 900003ff 800092b9 00000000 00000000
# b99080 00000000 00000000 00000000 00000000

```

063==0011 修改为0x0111 修改权限为用户权限

163==0011 修改为0x0111 修改权限为用户权限

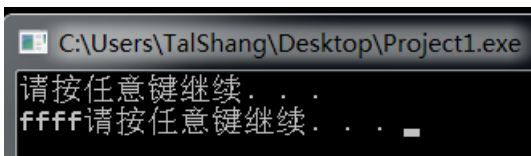
这两个权限是&的关系。

```

kd> !ed 17ed5808 0018a067
kd> !eq 18ae64 00b99167
No export eq found
kd> !ed 18ae64 00b99167
kd> g

```

3.查看效果



4.PDT&PTT的A位与D位

A位：指明这个页或页表是否曾经被访问过。内存管理软件会在页或页表载入内存时，清零该位(可能也没清0)。当第一次被访问过后，会置位该位。

D位：指明该页是否曾经被写入过。内存管理软件会在页或页表载入内存时，清零该位(可能也没清0)。当第一次被访问过后，会置位该位。

注意：申请的线性地址，在没有访问的情况下，没有物理页，当访问后产生一个页异常，挂上物理页。

然后把PDT&PTT的 A位和D位 置1，A位和D位一起起作用，方便页或页表进出物理内存。

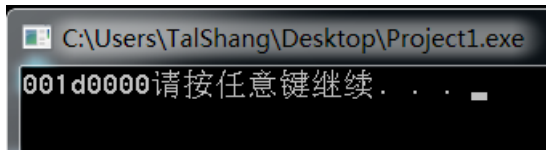
实验：申请内存看一下A\D位。

```

int main()
{
    int* p =
    (int*)VirtualAlloc(0,0x1000,MEM_COMMIT|MEM_RESERVE,PAGE_EXECUTE_READWRITE);
    printf("%08x\n", p);
    system("pause");
    printf("%x\n", *p);
    system("pause");
    *p = 0x100;
    system("pause");
    return 0;
}

```

1.拆分线性地址



001d0 000

0*4

1d0*4

000

2.查看没有访问地址时的页属性

```

kd> !dd 17ee9000+0*4
#17ee9000 02686867 1aa55867 00000000 00000000
#17ee9010 00000000 00000000 00000000 00000000
#17ee9020 00000000 00000000 00000000 00000000
#17ee9030 00000000 00000000 00000000 00000000
#17ee9040 00000000 00000000 00000000 00000000
#17ee9050 00000000 00000000 00000000 00000000
#17ee9060 00000000 00000000 00000000 00000000
#17ee9070 00000000 00000000 00000000 00000000
kd> !dd 02686000+1d0*4
# 2686740 00000000 00000000 00000000 00000000
# 2686750 00000000 00000000 00000000 00000000
# 2686760 00000000 00000000 00000000 00000000
# 2686770 00000000 00000000 00000000 00000000
# 2686780 00000000 00000000 00000000 00000000
# 2686790 00000000 00000000 00000000 00000000
# 26867a0 00000000 00000000 00000000 00000000
# 26867b0 00000000 00000000 00000000 00000000

```

此时PDT的属性为867。无PTT。

奇怪的是，6=0110，这里保留位0，也被置为1。

位6是被保留的并且应当被置为0。当控制寄存器CR4中的PSE和PAE标志置位时，如果保留位没有被置为0，处理器就产生一个页错误。

3.访问线性地址一次

```

kd> !dd 17ee9000+0*4
#17ee9000 02686867 1aa55867 00000000 00000000
#17ee9010 00000000 00000000 00000000 00000000
#17ee9020 00000000 00000000 00000000 00000000
#17ee9030 00000000 00000000 00000000 00000000
#17ee9040 00000000 00000000 00000000 00000000
#17ee9050 00000000 00000000 00000000 00000000
#17ee9060 00000000 00000000 00000000 00000000
#17ee9070 00000000 00000000 00000000 00000000
kd> !dd 02686000+1d0*4
# 2686740 122cd867 00000000 00000000 00000000
# 2686750 00000000 00000000 00000000 00000000
# 2686760 00000000 00000000 00000000 00000000
# 2686770 00000000 00000000 00000000 00000000
# 2686780 00000000 00000000 00000000 00000000
# 2686790 00000000 00000000 00000000 00000000
# 26867a0 00000000 00000000 00000000 00000000
# 26867b0 00000000 00000000 00000000 00000000
kd> !dd 122cd000+0
#122cd000 00000000 00000000 00000000 00000000
#122cd010 00000000 00000000 00000000 00000000
#122cd020 00000000 00000000 00000000 00000000
#122cd030 00000000 00000000 00000000 00000000
#122cd040 00000000 00000000 00000000 00000000
#122cd050 00000000 00000000 00000000 00000000
#122cd060 00000000 00000000 00000000 00000000
#122cd070 00000000 00000000 00000000 00000000

```

发现A位和D位被自动置为1。应该是页异常的时候置位了。

4.修改线性地址的值一次。

```

kd> !dd 17ee9000+0*4
#17ee9000 02686867 1aa55867 00000000 00000000
#17ee9010 00000000 00000000 00000000 00000000
#17ee9020 00000000 00000000 00000000 00000000
#17ee9030 00000000 00000000 00000000 00000000
#17ee9040 00000000 00000000 00000000 00000000
#17ee9050 00000000 00000000 00000000 00000000
#17ee9060 00000000 00000000 00000000 00000000
#17ee9070 00000000 00000000 00000000 00000000
kd> !dd 02686000+1d0*4
# 2686740 122cd867 00000000 00000000 00000000
# 2686750 00000000 00000000 00000000 00000000
# 2686760 00000000 00000000 00000000 00000000
# 2686770 00000000 00000000 00000000 00000000
# 2686780 00000000 00000000 00000000 00000000
# 2686790 00000000 00000000 00000000 00000000
# 26867a0 00000000 00000000 00000000 00000000
# 26867b0 00000000 00000000 00000000 00000000
kd> !dd 122cd000+0
#122cd000 00000100 00000000 00000000 00000000
#122cd010 00000000 00000000 00000000 00000000
#122cd020 00000000 00000000 00000000 00000000
#122cd030 00000000 00000000 00000000 00000000
#122cd040 00000000 00000000 00000000 00000000
#122cd050 00000000 00000000 00000000 00000000
#122cd060 00000000 00000000 00000000 00000000
#122cd070 00000000 00000000 00000000 00000000

