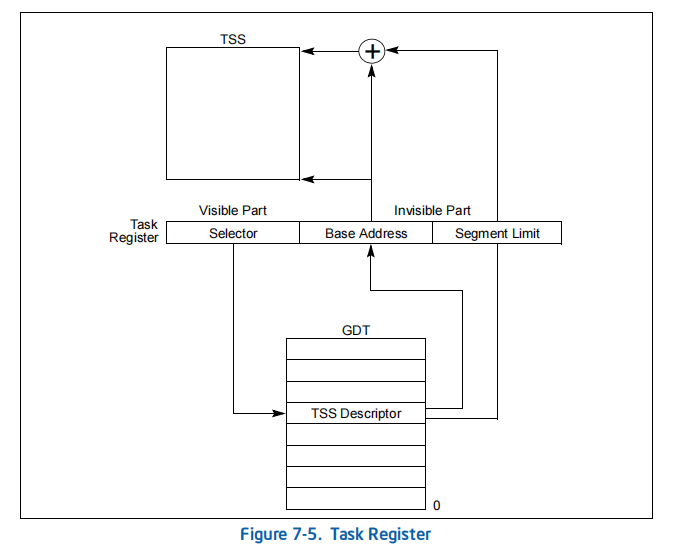
# 课堂笔记

### TSS(task state segment),TR(task register),TSS描述符之间的关系

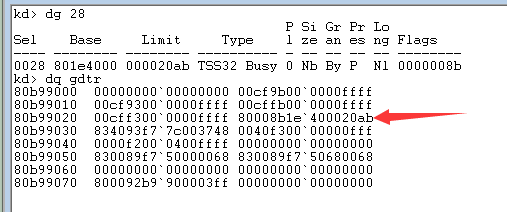
任务是CPU提供的切换上下文的机制，和OS实现的线程类似，OS为了尽量不依赖CPU特性而实现了线程机制。下面是一张从白皮书截下来的图片，描述了本节课涉及到的几个“东西”之间的关系（TSS表，TR寄存器，TSS描述符）



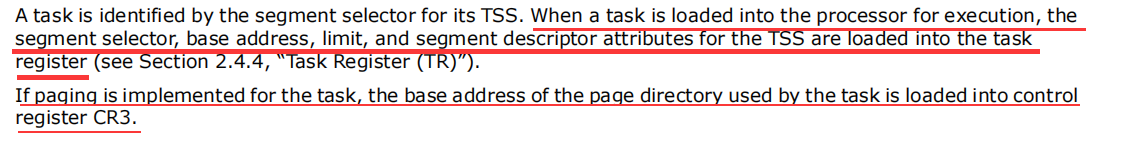
这张图的意思是，TR寄存器的选择子用来在GDT表里找到TSS描述符，描述符里的base address和limit可以找到TSS表。

### TR寄存器

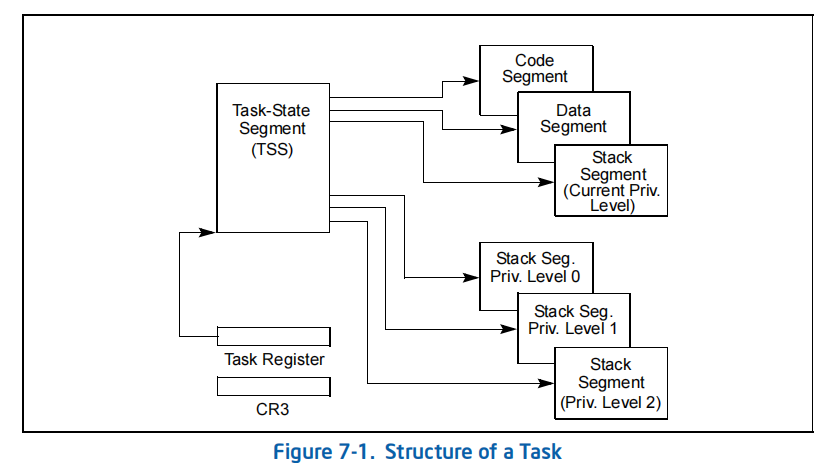
TR寄存器（task register）由三部分组成，选择子selector用来索引GDT，比如0x28就可以找到一个TSS描述符：



