МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ

БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

О.В. КАРАВАЕВА

# Компоновщик исполняемых файлов PE Linker

Учебное пособие

Кирова 2021

УДК 004.451(07)

К21

Рекомендовано к изданию методическим советом ВятГУ

Допущено редакционно-издательской комиссией методического совета ВятГУ в качестве учебного пособия для студентов направлений 09.03.01

«Информатика и вычислительная техника».

Рецензенты:

кандидат технических наук, доцент кафедры системы автоматического управления ВятГУ

В. И. Семеновых

кандидат технических наук, доцент кафедры математических и естественно-научных дисциплин КФ МФЮА

Т. А. Анисимова

**Караваева, О. В.**

К21 Системное программное обеспечение. Компоновщик исполняемых файлов PE Linker / О. В. Караваева. — Киров: ВятГУ, 2021. —90 с.!!!

УДК 004.451(07)

Пособие предназначено для студентов направлений 09.03.01

«Информатика и вычислительная техника» всех профилей подготовки, всех форм обучения для выполнения лабораторных работ по дисциплине

«Системное программное обеспечение»

Авторская редакция Тех. Редактор

ВятГУ, 2021

**СОДЕРЖАНИЕ**

[Компоновщик исполняемых файлов PE Linker 1](#_Toc89547664)

[Введение 4](#_Toc89547665)

[1. Описание лабораторной установки 5](#_Toc89547666)

[1.1. Описание интерфейса 5](#_Toc89547667)

[1.2. Замечания 6](#_Toc89547668)

[2. Выполнение лабораторной работы 7](#_Toc89547669)

[2.1. Написание программы на языке ассемблер 7](#_Toc89547670)

[2.2. Создание заголовка PE-файла 8](#_Toc89547671)

[2.3. Создание секций PE-файла 11](#_Toc89547672)

[2.4. Создание таблицы импорта 17](#_Toc89547673)

[2.5. Загрузка программы отладчиком SoftICE 18](#_Toc89547674)

[2.6. Примерные задания для выполнения лабораторной работы 19](#_Toc89547675)

[ЗАКЛЮЧЕНИЕ 29](#_Toc89547676)

# Введение

Компоновщик (Linker) и Загрузчик (Loader) — это два программных компонента, связанных с выполнением программы. Компоновщик — это программное обеспечение, которое связывает объектный код с дополнительными файлами, такими как заголовочные файлы, и создает исполняемый файл с расширением .exe. Загрузчик — это системное программное обеспечение, которое загружает исполняемый файл, созданный компоновщиком, в основную память.

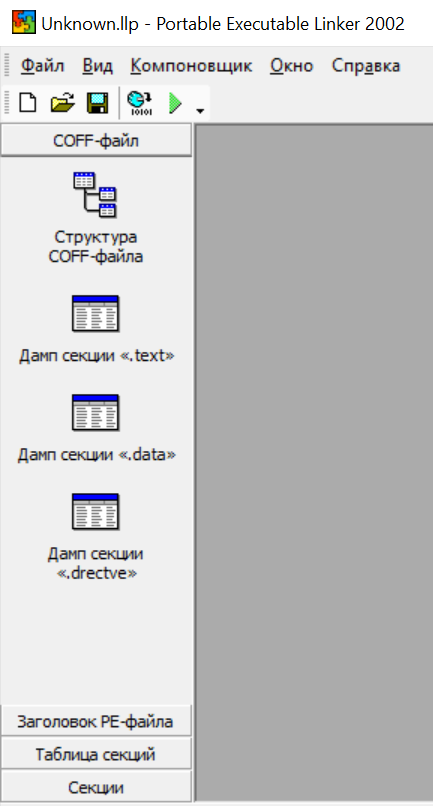
Для того, чтобы лучше понять, как работает компоновщик, как из программного кода получается исполняемый файл, необходимо самому проделать ту работу, которую выполняют компоновщик и загрузчик.

В ходе выполнения данной лабораторной работы студенту необходимо последовательно пройти этапы компиляции исходного текста задания, получения исполняемого модуля и загрузки полученной программы.

# Описание лабораторной установки

## Описание интерфейса

Все основные действия при выполнении лабораторной работы выполняются с помощью панели управления, которая становится доступной только после создания нового проекта или открытия существующего. Панель управления состоит из четырех групп. Содержимое первых двух фиксировано, содержание последних двух может меняться в процессе работы. Структура панели управления представлена ниже.



Структура панели управления

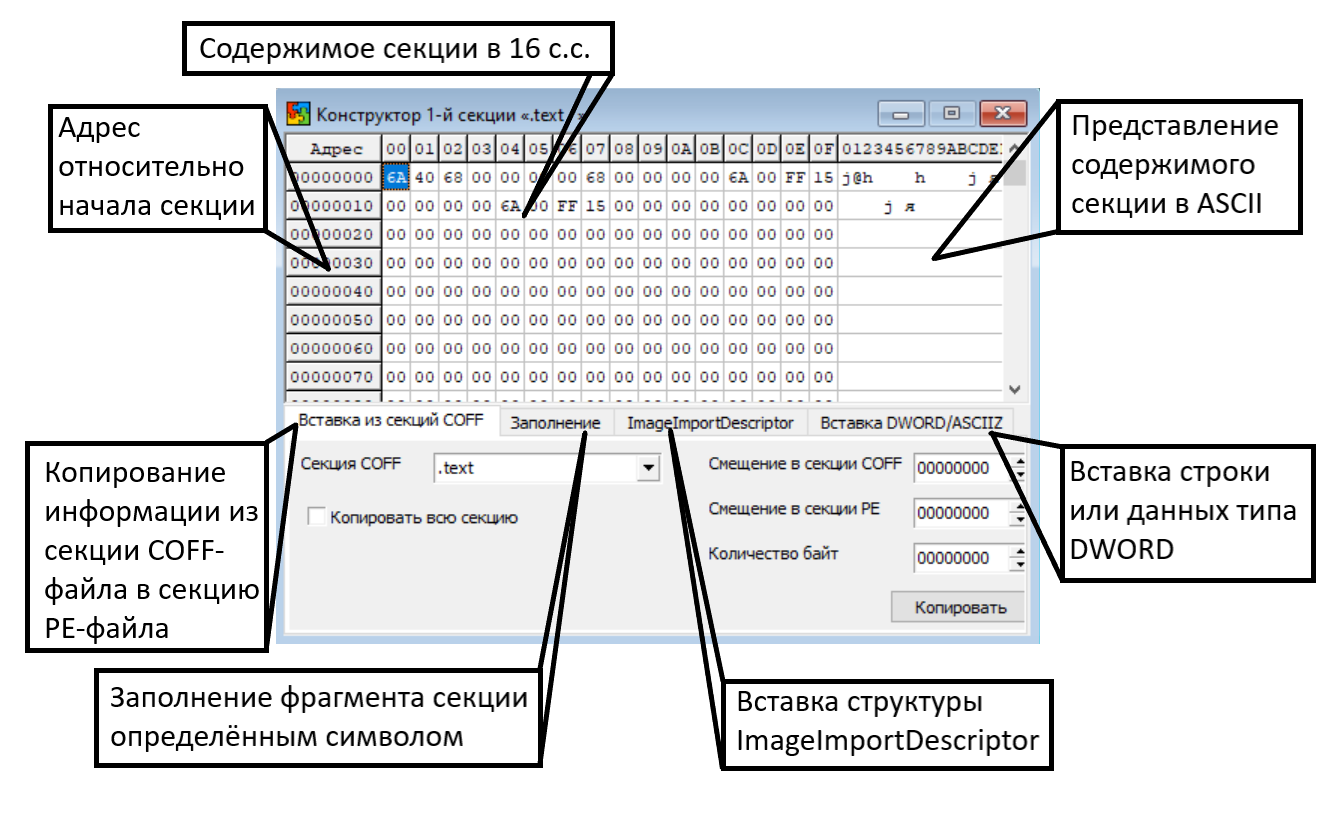
"COFF-файл" cодержит информацию об объектном файле: структуру и дампы всех его секций. Вся информация в этой группе доступна только для чтения.

