МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕКДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

по дисциплине

«Системное програмирование»

Методические указания к выполнению лабораторных работ

Направление подготовки 09.03.02

«Информационные системы и технологии»

Направленность (профиль) «Информационные системы управления технологическими и сервисными процессами»

Невинномысск 2026

Методические указания предназначены для студентов очно-заочной формы обучения специальности 09.03.02 Информационные системы и технологии и других технических специально-стей. Они содержат основы теории, порядок проведения практических работ и обработки экспериментальных данных, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины «Системное программирование». Объем и последовательность выполнения работ определяются преподавателем в зависимости от количества часов, предусмотренных учебным планом дисциплины, как для очной, так и для заочной форм обучения.

Методические указания разработаны в соответствии с требованиями Федерального Государственного образовательного стандарта в части содержания и уровня подготовки выпускников по направлению подготовки 09.03.02 Информационные системы и технологии

Код	Формулировка
ПК-3	Реализация и модификация компонентов информационных систем для автоматизации бизнес-процессов и организационного управления

Составитель: канд. техн. наук, доцент А.А. Евдокимов

Ответственный редактор: канд. техн. наук, доцент Д.В. Болдырев

Содержание

Практическая работа № 1

Разработка оконного приложения на языке низкого уровня

Цель работы: Получить навык создания оконных приложений на языке ассемблера 32-хразрядного процессора Intel.

Краткие сведения из теории

Минимально оконное приложение Windows состоит из трех частей.

1. Выполнение стартового кода.

2. Выполнение главной функции, которая выполняет следующие действия:

2.1. регистрирует класс окна;

2.2. создает окно;

2.3. отображает окно;

2.4. запускает цикл обработки сообщений;

2.5. завершает выполнение приложения.

3. Организация обработки сообщений в оконной процедуре.

Выполнение любого оконного Windows-приложения начинается с главной функции. Она содержит код, осуществляющий настройку (инициализацию) приложения в среде операционной системы Windows. Видимым для пользователя результатом работы главной функции является появление на экране графического объекта в виде окна. Последним действием кода главной функции является создание цикла обработки сообщений. После его создания приложением становится пассивным и начинает взаимодействовать с внешним миром посредством специальным образом оформленных данных – сообщений. Обработка поступающих приложению сообщений осуществляется специальной процедурой, называемой оконной. Оконная процедура уникальна тем, что может быть вызвана только из операционной системы, а не из приложения, которое ее содер-

жит (функция обратного вызова). Тело оконной процедуры также имеет определенную структуру.

Вызов функций Win32 API осуществляется аналогично вызову внешних функций, а передача параметров осуществляется через стек. Передача параметров производится в соответствии со стилем stdcall – параметры в стек помещается справа налево, т.е. первым в стек идет последний параметр функции, и удаляются из стека по завершению работы процедуры.

Модель сегментации, используемая в программах Win32 – flat, плоская модель памяти. В соответствии с этой моделью компилятор создает программу, которая содержит одни 32-разрядный сегмент для данных и кода программы. Код загрузочного модуля будет работать на процессорах i386 и выше. По этой причине директиве .MODEL должна предшествовать одна из директив: .386, .486 или .586.

В программе с плоской моделью памяти используется адресация программного кода и данных типа near. Это же касается и атрибута расстояния в директиве PROC, которые также имеет тип near.

Функции Win32 API должны быть объявлены внешними с помощью директивы EXTERN. Например,

```
EXTERN   MessageBoxA@16: NEAR
EXTERN   CreateWindowExA@48: NEAR
EXTERN   DefWindowProcA@16: NEAR
```

Рассмотрим запись имени функции, например функции вывода на экран сообщения с кнопкой MessageBox: MessageBoxA@16.

Для работы с однобайтной (ANSI) и двухбайтной (UNICODE) кодировкой символов программный интерфейс Win32 API имеет два варианта функций, которые различаются последним символом в названии. Ес-

ли это «A», то данная функция работает в кодировке ANSI, если «W» – UNICODE.

Наличие суффикса «Ex», говорит об использовании расширенного варианта функций, которые обладают дополнительными возможностями по сравнению со старыми вариантами функций Win32 API. Исчерпывающим источником информации по функциям Win32 API является MSDN (Microsoft Developer Network – информационная система поддержки разработчика по продуктам Microsoft), его можно найти в Интернете по адресу http://www.microsoft.com/msdn/.

При использовании MASM необходимо также в конце имени функции добавить @N, где N – размер всех параметров в байтах, передаваемых функции при ее вызове. Размер одного параметра – четыре байта.

Стартовый код

Стартовый код представляет собой последовательный вызов функций Win32 API, которые могут быть использованы для анализа информации о версии Windows, получения указателя на командную строку, получения указателя на блок с переменными окружения, получения значения базового адреса (дескриптора), по которому загружен модуль и т.д.

Функция GetModuleHandle (ModuleName: PChar): THandle считывает описатель модуля. Ее единственный параметр – адрес ASCIIZ-строки с именем исполняемого файла, базовый адрес загрузки которого необходимо получить.

Поскольку в адресное пространство процесса можно загрузить несколько файлов, для работы с ними требуется механизм их однозначной идентификации. При загрузке исполняемого файла в адресное пространство процесса ему присваивается уникальный номер. Этот номер (Handle Instance, hInst – описатель экземпляра) используется при вызове многих

функций Win32 API, загружающих те или иные ресурсы для данной программы. Значение hInst равно базовому адресу в адресном пространстве процесса, по которому загружен данный файл с расширением .exe.

При вызове функции GetModuleHandleA@4, передав ей значение NULL, получим значение hInst для текущей программы.

Регистрация класса окна

Под классом окна понимается совокупность присущих ему характеристик, таких как стиль его границ, формы указателя мыши, значков, цвет фона, наличие меню, адрес оконной процедуры, обрабатывающей сообщения этого окна.