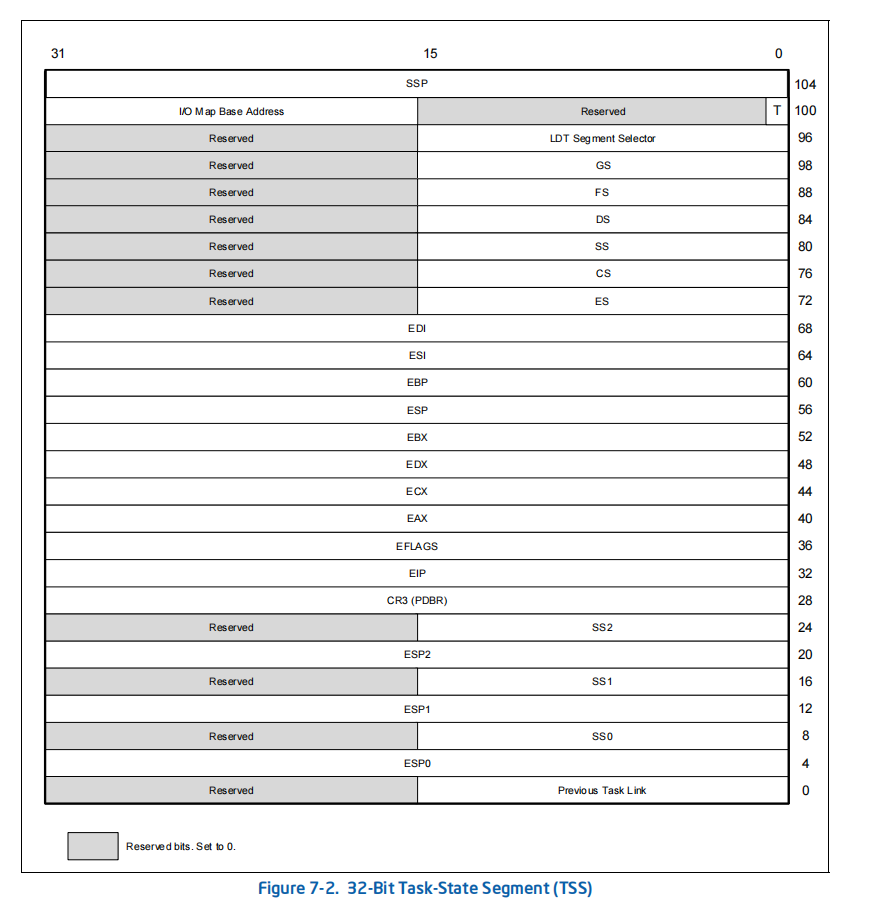
TR寄存器还存储了TSS的基址(base addres)和大小(limit)，编程中可以用ltr,str指令对TR寄存器进行读写。



