Выполнила: Белоусова Е., ИП-911

Цель: закрепить сведения о структуре исполнямых файлов, получить навыки работы с интерфейсом IMAGEHLP.
Задание 1: получить список экспортируемых функций библиотеки kernel32.dll.
Задание 2: получить список импортируемых функций notepad.exe.

Описание работы программы

Получим список экспортируемых функций библиотеки kernel32.dll. Для этого загрузим PE-файл, считаем базовый адрес загрузочного модуля. Определим относительный виртуальный адрес - RVA, таблицы экспорта. Определим виртуальный адрес массива строк по его RVA, виртуальный адрес строки - имени PE-файла по его RVA. Определим виртуальный адрес массива строк по его RVA. Считаем количество экспортируемых имен из таблицы экспорта и определим виртуальный адрес i-го имени по его RVA.

```
PS D:\семестр 5\os\na67> .\7b.exe kernel32.dll
0xa10000 - Base Address
0x92d30 - RVA
0xa88d30 - VA
Name of PEF: KERNEL32.dll
Exported data:
AcquireSRWLockExclusive
AcquireSRWLockShared
ActivateActCtx
ActivateActCtxWorker
AddAtomA
AddAtomW
AddConsoleAliasA
AddConsoleAliasW
AddDllDirectory
AddIntegrityLabelToBoundaryDescriptor
AddLocalAlternateComputerNameA
AddLocalAlternateComputerNameW
AddRefActCtx
AddRefActCtxWorker
AddResourceAttributeAce
AddSIDToBoundaryDescriptor
AddScopedPolicyIDAce
AddSecureMemoryCacheCallback
AddVectoredContinueHandler
AddVectoredExceptionHandler
AdjustCalendarDate
AllocConsole
AllocateUserPhysicalPages
AllocateUserPhysicalPagesNuma
AppPolicyGetClrCompat
AppPolicyGetCreateFileAccess
AppPolicyGetLifecycleManagement
AppPolicyGetMediaFoundationCodecLoading
AppPolicyGetProcessTerminationMethod
AppPolicyGetShowDeveloperDiagnostic
AppPolicyGetThreadInitializationType
AppPolicyGetWindowingModel
AppXGetOSMaxVersionTested
ApplicationRecoveryFinished
ApplicationRecoveryInProgress
AreFileApisANSI
AssignProcessToJobObject
AttachConsole
```

Получим список импортируемых функций notepad.exe. Для этого загрузим PE-файл, считаем базовый адрес загрузочного модуля. Определим относительный виртуальный адрес – RVA. Таблица импорта фактически есть массив структур IMAGE_IMPORT_DESCRIPTOR. Каждая структура содержит информацию о dll, из которых PE импортирует функции. Конец массива отмечается элементом, содержащим одни нули.