Регистрация класса окон осуществляется с помощью функции RegisterClassA, единственным параметром которой является указатель на структуру WNDCLASS, содержащую информацию об окне.

```
WNDCLASS  STRUC
    CLSSTYLE        DD  ?    ;стиль окна
    CLWNDPROC       DD  ?    ;указатель на процедуру окна
    CLSCSEXTRA      DD  ?    ;информация о доп. байтах для
                             ;данной структуры
    CLWNDEXTRA      DD  ?    ;информация о доп. байтах для окна
    CLSHINSTANCE    DD  ?    ;дескриптор приложения
    CLSHICON        DD  ?    ;идентификатор иконы окна
    CLSHCURSOR      DD  ?    ;идентификатор курсора окна
    CLBKGROUND      DD  ?    ;идентификатор кисти окна
    CLMENUNAME      DD  ?    ;имя-идентификатор меню
    CLNAME          DD  ?    ;специфицирует имя класса окон
WNDCLASS  ENDS
```

Таблица 1.1 – Константы стиля окна

Константа	Значение
CS_BYTEALIGNCLIENT	Использование границы по байту по оси X. Выравнивание клиентской области окна
CS_BYTEALIGNWINDOW	Использование границы по байту по оси X. Выравнивание окна
CS_CLASSDC	Классу окна присваивается собственный контекст изображения, который можно разделить между копиями.
CS_DBCLKS	Окну будут посылаться сообщения о двойном щелчке кнопки «мыши».
CS_GLOBALCLASS	Определяется глобальный класс окон.
CS_HREDRAW	Обеспечивается перерисовка содержимого клиентской области окна при изменении размера окна по горизонтали.
CS_KEYCVTWINDOW	Будет выполняться преобразование виртуальных клавиш.
CS_NOCLOSE	В системном меню блокируется выбор пункта для закрытия окна.
CS_NOKEYCVT	Отключается преобразование виртуальных клавиш.
CS_OWNDC	Каждому экземпляру окна присваивается собственный контекст изображения.
CS_PARENTDC	Классу окна передается контекст изображения родительского окна.
CS_SAVEBITS	Часть изображения на экране, закрытая окном, сохраняется.
CS_VREDRAW	Обеспечивается перерисовка содержимого клиентской области окна при изменении размера окна по вертикали.

Поле CLSSTYLE определяет стиль границ окна и его поведение при перерисовке. Значения стиля является целочисленным и формируется из констант (таблица 1.1). Константы данного поля и других полей можно найти в файле C:\masm32\INCLUDE\RESOURCE.H.

Поле CLWNDPROC – значение указателя на функцию окна, которая выполняет все задачи, связанные с окном.

Поле CLSCSEXTRA и CLWNDEXTRA служат для указания количества байтов, дополнительно резервируемых в структуре класса окна WNDCLASS и выделяемых для всех дополнительных структур, которые создаются с использованием данного класса окна. Обычно эти поля инициализированы нулевыми значениями (NULL).

В поля CLSHICON и CLSHCURSOR загружаются дескрипторы значка и указателя мыши. После запуска приложения значок будет отображаться на панели задач Windows и в левом верхнем углу окна приложения, а указатель мыши появится в области окна. Значки и указатели мыши представляют собой ресурсы и находятся в отдельных файлах. Для загрузки поименованного ресурса курсора используется функция

LoadCursor(Instance: THandle; CursorName: PChar): HCursor,

параметрами которой являются Instance – экземпляр модуля, исполнимый файл которого содержит курсор, или 0 для предопределенного курсора; CursorName – строка (заканчивающаяся пустым символом) или имя целочисленного идентификатора или предопределенный курсор, определенный одной из констант idc_.

Для загрузки поименованного ресурса пиктограммы используется функция

LoadIcon(Instance: THandle; IconName: PChar): HIcon,

параметрами которой являются Instanc – экземпляр модуля, исполнимый файл которого содержит пиктограмму, или 0 для предопределенной пиктограммы; IconName – строка или имя целочисленного идентификатора или предопределенная пиктограмма, определенная одной из констант idi_.

Поле CLBKGROUND определяет кисть, используемую для закраски фона окна. Значением данного параметра может быть как идентифика-

тор физической кисти, так и значение цвета (от 1 до 29). При использовании значения цвета нужно выбрать одно из следующего списка и прибавить к нему 1 (значения данных констант можно найти в файле C:\masm32\INCLUDE\WINDOWS.INC):

COLOR_SCROLLBAR; COLOR_BACKGROUND;

COLOR_ACTIVECAPTION; COLOR_iNACTIVECAPTION;

COLOR_MENU; COLOR_WINDOW; COLOR_WINDOWFRAME;

COLOR_MENUTEXT; COLOR_WINDOWTEXT;

COLOR_CAPTIONTEXT; COLOR_ACTIVEBORDER;

COLOR_INACTIVEBORDER; COLOR_APPWORKSPACE;

COLOR_HIGHLIQHT; COLOR_HIGHLIGHTTEXT;

COLOR_BTNFACE; COLOR_BTNSHADOW;

COLOR_GRAYTEXT; COLOR_DTNTEXT;

COLOR_INACTIVECAPTIONTEXT; COLOR_BTNHIGHLIGHT.

В поле CLMENUNAME записывается указатель на ASCIIZ-строку с именем меню. Если меню не используется, то в поле записывается значение NULL.

Поле CLNAME содержит длинный указатель на ASCIIZ-строку, которая определяет имя класса. Имя класса должно быть уникальным, чтобы не возникало проблем при разделении класса между приложениями.

После инициализации структуры регистрируется класс окна в системе. После регистрации класса окна структура WNDCLASS больше не нужна.

Создание окна

На основе зарегистрированного класса с помощью функции CreateWindowExA (или CreateWindowA) можно создать экземпляр окна.

Функция CreateWindow(ExStyle: Longint; ClassName, WindowName: PChar; Style: Longint; X, Y, Width, Height: Integer; WndParent: HWnd; Menu: HMenu; Instance: THandle; Param: Pointer):HWnd

создает перекрытое, всплывающее или дочернее окно с расширенным стилем.

Параметры функции CreateWindowExA:

ExStyle – один из следующих расширенных стилей окна: ws_ex_DlgModalFrame, или ws_ex_NoParentNotify. Позволяет задать дополнительные стили окна;

ClassName – указатель на ASCIIZ-строку с именем класса окна или предопределенное имя класса органа управления;

WindowName – указатель на ASCIIZ-строку с текстом, помещаемым в заголовок окна;

Style – одна из констант стиля окна или органа управления или их комбинация. К этим константам относятся константы ds_, ws_, bs_, cbs_, es_, lbs_, sbs_, ss_;

X, Y: – начальное положение окна или cw_UseDefault (80000000h);

Width – Начальная ширина окна (в единицах устройства);

Height – начальная высота окна (в единицах устройства);

WndParent – дескриптор родительского окна. Между двумя окнами Windows-приложения можно установить родовые отношения. Дочернее окно всегда должно появляться в области родительского окна;

Menu – идентификатор главного меню или дочернего окна;

Instance – дескриптор приложения создающего окно;

Param – используется при создании окна для передачи данных или указателя на них в оконную функцию. Все параметры, передаваемые функцией CreateWindowExA, сохраняются в создаваемой Windows внут-

ренней структуре TCreateStruct. Поля этой структуру идентичны параметрам функции CreateWindowExA. Указатель на структуру TCreateStruct передается оконной функции при обработке сообщения wm_Create. Сам указатель находится в поле lParam сообщения. Значение параметра Param функции CreateWindowExA находится в поле lCreateParams структуры TCreateStruct. Для создания дочернего окна MDI должно быть указателем на структуру TClientCreateStruct.