В "Заголовке РЕ-файла" находятся команды позволяющие изменять содержимое конкретных частей заголовка РЕ-файла.

В "Таблице секций" расположены команды для изменения заголовков отдельных секций РЕ-файла.

В разделе "Секции" собраны команды для редактирования содержимого секций.

При выборе любой команды в панели управления открывается окно с редактором соответствующего типа. Работы с редакторами осуществляется стандартными способами, принятыми в среде Windows.



Окно редактора секций

## Замечания

Все числа в лабораторной установке представлены в 16 с.с.(Hex).

Для создания секций нужно установить их количество в поле NumberOfSections файлового заголовка.

Если в окне есть кнопка "Применить", то для сохранения изменений необходимо нажать эту кнопку. В противном случае изменения сохраняются автоматически.

Программа не обнаружит ошибки, связанные с неправильным заполнением секций.

Все размеры указываются с учётом кратности FileAlignment, адреса указываются с учётом кратности SectionAlignment.

# Выполнение лабораторной работы

## Написание программы на языке ассемблер

Прежде чем начать работу с лабораторной установкой необходимо получить объектный модуль в формате COFF. Для создания такого модуля может использоваться компилятор MASM версии не ниже 6.14, или другие средства разработки, генерирующие объектные файлы в указанном формате.

Программа на ассемблере пишется в соответствие с заданием на лабораторную работу. При написании текста программы необходимо придерживаться следующих правил:

- все внешние функции (в данном случае – функции API) необходимо объявлять с помощью директивы

    extrn <имя функции>: dword

например:

    extrn MessageBoxA: dword    ;функция MessageBoxA модуля user32.dll

- все функции Windows API используют модель вызова STDCALL, которая имеет следующие особенности:

1. все параметры вызова, не зависимо от типа, имеют размер 4 байта;
2. параметры передаются в стек в порядке, обратном их объявлению;
3. параметры удаляются из стека вызываемой функцией;
4. результат функции возвращается в регистре EAX.

- имена функций чувствительны к регистру символов (как в языках C/C++);

- набор инструкций для процессоров i386 и выше должен быть обязательно включен директивой .386;

- обязательно должна использоваться модель памяти FLAT. Данная модель включается с помощью директивы

.model flat

После написания программы её необходимо откомпилировать с помощью команды:

ml /c /coff <имя файла>

Пример написания программы, выводящей на экран простое сообщение:

    .386

    .model flat

        extrn MessageBoxA: dword

        extrn ExitProcess: dword

    .code

    \_start:

        push    40h

        push    offset App

        push    offset Msg

        push    0

        call    MessageBoxA

        push    0

        call    ExitProcess

    .data

        App     db      'PE Linker Test', 0

        Msg     db      'Success!!!', 0

    end \_start

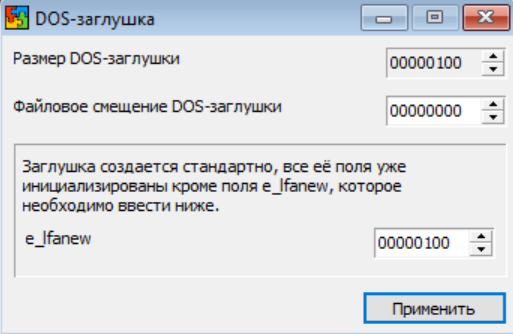
## Создание заголовка PE-файла

После успешной компиляции и получения COFF-файла можно приступать к этапу компоновки, выполняемому с помощью данной лабораторной установки.

PE-файл состоит из заголовка и некоторого набора секций, количество и размер которых зависит от информации, содержащейся в заголовке. Создание PE-файла должно начинаться именно с заголовка, как наиболее общего элемента.

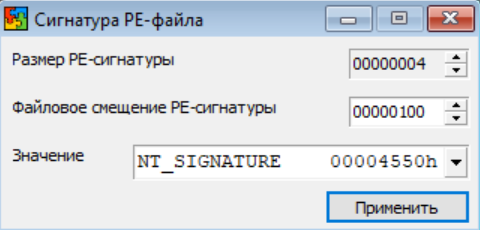
После создания нового проекта необходимо заполнить поля заголовка, не зависящие от конкретного задания - файловые смещения DOS-заглушки, сигнатуры, файлового и дополнительного заголовков. Большинство полей заголовка PE-файла имеют константные значения и приведены ниже.

В разделе «DOS-заглушка» необходимо указать значения файлового смещения и поля e\_Ifanew, которые равны 0 и 100 соответственно.

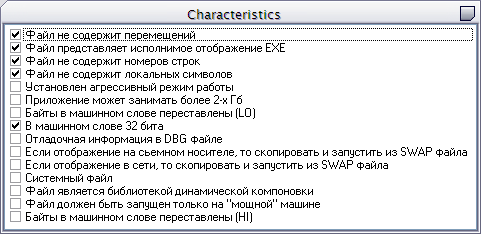


Далее файловое смещение вычисляется как сумма файлового смещения предыдущего раздела и его размера. Таким образом файловое смещение раздела «PE-сигнатура» будет равно 100.

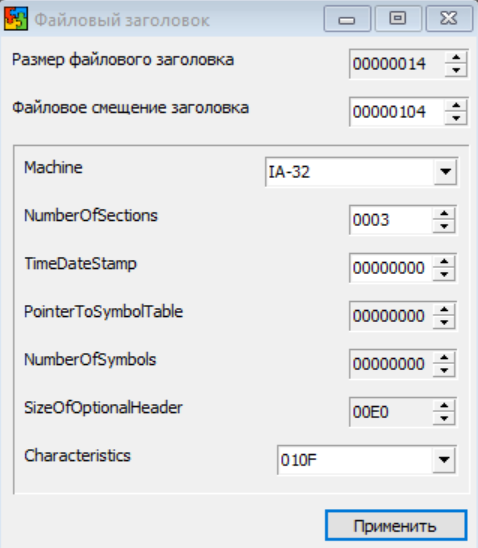
Также в разделе «PE-сигнатура» необходимо указать значение: NT\_SIGNATURE 00004550h.



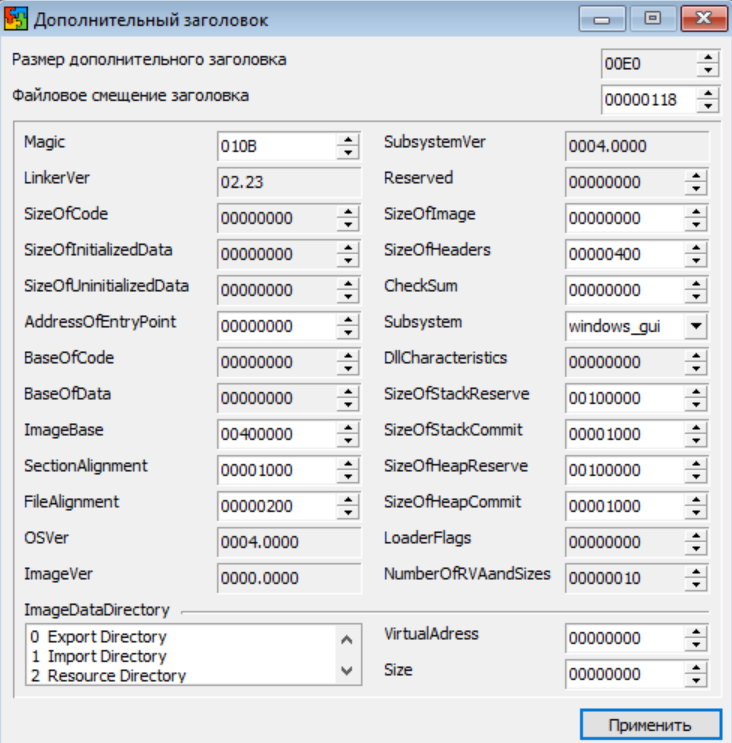
В разделе «Файловый заголовок» при заполнении поля Characteristics появится список, в котором нужно отметить следующие пункты:



Также в этом разделе необходимо заполнить поля файловое смещение заголовка, Machine, TimeDateStamp. Помимо этих полей необходимо определиться с количеством секций будущего PE-файла. Секции PE-файла копируют секции COFF-файла, за исключением секции .drectve, которую можно отбросить. Вместо неё для обращения к функциям из библиотек, необходимо создать секцию .idata – таблица импорта. Исходя из этого раздел «Файловый заголовок» будет заполнен следующим образом:



Раздел «Дополнительный заголовок» содержит большое количество полей, заполнить нужно только часть из них.



