# Sysenter的作用

上次课分析了系统调用R3部分，最后来到sysenter指令，调用这个指令，就进R0了。之前学保护模式说过，进R0需要CS,SS,EIP和一个新的ESP。Sysenter指令主要功能就是一次性替换了这四个寄存器，值是从MSR寄存器取的。微软有自己的线程机制，所以sysenter替换的esp没什么卵用。

sysenter干了几件事情：

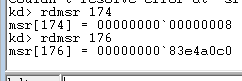
1． 将 SYSENTER\_CS\_MSR 的值装载到 cs 寄存器 174

2． 将 SYSENTER\_EIP\_MSR 的值装载到 eip 寄存器 176

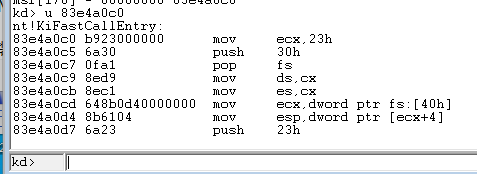
3． 将 SYSENTER\_CS\_MSR 的值加 8（Ring0 的堆栈段描述符）装载到 ss 寄存器。

1. 将 SYSENTER\_ESP\_MSR 的值装载到 esp 寄存器 175

这个寄存器可以在windbg里用rdmsr查看：



ss是cs+8得到，esp没什么卵用。可以看到，进R0后，会去到 83e4a0c0，u一下看看：



这就是系统调用R0部分的入口 KiFastCallEntry，这个函数待会详细分析一下，现在继续看看sysenter还干了些什么。

5． 将特权级切换到 Ring0

6． 如果 EFLAGS 寄存器的 VM 标志被置位，则清除该标志

1. 开始执行指定的 Ring0 代码

其实没啥了，就是清除VM位（虚拟8086）

然后就是去执行这个 KiFastCallEntry 啦。

# KiFastCallEntry分析

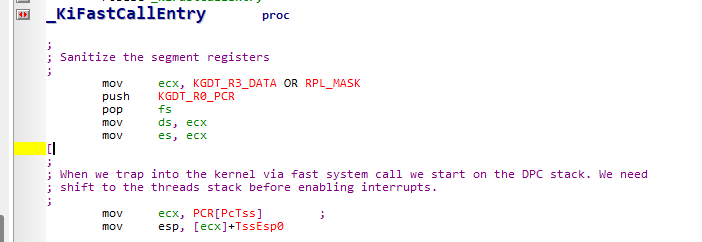
这个函数今天分析不完，只分析到调用系统函数之前。总体上说干了三件事情：

1.保存现场

2.复制参数

3.CALL

接下来开始分析。这个函数其实可以直接看WRK，人家微软也是用汇编写的。因为直接操作了esp.

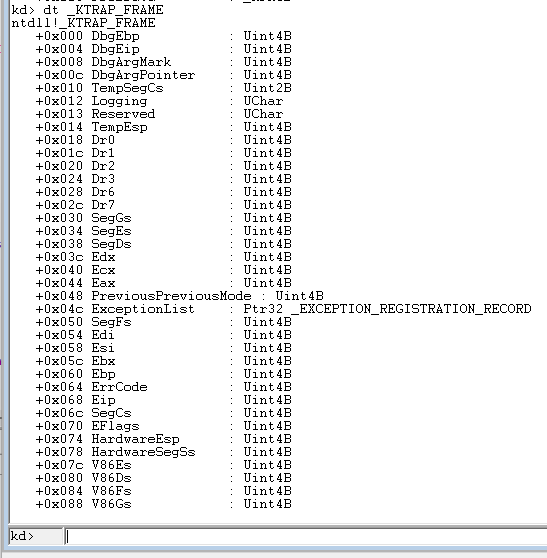


看汇编还是IDA好，高亮比较好用。

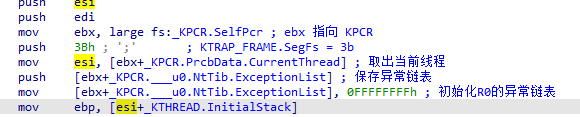
### 保存现场



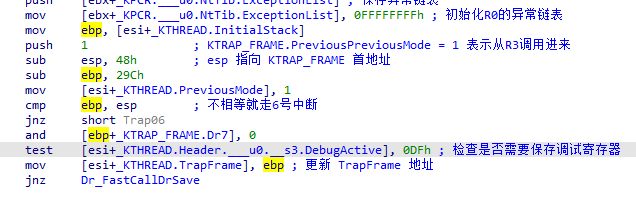
使用一个叫 KTRAP\_FRAME 的结构体保存R3的寄存器。



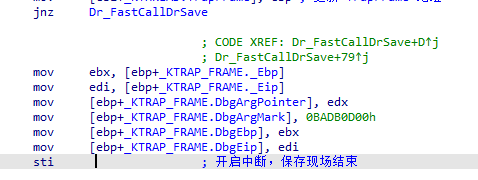
接下来是保存R3异常链表和初始化R0异常链表



然后检查一下堆栈，检查是否保存调试寄存器

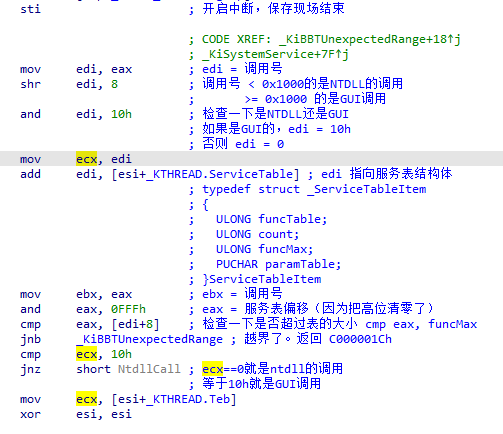


完成最后的保存现场工作，然后恢复中断：

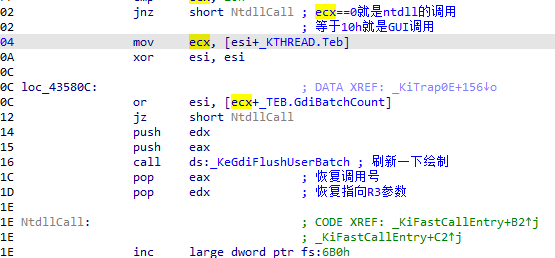


### 获取服务表，判断调用类型

开中断标志着保存现场结束，接下来，判断了一下R3通过eax传进来的调用号，先看看有没有超过系统服务表的大小（如果超了就调用\_KiBBTUnexpectedRange给R3返回错误C000001Ch），然后看看到底是ntdll的调用，还是GUI类型的调用，如果是GUI类型，则需要刷新一下绘制：



如果是GUI调用，就刷新一下绘制：



接下来还是要和NTDLL一样，走到同一个位置。

### 复制参数，调用函数