В случае успешного завершения функция возвращает дескриптор окна, в противном случае – 0.

Отображение окна

Для появления созданного окна на экране необходимо применить функцию ShowWindowA:

ShowWindow(Wnd: HWnd; CmdShow: Integer),

которая отображает или прячет окно образом, указанным параметром CmdShow. Wnd – идентификатор окна. CmdShow – одна из констант sw_, в зависимости от значения которой окно отображается в стандартном виде, развернутом на весь экран, свернутым в значок и т.д.

Функция UpdateWindow(Wnd: HWnd) посылает сообщение wm_Paint прямо оконной функции данного окна, если область обновления окна не пуста.

Цикл обработки сообщений

Windows поддерживает очередь сообщений для каждого приложения. Запуск приложения автоматически подразумевает формирование для него очереди.

Формат всех сообщений Windows одинаков и описывается структурой

MSGSTRUCT STRUC

```
MSHWND      DD  ?   ;идентификатор окна,
                    ;получающего сообщение
MSMESSAGE   DD  ?   ;идентификатор сообщения
MSWPARAM    DD  ?   ;доп. информация о сообщении
MSLPARAM    DD  ?   ;доп. информация о сообщении
MSTIME      DD  ?   ;время посылки сообщения
MSPT        DD  ?   ;положение курсора, во время
                    ;посылки сообщения
MSGSTRUCT  ENDS
```

Поле MSHWND содержит значение дескриптора окна, которому предназначено сообщение. Это дескриптор, возвращаемый функцией CreateWindowExA, однозначно идентифицирует окно в системе. приложение обычно имеет несколько окон, поэтому значение в поле MSHWND помогает приложению идентифицировать нужное окно.

В поле MSMESSAGE Windows помещает 32-разрядную константу – идентификатор сообщения, однозначно идентифицирующий тип сообщения. Все эти константы имеют символические имена, начинающиеся с префикса WM_ (Window Message). В программе на языке C/C++ эти константы используются в оконной функции оператором switch для принятия решения о том, какая из его ветвей будет исполняться. В программе на языке ассемблера этот оператор моделируется командами условного и безусловного переходов, а также командой cmp, в качестве второго операнда которой и выступает константа, обозначающая определенный тип сообщения.

Поля MSLPARAM и MSWPARAM предназначены для того, чтобы система Windows могла разместить в них дополнительную информацию о сообщении, необходимую для ее правильной обработки. Эти поля,

например, используются при обработке сообщений о выборе пунктов меню или о нажатии клавиш.

В поле MSTIME Windows записывает информацию о времени, когда сообщение было помещено в очередь сообщений.

Поле MSPT содержит координаты указателя мыши в момент помещения сообщения в очередь.

Функция GetMessage(var Msg: TMsg; Wnd: HWnd; MsgFilterMin, MsgFilterMax: Word): Bool считывает сообщение, в рамках диапазона фильтрации, из очереди сообщений прикладной задачи. Функция оставляет управление другим прикладным задачам, если сообщений нет или если следующим сообщением является wm_Paint или wm_Timer.

Параметры функции GetMessageA:

Msg – принимающая структура MSG;

Wnd – дескриптор окна, сообщения для которого должны будут выбираться функцией GetMessageA, или 0 для всех окон в прикладной задаче;

MsgFilterMin – задает минимальное значение параметра MSMESSAGE, нуль в случае отсутствия фильтрации, или wm_KeyFirst только для клавиатуры, или wm_MouseFirst только для мыши;

MsgFilterMax – задает максимальное значение параметра MSMESSAGE, нуль в случае отсутствия фильтрации, или wm_KeyLast только для клавиатуры, или wm_MouseLast только для мыши.

Функция GetMessageA выполняет следующие действия.

1. Постоянно просматривает очередь сообщений.

2. Выбирает сообщения, удовлетворяющие заданным в функции параметрам.

3. Заносит информацию о сообщении в экземпляр структуры MSG.

4. Передает управление в цикл обработки сообщений.

Цикл обработки сообщений состоит всего из двух функций: TranslateMessage и DispatchMessageA. Эти функции имеют единственный параметр – указатель на экземпляр структуры MSG, предварительно заполненный информацией о сообщении функцией GetMessageA.

Функция TranslateMessage предназначена для обнаружения сообщений от клавиатуры для данного приложения. Если приложение самостоятельно не обрабатывает ввод с клавиатуры, то эти сообщения передаются для обработки обратно Windows. Данная функция переводит комбинации wm_KeyDown/Up в wm_Char или wm_DeadChar и комбинации wm_SysKeyDown/Up в wm_SysChar или wm_SysDeadChar и направляет символьное сообщение в очередь прикладной задачи.

Функция DispatchMessageA предназначена для передачи сообщения оконной процедуре. Такая передача производится не напрямую, так как сама DispatchMessageA ничего не знает о месторасположении оконной процедуры, а косвенно – посредством системы Windows. Это делается следующим образом.

1. Функция DispatchMessageA возвращает сообщение операционной системе.

2. Windows, используя описание класса окна, передает сообщение нужной оконной процедуре приложения.

3. После обработки сообщения оконной процедурой управление возвращается операционной системе.

4. Windows передает управление функции DispatchMessageA.

5. DispatchMessageA завершает свое выполнение.

Так как вызов функции DispatchMessageA является последним в цикле, то управление опять передается функции GetMessageA, которая

выбирает очередное сообщение и, если оно удовлетворяет параметрам, заданным при вызове функции, выполняет тело цикла. Цикл обработки сообщений выполняется до тех пор, пока не приходит сообщение wm_Quit. Получение этого сообщения – единственное условие, при котором программа может выйти из цикла обработки сообщений.

Для завершения работы приложения достаточно использовать функцию ExitProcess@4 с нулевым параметром.

Обработка сообщений в оконной процедуре

Приложение может иметь несколько оконных процедур. Их количество определяется количеством классов окон, зарегистрированных в системе функцией RegisterClassA.