Поля AddressOfEntryPoint и SizeOfImage необходимо будет заполнить после заполнения таблицы секций. SectionAlignment – гранулярность физическое памяти, т.е. размер страницы. FileAlignment – гранулярность физического носителя, т.е. размер сектора винчестера. Поле SizeOfHeaders, как и все поля с указаниями размеров, вычисляется с учётом гранулярности. Данное поле вычисляется как суммарный размер всех заголовков и таблицы секций. В 16 с.с. суммарный размер будет равен 360, а с учётом кратности значению FileAlignment, в поле SizeOfHeaders будет записано значение 400.

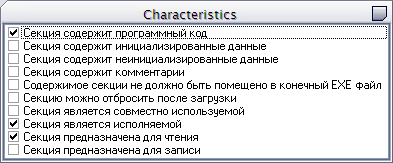
Когда значение поля NumberOfSections файлового заголовка отлично от нуля, становится доступной для редактирования таблица секций.

## 

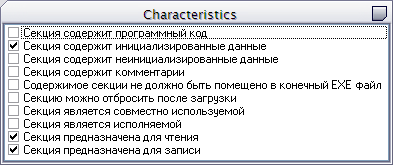
## Создание секций PE-файла

После создания заголовка необходимо заполнить пока что пустые секции PE файла. Прежде всего необходимо скопировать информацию из всех неслужебных секций COFF-файла в соответствующие им секции исполняемого модуля. Необходимо, чтобы порядок и имена секций объектного и исполняемого файлов совпадали.

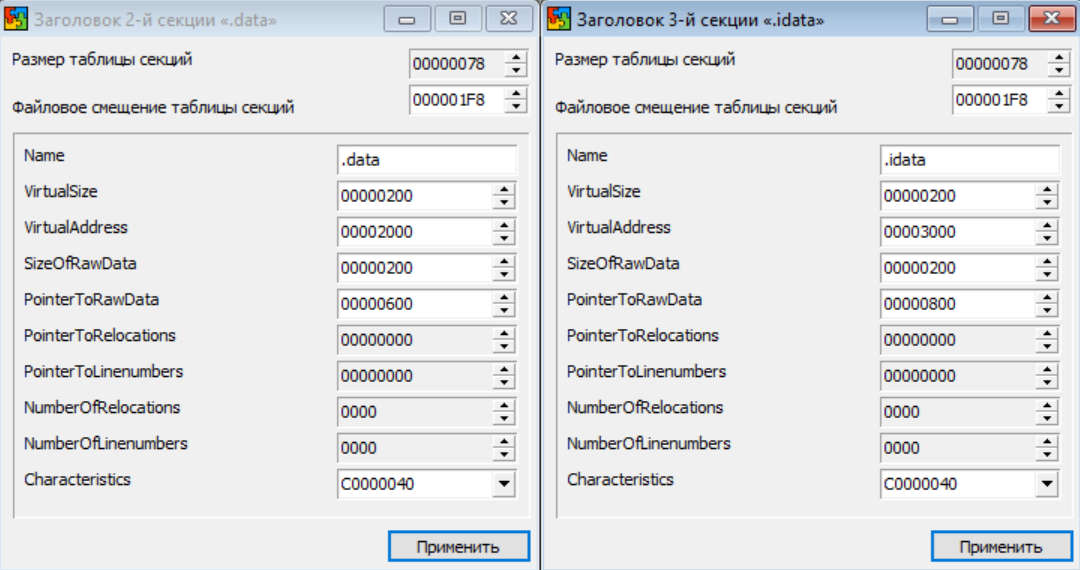
Значение поля VirtualSize равно значению SizeOfRawData. SizeOfRawData равен размеру дампа соответствующей секции COFF-файла, который можно посмотреть в первом пункте панели управления, с учётом гранулярности FileAlignment. VirtualAddress первой секции вычисляется как сумма файлового смещения таблицы секций и её размера с учётом гранулярности SectionAlignment. Для остальных секций берётся VirtualAddress предстоящей секции, складывается с её размером и округляется в большую сторону на значение SectionAlignment. PointerToRawData вычисляется как сумма SizeOfHeaders из раздела «дополнительный заголовок» и размера предыдущих секций. При заполнении поля Characteristics откроется список, где для секции кода .text нужно выбрать пункты:



Для секций данных .data и .idata нужно выбрать следующие пункты:



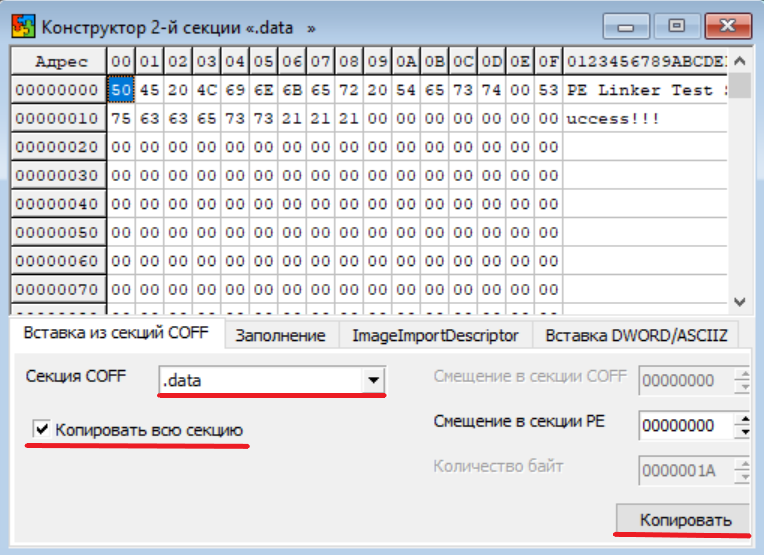
Ниже приведён пример заполнения заголовков секций PE-файла для .data и .idata.



Значение поля VirtualAddress из заголовка секции .text нужно записать в поле AddressOfEntryPoint в разделе «Дополнительный заголовок».