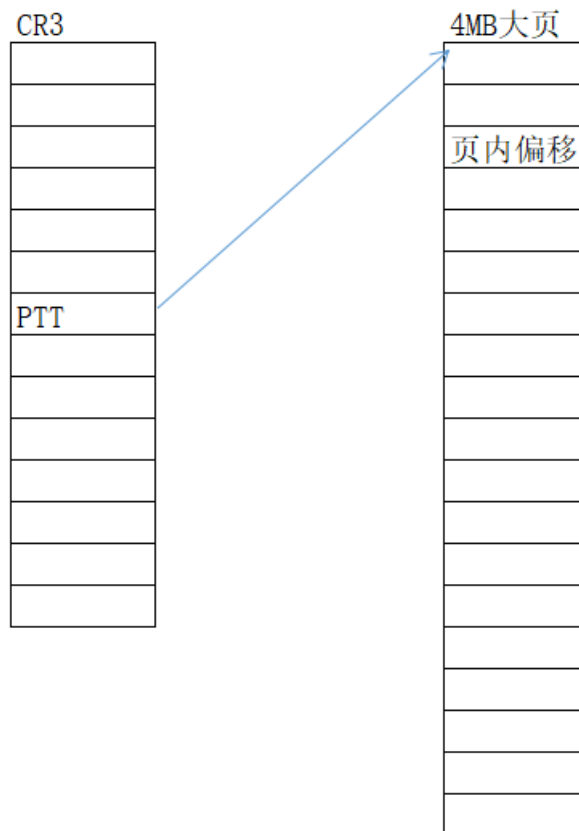
```

5.PDT&PTT的PS位

PS: page size确定页的大小

为0代表4KB的小页。

为1时，一个线性地址的前10位拿来寻址，也就是只有PDE，后面22位都是页内偏移== 2^{22} 次方 == 4,194,304 字节== 4096KB ==4MB



6.小细节

隐藏CR3就隐藏了内存，例如当获取DXF的CR3时，给与的就是假CR3。（注入仍能获取真正的CR3。
（老版本DXF

新版本DXF取消假CR3，使用PF缺页异常（设置P ==1。

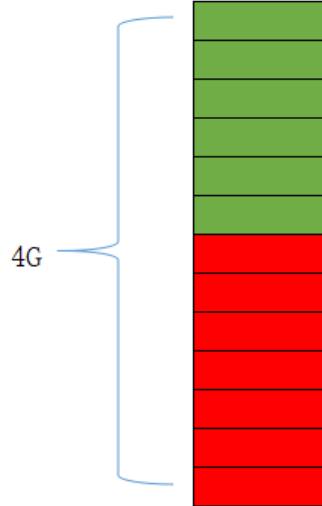
XF仍旧是假CR3。

7.思考

在保护模式下，操作系统也不能访问物理内存，如何拿到PDE和PTE？

10.101012分页 (4) 0xC0000000

操作系统如何管理内存



$4G = 4 * 1024M = 4 * 1024 * 1024KB = 4,194,304$
 $4G / 4K(\text{页大小}) = 1M$
 $1M * 4 = 4M$

$0xC0000000 / 4G * 4M$
 $0xC0000000 / 0x10000000 * 4M$
 $0xC / 0x10 * 4M$
 $0xC / 0x10 * 400000$
 $0xC * 400000 = 0x300000$
 $0x300000 + 0xC0000000 = 0xC0300000$

$0xC0000000 - 0xC0300000$ 是PTE
 $0xC0300000 - 0xC0400000$ 是PDE
 原本4M全是PTE，按照它的拆法，才有PDE和PTE区别。
 CR3 → PDE → PTE

在保护模式下，操作系统也不能访问物理内存，如何拿到PDE和PTE？

1. 拆分0xC0000000 (PTE首地址)

$300 * 4$

$0 * 4$

0

```

kd> !dd 00185000+300*4
# 185c00 0185063 3feb1863 24814863 00000000
# 185c10 00000000 00000000 00000000 00000000
# 185c20 00000000 00000000 00000000 00000000
# 185c30 00000000 00000000 00000000 00000000
# 185c40 00000000 00000000 00000000 00000000
# 185c50 00000000 00000000 00000000 00000000
# 185c60 00000000 00000000 00000000 00000000
# 185c70 00000000 00000000 00000000 00000000
kd> !dd 00185000+0*4
# 185000 24802867 00000000 00000000 00000000
# 185010 00000000 00000000 00000000 00000000
# 185020 00000000 00000000 00000000 00000000
# 185030 00000000 00000000 00000000 00000000
# 185040 00000000 00000000 00000000 00000000
# 185050 00000000 00000000 00000000 00000000
# 185060 00000000 00000000 00000000 00000000
# 185070 00000000 00000000 00000000 00000000
kd> !dd 24802000+0
#24802000 00000000 00000000 00000000 00000000
#24802010 00000000 00000000 00000000 00000000
#24802020 00000000 00000000 00000000 00000000
#24802030 00000000 00000000 00000000 00000000
#24802040 248fc867 248dd867 248fe867 248df867
#24802050 24860867 24841867 24832867 24823867
#24802060 24844867 24865867 24896867 248e7867
#24802070 24928867 248a9867 248ea867 248cb867
    
```

发现CR3和PDE一样。也就是说没有PDE的存在，只有PTE。

2.拆分0xC0300C00

300*4

300*4

C00

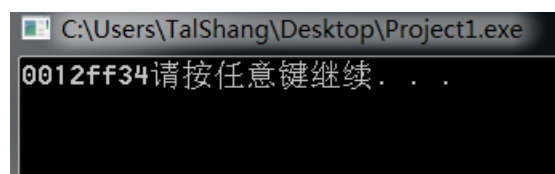
```
kd> !dd 00185000+300*4
# 185c00 00185063 3feb1863 24814863 00000000
# 185c10 00000000 00000000 00000000 00000000
# 185c20 00000000 00000000 00000000 00000000
# 185c30 00000000 00000000 00000000 00000000
# 185c40 00000000 00000000 00000000 00000000
# 185c50 00000000 00000000 00000000 00000000
# 185c60 00000000 00000000 00000000 00000000
# 185c70 00000000 00000000 00000000 00000000
kd> !dd 00185000+300*4
# 185c00 00185063 3feb1863 24814863 00000000
# 185c10 00000000 00000000 00000000 00000000
# 185c20 00000000 00000000 00000000 00000000
# 185c30 00000000 00000000 00000000 00000000
# 185c40 00000000 00000000 00000000 00000000
# 185c50 00000000 00000000 00000000 00000000
# 185c60 00000000 00000000 00000000 00000000
# 185c70 00000000 00000000 00000000 00000000
kd> !dd 00185000+c00
# 185c00 00185063 3feb1863 24814863 00000000
# 185c10 00000000 00000000 00000000 00000000
# 185c20 00000000 00000000 00000000 00000000
# 185c30 00000000 00000000 00000000 00000000
# 185c40 00000000 00000000 00000000 00000000
# 185c50 00000000 00000000 00000000 00000000
# 185c60 00000000 00000000 00000000 00000000
# 185c70 00000000 00000000 00000000 00000000
```

发现每项都是CR3，操作系统可通过模拟CPU拆分线性地址操作，得到CR3。（绕了两次）

3.随机拆线性地址