Когда для окна Windows-приложения появляется сообщение, операционная система Windows производит вызов соответствующей оконной процедуры. Сообщения, в зависимости от источника их появления в оконной процедуре, могут быть двух типов: синхронные и асинхронные. К синхронным сообщениям относятся те сообщения, которые помещаются в очередь сообщений приложения и ждут момента, когда они будут выбраны функцией GetMessage. После этого поступившие сообщения попадают в оконную процедуру, где и производится их обработка. Асинхронные сообщения попадают в оконную процедуру в экстренном порядке, минуя все очереди. Асинхронные сообщения, в частности, инициируются некоторыми функциями Win32 API, такими как CreateWindow или UpdateWindow.

Извлечение синхронного сообщения производится функцией GetMessage с последующей передачей обратно в Windows функцией DispatchMessage. Асинхронное сообщение, независимо от источника, кото-

рый инициирует его появление, сначала попадает в Windows и затем – в нужную оконную процедуру.

Windows требует, чтобы оконная процедура сохраняла значения регистров EBI, EDI и ESI. По завершении работы оконная процедура формирует значение в регистре EAX. Если сообщение обрабатывалось в оконной функции, то в EAX необходимо поместить нулевое значение. Если обработка осуществлялась по умолчанию, т.е. функцией DefWindowProc, то в EAX уже сформировано возвращаемое значение, и именно его нужно возвратить в качестве результата работы оконной процедуры.

В приложении А представлен код оконного приложения, которое обрабатывает нажатие левой и правой кнопки мыши.

Методика и порядок выполнения работы

1. Изучить принципы построения оконных приложений на языке ассемблера, взаимодействия приложения с операционной системой Windows.

2. Используя пакет MASM32 произвести компиляцию кода из приложения А. Исследовать полученный исполняемый файл под отладчиком.

3. Разработать программу в соответствии с индивидуальным заданием (Приложение Б) и защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Из каких частей состоит оконное Windows-приложение?

2. Как производится вызов функций Win32 API?

3. Для чего требуется регистрировать класс?

4. Какие функции используются в стартовом коде?

5. Что выполняет петля обработки сообщений?

6. Как реализуется взаимодействие операционной системы Windows с оконным приложением?

7. Что выполняет оконная процедура?

8. Как реализуется передача сообщения оконной процедуре?

9. Какие типы сообщений бывают?

10. Как вывести окно приложения на экран?

Практическая работа № 2

Разработка многопоточного приложения на языке низкого уровня

Цель работы: Получить навык создания оконных приложений Windows с дочерними окнами.

Краткие сведения из теории

Кнопки, списки, окна редактирования и другие элементы управления являются дочерними окнами главного окна приложения. Эти элементы, по сути, также являются окнами, но обладающими особыми свойствами. События, происходящие с этими элементами (и самим окном), приводят к приходу сообщений в оконную процедуру. Поэтому необходимо в оконную процедуру включить обработку событий с содержимым окна.

Сообщение WM_COMMAND генерируется системой, когда что-то происходит с управляющими элементами окна. В этом случае по значению параметров можно определить, какой это элемент и что с ним произошло (LPARAM – дескриптор элемента, старшее слово WPARAM – событие, младшее слово WPARAM – обычно идентификатор ресурса). Например, событие «кнопка нажата» может быть определено следующим образом. В начале производится проверка сообщения WM_COMMAND, а затем проверяем LPARAM – здесь хранится дескриптор (уникальный номер) окна (кнопка создается как окно).

В качестве примера рассмотрим приложение, интерфейс которого включает две кнопки и окно редактирования текста. При нажатии кнопки «Выполнить» создается окно-сообщение с кнопкой «ОК» и текстом из окна редактирования. По нажатию кнопки «Выход» происходит закрытие приложения.

Вначале программы опишем константы, которые используются для настройки окон и обработки событий.

```
.386P
.MODEL FLAT, stdcall
; сообщение приходит при закрытии окна
WM_DESTROY                      equ 2
; сообщение приходит при создании окна
WM_CREATE                       equ 1
;сообщение приходит при нажатии кнопки
WM_COMMAND                      equ 111h
```

Свойства главного окна:

```
CS_VREDRAW                      equ 1h
CS_HREDRAW                      equ 2h
CS_GLOBALCLASS                  equ 4000h
WS_OVERLAPPEDWINDOW             equ 000CF0000H
style equ CS_HREDRAW+CS_VREDRAW+CS_GLOBALCLASS
```

Свойства кнопки:

```
BS_DEFPUSHBUTTON                equ 1h
WS_VISIBLE                      equ 10000000h
WS_CHILD                        equ 40000000h
STYLBTN equ WS_CHILD + BS_DEFPUSHBUTTON + WS_VISIBLE
```

Свойства кнопки, которая будет создано как окно – это наиболее типичное сочетание свойств, но не единственное. Например, для того чтобы кнопка содержала иконку, то необходимым условием для этого будет свойство BS_ICON (или BS_BITMAP).

Свойства окна редактирования:

```
WS_BORDER                    equ 800000h
ES_AUTOHSCROLL               equ 80h
STYLEDT equ WS_CHILD + WS_VISIBLE + WS_BORDER + ES_AUTOHSCROLL
```

Также как и свойства кнопки, свойства окна редактирования могут быть дополнены другими константами. Например, константа ES_AUTOHSCROLL позволит избавиться от ограничения на число вводимых символов, которое определяется размером окна редактирования.

Зададим идентификаторы иконки и курсора:

```
IDI_APPLICATION              equ 32512
IDC_CROSS                    equ 32515
```

Режим показа окна установим нормальным:

```
SW_SHOWNORMAL                equ 1
```

Определим прототипы внешних функций

```
EXTERN      MessageBoxA@16: NEAR
EXTERN      CreateWindowExA@48: NEAR
EXTERN      DefWindowProcA@16: NEAR
EXTERN      DispatchMessageA@4: NEAR
EXTERN      ExitProcess@4: NEAR
EXTERN      GetMessageA@16: NEAR
EXTERN      GetModuleHandleA@4: NEAR
EXTERN      LoadCursorA@8: NEAR
EXTERN      LoadIconA@8: NEAR
EXTERN      PostQuitMessage@4: NEAR
EXTERN      RegisterClassA@4: NEAR
EXTERN      ShowWindow@8: NEAR
EXTERN      TranslateMessage@4: NEAR
EXTERN      UpdateWindow@4: NEAR
EXTERN      GetWindowTextA@12: NEAR
```

Директивы компоновщику для подключения библиотек, структуры сообщения и класса аналогичны директивам и структурам Приложения А.

