Лекция 7

Содержание

I. Исполняемые PE(***P****ortable* ***E****xecutable*)-файлы:

- структура PE-файлов,
- отображение исполняемых файлов на адресное пространство.

II. Получение информации о PE-файле:

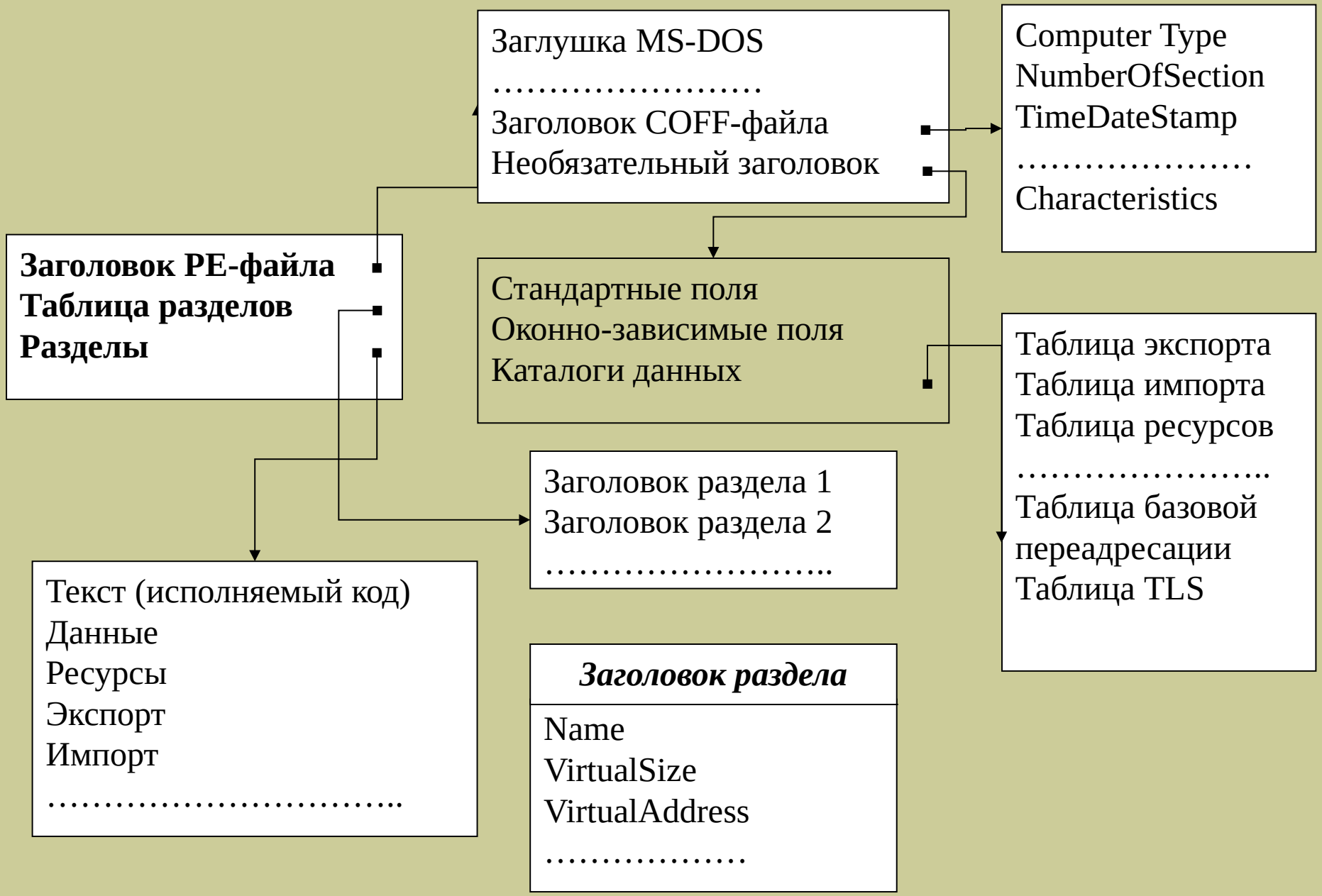
- интерфейс IMAGEHLP.