```
#include<windows.h>
#include<stdio.h>
int main()
{
    int a = 1;
    printf("%08x", &a);
    system("pause");
    return 0;
}
```

1.



0x0012f f34

0*4

12f*4

f34

2.正常通过CR3拆分

```
kd> ldd 2c141000+0*4
#2c141000 0663c867 30b0a867 00000000 00000000
#2c141010 00000000 00000000 00000000 00000000
#2c141020 00000000 00000000 00000000 00000000
#2c141030 00000000 00000000 00000000 00000000
#2c141040 00000000 00000000 00000000 00000000
#2c141050 00000000 00000000 00000000 00000000
#2c141060 00000000 00000000 00000000 00000000
#2c141070 00000000 00000000 00000000 00000000
kd> ldd 0663c000+12f*4
# 663c4bc 30ec3867 1c4ad025 1c26e025 1c4af025
# 663c4cc 1a5e0025 00000000 00000000 00000000
# 663c4dc 00000000 00000000 00000000 00000000
# 663c4ec 00000000 00000000 00000000 00000000
# 663c4fc 00000000 06c24025 00000000 00000000
# 663c50c 00000000 00000000 00000000 00000000
# 663c51c 00000000 00000000 00000000 00000000
# 663c52c 00000000 00000000 00000000 00000000
kd> ldd 30ec3000+f34
#30ec3f34 00000001 cccccccc 36413a79 0012ff88
#30ec3f44 00401577 00000001 0024fe40 00250ba0
#30ec3f54 36413ab1 00000000 00000000 7ffdb000
#30ec3f64 a454b600 0000004f 00c34183 0012ff54
#30ec3f74 92158c49 0012ffc4 00402f80 36171fa9
#30ec3f84 00000000 0012ff94 774f3c45 7ffdb000
#30ec3f94 0012ffd4 773b37f5 7ffdb000 76db3d52
#30ec3fa4 00000000 00000000 7ffdb000 00000000
```

PDE:0663c867

PTE:30ec3867

3.挂靠进程，通过虚拟地址拆分

0xC0000000-0xC0300000 是PTE

0xC0300000-0xC0400000 是PDE

```
kd> .process /i 8696d670
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
kd> g
Break instruction exception - code 80000003 (first chance)
nt!RtlpBreakWithStatusInstruction:
8409cd00 cc          int      3
```

dd 0xC0000000+12f*4

```
kd> dd C0000000+12f*4
c00004bc 30ec3867 1c4ad025 1c26e025 1c4af025
c00004cc 1a5e0025 00000000 00000000 00000000
c00004dc 00000000 00000000 00000000 00000000
c00004ec 00000000 00000000 00000000 00000000
c00004fc 00000000 06c24025 00000000 00000000
c000050c 00000000 00000000 00000000 00000000
c000051c 00000000 00000000 00000000 00000000
c000052c 00000000 00000000 00000000 00000000
```

dd 0xC0300000+0*4

```
kd> dd 0xC0300000+0*4
c0300000 0663c867 30b0a867 00000000 00000000
c0300010 00000000 00000000 00000000 00000000
c0300020 00000000 00000000 00000000 00000000
c0300030 00000000 00000000 00000000 00000000
c0300040 00000000 00000000 00000000 00000000
c0300050 00000000 00000000 00000000 00000000
c0300060 00000000 00000000 00000000 00000000
c0300070 00000000 00000000 00000000 00000000
```

4.实验：访问所有4G地址

```
void UpdateAddress()
{
    DWORD pdebase = 0xC0300000;
    DWORD ptebase = 0xC0000000;
    for (unsigned int i = 0x80000000; i <= 0xFFFFE000; i+=0x1000)
    {
        DWORD* pde = (DWORD*)((i >> 20) & 0xFFC) + pdebase;
        DWORD pdeAddress = *pde;
        if ((pdeAddress & 1) == 0)
        {
            continue;
        }
        pdeAddress |= 7;
        *pde = pdeAddress;
        if ((pdeAddress & 0x80) == 0x80)
        {
            continue;
        }

        DWORD* pte = (DWORD*)((i >> 10) & 0x3FFFC) + ptebase;
        DWORD pteAddress = *pte;
        if ((pteAddress & 1) == 0)
        {
            continue;
        }
        if ((pteAddress & 0x80) == 0x80)
        {
            continue;
        }
        pteAddress |= 7;
        *pte = pteAddress;
    }
}

_declspec(naked) void callgate()
{
    __asm
    {
        pushfd;
        pushad;
        push fs;
        mov ax, 0x30;
        mov fs, ax;
        call UpdateAddress;
        pop fs;
        popad;
        popfd;
        mov eax, cr3; //刷新CR3，防止缓存
        mov cr3, eax;
        retf;
    }
}

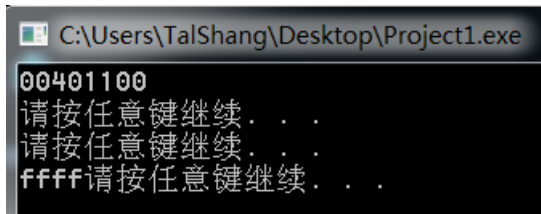
int main()
{
    /*__asm
```

```

{
    mov ecx, 0x12345678;
    shr ecx, 10;
    and ecx, 0x3FFFC;
}*/
char buf[6] = { 0,0,0,0,0x48,0 };
printf("%08x\n", callgate);
system("pause");
__asm
{
    call fword ptr buf;
}
system("pause");
int* x = (int *)0x80b99010;
printf("%x", *x);
return 0;
}

```

调用门: 0040ec00`00081100