Далее определяем сегмент данных. В сегменте данных необходимо зарезервировать три двойных слова для хранения дескрипторов двух кнопок и окна редактирования, которые будут получены после их создания функцией CreateWindowExA@48. Также необходимо определить уникальные имена классов кнопок и окна редактирования. Для хранения введенных символов в сегменте данных зарезервируем 50 байт (buf).

```
_DATA SEGMENT  DWORD PUBLIC  USE32 'DATA'
      NEWHWND          DD 0              ;дескриптор главного окна
      MSG              MSGSTRUCT  <?>
      WC               WNDCLASS  <?>
      HINST            DD 0              ;дескриптор приложения
      TITLENAME        DB 'Простой пример 32-битного приложения', 0
      CLASSNAME        DB 'CLASS32', 0
      ButtonClassName  DB "button",  0   ;имя класса кнопки
      ButtonText1      DB "Выполнить", 0
      ButtonText2      DB "Выход",  0
      EditClassName    DB "edit", 0      ;имя класса окна редактирования
      hwndButton1      DD ?         ;дескриптор первой кнопки
      hwndButton2      DD ?         ;дескриптор второй кнопки
      hwndEdit1        DD ?         ;дескриптор окна редактирования
      CAP              DB 'Сообщение', 0
      MES1             DB 'Выход из программы', 0
      buf              DB 50 DUP(0)          ;буфер
_DATA ENDS
```

Стартовый код, регистрация класса главного окна и создание последнего, а также цикл обработки сообщений в данном примере ничем не отличается от примера в Приложении А. Рассмотрим оконную процедуру.

Начало оконной процедуры включает проверку сообщений:

```
WNDPROC  PROC
      PUSH EBP
```

```
        MOV EBP, ESP
        PUSH EBX
        PUSH ESI
        PUSH EDI
CMP DWORD PTR [EBP+0CH], WM_DESTROY
        JE  WMDESTROY
CMP DWORD PTR [EBP+0CH], WM_CREATE
        JE  WMCREATE
CMP DWORD PTR [EBP+0CH] ,WM_COMMAND
        JE  WMCOMMAND
        JMP DEFWNDPROC
```

По приходу сообщения WM_CREATE необходимо создать дочерние окна (две кнопки и окно редактирования текста). Следующий код в оконной процедуре создает дочерние окна и сохраняет в сегменте данных их дескрипторы, которые потребуется при обработке сообщений WM_COMMAND.

```
WMCREATE:
;-------создаем кнопку "Выполнить"
        PUSH 0
        PUSH [HINST]
        PUSH 0
        PUSH DWORD PTR [EBP+08H]
        PUSH 20 ;DY – высота окна
        PUSH 100 ;DX – ширина окна
        PUSH 10 ;Y – координата левого верхнего угла
        PUSH 10 ;X – координата левого верхнего угла
        PUSH STYLBTN
        PUSH OFFSET ButtonText1  ; имя окна
        PUSH OFFSET ButtonClassName  ; имя класса
        PUSH 0
        CALL CreateWindowExA@48
        mov hwndButton1,  EAX
;--------создаем кнопку "Выход"
        PUSH 0
        PUSH [HINST]
```

```
        PUSH 0
        PUSH DWORD PTR [EBP+08H]
        PUSH 20 ; DY – высота окна
        PUSH 100 ; DX – ширина окна
        PUSH 40 ; Y – кордината левого верхнего угла
        PUSH 10 ; X – кордината левого верхнего угла
        PUSH STYLBTN
        PUSH OFFSET ButtonText2  ; имя окна
        PUSH OFFSET ButtonClassName  ; имя класса
        PUSH 0
        CALL CreateWindowExA@48
        mov hwndButton2, EAX
;--------создаем окно редактирования текста
        PUSH 0
        PUSH [HINST]
        PUSH 0
        PUSH DWORD PTR [EBP+08H]
        PUSH 20 ; DY – высота окна
        PUSH 200 ; DX – ширина окна
        PUSH 10 ; Y – координата левого верхнего угла
        PUSH 120 ; X – координата левого верхнего угла
        PUSH STYLEDT
        PUSH 0
        PUSH OFFSET EditClassName ; имя класса
        PUSH 0
        CALL CreateWindowExA@48
        mov hwndEdit1, EAX
        MOV EAX, 0
        JMP FINISH
```

Обработка сообщений от управляющих элементов включает проверку дескриптора элемента и действия в случае совпадения.

```
WMCOMMAND:
        mov eax, hwndButton1
        CMP DWORD PTR [EBP+14H], eax
        JNE  NEXTE
        PUSH 50
        PUSH OFFSET buf
```

```
        PUSH hwndEdit1
        CALL GetWindowTextA@12
        PUSH 0
        PUSH OFFSET CAP
        PUSH OFFSET buf
        PUSH NEWHWND  ; дескриптор окна
        CALL MessageBoxA@16
NEXTE:
        mov eax, hwndButton2
        CMP DWORD PTR [EBP+14H], eax
        JE  WMDESTROY
        MOV EAX, 0
        JMP FINISH
```

Функция GetWindowTextA@12 сохраняет в указанном буфере текст из окна редактирования.

Заключительный код оконной процедуры совпадает с кодом примера Приложения А.

```
DEFWNDPROC:
        PUSH DWORD PTR [EBP+14H]
        PUSH DWORD PTR [EBP+10H]
        PUSH DWORD PTR [EBP+0CH]
        PUSH DWORD PTR [EBP+08H]
        CALL DefWindowProcA@16
        JMP FINISH
WMDESTROY:
        PUSH 0        ; MB_OK
        PUSH OFFSET CAP
        PUSH OFFSET MES1
        PUSH DWORD PTR [EBP+08H] ; дескриптор окна
        CALL MessageBoxA@16
        PUSH 0
        CALL PostQuitMessage@4  ; сообщение WM_QUIT
        MOV EAX, 0
FINISH:
        POP EDI
        POP ESI
```

```
        POP EBX
        POP EBP
        RET 16
WNDPROC ENDP
_TEXT ENDS
        END START
```

Сочетание различных констант в свойствах дочерних окон позволяет получить различные управляющие элементы. Например, список – это окно со стилем

```
STYLLST equ WS_THICKFRAME + WS_CHILD + WS_VISIBLE + WS_BORDER + WS_TABSTOP + WS_VSCROLL + LBS_NOTIFY
```

Ход выполнения работы

1. Ознакомиться с константами, задающими стиль дочерних окон.

2. Создать оконное приложение с тремя кнопками и двумя окнами редактирования. Одна из трех кнопок реализует выход из приложения. Остальные две кнопки – действия, соответствующие индивидуальному заданию.

3. Защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Какие управляющие элементы можно создать?

