Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

Курсовая работа

по дисциплине «Операционные системы»

на тему «Просмотр PE-ELF файлов»

Выполнил:

студент группы ИП-711

Мартасов Илья Олегович

Работу проверил:

профессор кафедры ПМиК

Малков Е. А.

Новосибирск 2019 г.

**Содержание**

1.Постановка задачи …...........................................................................................3

2Основная информация…… .................................................................................4

3.Программная реализация…..............................................................................10

4.Результаты работы………..………………………….....……………...………14

**Постановка задачи**

Разработать настольное приложение, реализующее просмотр PE-файлов. В просмотр входит:

-DOS-заголовок

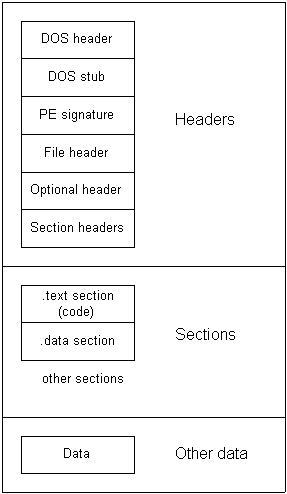
-PE-заголовок (IMAGE\_NT\_HEADER)

-Заголовки секций (IMAGE\_SECTION\_HEADER)

-Просмотр функций импорта и экспорта

**Основная информация**

PE формат — это формат исполняемых файлов всех 32- и 64- разрядных Windows систем. На данный момент существует два формата PE-файлов: PE32 и PE32+. PE32 формат для x86 систем, а PE32+ для x64. Описанные структуры можно наблюдать в заголовочном файле [WINNT.h](https://source.winehq.org/source/include/winnt.h), который поставляется вместе SDK.

Схематическое представление.   
  


***Dos-Header(IMAGE\_DOS\_HEADER)***  
Dos заголовок. Это самая первая структура в файле и она имеет размер 64 байта. В этой структуре наиболее важные поля это *e\_magic* и *e\_lfnew*.   
*e\_magic*: WORD — сигнатура находящаяся по смещению 0 от начала файла и равная “MZ”. Если данная сигнатура не равна MZ, то файл не загрузится.  
*e\_lfnew*: DWORD — смещение PE заголовка относительно начала файла. PE заголовок должен начинаться с сигнатуры (характерная запись/подпись) PE\x0\x0. PE заголовок может располагаться в любом месте файла.

#### *File-Header (IMAGE\_FILE\_HEADER)*

File-Header — набор полей, описывающий базовые характеристики файла.

Описание структуры:  
\**Machine*: WORD — это число (2 байта) задаёт архитектуру процессора, на которой данное приложение может выполняться.  
*NumberOfSections*: DWORD — количество секций в файле. Секции (в дальнейшем будем называть таблицей секций) следуют сразу после заголовка (PE-Header). В документации сказано что количество секций ограничено числом 96.  
*TimeDateStamp*: WORD — число хранящее дату и время создания файла.  
*PointerToSymbolTable*: DWORD — смещение (RAW) до таблицы символов, а SizeOfOptionalHeader — это размер данной таблицы. Данная таблица призвана служить для хранения отладочной информации, но отряд не заметил потери бойца с самого начала службы. Чаще всего это поле зачищается нулями.  
*SIzeOfOptionHeader*: WORD — размер опционального заголовка (что следует сразу за текущим. Для объектного файла он устанавливается в 0  
\**Characteristics*: WORD — характеристики файла.

**Optional-Header (IMAGE\_OPTIONAL\_HEADER)**

Этот заголовок является обязательным и имеет 2 формата PE32 и PE32+ (IMAGE\_OPTIONAL\_HEADER32 и IMAGE\_OPTIONAL\_HEADER64 соответственно). Формат хранится в поле *Magic*: WORD. Заголовок содержит необходимую информацию для загрузки файла.