```

; __stdcall HalpUnmapVirtualAddress(x, x, x)
_HalpUnmapVirtualAddress@12 proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch
arg_8= byte ptr  10h

mov     edi, edi
push    ebp
mov     ebp, esp
push    esi
mov     esi, [ebp+arg_0]
cmp     esi, 0FFD00000h
jb      short loc_80019C16

```

```

push    ebx
push    edi
and     esi, 0FFFFFF00h
push    esi
call    _MiGetPteAddress@4 ; MiGetPteAddress(x)
mov     ebx, [ebp+arg_4]
mov     edi, eax
mov     [ebp+arg_0], esi
test    ebx, ebx
jbe     short loc_80019BFB

```

这个地址给硬件用的

```

loc_80019BD5:
push    edi
call    _HalpFreePte@4 ; HalpFreePte(x)
cmp     [ebp+arg_8], 0
jz      short loc_80019BE7

```

5. 纠错0xC0300000中关于PTE的存储

80b99 010

0010 0000 0010 11

202*4

399*4

010

```

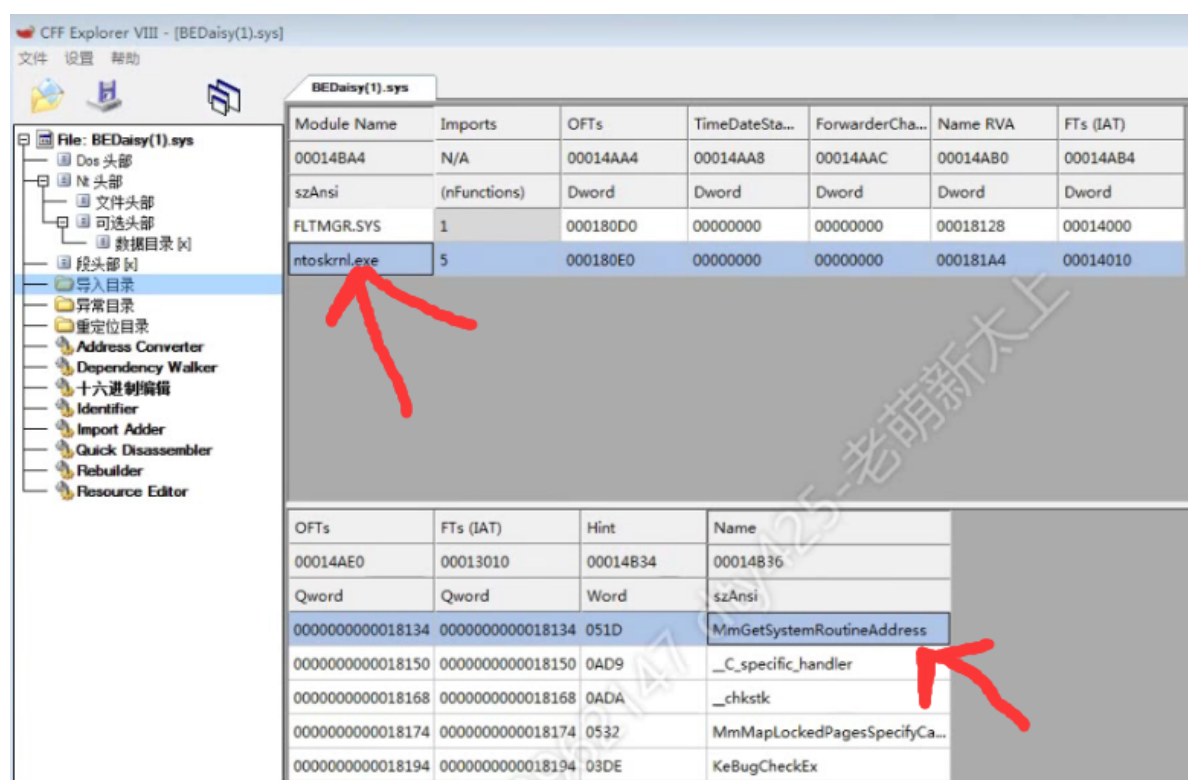
kd> dd 0xC0000000+202E64
c0202e64  00b99163  00000000  00000000  00000000
c0202e74  00000000  00000000  00000000  00000000
c0202e84  00000000  00000000  00000000  00000000
c0202e94  00000000  00000000  00000000  00000000
c0202ea4  00000000  00000000  00000000  00000000
c0202eb4  00000000  00000000  00000000  00000000
c0202ec4  00000000  00000000  00000000  00000000
c0202ed4  00000000  00000000  00000000  00000000
kd> dd 0xC0000000+399*4
c0000e64  00000000  00000000  00000000  00000000
c0000e74  00000000  00000000  00000000  00000000
c0000e84  00000000  00000000  00000000  00000000
c0000e94  00000000  00000000  00000000  00000000
c0000ea4  00000000  00000000  00000000  00000000
c0000eb4  00000000  00000000  00000000  00000000
c0000ec4  00000000  00000000  00000000  00000000
c0000ed4  00000000  00000000  00000000  00000000

```

根据PDE找PTE

11.29912分页(PAE)物理地址扩展

BEDaisy略微分析



Vmp保护了导入表

MmGetSystemRoutineAddress --> 获取内核库和HAL --> vmp修复导入表

理论基础

101012分页 intel 规定PTE PDE是四个字节。

32位->36位 就像16位->20位。

这时，可以通过修改PDE和PTE为8字节来增大物理内存。

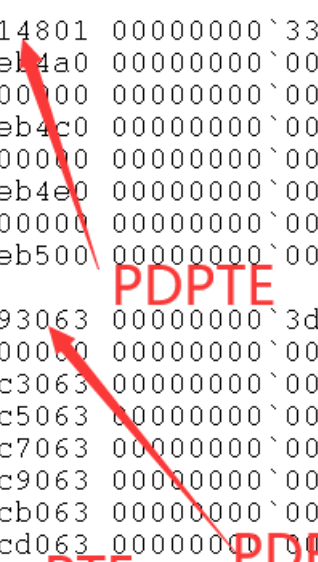
这就造成了不同的CPU，有不同的PDE和PTE的字节大小。

中文版的手册有点老了。

000