2. Как создаются кнопки на главном окне?

3. Обработка сообщений от управляющих элементов.

4. Как получить дескриптор дочернего окна, и для чего он нужен?

5. Какие поля структуры сообщения используются для управления?

Таблица 2.1 – Задания на лабораторную работу № 2

№	Кнопка № 1	Кнопка № 2
1	2	3
1	Создать каталог с именем из Edit1	Создать в каталоге с именем из Edit1 файл с расширением .txt и именем из Edit2
2	Создать файл с именем из Edit1	Записать в созданный файл строку из Edit2
3	Открыть файл с именем из Edit1	Вывести в Edit2 второе слово первой строки из открытого файла
4	Записать в файл с именем из Edit1 строку из буфера	Вывести в Edit2 размер файла с именем из Edit1
5	Вывести в Edit1 свободное пространство на диске C:\	Создать файл с именем из Edit2 в корне диска C:\
6	Удалить файл с именем из Edit1	Определить набор символов файла (ANSI или OEM) с именем из Edit2
7	Определить является ли файл с именем из Edit1 исполняемым	Вывести в Edit2 для какой подсистемы файл с именем из Edit1 является исполняемым – Win32, MS DOS, OS/2, UNIX (POSIX) и т.д.
8	Вывести в Edit1 текущий каталог	Определить и вывести тип диска из Edit2 с помощью функции MessageBoxA: съемный, фиксированный, CD-ROM, электронный или сетевой
9	Получить атрибуты файла именем из Edit1 и вывести их с помощью функции MessageBoxA	Вывести в Edit2 тип файла, указанного в Edit1
10	Вывести с помощью функции MessageBoxA полный путь и имя для указанного в Edit1 файла	Вывести в Edit2 доступные в настоящее время дисководы
11	Блокировать файл с именем из Edit1	Переименовать файл с именем из Edit2 в файл с именем из Edit1
12	Удалить существующий каталог с именем из Edit1	Изменить атрибуты файла с именем из Edit2
13	Закрыть открытый дескриптор файла с именем из Edit1	Вывести в Edit2 последнюю строку файла с именем из Edit1

1	**2**	**3**
14	Найти файл с именем из Edit1. Результат поиска вывести на экран с помощью функции MessageBoxA	Вывести в Edit2 псевдоним файла с именем из Edit1
15	Получить атрибуты каталога с именем из Edit1. Результат вывести на экран с помощью функции MessageBoxA	Получить информацию об изменениях в пределах каталога, имя которого указано в Edit2
16	Установить текущим каталог с именем из Edit1	Записать в существующий файл с именем из Edit2 имя текущего каталога в первую строку без удаления содержимого
17	Вывести на экран с помощью функции MessageBoxA время создания файла с именем из Edit1	Скопировать файл с именем из Edit1 в файл с именем из Edit2
18	Создать файл с именем из Edit1 и записать в него доступные в настоящее время дисководы	Перемесить файл с именем из Edit1 в файл с именем из Edit2
19	Вывести в Edit1 имя CD-ROM, если он имеется	Открыть файл с именем из Edit2
20	Определить наличие флеш-диска в системе и вывести его имя в Edit1	Создать на флеш-диске с именем из Edit1 каталог с именем из Edit2
21	Вывести в Edit1 текущую дату	Создать файл с именем из Edit2 и записать в него время создания
22	Изменить текущее время на время из Edit1	Создать каталог с именем из Edit2 в папке «Мои документы», предварительно определив тип операционной системы
23	Вывести в Edit1 тип операционной системы	Создать файл с именем из Edit2 и записать в него объем диска C:\ в байтах
24	Найти на диске файл с именем из Edit1 и удалить его	Создать файл с именем из Edit1 и записать в него строку из Edit2
25	Записать в файл, имя которого указано в Edit1, свои ФИО в последнюю строку	Создать файл с именем из Edit2 и записать в него первые две строки из файла с именем из Edit1

Практическая работа № 3

Разработка программ, использующих ресурсы

Цель работы: Получить навык создания оконных приложений Windows, использующих ресурсы.

Краткие сведения из теории

В операционную систему Windows введено понятие ресурса. Ресурс представляет собой некий визуальный элемент с заданными свойствами, хранящийся в исполняемом файле отдельно от кода и данных, который может отображаться специальными функциями.

Использование ресурсов дает две вполне определенные выгоды:

1. ресурсы загружаются в память лишь при обращении к ним, т.е. реализуется экономия памяти;
2. свойства ресурсов поддерживаются системой автоматически, не требуя от программиста написания дополнительного кода.

Описание ресурсов хранится отдельно от программы в текстовом файле (*.rc) и компилируется (*.res) специальным транслятором ресурсов. В исполняемый файл ресурсы включаются компоновщиком. Транслятором ресурсов в пакете MASM32 является RC.EXE.

Наиболее употребляемые ресурсы:

1. иконки;
2. курсоры;
3. битовая картинка;
4. строка;
5. диалоговое окно;
6. меню;
7. акселераторы.

Такой ресурс, как диалоговое окно, может содержать в себе управляющие элементы, которые также должны быть описаны, но в рамках описания окна.

Чтобы добавить этот файл в программу, его надо скомпилировать и указать имя скомпилированного *.RES-файла для компоновщика:

```
ml /c /coff /Cp lab3.asm
rc /r winmenu.rc
link lab3.obj winmenu.res /subsystem:windows
```

В качестве примера рассмотрим файл ресурсов (winmenu.rc), который содержит:

```
#define ZZZ_TEST  0
#define ZZZ_OPEN  1
#define ZZZ_SAVE  2
#define ZZZ_EXIT  3
ZZZ_Menu  MENU {
   POPUP "&File" {
      MENUITEM  "&Open",ZZZ_OPEN
      MENUITEM  "&Save", ZZZ_SAVE
      MENUITEM  SEPARATOR
      MENUITEM  "E&xit",ZZZ_EXIT
   }
   MENUITEM  "&Edit",ZZZ_TEST
}
```

Рассмотрим код программы, использующей меню (lab3.asm).

Прежде необходимо добавить константы меню:

```
ZZZ_TEST     equ     0
ZZZ_OPEN     equ     1
ZZZ_SAVE     equ     2
ZZZ_EXIT     equ     3
```

Сообщения от нашего меню должны совпадать с определениями из winmenu.rc. Кроме того, в данном примере их номера важны, потому что они используются как индекс для таблицы переходов к обработчикам.