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
                union {
                                DWORD   Characteristics;
                        DWORD   OriginalFirstThunk;
                } DUMMYUNIONNAME;
                DWORD   TimeDateStamp;
                DWORD   ForwarderChain;
                DWORD   Name;
                DWORD   FirstThunk;
        } IMAGE_IMPORT_DESCRIPTOR,*PIMAGE_IMPORT_DESCRIPTOR;
```

Смещение (hex)	Размер	Тип	Название	Описание
00	4	DWORD	OriginalFirstThunk	RVA таблицы имен импорта (INT).
04	4	DWORD	TimeDateStamp	Дата и время.
08	4	DWORD	ForwarderChain	Индекс первого перенаправленного символа.
0C	4	DWORD	Name	RVA ASCIIZ-строки, содержащей имя DLL.
10	4	DWORD	FirstThunk	RVA таблицы адресов импорта (IAT).

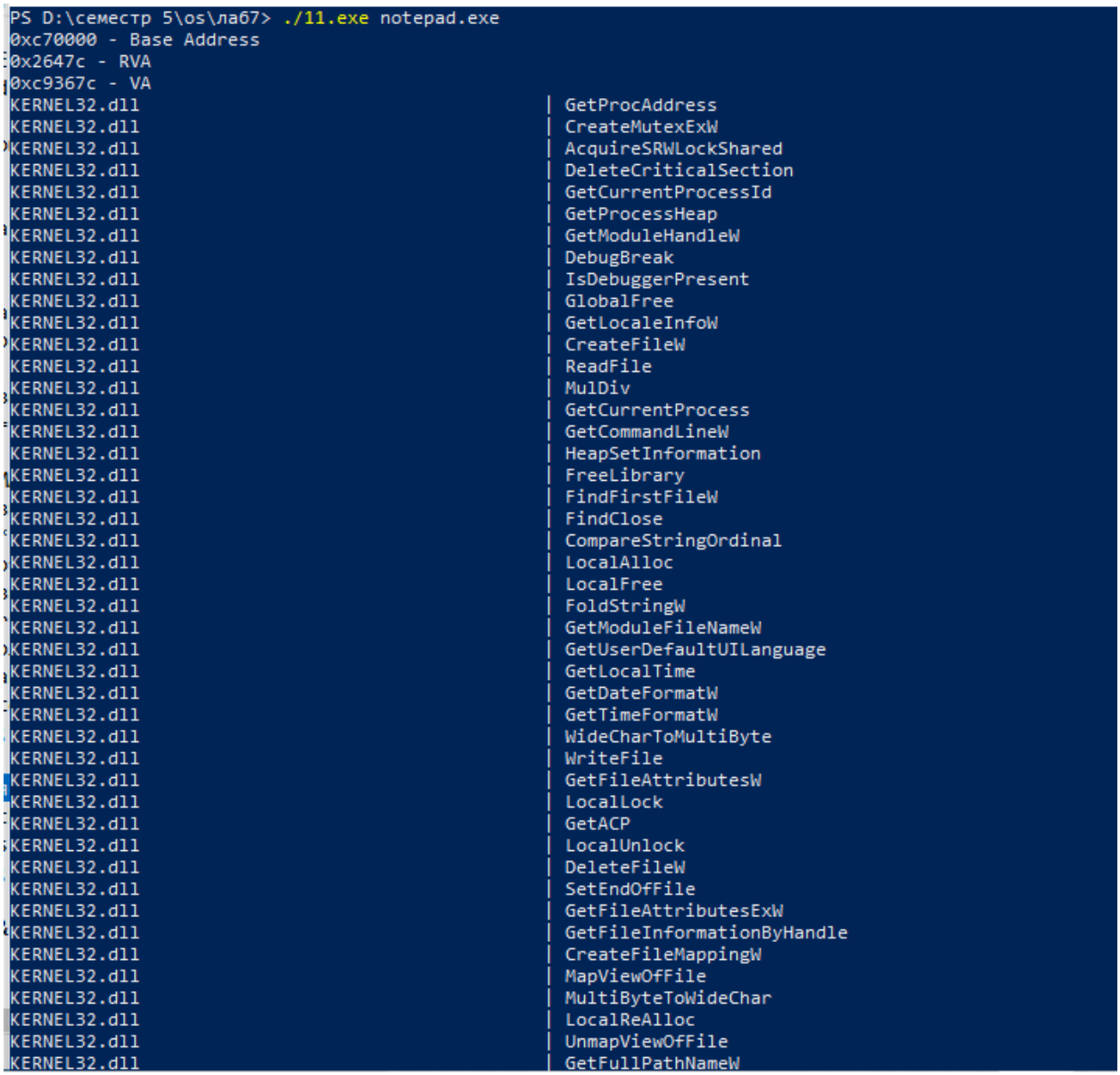
Таблицы имен и адресов (OriginalFirstThunk и FirstThunk) ссылаются на массив структур IMAGE_THUNK_DATA. Дальнейшие действия зависят от старшего бита структуры. Если он установлен, то оставшиеся биты представляют из себя номер импортируемого символа (импорт по номеру). В противном случае (старший бит сброшен) оставшиеся биты задают RVA импортируемого символа (импорт по имени). Если мы имеем импорт по имени, то указатель хранит адрес на следующую структуру:

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    BYTE    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

Поле *OriginalFirstThunk* ссылается на массив, где хранится информация по импортируемым функциям. Поле *FirstThunk* ссылается на аналогичный массив той же размерности, но разве что заполняется он во время загрузки эффективным адресами функций. Т.е. загрузчик

анализирует *OriginalFirstThunk*, определяет реальный адрес функции для каждого его элемента и заносит этот адрес в *FirstThunk*.

Поочередно будем перебирать строки таблицы импорта, получать ссылки на массивы с информацией об импортируемых функциях. С помощью макроса IMAGE_SNAP_BY_ORDINAL будем проверять установлен старший бит или нет. Если нет, то будем получать имя функции.