```
kd> r gdtr
gdtr=80b99000
kd> !dq 3f2eb460+2*8
#3f2eb470 00000000`32b14801 00000000`33295801
#3f2eb480 83f95f80`86eeb4a0 00000000`00000000
#3f2eb490 00000000`00000000 00000000`00000000
#3f2eb4a0 83f95f80`86eeb4c0 00000000`00000000
#3f2eb4b0 00000000`00000000 00000000`00000000
#3f2eb4c0 83f95f80`86eeb4e0 00000000`00000000
#3f2eb4d0 00000000`00000000 00000000`00000000
#3f2eb4e0 83f95f80`86eeb500 00000000`00000000
kd> !dq 32b14000+5*8
#32b14028 00000000`00193063 00000000`3d5e2863
#32b14038 00000000`00000000 00000000`001c2063
#32b14048 00000000`001c3063 00000000`001c4063
#32b14058 00000000`001c5063 00000000`001c6063
#32b14068 00000000`001c7063 00000000`001c8063
#32b14078 00000000`001c9063 00000000`001ca063
#32b14088 00000000`001cb063 00000000`001cc063
#32b14098 00000000`001cd063 00000000`001ce063
kd> !dq 00193000+199*8
# 193cc8 00000000`00b99163 00000000`00000000
# 193cd8 00000000`00000000 00000000`00b9c963
# 193ce8 00000000`00b9d121 00000000`00b9e121
# 193cf8 00000000`00b9f963 00000000`00ba0860
# 193d08 00000000`00000000 00000000`00000000
# 193d18 00000000`00000000 00000000`00000000
# 193d28 00000000`00000000 00000000`00000000
# 193d38 00000000`00000000 00000000`00000000

kd> !dq 00b99000+0
# b99000 00000000`00000000 00cf9b00`0000ffff
# b99010 00cf9300`0000ffff 00cffb00`0000ffff
# b99020 00cff300`0000ffff 80008b1e`400020ab
# b99030 834093f7`9c003748 0040f300`00000fff
# b99040 0000f200`0400ffff 00000000`00000000
# b99050 830089f7`70000068 830089f7`70680068
# b99060 00000000`00000000 00000000`00000000
# b99070 800092b9`900003ff 00000000`00000000
```



12.PAT/PCD/PWT

定义:

PAT== page attribute table

PCD== page cache disable =1则禁止某页写入缓存，只能写内存。

PWT== Page Write Through ,写数据到缓存时，也写到内存中。

PAT最高位 PCD中间位 PWT最低位

111==7 (最大

缓存类型:

这里的WriteProtected是针对局部的写内存。CR4中有个全局的。

11.7 page 的内存 cache 类型

在 x86/x64 里可以为一个物理地址区域定义一个内存的 cache 类型，每个 cache 类型对应一个值。在 Intel 处理器上这个 cache 类型可以为以下类型。

W开头，通通都是读缓存。

定义一个物理地址区域的 `cache` 类型，可以使用以下两种方法。

- ①使用 MTRR 来定义。
- ② 使用 page 的 PAT、PCD 与 PWT 标志来定义。

使用 MTRR 可以对 Fixed-range 和 Variable-range 进行定义, 详见 7.2 节描述。

检测是否支持 PAT 功能

页属性的cache

$XD = NX$, $=1$ 则页不可执行, 0 可执行

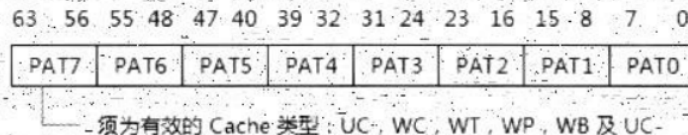
Ignored 无PTE时的大页有PAT

DEP数据保护的原理, DEP有进程的和全局的。

PAT MSR

11.7.2 PAT MSR

在 MSR 中提供了一个 64 位的 IA32_PAT 寄存器用来定义 PAT，如下图所示。



IA32_PAT 寄存器的每个字节对应一个 PAT 元素，所定义的 Cache 类型必须是有效的 cache 类型（保留的必须为 0 值）。如前面所述，在处理器 power-up 或 reset 时，这个 IA32_PAT 里的元素使用默认的 cache 类型。

IA32_PAT 寄存器的地址是 277H，软件可以在 0 级权限下使用 WRMSR 指令对 IA32_PAT 寄存器进行重新设置。rdmsr 277h

1.

```
kd> rdmsr 277  
msr[277] = 00070106`00070106
```

00 07 01 06 00 07 01 06 （对应缓存类型

7 6 5 4 3 2 1 0

如果PAT =0,PCD=0, PWT=0, 则为000, 查右边第一个==0。

JLEAF, 返回1, 线性地址有效

```
00489BB7 loc_489BB7: ;  
00489BB7 shr     ecx, 10      ; >>12*4  
00489BBA and     ecx, 3FFFFCh  
00489BC0 sub     ecx, 40000000h ; add ecx,C000 0000  
00489BC6 mov     eax, [ecx]  
00489BC8 test    al, 1  
00489BCA jz      short loc_489BAD  
  
00489BCC and     al, 80h      ; 保证PAT位=0  
00489BCE cmp     al, 80h      ; 因为PAT默认=0  
00489BD0 setnz   al          ; 如果PAT被设置为1,则认为线性地址无效  
00489BD3 retn              ; 但是PAT=1,并无错  
00489BD3 _MiIsAddressValid@8 endp  
00489BD3  
  
00489BAD loc_489BAD:  
00489BAD xor     al, al  
00489BAF retn
```

反线性地址可效

PAT位置1。

```
kd> x nt!MmisAddress*  
83e5f717 nt!MmisAddressValid (_MmisAddressValid@4)  
kd> x nt!*isaddress*  
83e5271b nt!PopIsAddressRangeValid (@PopIsAddressRangeValid@8)  
83e5271b nt!IopIsAddressRangeValid (_IopIsAddressRangeValid@8)  
83f02aa8 搜索导出函数!MiIsAddressValid (_MiIsAddressValid@8)  
83e5f717 nt!MmisAddressValid (_MmisAddressValid@4)
```