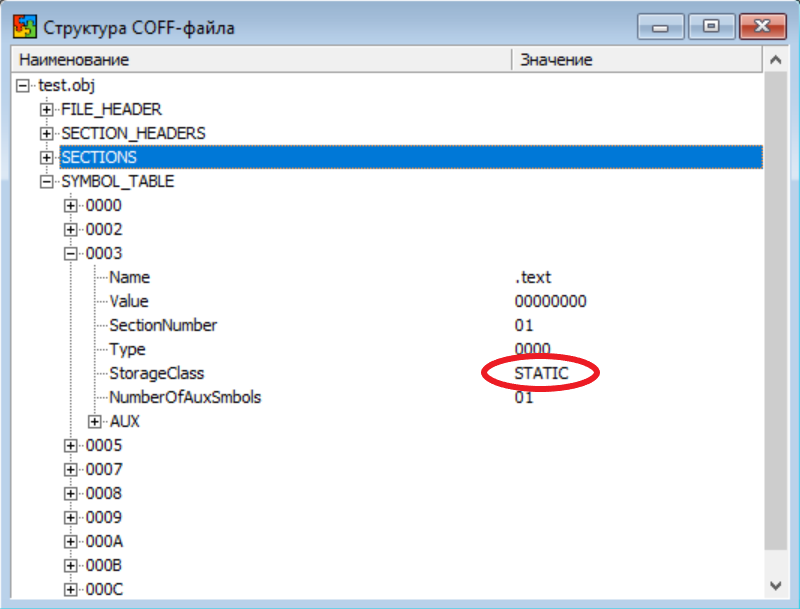
После этого необходимо заполнить поле SizeOfImage – суммарный объём от заголовка до конца данных последней секции в виртуальном пространстве округлённый на значение поля SectionAlignment. Т.е. сумма VirtualAddress из заголовка секции .idata и SizeOfRawData из заголовка той же секции с учётом гранулярности SectionAlignment.

Теперь необходимо заполнить содержимое секций. Для этого необходимо перейти в раздел секций, выбрать нужную секцию, в пункте «Секция COFF» указать соответствующую секцию COFF-файла, поставить галочку в пункте «Копировать всю секцию» и скопировать данные из COFF-файла. Проделать данные действия нужно только для секций .text и .data, для секции .idata этого делать не нужно т.к. её нет в COFF-файле. После копирования секции в памяти отобразятся те данные, которые находились в соответствующей секции COFF-файла, как это показано ниже.



Далее нужно выполнить разрешение статических и внешних ссылок. На этапе компиляции неизвестны реальные адреса переменных и функций API, поэтому компилятор превращает адреса переменных в статические, а адреса функций API – во внешние ссылки. Информация о неразрешенных ссылках хранится в двух местах в объектном модуле: в COFF-таблице символов и в списках привязок для каждой секции.

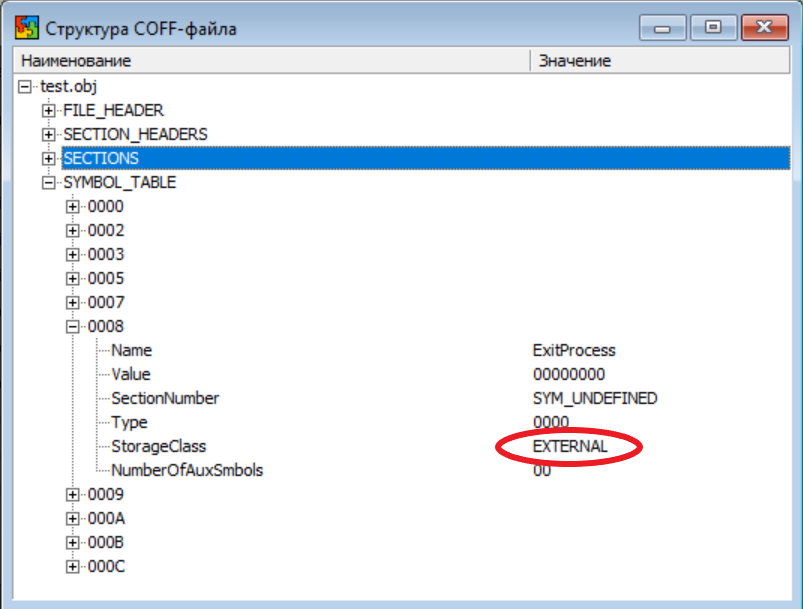
Также в COFF-таблицу символов компилятор помещает сведения обо всех используемых переменных и функциях, адреса которых он не смог определить. Для данной работы интерес представляют лишь элементы типа STATIC и EXTERNAL, хранящие сведения о переменных и функциях соответственно. Содержимое элемента типа STATIC COFF-таблицы символов представлено ниже.



Содержимое элемента типа STATIC

Поле Name содержит символьное имя переменной, взятое из исходного текста программы, поле SectionNumber задает номер секции COFF-файла, в которой находится данная переменная (в COFF-файле секции нумеруются с единицы), поле Value содержит адрес переменной относительно начала секции с номером SectionNumber.

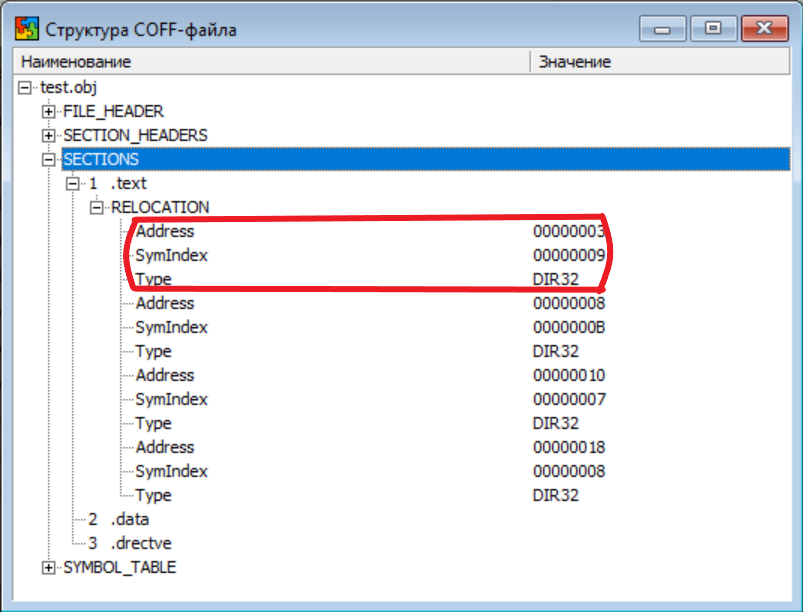
Структура элемента типа EXTERNAL, описывающего API-функцию приведена ниже.



Содержимое элемента типа EXTERNAL

Поле Name содержит символьное имя функции, взятое из исходного текста программы, поле SectionNumber для данного элемента не определено, поскольку функция находится вне объектного модуля, а поле Value всегда содержит нуль.

Список всех ссылок на переменные и функции API данной секции при компиляции попадает в перечень привязок (RELOCATION) соответствующей секции. Ниже приведен пример списка RELOCATION для секции .text объектного модуля программы.



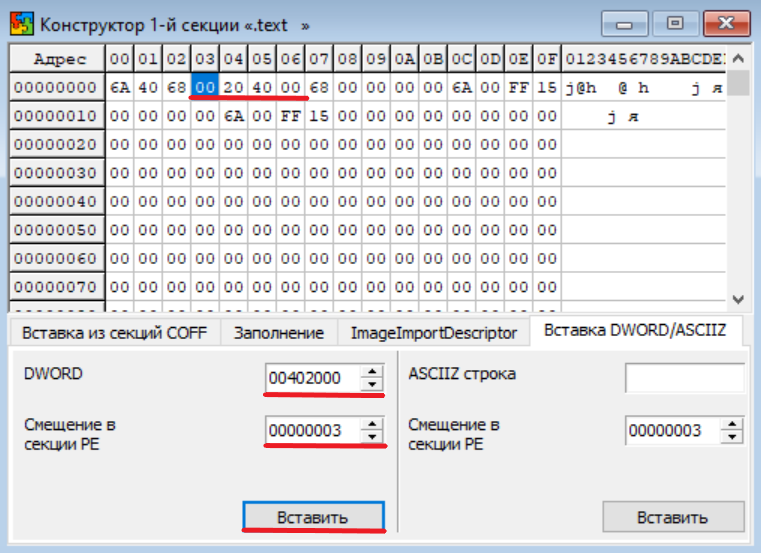
Содержимое списка RELOCATION

Из рисунка видно, что данная секция содержит четыре неразрешенные ссылки типа 32-разрядный прямой адрес. Это означает, что в секции .text объектного модуля имеются четыре четырехбайтных поля, значение которых необходимо скорректировать.

