

第一题

- 找到 Optional Header 中 Data Director （相对与PE头偏移量为 78h ，文件中地址为 0x168 ） 的第一个 8h 字节即为导出表信息。如图：

```

00000130 00 00 04 00 00 10 00 00 00 00 10 00 00 10 00 00
00000160 00 00 00 00 10 00 00 00 90 A4 09 00 A0 72 00 00
00000170 04 4E 03 00 00 03 00 00 00 00 03 00 00 11 0E 00

```

可知导出表的RVA为 0x9A490 ，大小为 0x72A0 。通过LordPE转换得到在文件中的地址为 0x99890

- 在 0x99890 处起，大小 28h 范围内即为导出表，其中 Address Of Functions, Address Of Names, Address Of Ordinals的值如图：

```

00099890 00 00 00 00 D9 B7 3A A3 00 00 00 00 00 CF 09 00
000998A0 DE 05 00 00 BF 04 00 00 E2 03 00 00 B8 A4 09 00 AddressOfFunctions
000998B0 B4 B7 09 00 3C C7 09 00 B0 49 06 00 A0 86 04 00
                AddressOfNames AddressOfOrdinals

```

- 通过LordPE地址变换，可以得到：

| | RVA | 文件中地址 |
|--------------------|---------|---------|
| AddressOfNames | 0x9B7B4 | 0x9ABB4 |
| AddressOfOrdinals | 0x9C73C | 0x9BB3C |
| AddressOfFunctions | 0x9A4B8 | 0x998B8 |

- 0x9ABB4 处开始为导出函数名字的RVA数组，每个DWORD是一个字符串的RVA。通过LordPE可得到字符串在文件中的地址。

函数名RVA数组前5项：

```

0009ABB0 A0 55 04 00 37 CF 09 00 4E CF 09 00 69 CF 09 00
0009ABC0 7D CF 09 00 8E CF 09 00 A1 CF 09 00 BA CF 09 00

```

实际字符串前5项：

```

0009C300 55 53 45 52 33 32 2E 64 6C 6C 00 47 65 74 50 6F USER32.dll GetPo
0009C310 69 6E 74 65 72 46 72 61 6D 65 41 72 72 69 76 61 interFrameArriva
0009C320 6C 54 69 6D 65 73 00 57 6F 77 36 34 54 72 61 6E lTimes Wow64Tran
0009C330 73 69 74 69 6F 6E 00 41 63 74 69 76 61 74 65 4B sition ActivateK
0009C340 65 79 62 6F 61 72 64 4C 61 79 6F 75 74 00 41 64 eyboardLayout Ad
0009C350 64 43 6C 69 70 62 6F 61 72 64 46 6F 72 6D 61 74 dClipboardFormat
0009C360 4C 69 73 74 65 6E 65 72 00 41 64 64 56 69 73 75 Listener AddVisu
0009C370 61 6C 49 64 65 6E 74 69 66 69 65 72 00 41 64 6A alIdentifier Adj
0009C380 75 73 74 57 69 6E 64 6F 77 52 65 63 74 00 41 64 ustWindowRect Ad
0009C390 6A 75 73 74 57 69 6E 64 6F 77 52 65 63 74 45 78 justWindowRectEx
0009C3A0 00 41 64 6A 75 73 74 57 69 6E 64 6F 77 52 65 63 AdjustWindowRec
0009C3B0 74 45 78 46 6F 72 44 70 69 00 41 6C 69 67 6E 52 tExForDpi AlignR
0009C3C0 65 63 74 73 00 41 6C 6C 6F 77 46 6F 72 65 67 72 ects AllowForeqr

```

- 0x9BB3C 处开始为导出函数的序号，前5项如图：

```

0009BB30 10 17 0A 00 1A 17 0A 00 25 17 0A 00 03 00 04 00
0009BB40 05 00 06 00 07 00 08 00 09 00 0A 00 0B 00 0C 00
0009BB50 0D 00 0E 00 0F 00 10 00 11 00 12 00 13 00 14 00