书上说当任务执行的时候，TR寄存器会存储这一堆东西。第二句话说，如果启用了分页机制，那任务就会使用CR3给定的页目录基址。

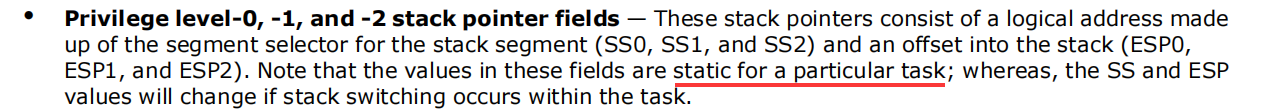


### TSS表



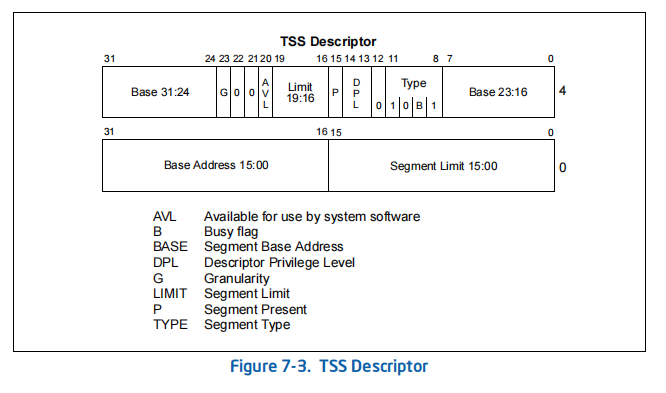
TSS（任务状态段）是一块内存，用来一次性替换一堆寄存器。

这里的SS0，SS1, ESP0,ESP1对于特定一个任务是静态的，而SS,ESP在任务切换的时候是会变的：



课上的实验我们自己构造一个TSS，和一个GDT里的TSS描述符，然后通过call far进去，iretd返回。

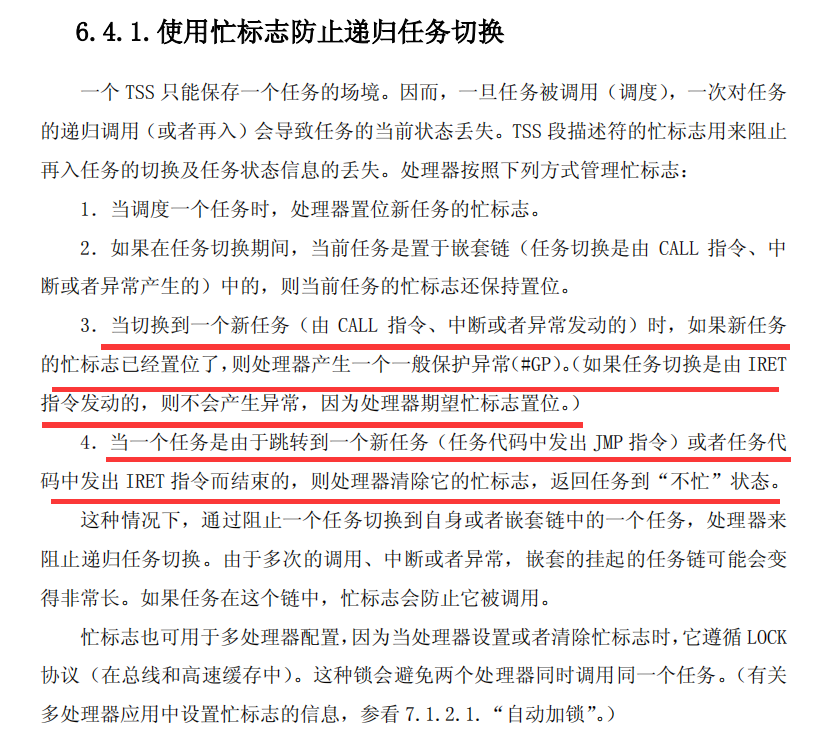
### TSS描述符

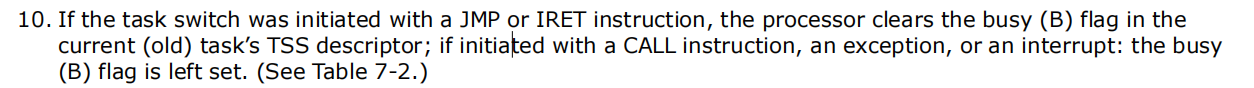


**tss描述符只会存储在GDT表中**，可以看到S=0，type=9或11。

Base是TSS表的偏移，limit是大小（单位字节）.

Busy 位的用途如下图所说，是用来防止递归任务切换的：

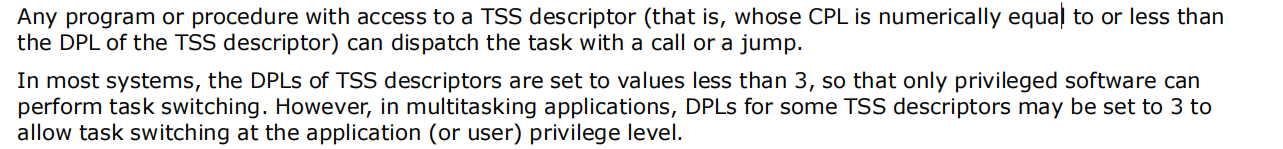




补充白皮书原文。

**就是说call/异常/中断发起调度新任务时，新任务的busy会由0变1（JMP/CALL切换到busy=1的任务会报#GP异常）**

**Jmp/iretd切换任务时，当前任务的busy从1变0.**

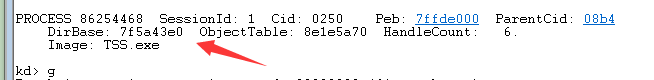


上图这段话说的是，CPL必须小于等于TSS.DPL。如果TSS.DPL==3，我们就可以在应用层R3去调用任务门。

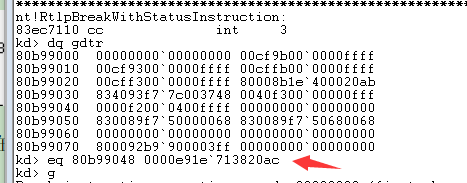
### 课上实验复现

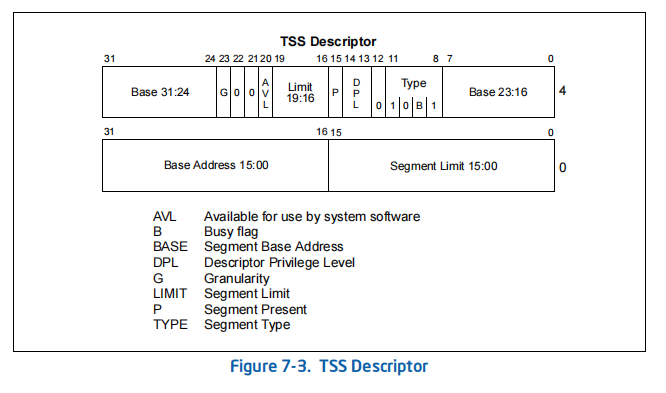
|  |
| --- |
| #include "stdafx.h"  #include <Windows.h>  struct \_KiIoAccessMap  {  UCHAR DirectionMap[32]; //0x0  UCHAR IoMap[8196]; //0x20  };  typedef struct \_KTSS  {  USHORT Backlink; //0x0  USHORT Reserved0; //0x2  ULONG Esp0; //0x4  USHORT Ss0; //0x8  USHORT Reserved1; //0xa  ULONG NotUsed1[4]; //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;  KTSS tss={0};  char bufEsp0[0x2000]={0};  char bufEsp3[0x2000]={0};  void \_\_declspec(naked) test()  {  \_\_asm  {  int 3;  pushfd;  pop eax;  or eax,0x4000; // eflags.NT = 1  push eax;  popfd;  iretd;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  memset(bufEsp0,0,0x2000);  memset(bufEsp3,0,0x2000);  tss.Esp0 = (ULONG)bufEsp0 + 0x1FF0;  tss.Esp = (ULONG)bufEsp3 + 0x1FF0;  tss.Ss0 = 0x10;  tss.Ss = 0x10;  tss.Cs = 0x8;  tss.Ds = 0x23;  tss.Es = 0x23;  tss.Fs = 0x30;  tss.EFlags = 2;  tss.Eip = (ULONG)test;  tss.IoMapBase = 0x20ac;  printf("%x\r\n",sizeof(tss));  printf("请输入的你的CR3:");  DWORD dwCr3 = 0;  scanf("%x",&dwCr3);  tss.CR3 = dwCr3;  printf("%x\r\n",&tss);  system("pause");    char bufcode[]={0,0,0,0,0x48,0};    \_\_asm  {  call fword ptr bufcode  //int 32;  }  system("pause");  return 0;  } |

