2021/01/12_32位汇编_第3课_在汇编中使用C库函数和资源、RadASm的使用

笔记本： 32位汇编
创建时间： 2021/1/12 星期二 10:54
作者： ileemi

在微软汇编编译器中，ret 是伪指令
调用约定：
C：调用者平栈 ret --> retn（需要自己手动添加平衡堆栈的字节大小）
stdcall：翻译成 retn "平衡堆栈的字节大小"

尽量不使用 eax 寄存器传递参数
leave（出口）：等价 mov ebp, esp; pop ebp; 自动平衡堆栈
enter（入口）：等价 push ebp; mov ebp, esp

过程函数（带参宏同理）内申请局部变量，在函数开头会自动添加两行代码，以固定局部变量的地址。
在标号中申请局部变量需要手动添加两行代码（push ebp;mov ebp, esp）。

ret：伪指令（自动平衡堆栈）
retn：汇编指令（手动平衡堆栈，后跟平衡堆栈的大小）

调用约定：
C：调用者平栈 ret --> retn
stdcall：翻译成 retn "平衡堆栈的字节大小"
初始化局部变量可以使用stos
type：类型
sizeof：大小

在汇编中使用C库函数

masm32中有对应的C库：
C库函数分版本：最少四个（静态、动态、ASCII、Unicode、Debug、Release、多/单线程）
静态库：**libc** 开头，2019以后库的名称有所改变，注意库名称中的关键字 "C、R、T"（表示C运行库）。
动态库："**msvc**" 开头，具体看后缀名进行分析
例如：

libc.lib：Release
libcd.lib：Debug
libcmt.lib：Release多线程

在汇编中使用C库函数时要确定使用的版本，多使用 Release 版本的库。

动态使用C库函数

> 需要包含 msvcrt.lib，同时将 msvcrt.inc 进行声明。（C库的调用约定为C调用约定）

在使用C库函数的时候，可以到 msvcrt.inc 文件中查看API的声明，看其是否有改动，例如：
strcmp 在 msvcrt.inc 文件中的声明为：crt_strcmp，所以在汇编中使用库函数 strcmp 就需要使用 crt_strcmp、crt_printf等（通过ollydbg可以查看编译后的可执行程序加载了msvcrt.dll）。

> 使用 invoke 将局部变量的地址当作参数传递的时候，需要在局部变量的地址前添加 addr（**offset 和 addr 都可以获取全局变量的地址，但是获取局部变量的地址只能使用 addr**）。

通过 ollydbg 调试程序的时候，可以查看C库函数的实现代码（运行到库函数的时候跟进去即可）。

crt_printf 在 msvcrt.inc 中的声明：

```
externdef _imp__printf:PTR c_msvcrt
crt_printf equ <_imp__printf>
```

解析：
externdef：伪指令
c_msvcrt typedef PROTO **C :VARARG**
_imp__printf:PTR **c_msvcrt**：没有定义参数的个数

> 动态使用C库函数 printf 时，参数个数没有上线，只有C调用约定才可以不定参数。

自己的函数设置为不定参数，使用call，将参数push到堆栈中：

```
MY_ADD proc
  ret
MY_ADD endp
.code
START
  push 2
  push 3
  push 5
  call MY_ADD
```

```
  add esp, 12 ; 平衡堆栈
end START
```

静态使用C库函数

> 需要包含 libc.lib，同时将 libc.inc 进行声明（masm32生成的 libc.lib 文件如果有问题，就去 VC++6.0 或者 VS中拷贝一个 libc.lib 进行使用）。
> **使用的C库函数需要向前声明。**

```
includelib libc.lib
.const
MY_MSG "Hello World", 0

strlen proto c :dword ; 向前声明
printf proto c :VARAGE ; VARAGE -- 不指定参数的个数

.code
START
  local buf1[10]:byte
  incoke strlen, add MY_MSG ; 静态使用C库函数
  ;incoke printf, add MY_MSG ; 链接失败，静态使用printf时，其会依赖于main函数的实现，没有初始化代码
end START
```

通过 ollydbg 调试可执行程序可以发现 libc.lib 中的 strlen 函数的代码被静态链接到主模块中，

> 注意：静态使用printf时，其会依赖于main函数的实现，没有初始化代码（解决办法：将初始化代码链接到程序中），动态使用printf时，.dll中会自动调用一些初始化的代码。

ret（retn） 会从栈顶获取返回地址交给 edi，在retn之前需要调用 ExitProcess（kernrl32.lib）直接退出程序。

int21H为特权指令时，在保护模式下只有操作系统可以调用

在汇编中使用资源快速创建对话框

在汇编中使用资源快速创建对话框可以提供程序的开发时间效率。

> 使用 VC++6.0 创建一个资源脚本（.rc文件）。使用VS也可以但是需要注意 link.exe（链接器） 以及 rc.exe（资源编译器）的版本（**向下兼容原则**）。
> 操作步骤：
>
> - 打开VC --> File --> NEW --> "Resource Script"；

> - 编译：使用 ".rc" 资源编译器 编译 ".rc" 文件，编译成功会生成 ".h（保存对话框、控件的ID等信息）" ".RES（以二进制形式保存资源）" 文件，示例：rc xxx.rc --> xxx.RES；
> - 链接：将 "xxx.RES" 文件中保存资源的二进制信息保存到可执行文件中。

资源使用方法示例：

- 方法1：使用对话框可以将 ".h" 文件中对应的ID号 拿到程序中使用，使用示例："**IDD_DIALOG1 equ 101**"；
- 方法2：可以将 ".h" 文件进行修改成 ".inc"，文件格式需要满足汇编语言的语法即可（比较麻烦）。

调用对话框使用API：**DialogBox**（invoke即可，注意API所在的.lib文件，以及库的版本关系到API的版本）
DialogBox 高版本 DialogBoxParam
关闭对话框，在对话框过程函数中调用：EndDialog即可

> VS 编译器支持将 可执行文件中的资源提取出来，可以对其界面进行修改。

RadASM

> 基于 masm32

代码注入

> dll 注入的缺点（Loadlibrary）：不够隐秘，注入成功后对方进程多了一个dll，通过遍历模块列表很方便检测到注入的dll。为了增加隐秘性，可以使用汇编将需要注入到程序中的代码以二进制的方式进行注入。

不注入dll，直接注入代码（隐秘性更高，高级语言实现不了（原因：控制不了汇编的二进制））。

创建远程线程代码注入流程：

- 为目标进程申请内存
- 写入注入的代码
- 创建远程线程（CreateRemoteThread）

注入代码不推荐使用伪指令编写（容易产生额外的代码），使用 "call" 手动传递参数。

程序编译成功后，可以通过调试器（ollydbg）进行调试，快速定位目标程序被注入代码的地址的方法：

1. 目标程序正常运行，在注入的代码中编写让目标程序崩溃的代码（直接运行 -- 崩溃 -- 调试）：
   mov eax, 0
   mov dword ptr [eax], 0
   或者：

int 3
mov eax, 0

- 将 "ollydbg" 设置为实时调试器，当目标程序崩溃的时候，点击调试，这个时候 "ollydbg" 会自动断到目标程序出问题的代码位置上
- 确定位置后，将目标程序出错的代码 nop 掉，接着就可以继续调试。

2. 使用 "ollydbg" 将目标程序调试起来，接着进行代码注入，目标程序崩溃后，"ollydbg" 就会自动在目标程序崩溃的地址停下（通过调试器运行 -- 崩溃 -- 调试）。

> 方法1：可以绕过有些软件的反调试功能
> 方法2：不一定可行，有的软件具有反调试功能。