```
KERNEL32.dll                                | ResolveDelayLoadedAPI
KERNEL32.dll                                | DelayLoadFailureHook
KERNEL32.dll                                | GetModuleFileNameA
GDI32.dll                                   | CreateDCW
GDI32.dll                                   | StartPage
GDI32.dll                                   | StartDocW
GDI32.dll                                   | SetAbortProc
GDI32.dll                                   | DeleteDC
GDI32.dll                                   | EndDoc
GDI32.dll                                   | AbortDoc
GDI32.dll                                   | EndPage
GDI32.dll                                   | GetTextMetricsW
GDI32.dll                                   | SetBkMode
GDI32.dll                                   | LPtoDP
GDI32.dll                                   | SetWindowExtEx
GDI32.dll                                   | SetViewportExtEx
GDI32.dll                                   | SetMapMode
GDI32.dll                                   | GetTextExtentPoint32W
GDI32.dll                                   | TextOutW
GDI32.dll                                   | EnumFontsW
GDI32.dll                                   | GetTextFaceW
GDI32.dll                                   | SelectObject
GDI32.dll                                   | DeleteObject
GDI32.dll                                   | CreateFontIndirectW
GDI32.dll                                   | GetDeviceCaps
USER32.dll                                  | GetFocus
USER32.dll                                  | PostMessageW
USER32.dll                                  | GetMenu
USER32.dll                                  | CheckMenuItem
USER32.dll                                  | GetSubMenu
USER32.dll                                  | EnableMenuItem
USER32.dll                                  | ShowWindow
USER32.dll                                  | GetDC
USER32.dll                                  | ReleaseDC
USER32.dll                                  | SetCursor
USER32.dll                                  | GetDpiForWindow
USER32.dll                                  | SetActiveWindow
USER32.dll                                  | LoadStringW
USER32.dll                                  | DefWindowProcW
USER32.dll                                  | IsIconic
USER32.dll                                  | SetFocus
USER32.dll                                  | PostQuitMessage
USER32.dll                                  | DestroyWindow
api-ms-win-shcore-obsolete-l1-1-0.dll       | SHStrDupW
api-ms-win-shcore-scaling-l1-1-1.dll        | GetDpiForMonitor
api-ms-win-core-errorhandling-l1-1-0.dll    | RaiseException
api-ms-win-core-errorhandling-l1-1-0.dll    | SetUnhandledExceptionFilter
api-ms-win-core-errorhandling-l1-1-0.dll    | UnhandledExceptionFilter
api-ms-win-core-processthreads-l1-1-0.dll   | TerminateProcess
api-ms-win-core-processthreads-l1-1-1.dll   | IsProcessorFeaturePresent
api-ms-win-core-processthreads-l1-1-1.dll   | GetProcessMitigationPolicy
api-ms-win-core-synch-l1-1-0.dll            | CreateEventExW
api-ms-win-core-synch-l1-1-0.dll            | CreateEventW
api-ms-win-core-synch-l1-1-0.dll            | ResetEvent
api-ms-win-core-synch-l1-1-0.dll            | SetEvent
api-ms-win-core-synch-l1-1-0.dll            | InitializeCriticalSectionAndSpinCount
api-ms-win-core-profile-l1-1-0.dll          | QueryPerformanceCounter
api-ms-win-core-sysinfo-l1-1-0.dll          | GetTickCount
api-ms-win-core-sysinfo-l1-1-0.dll          | GetSystemTimeAsFileTime
api-ms-win-core-interlocked-l1-1-0.dll      | InitializeSListHead
api-ms-win-core-libraryloader-l1-2-0.dll    | LoadLibraryExW
api-ms-win-core-winrt-string-l1-1-0.dll     | WindowsDeleteString
api-ms-win-core-winrt-string-l1-1-0.dll     | WindowsCreateStringReference
api-ms-win-core-winrt-string-l1-1-0.dll     | WindowsGetStringRawBuffer
api-ms-win-core-winrt-string-l1-1-0.dll     | WindowsCreateString
api-ms-win-core-winrt-error-l1-1-0.dll      | SetRestrictedErrorInfo
api-ms-win-core-winrt-l1-1-0.dll            | RoInitialize
api-ms-win-core-winrt-l1-1-0.dll            | RoGetActivationFactory
api-ms-win-core-winrt-l1-1-0.dll            | RoUninitialize
api-ms-win-core-winrt-error-l1-1-1.dll      | RoGetMatchingRestrictedErrorInfo
api-ms-win-eventing-provider-l1-1-0.dll     | EventProviderEnabled
api-ms-win-core-synch-l1-2-0.dll            | Sleep
COMCTL32.dll                                | CreateStatusWindowW
PS D:\семестр 5\os\лаб7>
```

Листинг