```

kd> !x nt!MmIsAddress*
83e5f717 nt!MmIsAddressValid (_MmIsAddressValid@4)
kd> x nt!*isaddress*
83e5271b nt!PopIsAddressRangeValid (@PopIsAddressRangeValid@8)
83e5271b nt!IopIsAddressRangeValid (_IopIsAddressRangeValid@8)
83f02aa8 nt!MiIsAddressValid (_MiIsAddressValid@8)
83e5f717 nt!MmIsAddressValid (_MmIsAddressValid@4)
kd> bp 83e5f717
kd> bp MmIsAddressValid
breakpoint 0 redefined
kd> bp MmIsAddressValid
breakpoint 0 redefined
kd> g
Breakpoint 0 hit
nt!MmIsAddressValid:
83e5f717 8bff mov edi,edi
kd> kv
# ChildEBP RetAddr Args to Child
00 807ec9a4 967a4d8e 8663b008 8663b000 887b39d8 nt!MmIsAddressValid
WARNING: Stack unwind information not available. Following frames may be wrong.
01 807ec9d8 840122e6 887b39d8 8663b000 00000000 pchunter32as+0x75d8e
02 807ecbbc 84015d98 00000001 00000000 807ecbe4 nt!IopLoadDriver+0x7ed
03 807ecc00 83ecbaab 8edf7bd0 00000000 863eba70 nt!IopLoadUnloadDriver+0x70
04 807ecc50 84057f5e 00000001 f1a5f913 00000000 nt!ExpWorkerThread+0x10d
05 807ecc90 83eff219 83ecb99e 00000001 00000000 nt!PspSystemThreadStartup+0x9e
06 00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0x19
kd> bc *
kd> bl
kd> g

```

取消断点
查看断点

反线性地址实验

```

kd> !pte 83e5f717
VA 83e5f717
PDE at C06020F8 PTE at C041F2F8
contains 00000000001D9063 contains 0000000003E5F121
pfn 1d9 ---DA--KWEV pfn 3e5f -G--A--KREV
kd> dq C041F2F8
c041f2f8 00000000`03e5f121 00000000`03e60121
c041f308 00000000`03e61121 00000000`03e62121
c041f318 00000000`03e63121 00000000`03e64121
c041f328 00000000`03e65121 00000000`03e66121
c041f338 00000000`03e67121 00000000`03e68121
c041f348 00000000`03e69121 00000000`03e6a121
c041f358 00000000`03e6b121 00000000`03e6c121
c041f368 00000000`03e6d121 00000000`03e6e121
kd> eq 00000000`03e5f1a1
kd> q

```

这里是PAT=0
这里是PAT=1

!vtop cr3 线性地址

反调试,反内存扫描。

申请内存,但不访问,此时没有物理页,如果进程被调试器附加,或者模块被特征码扫描,调试可能会给上物理页,特征码扫描一定会挂上物理页,此时检测申请的线性地址的物理页

(NtQueryVirtualMemory) (MmIsAddressValid)即可达到反部分调试和反内存扫描的效果。

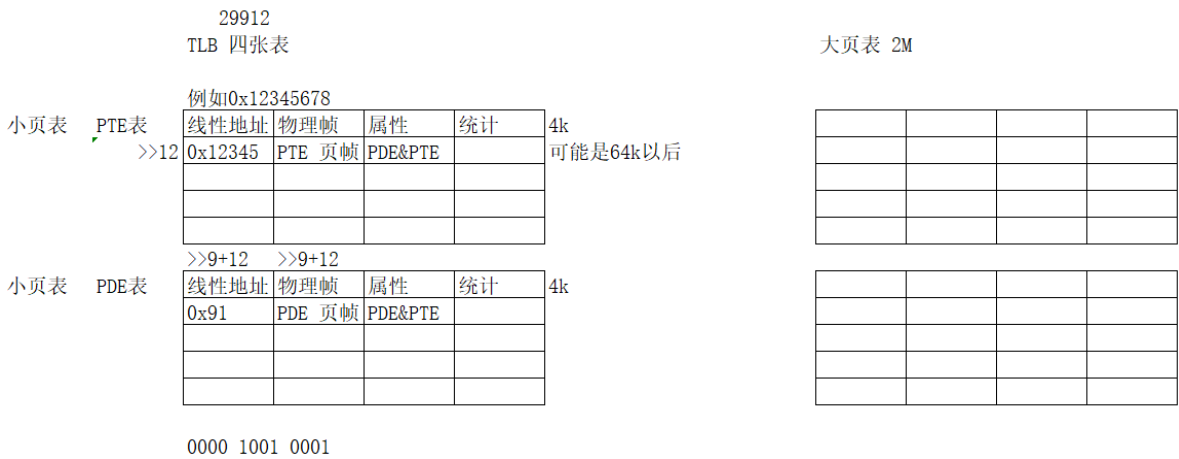
标志位是MemoryWorkingSetExInformation 进程的工作集 里面都是挂上物理页的线性地址。

MemoryMappedFilenameInformation 从线性地址返回模块名。

13.TLB缓存

页帧

页帧数---->PDE页帧，PTE页帧。
大页下，PDE抹掉页内偏移就是 PDE页帧。29912>>9+12>>PDE页帧。
小页下，PTE抹掉后12位，前面的位数就是PTE页帧。



0000 1001 0001

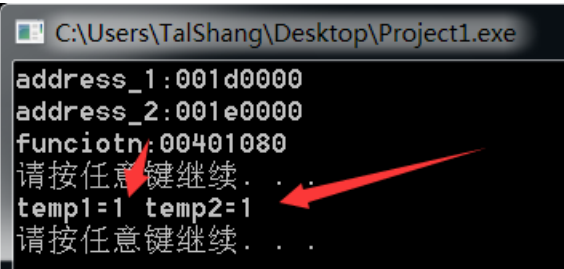
查找流程

- 1.线性地址-->TLB缓存
- 2.没有找到 则 线性地址-->物理帧缓存（page struct cache） -->PDE页帧-->PTE页帧（PTE并没有被缓存保存。
- 3.都没有则 线性地址-->PDPTE-->PDE-->PTE。

cache实验

- 1.申请两个线性地址并访问。
- 2.把两个线性地址的pte，分别赋给0地址，然后mov取0地址的值，观察结果

```
kd> eq 80b99048 0040ec00`00081080
kd> g
```



发现都是 1。

代码：不刷新CR3

```
int* address_1=0;
int* address_2=0;
int temp1;
int temp2;
__declspec(naked) void a()
{
```