```

6. 0x998B8 处开始为所有函数地址的RVA数组，每个RVA经转换后即可得到对应函数的起始地址。该数组与导出函数序号对应后的5个函数地址RVA如图：

```

000998B0 B4 B7 09 00 3C C7 09 00 B0 49 06 00 A0 86 04 00
000998C0 C0 4F 0A 00 A0 45 04 00 D0 44 04 00 50 49 04 00
000998D0 10 16 04 00 70 88 02 00 80 79 08 00 90 64 09 00

```

通过LordPE即可根据RVA得到每个函数在文件中的地址。

综上，可得User32.dll前5个导出函数的信息如下：

| | Name | Ordinal | Function RVA地址 | Function 文件地址 |
|---|----------------------------|---------|----------------|---------------|
| 1 | ActivateKeyboardLayout | 3 | 0x445A0 | 0x439A0 |
| 2 | AddClipboardFormatListener | 4 | 0x444D0 | 0x438D0 |
| 3 | AddVisualIdentifier | 5 | 0x44950 | 0x43D50 |
| 4 | AdjustWindowRect | 6 | 0x41610 | 0x40A10 |
| 5 | AdjustWindowRectEx | 7 | 0x28870 | 0x27C70 |

第二题

```
.386
.model flat, stdcall
option casemap :none

include C:\Irvine\Irvine32.inc

includelib kernel32.lib
includelib user32.lib
includelib masm32.lib
includelib C:\Irvine\Irvine32.lib

.data
    arr WORD 524 DUP(0)
    lpText BYTE "I'm a msg box.", 0 ; The message to be displayed.
.code
main PROC
    ; 赋随机值
    lea esi, OFFSET arr
    mov ecx, LENGTHOF arr
loop1:
    call Random32
    mov [esi], eax
    add esi, TYPE arr
    loop loop1

    ; 求最小值
    lea esi, OFFSET arr
    mov ecx, LENGTHOF arr
    mov eax, [esi]
loop2:
    cmp eax, [esi]
    jle next
    mov eax, [esi]
next:
    add esi, TYPE arr
    loop loop2

    invoke MessageBox, NULL, OFFSET lpText, NULL, MB_OK
    invoke ExitProcess, 0
main ENDP
END main
```

第三题

各个节表的信息如图：

| [区段表] | | | | | | |
|---------|----------|----------|----------|----------|----------|--|
| 名称 | VOffset | VSize | ROffset | RSize | 标志 | |
| .text | 00001000 | 000010C6 | 00000400 | 00001200 | 60000020 | |
| .rdata | 00003000 | 00000336 | 00001600 | 00000400 | 40000040 | |
| .data | 00004000 | 00000F25 | 00001A00 | 00000E00 | C0000040 | |

导入函数信息：

1. 找到 Optional Header 中 Data Director（相对与PE头偏移量为 78h）的第二个 8h 字节即为导入表的信息。可知导出表的RVA为 0x3074，大小为 0x3C。通过LordPE转换得到在文件中的地址为 0x1674。
2. 导入表有两个Import Descriptor，内容如图：

| | | |
|----------|-------------------------|-------------------------|
| 00001670 | 00 00 00 00 B0 30 00 00 | 00 00 00 00 00 00 00 00 |
| 00001680 | 32 31 00 00 00 30 00 00 | 1C 31 00 00 00 00 00 00 |
| 00001690 | 00 00 00 00 4E 31 00 00 | 6C 30 00 00 00 00 00 00 |

根据Descriptor的结构：

```
IMAGE_IMPORT_DESCRIPTOR STRUCT
    union
        Characteristics          dd    ?
        OriginalFirstThunk       dd    ? ; 0000h - 桥 1
    ends
    TimeDateStamp               dd    ? ; 0004h - 时间戳
    ForwarderChain              dd    ? ; 0008h - 链表的前一个结构
    Name1                      dd    ? ; 000ch - 指向链接库名字的指针
    FirstThunk                  dd    ? ; 0010h - 桥 2
IMAGE_IMPORT_DESCRIPTOR ENDS
```