```
//7b.c
#include <Windows.h>
#include <ImageHlp.h>
//cmd.exe /k "C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvars32.bat" `& powershell
int main(int argc, char* argv[])
{
   LOADED_IMAGE LoadedImage;
   PUCHAR BaseAddress;
   DWORD RVAExpDir, VAExpAddress;
   IMAGE_EXPORT_DIRECTORY* ExpTable;
   char* sName;
   DWORD nNames;
   char* pName;
   char** pNames;
   DWORD i;
   //Загружаем PE-файл
   if(!MapAndLoad(argv[1], NULL, &LoadedImage, TRUE, TRUE))
   {
      printf("Something's wrong!\n");
      exit(1);
   }
   //Считываем базовый адрес загрузочного модуля
   BaseAddress = LoadedImage.MappedAddress;
   printf("0x%lx - Base Address\n", BaseAddress);
   //Считываем базовый адрес загрузочного модуля
   RVAExpDir = LoadedImage.FileHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddr
ess;
```

```
   printf("0x%lx - RVA\n", RVAExpDir);
   //Определяем виртуальный адрес массива стрк по его RVA
   VAExpAddress = (DWORD)ImageRvaToVa(LoadedImage.FileHeader,
BaseAddress, RVAExpDir, NULL);
   printf("0x%lx - VA\n", VAExpAddress);
   ExpTable = (IMAGE_EXPORT_DIRECTORY*)VAExpAddress;
   //Определяем виртуальный адрес строки - (имени PE-файла по его RVA)
   sName = (char*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress,
ExpTable->Name, NULL);
   printf("Name of PEF: %s\n", sName);
   //Определяем виртуальный адрес массива строк по его RVA
   pNames = (char**)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress,
ExpTable->AddressOfNames, NULL);
   //Считываем количество экспортируемых имен из таблицы экспорта
   nNames = ExpTable->NumberOfNames;
   printf("Exported data: \n", pName);
   for(int i = 0; i < nNames; i++)
   {
      //Определяем виртуальный адрес i-го имени по его RVA
      pName = (char*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress,
(DWORD)* pNames, NULL);
      printf("%s\n", pName);
      *pNames++; //переходим к следующей строке
   }
   UnMapAndLoad(&LoadedImage);
   return 0;
}
//11.c
#include <Windows.h>
#include <stdio.h>
```

```
#include <ImageHlp.h>


int main(int argc, char* argv[])
{
   LOADED_IMAGE LoadedImage;
   PUCHAR BaseAddress;
   DWORD RVAImDir, VAImAddress;
   IMAGE_IMPORT_DESCRIPTOR* ImportDescriptor;
   if(!MapAndLoad(argv[1], NULL, &LoadedImage, TRUE, TRUE))
   {
      printf("Something's wrong!\n");
      exit(1);
   }
   BaseAddress = LoadedImage.MappedAddress;
   printf("0x%lx - Base Address\n", BaseAddress);
   RVAImDir = LoadedImage.FileHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddr
ess;
   printf("0x%lx - RVA\n", RVAImDir);
   VAImAddress = (DWORD)ImageRvaToVa(LoadedImage.FileHeader,
BaseAddress, RVAImDir, NULL);
   printf("0x%lx - VA\n", VAImAddress);
   ImportDescriptor = (IMAGE_IMPORT_DESCRIPTOR*)VAImAddress;
   while (ImportDescriptor->Name) {
         PIMAGE_THUNK_DATA OriginalFirstThunk =
(PIMAGE_THUNK_DATA)ImageRvaToVa(LoadedImage.FileHeader,
LoadedImage.MappedAddress, ImportDescriptor->OriginalFirstThunk, NULL);
         PIMAGE_THUNK_DATA FirstThunk =
(PIMAGE_THUNK_DATA)ImageRvaToVa(LoadedImage.FileHeader,
LoadedImage.MappedAddress, ImportDescriptor->FirstThunk, NULL);
         PIMAGE_THUNK_DATA ThunkData = OriginalFirstThunk;
```

```
        if (!ThunkData) ThunkData = FirstThunk;
        while (ThunkData->u1.AddressOfData) {
            if (!IMAGE_SNAP_BY_ORDINAL(ThunkData->u1.Ordinal)) {
                PIMAGE_IMPORT_BY_NAME ImportFunctionName =
(PIMAGE_IMPORT_BY_NAME)ImageRvaToVa(LoadedImage.FileHeader,
LoadedImage.MappedAddress, ThunkData->u1.AddressOfData, NULL);
                CHAR ImportAddressTableBuffer[MAXWORD] = "";
                sprintf(ImportAddressTableBuffer, "%-48s | %s",
(PUCHAR)ImageRvaToVa(LoadedImage.FileHeader,
LoadedImage.MappedAddress, ImportDescriptor->Name, NULL),
                    ImportFunctionName->Name);
                printf("%s\n", ImportAddressTableBuffer);
            }
            ThunkData++;
            OriginalFirstThunk++;
            FirstThunk++;
        }
        ImportDescriptor++;
    }
  UnMapAndLoad(&LoadedImage);
}
```