链滴

Win32 汇编学习 (1)：基本概念

作者：Akkuman
原文链接：https://ld246.com/article/1518195164706
来源网站：链滴

背景知识

Windows 把每一个 Win32 应用程序放到分开的虚拟地址空间中去运行，也就是说每一个应用程序拥有其相互独立的 4GB 地址空间，当然这倒不是说它们都拥有 4GB 的物理地址空间，而只是说能够 4GB 的范围内寻址。操作系统将会在应用程序运行时完成 4GB 的虚拟地址和物理内存地址间的转换这就要求编写应用程序时必须格守 Windows 的规范，否则极易引起内存的保护模式错误。而过去的 in16 内存模式下，所有的应用程序都运行于同一个 4GB 地址空间，它们可以彼此"看"到别的程序的容，这极易导致一个应用程序破坏另一个应用程序甚至是操作系统的数据或代码。

<!--more-->

和 16 位 Windows 下的把代码分成 DATA，CODE 等段的内存模式不同，WIN32 只有一种内存模，即 FLAT 模式，意思是"平坦"的内存模式，再没有 64K 的段大小限制，所有的 WIN32 的应用程序行在一个连续、平坦、巨大的 4GB 的空间中。这同时也意味着您无须和段寄存器打交道，您可以用意的段寄存器寻址任意的地址空间，这对于程序员来说是非常方便的，这也使得用32位汇编语言和用语言一样方便。 在Win32下编程，有许多重要的规则需要遵守。有一条很重要的是：Windows 在内频繁使用 ESI，EDI，EBP，EBX 寄存器，而且并不去检测这些寄存器的值是否被更改，这样当您要使这些寄存器时必须先保存它们的值，待用完后再恢复它们，一个最显著的应用例子就是 Windows 的 allBack 函数中。

内容

一般的Win32汇编都有下面的程序段，这是一个Win32汇编编程的基础框架，若您现在还不知道这些令的确切意义的话，没关系， 随后我就会给大家详细解释。

```
.386
.MODEL Flat, STDCALL
.DATA
    <Your initialized data>
    ......
.DATA?
   <Your uninitialized data>
   ......
.CONST
   <Your constants>
   ......
.CODE
   <label>
    <Your code>
   .....
    end <label>
```

这就是一般Win32汇编编程的基础框架，其中各个关键词的解释说明如下：

.386

这是一个汇编语言伪指令，他告诉编译器我们的程序是使用80386指令集编写的。您还可以使用 .486 .586， 但最安全的还是使用.386。对于每一种CPU有两套几乎功能相同伪指令： .386/.386P、 486/.86P、 586/.586P。 带P的指令标明您的程序中可以用特权级指令。特权级指令是保留给操作系统的如虚拟设备驱动程序。在大多数时间，您的程序都无须运行在RING0层，故用不带后缀P的伪指令已够了。

.MODEL FLAT，STDCALL

.MODEL 是用来指定内存模式的伪指令，在Win32下，只有一种内存模型，那就是FLAT。 STDCALL告诉编译器参数的传递约定。参数的传递约定是指参数传达时的顺序(从左到右或从右到左)和由谁恢堆栈指针(调用者或被调用者)。在Win16下有两种约定：C 和 PASCAL。C 约定规定参数传递顺序是右到左，即最右边的参数最先压栈，由调用者恢复堆栈指针。

例如：为调用函数 foo (int first_param， int second_param， int third_param)； 按C约定的汇代码应该是这样的：

```
push [third_param]
push [second_param]
push [first_param]
call foo
add esp， 3 * 4 ;调用者自己恢复堆栈指针
```

PASCAL约定和C约定正好相反，它规定参数是从左向右传递，由被调用者恢复堆栈。Win16采用了PSCAL约定， 因为PASCAL约定产生的代码量要小。当不知道参数的个数时，C约定特别有用。如在函数sprintf () 中， wsprintf预先并不知道要传递几个参数，所以它不知道如何恢复堆栈。STDCALL是C定和PASCAL约定的混合体，它规定参数的传递是从右到左，恢复堆栈的工作交由被调用者。Win32用STDCALL约定，但除了一个特例，即：wsprintf。

```
.DATA
.DATA?
.CONST
.CODE
```

上面的四个伪指令是"分段"(SECTION)伪指令。我们上面刚讲过Win32下没有"段"(SEGMENT)的概念但是您可以把您的程序分成不同的"分段"， 一个"分段"的开始即是上一个"分段"的结束。WIN32中只两种性质的"分段"：DATA和CODE。

其中DATA"分段"又分为三种：

- .DATA 其中包括已初始化的数据。
- .DATA? 其中包括未初始化的数据。比如有时您仅想预先分配一些内存但并不想指定初始值。使用初始化的数据的优点是它不占据可执行文件的大小，如：若您要在 .DATA? 段中分配10,000字节的空，您的可执行文件的大小无须增加10,000字节，而仅仅是要告诉编译器在装载可执行文件时分配所需节。
- .CONST 其中包括常量定义。这些常量在程序运行过程中是不能更改的。 应用程序并不需要以上所的三个"分段"， 可以根据需要进行定义。
- .CODE 这是代码"分段"。

实际上，分段并不是象在 Dos 下一样，为不同的段分别指出不同的段寄存器，因为 Windows 下只一个 4GB 的段，Windows 程序中的分段表现在当程序装载时，赋予不同的分段不同的属性，比如说你的程序加载时，对于 Ring3 程序来说，.code 段是不可写的，而 .data 段是可写的，如果你尝试象 Dos 下一样写自己的代码部分，你会得到一个蓝屏错误

```
<label>
end <label>
```

是用来唯一标识您的代码范围的标签， 两个标签必须相同，应用程序的所有可执行代码必须在两个标之间。