*AddressOfEntryPoint*: DWORD — RVA адрес точки входа. Может указывать в любую точку адресного пространства. Для .exe файлов точка входа соответствует адресу, с которого программа начинает выполняться и не может равняться нулю!  
*BaseOfCode*: DWORD — RVA начала кода программы (секции кода).  
*BaseOfData*: DWORD — RVA начала кода программы (секции данных).  
*ImageBase*: DWORD — предпочтительный базовый адрес загрузки программы. Должен быть кратен 64кб. В большистве случаев равен 0x00400000.  
*SectionAligment*: DWORD — размер выравнивания (байты) секции при выгрузке в виртуальную память.  
*FileAligment*: DWORD — размер выравнивания (байты) секции внутри файла.  
*SizeOfImage*: DWORD — размер файла (в байтах) в памяти, включая все заголовки. Должен быть кратен SectionAligment.  
*SizeOfHeaders*: DWORD — размер всех заголовков (DOS, DOS-Stub, PE, Section) выравненный на FileAligment.  
*NumberOfRvaAndSizes*: DWORD — количество каталогов в таблице директорий (ниже сама таблица). На данный момент это поле всегда равно символической константе IMAGE\_NUMBEROF\_DIRECTORY\_ENTRIES, которая равна 16-ти.  
*DataDirectory*[NumberOfRvaAndSizes]: IMAGE\_DATA\_DIRECTORY — каталог данных. Проще говоря это массив (размером 16), каждый элемент которого содержит структуру из 2-ух DWORD-ых значений.

**Section-header (IMAGE\_SECTION\_HEADER)**

*Name*: BYTE[IMAGE\_SIZEOF\_SHORT\_NAME] — название секции.   
*VirtualSize*: DWORD — размер секции в виртуальной памяти.  
*SizeOfRawData*: DWORD — размер секции в файле.  
*VirtualAddress*: DWORD — RVA адрес секции.  
*SizeOfRawData*: DWORD — размер секции в файле. Должен быть кратен *FileAligment*.  
*PointerToRawData*: DWORD — RAW смещение до начала секции. Также должен быть кратен *FileAligment*…  
*Characteristics*: DWORD — атрибуты доступа к секции и правила для её загрузки в вирт. Память.

**Программная реализация**

**Раздел просмотра функций из dll-файлов**

case '5':

system("CLS");

if(f == 0)

{

printf("Out of file\n");

system("PAUSE");

break;

}

ch1 = '0';

while(ch1 != '4')

{

system("CLS");

printf("1 - DLL LIST\n");

printf("2 - IMPORT\n");

printf("3 - EXPORT\n");

printf("4 - BACK\n");

ch1 = getch();

switch(ch1)

{

case '1':

system("CLS");

if(importSection == NULL)

{

importDescriptor = (PIMAGE\_IMPORT\_DESCRIPTOR)fileData + imageNTHeaders->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_IMPORT].VirtualAddress;

}

else

{

importDescriptor = (PIMAGE\_IMPORT\_DESCRIPTOR)(fileData + importSection->PointerToRawData + (imageNTHeaders->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_IMPORT].VirtualAddress - importSection->VirtualAddress));

}

while(importDescriptor->Name != 0)

{

printf("%s\n", fileData + importSection->PointerToRawData + (importDescriptor->Name - importSection->VirtualAddress));

importDescriptor++;

}

system("PAUSE");

break;

case '2':

if(imageNTHeaders->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_IMPORT].VirtualAddress == 0)

{

printf("NO IMPORT\n");

system("PAUSE");

break;

}

if(importSection == NULL)

{

importDescriptor = (PIMAGE\_IMPORT\_DESCRIPTOR)fileData + imageNTHeaders->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_IMPORT].VirtualAddress;

}

else

{

importDescriptor = (PIMAGE\_IMPORT\_DESCRIPTOR)(fileData + importSection->PointerToRawData + (imageNTHeaders->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_IMPORT].VirtualAddress - importSection->VirtualAddress));

}

while(importDescriptor->Name != 0)

{

system("CLS");

printf("%s\n", fileData + importSection->PointerToRawData + (importDescriptor->Name - importSection->VirtualAddress));

thunk = importDescriptor->OriginalFirstThunk == 0 ? importDescriptor->FirstThunk : importDescriptor->OriginalFirstThunk;

thunkData = (PIMAGE\_THUNK\_DATA)(fileData + importSection->PointerToRawData + (thunk - importSection->VirtualAddress));

for (;thunkData->u1.AddressOfData != 0; thunkData++)

{

if (thunkData->u1.AddressOfData > 0x80000000)

{

printf("Ordinal: %x\n", (WORD)thunkData->u1.AddressOfData);

}

else

{

printf("%s\n", (fileData + importSection->PointerToRawData + (thunkData->u1.AddressOfData - importSection->VirtualAddress + 2)));

}

}

printf("1 - GO NEXT\n");

printf("2 - GO PAST\n");

printf("3 - EXIT\n");

ch2 = getch();

switch(ch2)

{

case '1':

importDescriptor++;

break;

case '2':

importDescriptor--;

break;

case '3':

f1 = 0;

break;

default:

break;

}

if(f1 == 0)

{

f1 = 1;

break;