程序跑起来后，memset挂页并初始化，然后设置寄存器上下文，eip是要去的位置，CR3要手动填写，!process 0 0 查看当前进程的CR3（dirbase）：



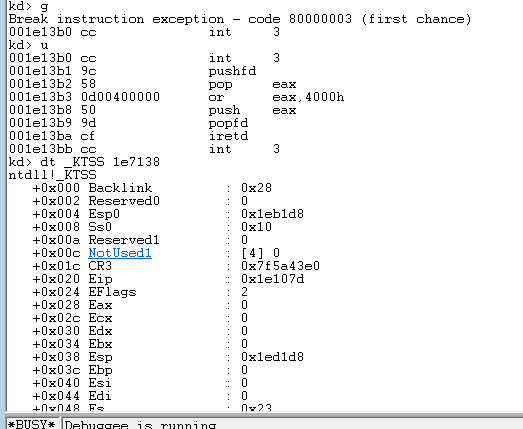
然后打印TSS的地址，这时候需要断下，在GDT 48处填入TSS描述符：





粒度G=0，limit=0x020ac，DPL=3，BUSY=0.

然后回车，跳过system(“pause”)，call far之后在windbg断下：

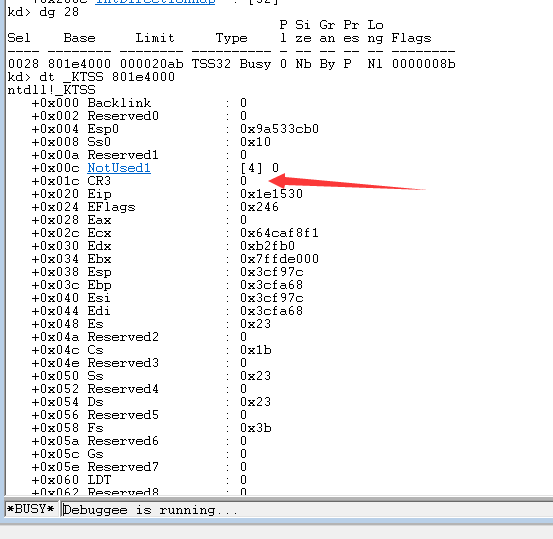


u看看断的位置，是正确的，进入了裸函数test，dt看看TSS，也没问题。

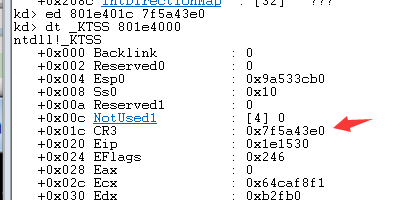
**因为int3 会清空NT位，iretd返回前，需要把NT置1，这样CPU就知道这是从任务切换中返回，否则CPU以为这是中断返回。**

在win7 32位做这个实验，进入裸函数后断下，iretd返回之前，记得把previous task link （0x28）的CR3写值，因为这个CR3被清零了，是win7的bug。