В сегмент данных необходимо добавить имя класса меню:

```
menu_name      db     "ZZZ_Menu",0;
```

и выделить четыре байта для дескриптора меню:

```
menu_hinst      dd     0
```

После регистрации класса окна необходимо загрузить меню, для чего служит следующий код:

```
push    offset  menu_name
push    [HINST]
call        LoadMenuA@8
mov         menu_hinst,  eax
```

Функция LoadMenuA@8 получает указатель на меню из файла ресурсов, который записывается в EAX.

Далее при создании окна необходимо в параметр Menu загрузить идентификатор разработанного меню.

Теперь перейдем к процедуре окна. Как и от любого другого управляющего элемента при выборе определенного элемента меню приходит сообщение WM_COMMAND, которое в LPARAM будет содержать дескриптор меню, а в WPARAM – идентификатор пункта меню, которые ранее мы задали как константы (ZZZ_TEST, ZZZ_OPEN, ZZZ_SAVE, ZZZ_EXIT). Следовательно, для обработки событий меню необходимо обнаружить его дескриптор и по коду пункта выполнить соответствующее действие. Например, обработка пункта выхода из приложения «Exit»:

```
mov eax, menu_hinst
CMP dword ptr [ebp+14h],eax
jne go_away
cmp dword ptr [ebp+10h],ZZZ_EXIT
je WMDESTROY
```

На рисунке 3.1 представлено меню рассмотренного примера.

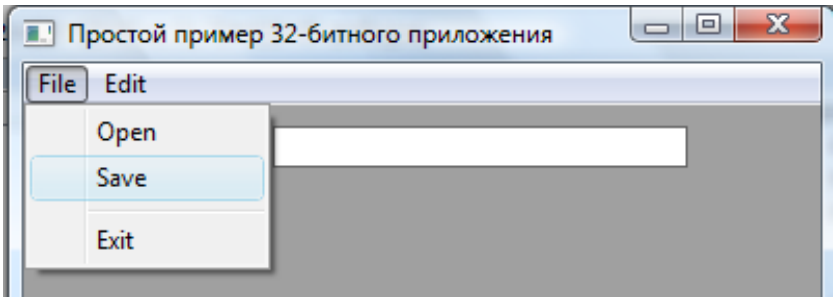

Рисунок 3.1 – Одноуровневое меню

Рассмотрим структуру меню с несколькими уровнями. На рисунке 3.2 представлена иерархическая структура меню.

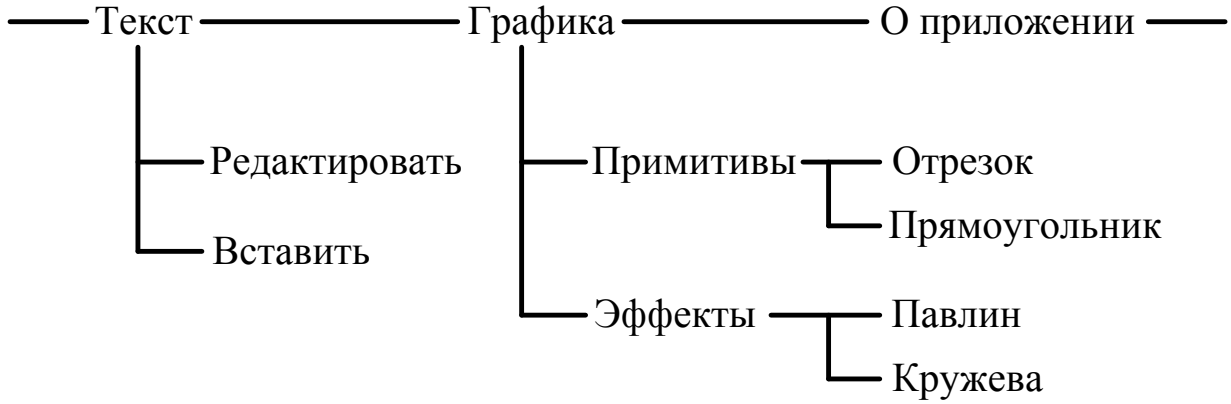

Рисунок 3.2 – Структура меню с двумя уровнями

Далее приведем текст файла ресурса меню рисунка 3.2.

```
#define IDM_DRAWTEXT  0
#define IDM_TEXTOUT  1
#define IDM_LENGTH  2
#define IDM_RECTANGLE  3
#define IDM_PEACOCK  4
#define IDM_LACES  5
#define IDM_ABOUT  6
MYMENU  MENU
{
POPUP “&Текст”
            {
            MENUITEM  “&Редактировать”, IDM_DRAWTEXT
            MENUITEM  “&Вставить”, IDM_TEXTOUT
            }
POPUP “&Графика”
```

```
        {
        POPUP “&Примитивы”
                {
                MENUITEM  “&Отрезок”, IDM_LENGTH
                MENUITEM  “&Прямоугольник”, IDM_RECTANGLE
                }
        POPUP “&”Эффекты”
                {
                MENUITEM  “&Павлин”, IDM_PEACOCK
                MENUITEM  “&Кружева”, IDM_LACES
                }
        }
MENUITEM  “&О приложении”, IDM_ABOUT
}
```

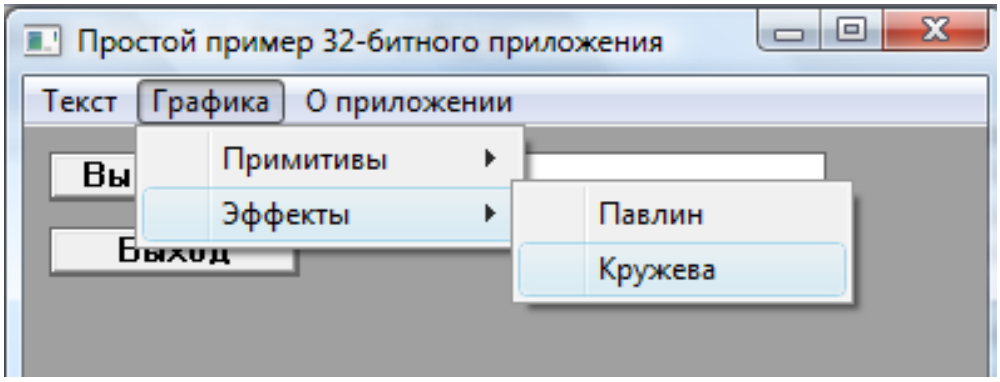

Рисунок 3.3 – Двухуровневое меню

Ход выполнения работы

1. Ознакомиться с принципами организации меню.

2. К приложению, разработанному во второй лабораторной работе, добавить меню, изображенное на рисунке 4. При выборе пункта «Очистить» должна произойти очистка поля для редактирования текста. Пункты «Действие 1» и «Действие 2» соответствуют заданиям на лабораторную работу № 2.