}

}

break;

case '3':

if(imageNTHeaders->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_EXPORT].VirtualAddress == 0)

{

printf("NO EXPORT\n");

system("PAUSE");

break;

}

if(!MapAndLoad(fileName, NULL, &LoadedImage, TRUE, TRUE))

{

printf("Something's wrong!\n");

system("PAUSE");

break;

}

BaseAddress = LoadedImage.MappedAddress;

RVAExpDir = LoadedImage.FileHeader->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_EXPORT].VirtualAddress;

VAExpAddress = (DWORD)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, RVAExpDir, NULL);

ExpTable = (IMAGE\_EXPORT\_DIRECTORY\*)VAExpAddress;

pNames = (char\*\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, ExpTable->AddressOfNames, NULL);

for(DWORD i = 0; i < ExpTable->NumberOfNames; i++)

{

pName = (char\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, (DWORD)\*pNames, NULL);

printf("%s\n", pName);

\*pNames++;

}

UnMapAndLoad(&LoadedImage);

system("PAUSE");

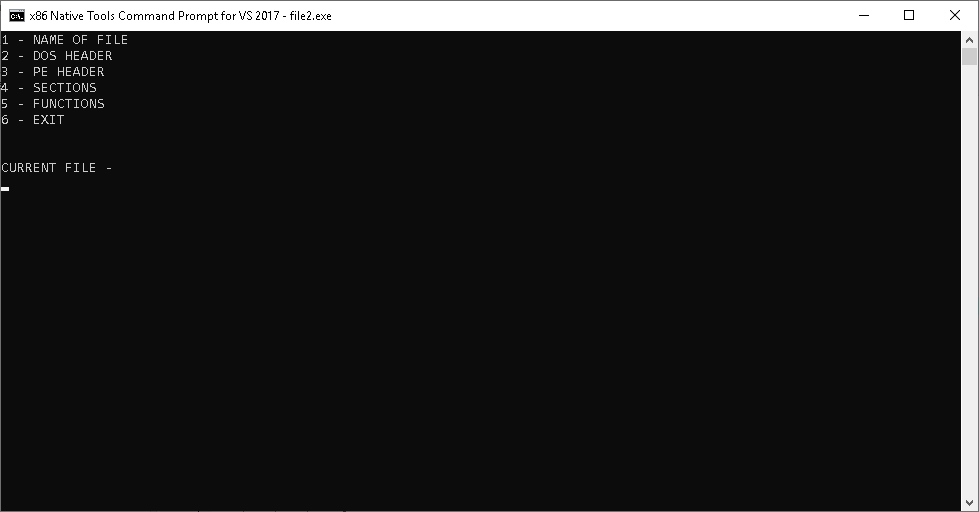
break;

}

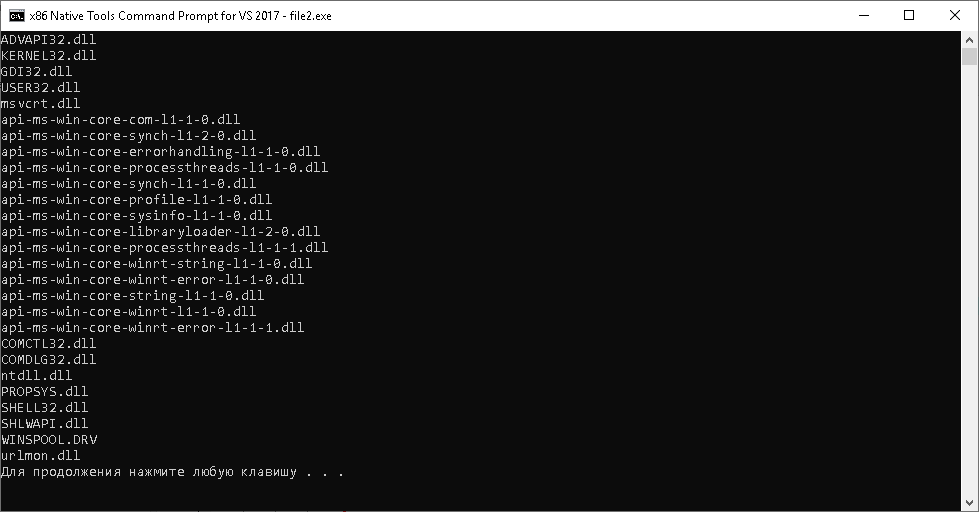
}

**Результаты работы**

**Главное меню:**



**Список импортируемых dll**



**Список функций от импортируемой dll**