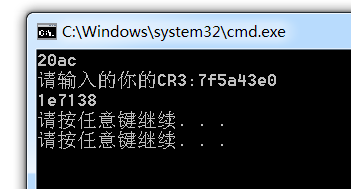




Previous task link CR3写值：

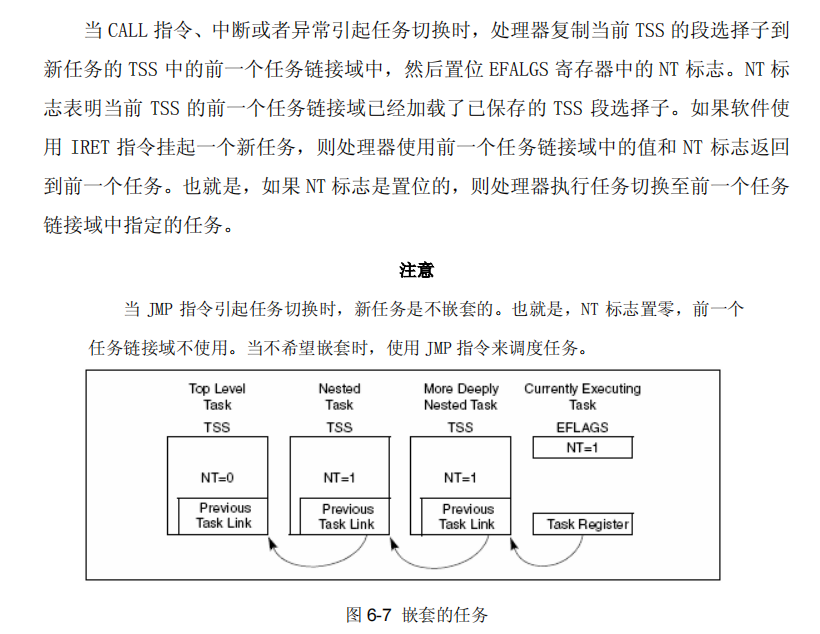


然后g放行，正确返回：

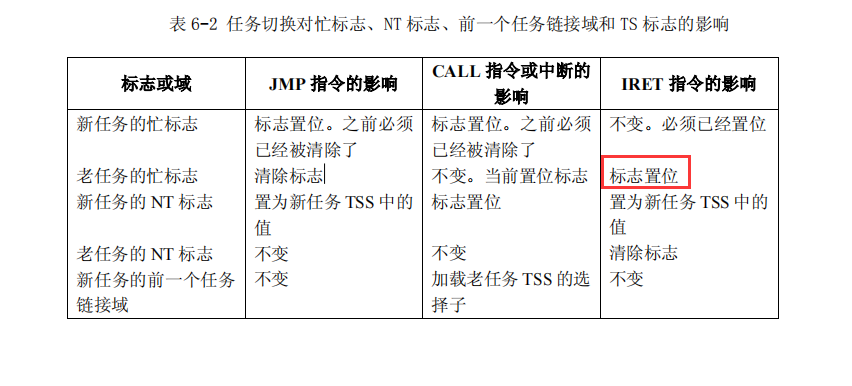


# 作业

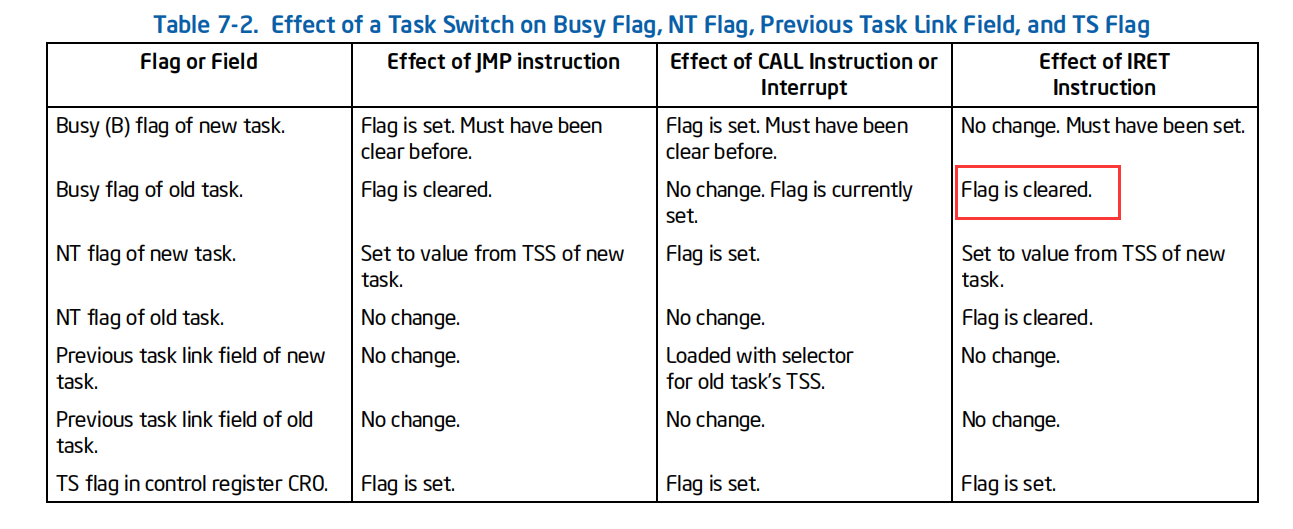
### 请阅读 IA-32卷3：系统编程指南 161页以及之后任务相关的内容



Call far 和 jmp far 进行显式任务切换时，一个区别是CALL会把eflags.NT置1，表示新任务是嵌套的，并且会把旧任务的TSS段选择子写到新TSS的previous task link中。



**注意中文版“老任务的忙标志 IRET指令的影响”写错了，应该是清零，以下图的白皮书原文为准。**



### 1.通过JMP 跳转任务段，尝试用jmp返回，iretd 返回

看表说话，Call和jmp任务切换的区别是：

CALL会把NT置1，JMP是从新任务TSS里取值；

JMP会把老任务的busy置0，call不改；

JMP切换不会改新任务TSS的previous task link。如果想iretd，则需要自己填这个值。

并且，JMP和CALL都需要满足新任务的段描述符busy位等于0.