III. Исполняемые ELF (***E****xecutable and* ***L****incable* ***F****ormat*) - файлы.

Исполняемые файлы PE–файлы (*Portable Executable File*)

Исполняемые файлы (.exe, .dll, .ocx и т.д.) - файлы образа задачи (image file) компонуются из объектных файлов (.obj) - COFF (***C****ommon* ***O****bject* ***F****ile* ***F****ormat*) – файлов.
Отображение исполняемого файла на адресное пространство – загрузка исполняемого модуля, происходит по *базовому адресу* (если возможно, то по *базовому адресу по умолчанию*). Объектный код содержит относительные адреса – смещения по отношению к базовому адресу. При компоновке относительные адреса заменяются на абсолютные адреса с использованием *базового адреса по умолчанию*.

Структура PE-файла



Разделы

Раздел текста: исполняемый код данного файла, обозначается .text
Разделы данных: .bss содержит неинициализированные данные, .rdata – данные только для чтения (символьные строки, константы), .data содержит все остальные переменные.
Раздел ресурсов: содержит информацию о ресурсах, обозначается .rsrc.
Раздел перемещения: хранит таблицу адресных записей с адресными привязками к реальному адресу загрузки, обозначается .reloc.
Раздел экспорта: содержит информацию об экспортируемых функциях и глобальных переменных, обозначается .edata.
Раздел импорта: содержит информацию об импортируемых функциях, обозначается .idata.

Получение информации о PE-файле

```
#include <windows.h>
#include <imagehlp.h>

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);

  //Определяем относительный виртуальный адрес - RVA,
таблицы экспорта
  RVAExpDir=  LoadedImage.FileHeader->
       OptionalHeader.DataDirectory
[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;

 printf("0x%lx -RVA\n", RVAExpDir);
```

```
BOOL MapAndLoad(
PSTR ImageName,
PSTR DllPath,
PLOADED_IMAGE LoadedImage,
BOOL DotDll,
BOOL ReadOnly );
```

```
typedef struct _LOADED_IMAGE {
PSTR ModuleName;
HANDLE hFile;
PUCHAR MappedAddress;
PIMAGE_NT_HEADERS32 FileHeader;
ULONG NumberOfSections;
PIMAGE_SECTION_HEADER Sections;
ULONG SizeOfImage;
} LOADED_IMAGE, *PLOADED_IMAGE;
```

LoadedImage.FileHeader

```
typedef struct _IMAGE_NT_HEADERS {
DWORD Signature;
IMAGE_FILE_HEADER FileHeader;
IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

FileHeader->OptionalHeader

```
typedef struct _IMAGE_OPTIONAL_HEADER
{  ……
BYTE        MajorLinkerVersion;
BYTE        MinorLinkerVersion;
DWORD    SizeOfCode;
DWORD    SizeOfInitializedData;
DWORD    SizeOfUninitializedData;
DWORD    AddressOfEntryPoint;
DWORD    BaseOfCode;
DWORD    BaseOfData;

……
DWORD    NumberOfRvaAndSizes;
IMAGE_DATA_DIRECTORY
DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER,
*PIMAGE_OPTIONAL_HEADER;
```

```
RVAExpDir=  LoadedImage.FileHeader->
OptionalHeader.DataDirectory
[IMAGE_DIRECTORY_ENTRY_EXPORT] .VirtualAddress;
```

Индексы массива точек входа таблиц данных (Directory entries):

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT        0
#define IMAGE_DIRECTORY_ENTRY_IMPORT        1
……..
#define IMAGE_DIRECTORY_ENTRY_BASERELOC  5
……..
```

```
typedef struct _IMAGE_DATA_DIRECTORY{
DWORD VirtualAddress;
DWORD Size;
 }IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

```
//Определяем виртуальный адрес массива строк 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);
```

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