Каждая привязка содержит следующие поля: Address – смещение начала четырехбайтной ссылки в текущей секции, SymIndex – номер символа из COFF-таблицы символов, описывающего данную ссылку. Поле Type описывает тип ссылки, в данной работе встречаются только ссылки типа DIR32 – прямой 32-разрядный адрес.

По значению поля SymIndex ссылки найти соответствующий ей элемент в SYMBOL\_TABLE. Если тип данной ссылки EXTERNAL, то разрешить его пока не удастся, перейти к проверке остальных ссылок. Если тип ссылки STATIC, ссылка является разрешимой. Для таких ссылок необходимо сосчитать адрес, который нужно вставить в секцию .text. Адрес вычисляется как сумма ImageBase из «Дополнительного заголовка», VirtualAddress’а из заголовка секции, номер которой указан в поле SectionNumber соответствующей ссылки, и значения поля Value из SYMBOL\_TABLE. Вычислив адрес, необходимо перейти в разделе «Секции» в секцию .text, на вкладку «Вставка DWORD/ASCIIZ». Записать полученное значение в поле «DWORD», указать смещение в секции PE-файла равным значению Address из таблицы привязок (RELOCATION) для данной ссылки. На этом этапе необходимо подобным образом разрешить все ссылки типа STATIC. Пример разрешения ссылки приведён ниже.

ImageBase равен 400000. В SectionNumber первой неразрешённой ссылки типа STATIC указано значение 02, что соответствует секции .data. В разделе «Таблица секций» в заголовке секции .data значение VirtualAddress равно 2000. Значение поля Value для данной ссылки в SYMBOL\_TABLE равно 0. При сложении данных значений получаем 402000. Значение поля Address из таблицы привязок (RELOCATION) равен 3.



Разрешение внешних ссылок можно будет провести только после построения таблицы импорта.

## 

## Создание таблицы импорта

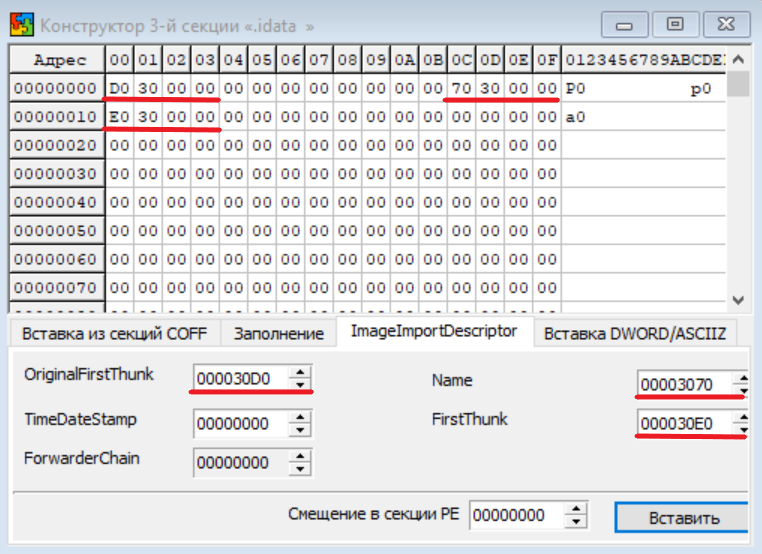
Для связи любого Windows-приложения с системными библиотеками, в которых содержатся функции API, используется специальная структура данных, называемая таблицей импорта. Эта таблица содержит описание всех DLL, используемых при работе приложения, и данные обо всех функциях, импортируемых приложением из этих DLL.

Таблица импорта может размещаться в любой секции, доступной для чтения и записи. Её можно помещать и в имеющиеся секции данных или в специально созданную секцию.

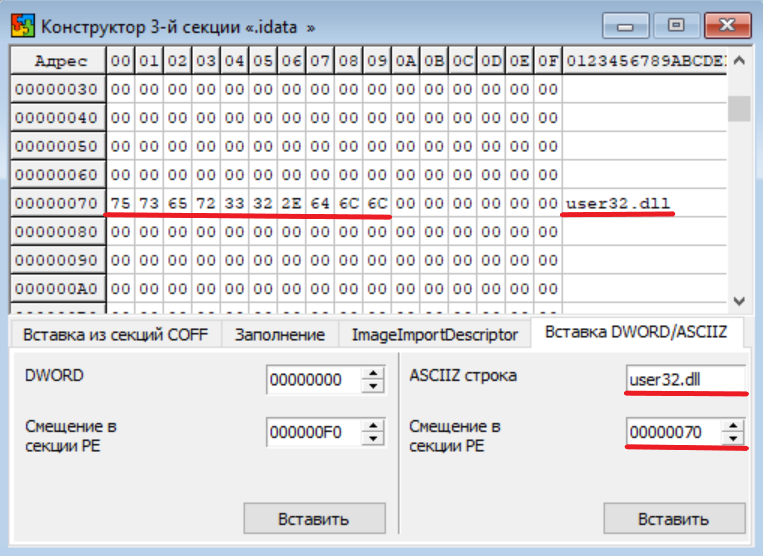
Перед построением таблицы импорта необходимо составить список всех импортируемых функций и всех DLL, содержащих эти функции.

Далее необходимо создать саму таблицу, пользуясь вышеуказанной информацией и теоретическими сведениями об импортировании в PE-файлах. Пример создания таблицы импорта приведён ниже.

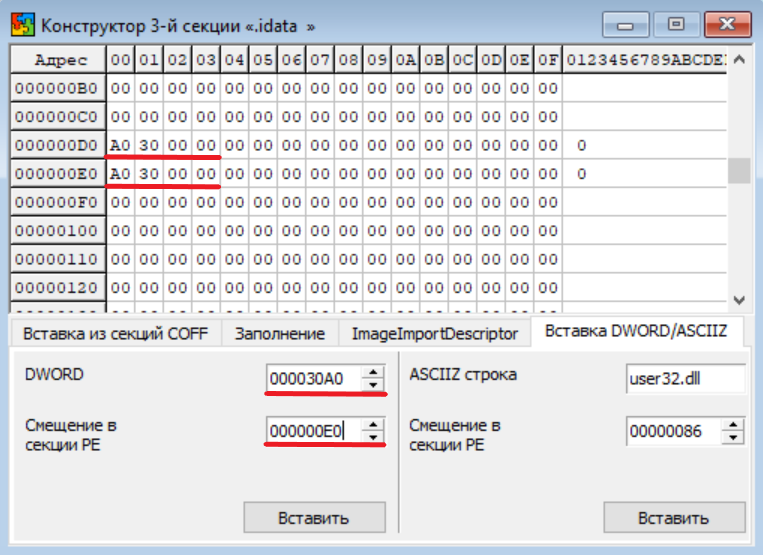
В данной программе есть две функции, которые импортируются из библиотек. MessageBoxA находится в библиотеке user32.dll, ExitProcess находится в библиотеке kernel32.dll. Соответственно в данном примере необходимо подключить две библиотеки. В разделе «Секции» в секцию .idata необходимо загрузить две структуры ImageImportDescriptor по одной для каждой библиотеки. Поле OriginFirstThunk должно содержать адрес, куда позже будет помещена первая структура FirstThunk, относительно начала PE-файла. Например, адрес 30D0. Тогда в поле FirstThunk укажем адрес копии данной структуры 30E0. Поле Name должно содержать адрес, куда в дальнейшем будет записано имя подключаемой библиотеки, относительно начала PE-файла. Например, адрес 3070. Смещение в секции PE принять равным 0.