可以解析出：

| Name | INT表RVA | INT表文件中地址 | IAT表RVA | IAT表文件中地址 | IAT表内存中虚拟地址 |
|--------------|---------|-----------|---------|-----------|-------------|
| kernel32.dll | 0x30B0 | 0x16B0 | 0x3000 | 0x1600 | 0x403000 |
| user32.dll | 0x311C | 0x171C | 0x306C | 0x166C | 0x40306C |

加载前：

- kernel32.dll的INT表（每个DWORD是一个字符串的RVA）：

| | |
|----------|---|
| 000016B0 | 24 31 00 00 0C 33 00 00 EC 32 00 00 5A 31 00 00 |
| 000016C0 | 68 31 00 00 76 31 00 00 88 31 00 00 A2 31 00 00 |
| 000016D0 | B4 31 00 00 C6 31 00 00 E4 31 00 00 F4 31 00 00 |
| 000016E0 | 04 32 00 00 14 32 00 00 24 32 00 00 30 32 00 00 |
| 000016F0 | 44 32 00 00 54 32 00 00 68 32 00 00 74 32 00 00 |
| 00001700 | 90 32 00 00 A2 32 00 00 BC 32 00 00 C4 32 00 00 |
| 00001710 | DC 32 00 00 2A 33 00 00 00 00 00 00 40 31 00 00 |

对应的函数名字符串：

ExitProcess、WriteConsoleOutputAttribute、WriteConsoleOutputCharacterA

- kernel32.dll的IAT表内容与INT表相同。
- user32.dll的INT表（每个DWORD是一个字符串的RVA）：

| | |
|----------|---|
| 00001710 | DC 32 00 00 2A 33 00 00 00 00 00 00 40 31 00 00 |
| 00001720 | 00 00 00 00 9B 00 45 78 69 74 50 72 6F 63 65 73 |

对应的函数名字符串：

MessageBoxA

- user32.dll的IAT表内容与INT表相同。

加载后：

- kernel32.dll的IAT表：

用ida加载，在IAT表的虚拟地址 0x403000 处可以看到导入的函数：

```
.idata:00403000 ;
.idata:00403000 ; Imports from kernel32.dll
.idata:00403000 ;
.idata:00403000 ; =====
.idata:00403000 ;
.idata:00403000 ; Segment type: Externs
.idata:00403000 ; _idata
.idata:00403000 ; void __stdcall __noreturn ExitProcess(UINT uExitCode)
.idata:00403000 ; extrn __imp_ExitProcess:dword
.idata:00403000 ; ; DATA XREF: ExitProcess↑r
.idata:00403000 ; ; .rdata:00403084↓o
.idata:00403004 ; BOOL __stdcall WriteConsoleOutputAttribute(HANDLE hConsoleOutput, const WORD *lpAttribute, DWORD nLength, COORD dwWriteCoord, LPDWORD lp
.idata:00403004 ; extrn __imp_WriteConsoleOutputAttribute:dword
.idata:00403004 ; ; DATA XREF: WriteConsoleOutputAttribute↑r
.idata:00403008 ; BOOL __stdcall WriteConsoleOutputCharacterA(HANDLE hConsoleOutput, LPCSTR lpCharacter, DWORD nLength, COORD dwWriteCoord, LPDWORD lpNuml
.idata:00403008 ; extrn __imp_WriteConsoleOutputCharacterA:dword
.idata:00403008 ; ; DATA XREF: WriteConsoleOutputCharacterA↑r
.idata:0040300C ; BOOL __stdcall CloseHandle(HANDLE hObject)
```

- user32.dll的IAT表：

用ida加载，在IAT表的虚拟地址 0x40306C 处可以看到导入的函数：

```
.idata:0040306C ;
.idata:0040306C ; Imports from user32.dll
.idata:0040306C ;
.idata:0040306C ; int __stdcall MessageBoxA(HWND hWnd, LPCSTR lpText, LPCSTR lpCaption, UINT uType)
.idata:0040306C ; extrn __imp_MessageBoxA:dword
.idata:0040306C ; ; DATA XREF: MessageBoxA↑r
.idata:0040306C ; ; .rdata:00403098↓o
```