这几点搞明白之后，别忘了win7任务切换会把 previous.CR3 清零，得手动写值。

###### JMP跳转+jmp返回

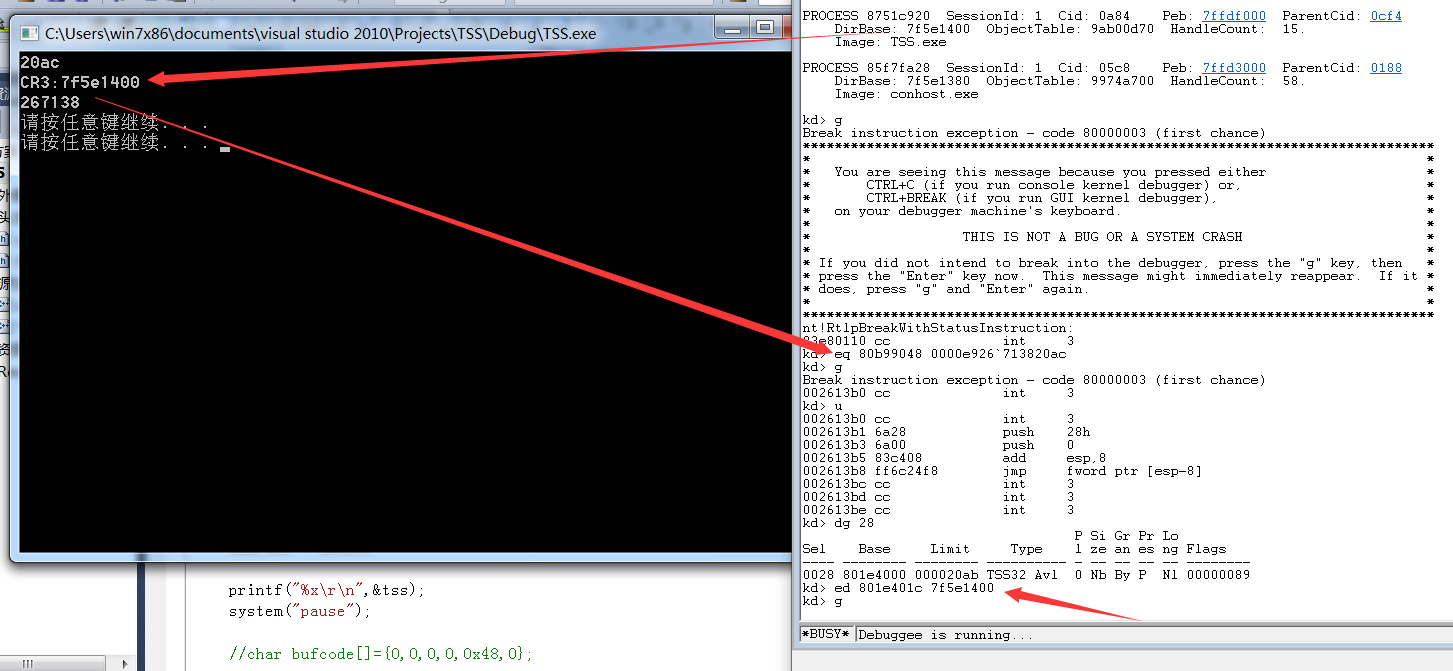
第一次跳转，新任务段的busy必须是0，JMP之后，老任务段的busy清零，待会第二次JMP返回时就不用改了。

由于两次都是JMP，对NT没有要求。

Previous task link 也不用动，因为jmp不会影响这个值。

修改代码如下：

|  |
| --- |
| #include "stdafx.h"  #include <Windows.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;  KTSS tss={0};  char bufEsp0[0x2000]={0};  char bufEsp3[0x2000]={0};  void \_\_declspec(naked) test()  {  \_\_asm  {  int 3; // int will clear NT  push 0x28;  push 0;  add esp,8;  jmp fword ptr [esp-8];  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  memset(bufEsp0,0,0x2000);  memset(bufEsp3,0,0x2000);  tss.Esp0 = (ULONG)bufEsp0 + 0x1FF0;  tss.Esp = (ULONG)bufEsp3 + 0x1FF0;  tss.Ss0 = 0x10;  tss.Ss = 0x10;  tss.Cs = 0x8;  tss.Ds = 0x23;  tss.Es = 0x23;  tss.Fs = 0x30;  tss.EFlags = 2;  tss.Eip = (ULONG)test;  tss.IoMapBase = 0x20ac;  printf("%x\r\n",sizeof(tss));  printf("CR3:");  DWORD dwCr3 = 0;  scanf("%x",&dwCr3);  tss.CR3 = dwCr3;  printf("%x\r\n",&tss);  system("pause");    //char bufcode[]={0,0,0,0,0x48,0};    \_\_asm  {  push 0x48;  push 0;  add esp,8;  jmp fword ptr [esp-8];  }  system("pause");  return 0;  } |