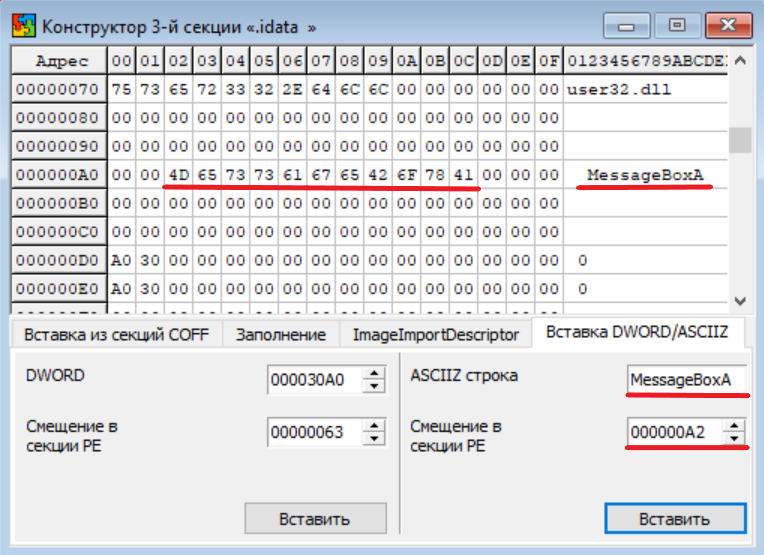
Далее по адресу из поля Name во вкладке «Вставка ASCIIZ» записываем название импортируемой библиотеки.



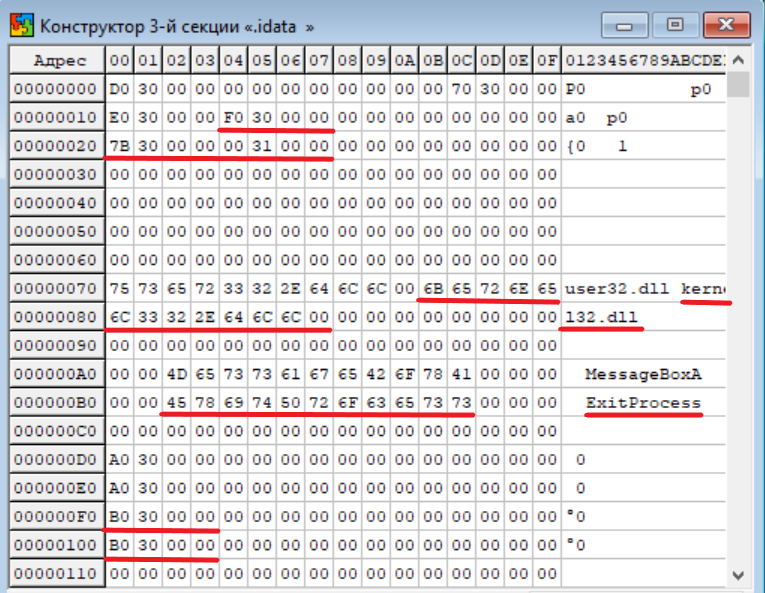
После этого необходимо по адресу из поля FirstThunk и OriginalFirstFunk создать структуру, которая будет состоять из последовательной записи четырёхбайтных адресов, указывающих на названия функций, которые будут вызваны из этой библиотеки. Названия функций будут прописаны позже, поэтому примем этот адрес равным 30A0. Необходимо вставить это значение по адресу OriginFirstFunk (и продублировать это значение по адресу FirstFunk) на вкладке «Вставка DWORD».



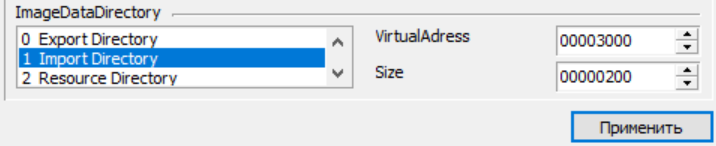
Далее по адресу, который был записан в структуру FirstThunk записываем название необходимых функций, со смещением в два байта.



Такую структуру необходимо создать подобным образом для второй библиотеки. Вторая структура ImageImportDescriptor записывается сразу после второй, т.е. со смещением секции PE равным 14. Название второй библиотеки записывается через байт после названия первой.



Адрес точки входа RVA и размер созданной таблицы импорта необходимо поместить в элемент «Import Directory» структуры «ImageDataDirectory» дополнительного заголовка. VirtualAdress и Size таблицы импорта ранее был занесён в заголовок секции .idata.

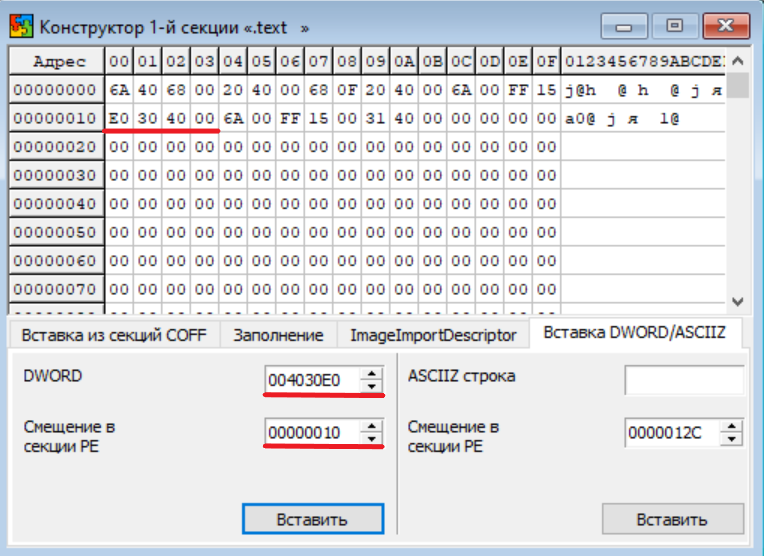


После создания таблицы необходимо разрешить все оставшиеся внешние ссылки.

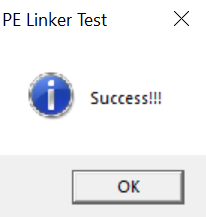
По значению поля SymIndex ссылки найти соответствующий ей элемент в SYMBOL\_TABLE. Если тип данной ссылки STATIC, он уже должен быть разрешён, перейти к проверке остальных ссылок. Если тип ссылки EXTERNAL, ссылку необходимо разрешить.

Для таких ссылок необходимо сосчитать адрес, который нужно вставить в секцию .text. Адрес вычисляется как сумма ImageBase из «Дополнительного заголовка» и смещения относительно начала PE-файла ячейки с элементом массива FirstThunk, куда была помещена ссылка на название необходимой в данном случае функции. Вычислив адрес, необходимо перейти в разделе «Секции» в секцию .text, на вкладку «Вставка DWORD/ASCIIZ». Записать полученное значение в поле «DWORD», указать смещение в секции PE-файла равным значению Address из таблицы привязок (RELOCATION) для данной ссылки. На данном этапе необходимо таким образом разрешить все ссылки типа EXTERNAL. Пример разрешения ссылки приведён ниже.

ImageBase равен 400000. В SYMBOL\_TABLE поле Name первой неразрешённой ссылки типа EXTERNAL указано значение MessageBoxA, что соответствует элементу массива FirstThunk со смещением 30E0. При сложении данных значений получаем 4030E0. Значение поля Address из таблицы привязок (RELOCATION) равен 10.



Если все шаги сделаны правильно, то после компоновки в каталоге проекта появится исполняемый файл, работоспособность которого необходимо проверить, запустив его на выполнение.



## Загрузка программы отладчиком SoftICE

Для защиты лабораторной работы необходимо продемонстрировать загрузку полученной программы с помощью системного отладчика SoftICE.