```

__asm
{
    pushad;
    pushfd;
    push fs;
    mov ax, 0x30;
    mov fs, ax;

    mov eax, [address_1];
    shr eax, 9; /*8
    and eax, 0x7FFFF8; //保留20位和清零最后3位
    add eax, 0xC0000000; //存放8字节PTE
    mov ecx, [eax]; //取低位
    mov ebx, [eax + 4]; //取高位
    mov dword ptr ds:[0xC0000000], ecx;
    mov dword ptr ds:[0xC0000004], ebx;
    mov eax, dword ptr ds : [0] ;
    mov dword ptr ds : [temp1] , eax;

    mov eax, [address_2];
    shr eax, 9;
    and eax, 0x7FFFF8;
    add eax, 0xC0000000; //存放8字节PTE
    mov ecx, [eax]; //取低位
    mov ebx, [eax + 4]; //取高位
    mov dword ptr ds:[0xC0000000], ecx;
    mov dword ptr ds:[0xC0000004], ebx;
    mov eax, dword ptr ds : [0] ;
    mov dword ptr ds : [temp2] , eax;

    pop fs;
    popfd;
    popad;

    retf;
}
}
int main()
{
    address_1 = (int*)VirtualAlloc(0,0x1000,MEM_COMMIT,PAGE_EXECUTE_READWRITE);
    address_2 = (int*)VirtualAlloc(0,0x1000, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    *address_1 = 0x1;
    *address_2 = 0x2;
    temp1 = 3;
    temp2 = 3;
    printf("address_1:%08x\n", address_1);
    printf("address_2:%08x\n", address_2);
    printf("funciotn:%08x\n", a);
    system("pause");

    char bufcode[6] = { 0,0,0,0,0x48,0 };
    __asm
    {
        call fword ptr bufcode;
    }
    printf("temp1=%x temp2=%x \n",temp1,temp2);
    system("pause");
}

```

```
    return 0;
}
```

代码：刷新CR3，刷新cache

```
int* address_1=0;
int* address_2=0;
int temp1;
int temp2;
__declspec(naked) void a()
{
    __asm
    {
        pushad;
        pushfd;
        push fs;
        mov ax, 0x30;
        mov fs, ax;

        mov eax, [address_1];
        shr eax, 9;
        and eax, 0x7FFFF8;
        add eax, 0xC0000000;
        mov ecx, [eax];
        mov ebx, [eax + 4];
        mov dword ptr ds:[0xC0000000], ecx;
        mov dword ptr ds:[0xC0000004], ebx;
        mov eax, dword ptr ds : [0] ;
        mov dword ptr ds : [temp1] , eax;

        //
        mov eax,cr3;
        mov cr3,eax;
        //

        mov eax, [address_2];
        shr eax, 9;
        and eax, 0x7FFFF8;
        add eax, 0xC0000000;
        mov ecx, [eax];
        mov ebx, [eax + 4];
        mov dword ptr ds:[0xC0000000], ecx;
        mov dword ptr ds:[0xC0000004], ebx;
        mov eax, dword ptr ds : [0] ;
        mov dword ptr ds : [temp2] , eax;

        pop fs;
        popfd;
        popad;

        retf;
    }
}
int main()
{
    address_1 = (int*)VirtualAlloc(0,0x1000,MEM_COMMIT,PAGE_EXECUTE_READWRITE);
```



```

    address_2 = (int*)VirtualAlloc(0,0x1000, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    *address_1 = 0x1;
    *address_2 = 0x2;
    temp1 = 3;
    temp2 = 3;
    printf("address_1:%08x\n", address_1);
    printf("address_2:%08x\n", address_2);
    printf("funciotn:%08x\n", a);
    system("pause");

    char bufcode[6] = { 0,0,0,0,0x48,0 };
    __asm
    {
        call fword ptr bufcode;
    }
    printf("temp1=%x temp2=%x \n",temp1,temp2);
    system("pause");
    return 0;
}

```

代码：增加int3，刷新cache（不实用）

```

//
    int 3;
//

```

页属性G位：

G位：Gloabl Page（第九位）

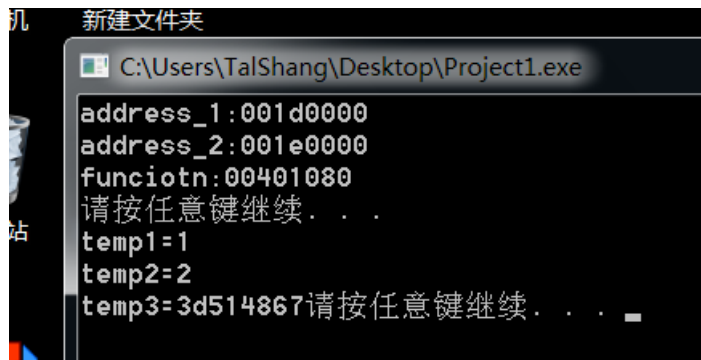
当G==1，则是全局页，进了TLB缓存后，即使刷新CR3，也不会刷新TLB。（注意是在CR4的标志位生效的情况下才有效。）

所以0环下的页，一般都是全局页。