（别忘了previous task link CR3

###### Jmp跳转+iretd返回

做完上一个实验后，重启一下电脑，免得刚才改乱了的各种值影响本次实验。

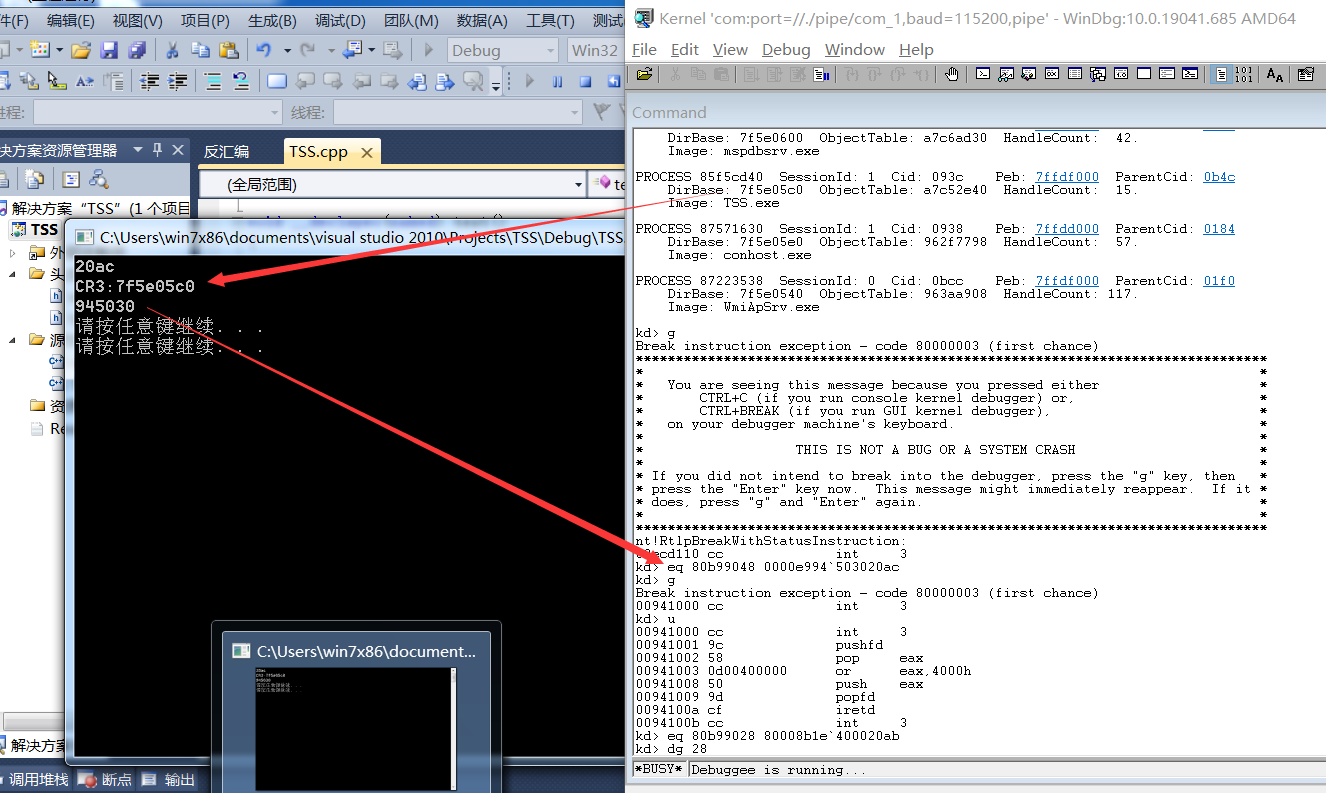
JMP会把老任务的busy置0，而iret要求新任务的busy必须是1，所以iret之前，别忘了给GDT 28的busy置1

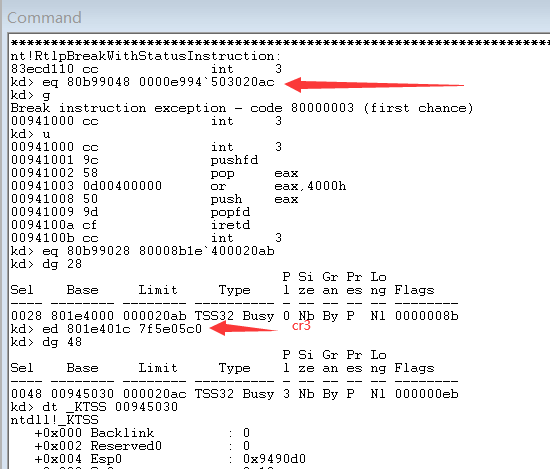
然后就是NT位必须改成1，因为JMP过来的，NT不会变，而且int 3也会置0 NT位。

最后，别忘了 previous task link 也得自己填。

代码如下：

|  |
| --- |
| #include "stdafx.h"  #include <Windows.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;  KTSS tss={0};  char bufEsp0[0x2000]={0};  char bufEsp3[0x2000]={0};  void \_\_declspec(naked) test()  {  \_\_asm  {  int 3; // int will clear NT  pushfd;  pop eax;  or eax,0x4000; // eflags.NT = 1  push eax;  popfd;  iretd;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  memset(bufEsp0,0,0x2000);  memset(bufEsp3,0,0x2000);  tss.Esp0 = (ULONG)bufEsp0 + 0x1FF0;  tss.Esp = (ULONG)bufEsp3 + 0x1FF0;  tss.Ss0 = 0x10;  tss.Ss = 0x10;  tss.Cs = 0x8;  tss.Ds = 0x23;  tss.Es = 0x23;  tss.Fs = 0x30;  tss.EFlags = 2;  tss.Eip = (ULONG)test;  tss.IoMapBase = 0x20ac;  printf("%x\r\n",sizeof(tss));  printf("CR3:");  DWORD dwCr3 = 0;  scanf("%x",&dwCr3);  tss.CR3 = dwCr3;  printf("%x\r\n",&tss);  system("pause");    //char bufcode[]={0,0,0,0,0x48,0};    \_\_asm  {  push 0x48;  push 0;  add esp,8;  jmp fword ptr [esp-8];  }  system("pause");  return 0;  } |