Чтобы загрузить приложение с помощью SoftICE необходимо открыть использовать Symbol Loader из комплекта поставки отладчика. Открыв исследуемый модуль в этой утилите необходимо запустить его на выполнение с помощью команды Translate.

Когда появится «окно» SoftICE, то необходимо продемонстрировать работу следующих команд:

- d <адрес> - отображение информации (дампа) по указанному адресу;

- u <адрес> - дизассемблирование кода по указанному адресу;

- map32 <имя процесса> - отображение информации об указанном процессе;

- mod <имя модуля> - отображение информации об указанном модуле.

Также необходимо продемонстрировать процесс трассировки программы с заходом в подпрограммы и без него.

## Примерные задания для выполнения лабораторной работы

|  |  |  |
| --- | --- | --- |
| № | Функция для получения системной информации | Адрес загрузки |
| 1 | Получить имя компьютера с помощью функции  BOOL GetComputerNameA(  LPTSTR lpBuffer, // Адрес буфера  LPDWORD nSize // Адрес размера буфера  ); | 00400000h |
| 2 | Получить имя компьютера с помощью функции  BOOL GetComputerNameA(  LPTSTR lpBuffer, // Адрес буфера  LPDWORD nSize // Адрес размера буфера  ); | 00500000h |
| 3 | Получить имя текущего каталога  DWORD GetCurrentDirectoryA(  DWORD nBufferLength, // Размер буфера в символах  LPTSTR lpBuffer // Адрес буфера для тек. каталога  ); | 00600000h |
| 4 | Получить имя текущего каталога  DWORD GetCurrentDirectoryA(  DWORD nBufferLength, // Размер буфера в символах  LPTSTR lpBuffer // Адрес буфера для тек. каталога  ); | 00700000h |
| 5 | Получить список значение переменной окружения PATH.  DWORD GetEnvironmentVariableA(  LPCTSTR lpName, // Адрес строки с именем переменной  LPTSTR lpBuffer, // Адрес буфера для значения переменной  DWORD nSize // Размер буфера в символах  ); | 00800000h |
| 6 | Получить имя активной раскладки клавиатуры  BOOL GetKeyboardLayoutNameA(  LPTSTR pwszKLID // Адрес буфера для раскладки  );  Примечание: Размер буфера не меньше 9 символов | 00900000h |
| 7 | Получить путь к текущему временному каталогу  DWORD GetTempPathA(  DWORD nBufferLength, // Размер буфера в символах  LPTSTR lpBuffer // Адрес буфера для временного каталога  ); | 00B00000h |
| 8 | Получить каталог Windows  UINT GetWindowsDirectoryA(  LPTSTR lpBuffer, // Адрес буфера для имени каталога  UINT uSize // Размер буфера  );  Примечание: Размер буфера не меньше 260 символов | 00C00000h |
| 9 | Получить имя системного каталога Windows  UINT GetSystemDirectoryA(  LPTSTR lpBuffer, // Адрес буфера для системного каталога  UINT uSize // Размер буфера  );  Примечание: Размер буфера не меньше 260 символов | 00400000h |
| 10 | Получить имя текущего пользователя  BOOL GetUserNameA(  LPTSTR lpBuffer, // Адрес буфера для имени пользователя  LPDWORD nSize // Адрес размера буфера  ); | 00500000h |
| 11 | Получение числа тактов процессора после запуска системы  DWORD GetTickCount(VOID)  Значение возвращается в регистровой паре edx:eax | 00600000h |
| 12 | Функция создает или открывает каталог, физический диск, том, буфер консоли (CONIN$ или CONOUT$), устройство на магнитной ленте, коммуникационный ресурс, почтовый слот или именованный канал.  DWORD CreateFile(  LPCTSTR lpFileName, // имя файла  DWORD dwDesiredAccess, // режим доступа  DWORD dwShareMode, // совместный доступ  LPSECURITY\_ATTRIBUTES lpSecurityAttributes, // SD (дескр. защиты)  DWORD dwCreationDisposition, // как действовать  DWORD dwFlagsAndAttributes, // атрибуты файла  HANDLE hTemplateFile //дескр.шаблона файла  );  Функция возвращает дескриптор, который может быть использован для доступа к объекту. | 00700000h |
| 13 | Функция GetLogicalDriveStrings заполняет буфер строками, указывающими допустимые диски в системе  DWORD GetLogicalDriveStrings(  DWORD nBufferLength, // размер буфера  LPTSTR lpBuffer // адрес буфера для загрузки строк  );  Функция возвращает имена логических дисков | 00800000h |
| 14 | Функция GetLogicalDriveStringsW заполняет буфер строками, указывающими допустимые диски в системе  DWORD GetLogicalDriveStringsW(  DWORD nBufferLength, // размер буфера  LPTSTR lpBuffer //адрес буфера для загрузки строк  );  Функция возвращает имена логических дисков | 00900000h |
| 15 | Получить каталог Windows в многопользовательском режиме  UINT GetSystemWindowsDirectoryA(  LPTSTR lpBuffer, // Адрес буфера для имени каталога  UINT uSize // Размер буфера  ); | 00А00000h |
| 16 | Форматирует строку сообщения.  DWORD FormatMessage (  DWORD nSize, // размер буфера  LPTSTR lpBuffer // адрес буфера для загрузки строки  ); | 00B00000h |
| 17 | Функция GetLongPathNameA по короткому имени пути извлекает длинный путь  DWORD GetLongPathName(  LPCTSTR *lpszShortPath*, // строка пути с нулем в конце  LPTSTR *lpszLongPath*, // буфер короткой формы  DWORD *cchBuffer* // размер буфера короткой формы  ); | 00С00000h |
| 18 | Функция FormatMessageA требует определения сообщения как вводимых данных. Определение сообщения может прийти из буфера, который передается в функцию, формирует строку сообщения.  DWORD FormatMessage (  DWORD nSize, // размер буфера выводимых данных  LPTSTR lpBuffer // указатель на буфер, который получает строку с завершающим нулем, устанавливающую форматированное сообщение  ); | 00D00000h |
| 19 | GetLogicalDriveStrings  DWORD GetLogicalDriveStringsW(  DWORD nBufferLength, // входной параметр – размер строки  LPTSTR lpBuffer // выходной параметр – адрес буфера строки  );  Функция возвращает буфер строки из установленных логических дисков на компьютере | 00400000h |
| 20 | GetShortPathName  DWORD WINAPI GetShortPathName (  LPCTSTR lpszLongPath, //Путь строки  LPTSTR lpszShortPath, //Указатель на буфер для получения нулевым краткую форму пути, который указывает lpszLongPath  DWORD cchBuffer); //Размер буфера, который указывает на lpszShortPath в TCHARs  );  Функция возвращает краткую форму пути по указанному пути. | 00500000h |
| 21 | PUSH OFFSET IDP  PUSH DWORD PTR [EBP+8]  CALL GetWindowThreadProcessId@8    Функция GetWindowThreadProcessId  Синтаксис:   |  | | --- | | DWORD GetWindowThreadProcessId(  Wnd: HWND; //Идентификатор окна  lpdwProcessId : LPDWORD //32-битное значение идентификатора процесса  ); |   Функция возвращает идентификатор процесса к которому принадлежит данное окно  Возвращаемое значение: идентификатор нити  Эта функция хороша тем, что для всех окон одного приложения этот идентификатор будут един, если они запущены внутри одной нити (что зачастую и делается). То есть Вы можете разбить окна по нитям и /или по процессам, к которым они относятся. А также вы можете определять, какие окна данного приложения в настоящий момент доступны, видны, свернуты и т.д. | 00600000h |
| 22 | Смена генерируемых событий между кнопками мыши  BOOL SwapMouseButton(  BOOL fSwap //Если сменить, то передавать true  ); | 00700000h |
| 23 | Установить время между нажатиями мыши при двойном клике  BOOL SetDoubleClickTime(  UINT uInterval //Интервал времени  ); | 00800000h |
| 24 | Получить количество процессов в системе  WORD GetMaximumProcessorGroupCount(); | 00900000h |
| 25 | Установить имя компьютера  BOOL SetComputerName(  LPCTSTR lpComputerName //Адрес строки с именем  ); | 00А00000h |
| 26 | Установить время между нажатиями мыши при двойном клике  UINT GetDoubleClickTime(); | 00В00000h |
| 27 | Создать новую очередь сообщений прикладной задачи указанного размера. Старая очередь удаляется.    BOOL SetMessageQueue(  UINT Msg //Размер очереди  ); | 00С00000h |
| 28 | Установить код последней ошибки для вызванного потока  void SetLastError(  DWORD dwErrCode //Код ошибки  ); | 00400000h |
| 29 | Получить идентификатор текущего языка  LANGID GetUserDefaultUILanguage(); | 00500000h |
| 30 | Получить текущий тип даты в строковом представлении заданного формата. Возвращает количество символов в выходной строке.  UINT GetDateFormatA  (  LCID lcid, //Идентификатор языка  DWORD dwFlags, //Флаги формата даты  const SYSTEMTIME\* lpTime, //Указатель на информацию о дате в формате  LPCSTR lpFormat, //Формат строки  LPSTR lpDateStr, //Указатель на выходную строку  INT cchOut //Размер выходной строки  ) | 00600000h |
| 31 | Установить атрибуты для файла или каталога  BOOLSetFileAttributes(  LPCTSTR lpFileName, //Адрес строки с путем к файлу  DWORD dwFileAttributes //Байты атрибутов  ); | 00700000h |
| 32 | Получить атрибуты файла или каталога  DWORD GetFileAttributes(  LPCTSTR lpFileName //Адрес строки с путем к файлу  ); | 00800000h |
| 33 | Получить размер свободного пространства на диске  BOOL GetDiskFreeSpace(  LPCTSTR lpRootPathName, //Адрес строки с путем к корневому каталогу диска  LPDWORD lpSectorsPerCluster, //Адрес числа, содержащего количество секторов в кластере  LPDWORD lpBytesPerSector, //Адрес числа, содержащего количество байт в кластере  LPDWORD lpNumberOfFreeClusters, //Адрес числа, содержащего количество свободных кластеров  LPDWORD lpTotalNumberOfClusters //Адрес числа, содержащего общее количество кластеров  ); | 00900000h |
| 34 | Воспроизвести звуковой сигнал  BOOL Beep(  DWORD dwFreq, //Частота звучания, Гц  DWORD dwDuration //Длительность звучания  ); | 00А00000h |
| 35 | Определяет позицию курсора  BOOL GetCursorPos(  LPPOINT lpPoint //адрес структуры (X; Y), куда запишется координата курсора  ); | 00В00000h |
| 36 | Получить имя компьютера с помощью функции  BOOL GetComputerNameA(  LPTSTR lpBuffer, // Адрес буфера  LPDWORD nSize // Адрес размера буфера  ); | 00400000h |
| 37 | Получить имя компьютера с помощью функции  BOOL GetComputerNameA(  LPTSTR lpBuffer, // Адрес буфера  LPDWORD nSize // Адрес размера буфера  ); | 00500000h |
| 38 | Получить имя текущего каталога  DWORD GetCurrentDirectoryA(  DWORD nBufferLength, // Размер буфера в символах  LPTSTR lpBuffer // Адрес буфера для тек. каталога  ); | 00600000h |
| 39 | Получить имя текущего каталога  DWORD GetCurrentDirectoryA(  DWORD nBufferLength, // Размер буфера в символах  LPTSTR lpBuffer // Адрес буфера для тек. каталога  ); | 00700000h |
| 40 | Получить список значение переменной окружения PATH.  DWORD GetEnvironmentVariableA(  LPCTSTR lpName, // Адрес строки с именем переменной  LPTSTR lpBuffer, // Адрес буфера для значения переменной  DWORD nSize // Размер буфера в символах  ); | 00800000h |
| 41 | Получить имя активной раскладки клавиатуры  BOOL GetKeyboardLayoutNameA(  LPTSTR pwszKLID // Адрес буфера для раскладки  );  Примечание: Размер буфера не меньше 9 символов | 00900000h |
| 42 | Получить путь к текущему временному каталогу  DWORD GetTempPathA(  DWORD nBufferLength, // Размер буфера в символах  LPTSTR lpBuffer // Адрес буфера для временного каталога  ); | 00A00000h |
| 43 | Получить каталог Windows  UINT GetWindowsDirectoryA(  LPTSTR lpBuffer, // Адрес буфера для имени каталога  UINT uSize // Размер буфера  );  Примечание: Размер буфера не меньше 260 символов | 00B00000h |
| 44 | Получить имя системного каталога Windows  UINT GetSystemDirectoryA(  LPTSTR lpBuffer, // Адрес буфера для системного каталога  UINT uSize // Размер буфера  );  Примечание: Размер буфера не меньше 260 символов | 00C00000h |
| 45 | Получить имя текущего пользователя  BOOL GetUserNameA(  LPTSTR lpBuffer, // Адрес буфера для имени пользователя  LPDWORD nSize // Адрес размера буфера  ); | 00D00000h |