```
or ptr,0x100
```

invlpg

把指定地址从TLB中移除



```
invlpg dword ptr ds:[0]; //0是线性地址
```

CR4(PGE位):

为1, 全局页生效。

为0, 全局页无效。

```
__emit 0x0f; //mov eax,cr4
__emit 0x20;
__emit 0xe0;
__asm
{
mov ebx,0x80;
not ebx;
and eax,ebx;//PGE位
}
__emit 0x0f;//mov cr4,eax
__emit 0x22;
__emit 0xe0;
```

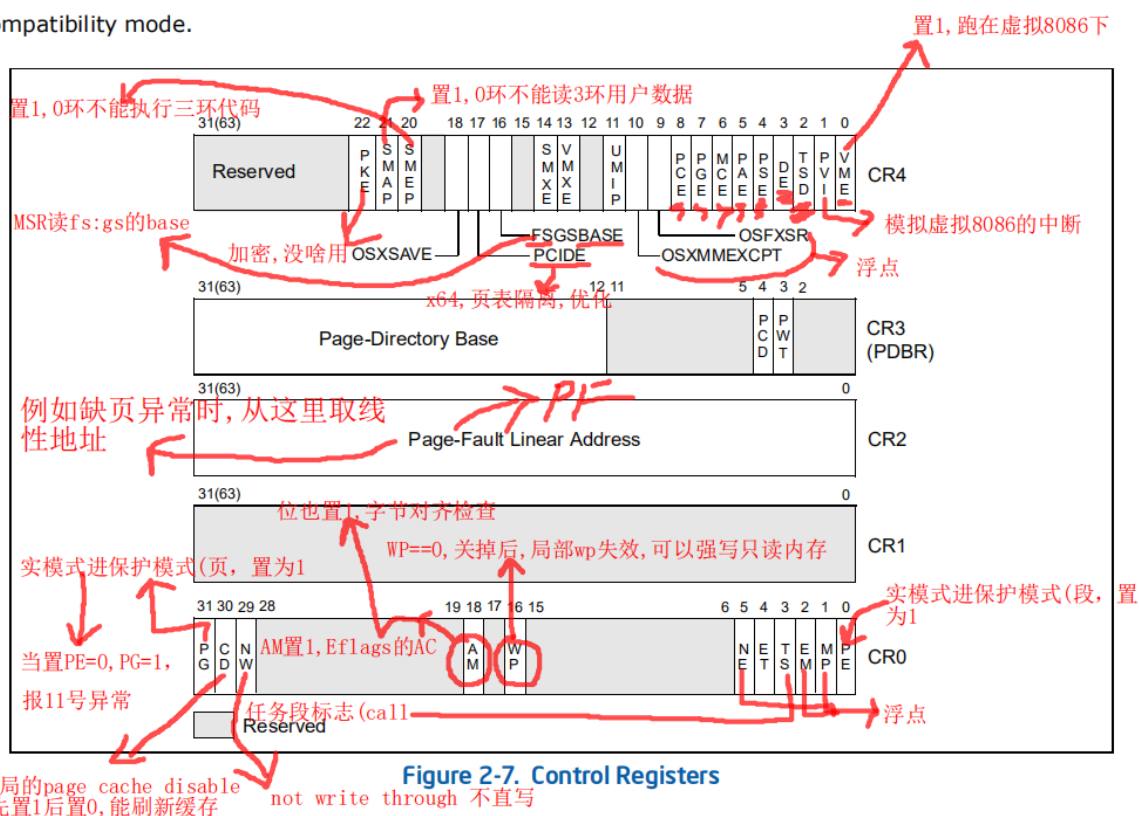
14.控制寄存器(控制操作系统功能)

CR0 ---CR8 一共9个

CR1 CR5 CR6 没有用到

x86 只用到了4个 CR0 CR2 CR3 CR4

compatibility mode.



CR2

当产生与页相关的异常时, 在进入中断后, 从CR2寄存器拿线性地址。

CR0

掌管全局

CR4

个性化设置。

RDTSC (CR4:TSD

TSD:置1，允许三环使用RDTSC。置0，允许0环使用RDTSC

获取时间(毫秒级) (8字节)

针对进程

CR4:DE

DR0-DR7，一共8个调试寄存器，DR4,DR5没有被用过。

当DE置1，可以使用DR4,DR5。

当DE为0，一般访问DR4,DR5，访问的是DR6，DR7。

CR4:PSE

page size extensions，置1，允许4MB大页的存在，局部对标PTE的PS位。

CR4:PAE

physical address extensions。置1，支持物理地址扩展，即是x86下的29912分页。

CR4:MCE

Machine-Check enable，置1，代表多了一个中断（必须挂上中断

置0，可挂可不挂

iiramybu							
进程 驱动模块 内核 内核钩子 应用层钩子 网络 注册表 文件 启动信息 系统杂项 电脑体检 配置 关于							
SSDT ShadowSSDT FSD 键盘 I8042prt 鼠标 Partmgr Disk Atapi Acpi Scsi 内核钩子 Object钩子 系统中断表							
Cpu...	序号	函数名称	段选择子	当前函数地址	Hook	原始函数地址	当前函数地址所在模块
0	00	Divide error	0x01	0x83E90FC0	-	0x83E90FC0	C:\Windows\system32\ntkrnlpa.exe
0	01	Debug	0x01	0x83E91150	-	0x83E91150	C:\Windows\system32\ntkrnlpa.exe
0	02	Not used	0x08	0x83F7B068	-	0x83E91240	C:\Windows\system32\ntkrnlpa.exe
0	03	Breakpoint	0x01	0x83E915C0	-	0x83E915C0	C:\Windows\system32\ntkrnlpa.exe
0	04	Overflow	0x01	0x83E91748	-	0x83E91748	C:\Windows\system32\ntkrnlpa.exe
0	05	Bounds check	0x01	0x83E918A8	-	0x83E918A8	C:\Windows\system32\ntkrnlpa.exe
0	06	Invalid opcode	0x01	0x83E91A1C	-	0x83E91A1C	C:\Windows\system32\ntkrnlpa.exe
0	07	Device not available	0x01	0x83E92018	-	0x83E92018	C:\Windows\system32\ntkrnlpa.exe
0	08	Double fault	0x0A	0x83F7B000	-	0x83E92357	C:\Windows\system32\ntkrnlpa.exe
0	09	Coprocessor segment over...	0x01	0x83E92478	-	0x83E92478	C:\Windows\system32\ntkrnlpa.exe
0	0A	Invalid TSS	0x01	0x83E9259C	-	0x83E9259C	C:\Windows\system32\ntkrnlpa.exe
0	0B	Segment not present	0x01	0x83E926DC	-	0x83E926DC	C:\Windows\system32\ntkrnlpa.exe
0	0C	Stack segment fault	0x01	0x83E9293C	-	0x83E9293C	C:\Windows\system32\ntkrnlpa.exe
0	0D	General protection	0x01	0x83E92C2C	-	0x83E92C2C	C:\Windows\system32\ntkrnlpa.exe
0	0E	Page Fault	0x01	0x83E932FC	-	0x83E932FC	C:\Windows\system32\ntkrnlpa.exe
0	0F	Reserved by Intel	0x01	0x83E936B0	-	0x83E936B0	C:\Windows\system32\ntkrnlpa.exe
0	10	Floating point error	0x01	0x83E937D4	-	0x83E937D4	C:\Windows\system32\ntkrnlpa.exe
0	11	Alignment check	0x01	0x83E93914	-	0x83E93914	C:\Windows\system32\ntkrnlpa.exe
0	12	Machine check	0x14	0x8640D1C0	idt hook	0x83E936B0	未知模块
0	13	SIMD floating point exception	0x01	0x83E93A80	-	0x83E936B0	C:\Windows\system32\ntkrnlpa.exe
0	14	Reserved by Intel	0x01	0x83E936B0	-	0x83E936B0	C:\Windows\system32\ntkrnlpa.exe
0	15	Reserved by Intel	0x01	0x83E936B0	-	0x83E936B0	C:\Windows\system32\ntkrnlpa.exe
0	16	Reserved by Intel	0x01	0x83E936B0	-	0x83E936B0	C:\Windows\system32\ntkrnlpa.exe
0	17	Reserved by Intel	0x01	0x83E936B0	-	0x83E936B0	C:\Windows\system32\ntkrnlpa.exe

CR4:PCE

Performance-Monitoring Counter Enable 性能监控？

CR4:VMXE(VT位 (-1环

使用VT, 必须置1。

CR4:SMXE(上帝位(-2环

15.通过分页知识 做一个 读其他进程内存的数据的函数