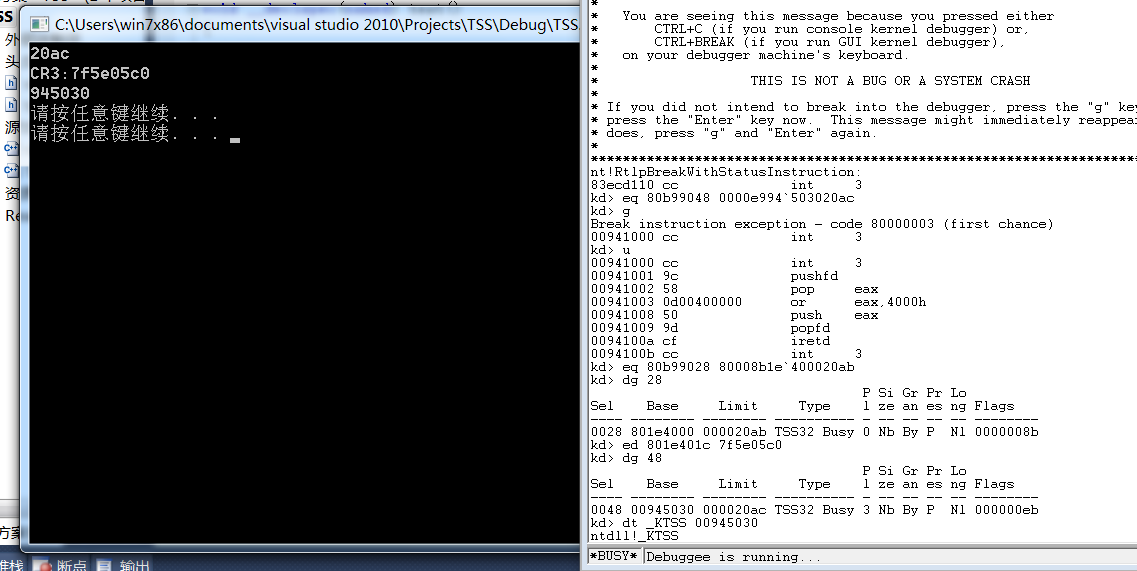


写回CR3



填充 previous task link

这样就可以正常iretd返回了



**总结：**

**jmp far进去断下之后，iretd返回之前需要做的步骤：nt置1，previous task link 自己填28，28的TSS.CR3记得写值，28的描述符busy记得置1**

### 2.通过任务段提权到1环，也就是成功之后CS段寄存器后2位 为1

**这个实验裸函数里不能int3，不然会gg（我调了一晚上最后问火哥才知道的）**

那不能断下的话，就没法修复previous task link 的CR3了呀，只能在获取CR3的时候，顺便把28的CR3偏移记下来，在裸函数里修复了。

-----------------------分割线 -----------------

首先在60处构建一个R1代码段,68处构建一个R1栈段：

eq 80b99060 00cfbb00`0000ffff

eq 80b99068 00cfb300`0000ffff

在课堂实验代码的基础上修改，添加了一个ESP1缓冲区，补上了SS1，然后初始化的时候，让CS=0x61(RPL==1), Ss1=0x69,Fs=0x69,esp1指向那块内存。

|  |
| --- |
| #include "stdafx.h"  #include <Windows.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 Esp1;  USHORT Ss1;  USHORT ReservedSs1;  ULONG Esp2;  USHORT Ss2;  USHORT ReservedSs2;  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;  KTSS tss={0};  char bufEsp1[0x2000]={0};  char bufEsp3[0x2000]={0};  ULONG PreviousCR3Offset = 0;  WORD wCS1 = 0;  void \_\_declspec(naked) test()  {  \*(DWORD\*)PreviousCR3Offset = tss.CR3;  \_\_asm  {  //int 3;  mov wCS1,cs;  pushfd;  pop eax;  or eax,0x4000; // eflags.NT = 1  push eax;  popfd;  iretd;  }  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  memset(bufEsp1,0,0x2000);  memset(bufEsp3,0,0x2000);  PreviousCR3Offset = 0;  wCS1 = 0;  //tss.Esp0 = (ULONG)bufEsp0 + 0x1FF0;  tss.Esp = (ULONG)bufEsp3 + 0x1FF0;  tss.Esp1 = (ULONG)bufEsp1 + 0x1FF0; // ESP1  tss.Ss1 = 0x69; // Ss1  tss.Ss = 0x69; // Ss1  tss.Cs = 0x61; // Cs1  tss.Ds = 0x23;  tss.Es = 0x23;  tss.Fs = 0x69; // Fs1  tss.EFlags = 2;  tss.Eip = (ULONG)test;  tss.IoMapBase = 0x20ac;  printf("%x\r\n",sizeof(tss));  printf("CR3, Previous CR3 offset: ");  DWORD dwCr3 = 0;  scanf("%x %x",&dwCr3, &PreviousCR3Offset);  tss.CR3 = dwCr3;  printf("%x\r\n",&tss);  system("pause");    char bufcode[]={0,0,0,0,0x48,0};    \_\_asm  {  call fword ptr bufcode  //int 32;  }  printf("CS1 = %x\n", wCS1);  system("pause");  return 0;  } |