Примечания:

Все функций API используют модель вызова STDCALL:

а) параметры передаются в стек в обратном порядке;

б) все параметры имеют размер 4 байта (DD, DWORD);

в) стек очищает вызываемая функция;

г) результат возвращается в регистре EAX.

Функция MessageBoxA

int MessageBoxA(

HWND hWnd, // Ссылка на родительское окно (в данном случае 0)

LPCTSTR lpText, // Адрес строки с текстом сообщения

LPCTSTR lpCaption, // Адрес строки с текстом заголовка окна

UINT uType // Стиль окна (рекомендуется 40h)

);

Функция ExitProcess

VOID ExitProcess(

UINT uExitCode // Код завершения (0 в случае успеха)

);

MessageBoxA и GetKeyboardLayoutNameA импортируется из user32.dll, GetUserNameA – из advapi32.dll, все остальные функции – из kernel32.dll

Имена функций чувствительны к регистру!

Все функции объявлять с помощью директивы “EXTRN <имя функции>: DWORD“

Пример:

extrn MessageBoxA:DWORD ; Функция MessageBox для формата ANSI

# Заключение

В языках высокого уровня хранятся некоторые встроенные заголовочные файлы или библиотеки. Эти библиотеки предопределены и содержат основные функции, необходимые для выполнения программы. Эти функции связаны с библиотеками программой под названием Linker. Если компоновщик не находит библиотеку функции, он сообщает об этом компилятору, а затем компилятор генерирует ошибку. Компилятор автоматически вызывает компоновщик в качестве последнего шага при компиляции программы. Не встроенные библиотеки, он также связывает пользовательские функции с пользовательскими библиотеками. Обычно более длинная программа делится на более мелкие подпрограммы, называемые модулями. И эти модули должны быть объединены для выполнения программы. Процесс объединения модулей выполняется компоновщиком.

Загрузчик — это программа, которая загружает машинные коды программы в системную память. В вычислительной технике загрузчик — это часть операционной системы, которая отвечает за загрузку программ. Это один из важнейших этапов в процессе запуска программы. Потому что он помещает программы в память и подготавливает их к выполнению. Загрузка программы включает в себя чтение содержимого исполняемого файла в память. После завершения загрузки операционная система запускает программу, передавая управление загруженному программному коду. Все операционные системы, поддерживающие загрузку программ, имеют загрузчики. Во многих операционных системах загрузчик постоянно находится в памяти.

Библиографический список

Список сокращений