```
PVOID ImageRvaToVa(
  PIMAGE_NT_HEADERS NtHeaders,
  PVOID Base,
  ULONG Rva,
  PIMAGE_SECTION_HEADER* LastRvaSection
);
```

ExpTable=(IMAGE_EXPORT_DIRECTORY*)VAExpAddress

```
typedef struct _IMAGE_EXPORT_DIRECTORY
{
DWORD   Characteristics;
DWORD   TimeDateStamp;
WORD    MajorVersion;
WORD    MinorVersion;
DWORD   Name;
DWORD   Base;
DWORD   NumberOfFunctions;
DWORD   NumberOfNames;
DWORD   AddressOfFunctions;
DWORD   AddressOfNames;
DWORD   AddressOfNameOrdinals;
} IMAGE_EXPORT_DIRECTORY,
*PIMAGE_EXPORT_DIRECTORY;
```

```
//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(i=0;i<nNames;i++){
    //Определяем виртуальный адрес i-ого 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;
}
```

КОМПИЛЯЦИЯ

> cl 1.c imagehlp.lib

OUTPUT

> 1 td1.dll

0x20000 - Base Address

0x9a50 -RVA

0x29a50 -VA

Name of PEF: td1.dll

Exported data:

a

f

g

Карта адресного пространства процесса (*Linux 3.11.6-4, x86_64*)

```
malkov@linux-bx7d:~/WORKSHOP/PROJECTS/Transport/ISSUE> ./sch_test6  256  32 64 0.01 l

malkov@linux-bx7d:~> ps -aux | grep sch_test6
malkov   17686  0.0  0.0  12952  1328 pts/10   S+   16:23   0:00 ./sch_test6 256 32 64 0.01 2
malkov@linux-bx7d:~> cd  /proc/17686
malkov@linux-bx7d:/proc/17686> cat maps
00400000-00402000 r-xp 00000000 08:06 7210176                            /home/malkov/.../sch_test6
00601000-00602000 r--p 00001000 08:06 7210176                           /home/malkov/.../sch_test6
00602000-00643000 rw-p 00002000 08:06 7210176                            /home/malkov/.../sch_test6
01f40000-01f82000 rw-p 00000000 00:00 0                                   [heap]
7f8bc8cec000-7f8bc8e91000 r-xp 00000000 08:05 1179690                    /lib64/libc-2.18.so
.....
7f8bc9097000-7f8bc909b000 rw-p 00000000 00:00 0
.....
.
7f8bc92b2000-7f8bc93b4000 r-xp 00000000 08:05 1179681                    /lib64/libm-2.18.so
.....
7f8bc95b5000-7f8bc969f000 r-xp 00000000 08:05 6427589                     /usr/lib64/libstdc++.so.6.0.18
.....
......
7f8bc9ade000-7f8bc9adf000 rw-p 00000000 00:00 0
7fff1b637000-7fff1b659000 rw-p 00000000 00:00 0                          [stack]
7fff1b7b3000-7fff1b7b5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ff ff ff ff ff 60 10 00 r-xp 00000000 00:00 0                 [vsyscall]
malkov@linux-bx7d:/proc/17686>
```

диапазон адресов области | доступ | смещение в отображении файла | номер устройства | индексный дескриптор файла | путь к файлу \

Адрес и размер области кратен 4K

```
...б5-linux> nm -D lib1d.so  | grep T
0000000000005c8 T _fini
0000000000000460 T _init
0000000000000574 T sh_fun
```

```
б5-linux> objdump  -t lib1d.so
……..
0000000000201010 g     O .data  0000000000000008              a
0000000000201018 g       *ABS*  0000000000000000          __bss_start
0000000000201028 g       *ABS*  0000000000000000          _end
0000000000201018 g       *ABS*  0000000000000000          _edata
0000000000000460 g     F .init  0000000000000000          _init
0000000000000574 g     F .text  0000000000000010          sh_fun
```

Вторая тема курсовой