3. Защитить лабораторную работу преподавателю.

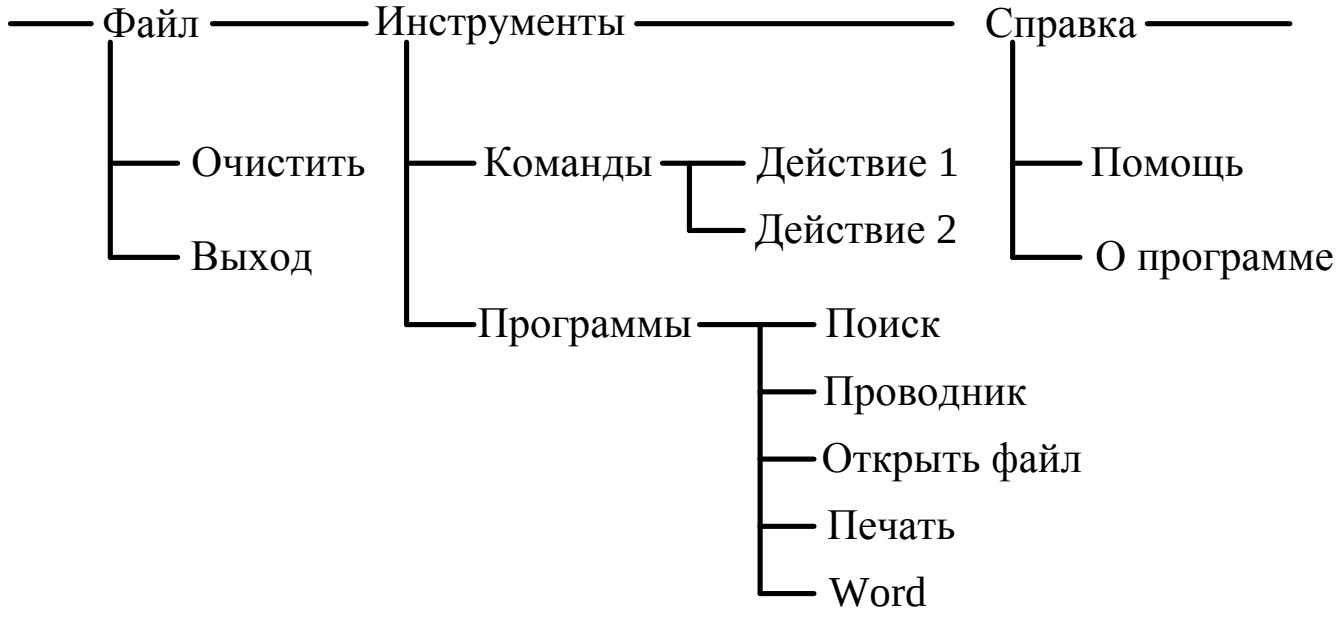

Рисунок 4 – Структура меню задания на лабораторную работу

Программы «Поиск» и «Проводник» можно вызвать функцией ShellExecute. Функция ShellExecute из библиотеки Shell32.dll выполняет операцию над указанным файлом. Вот её прототип:

HINSTANCE ShellExecute (HWND *hwnd*, LPCTSTR *lpOperation*, LPCTSTR *lpFile*, LPCTSTR *lpParameters*, LPCTSTR *lpDirectory*, INT *nShowCmd*).

В случае успешного завершения функция возвращает значение, большее 32. При возникновении ошибки функция вернёт одно из следующих значений:

Код ошибки	Описание
0	Недостаточно памяти или ресурсов Windows
2	Указанный файл не найден
3	Указанный путь не существует
5	Операционная система не имеет доступа к указанному файлу
8	Недостаточно памяти для завершения операции
26	Невозможен совместный доступ к файлу
27	Невозможно загрузить приложение, ассоциированное с типом файла
28	DDE транзакция не может быть завершена из-за истечения времени
29	Неудача при выполнении DDE транзакции
30	DDE транзакция не может быть завершена, так как обрабатываются другие DDE-транзакции
31	Нет никакого приложения, ассоциированного с расширением файла
32	Не найдена указанная DLL-библиотека

Функции передаются следующие параметры:

Параметр	Описание
hwnd	Дескриптор родительского окна. При вызове функции из Visual FoxPro должен быть равен нулю.
Operation	Может принимать одно из следующих значений: "find", "explore", "edit", "open" или "print"
File	Имя файла или папки – в зависимости от значения параметра *Operation*.
Parameters	Список параметров, передаваемых загружаемому приложению
Directory	Путь к файлу, указанному в *File*
ShowCmd	Определяет вид главного окна загружаемого приложения

Если *Operation="find"*, функция выводит диалоговое окно для поиска файлов по условиям. Параметр *File* должен указывать путь к папке, начиная с которой будет выполняться поиск. Остальные параметры не используются.

Если *Operation="explore"*, функция выводит диалоговое окно – список папок. Параметр *File* должен указывать путь к папке, содержимое которой нужно посмотреть. Остальные параметры не используются.

Если *Operation="edit"*, функция открывает файл на редактирование, загружая приложение, ассоциированное с расширением файла. Параметр *Edit* должен содержать имя файла, параметр *Directory* – указывать путь к этому файлу; если параметр *Directory* не используется, то параметр *Edit* должен указывать путь и имя файла.

Если *Operation="open"*, функция выполняет следующие действия: если в *File* указан исполняемый файл (например, типа EXE), то он запускается на выполнение; загружаемой программе передаётся список параметров, указанных в *Parameters*; в противном файл открывается на редактирование.

Если *Operation="print"*, то выполняется печать файла на принтере (фактически загружается ассоциированное с расширением файла приложение, которое и печатает документ).

Параметр *ShowCmd* может принимать значения от 0 до 10, реальный интерес представляют значения, перечисленные в таблице:

ShowCmd	Описание
0	Скрывает окно загружаемого приложения и активизирует другое окно.
1	Отображает главное окно приложения и делает его активным. Если окно приложения минимизировано или максимизировано, Windows восстанавливает его первоначальный размер и позицию.
2	Окно загружаемого приложения минимизировано.
3	Раскрывает окно приложения на весь экран и делает его активным.
4	Отображает окно приложения в его последних сохранённых размерах, но не делает его активным.

Примеры использования функции в вашем приложении.

В следующем примере запускается приложение Notepad (блокнот); окно приложения распахивается на весь экран и становится активным:

nReturn = ShellExecute(0,'open','c:\Windows\Notepad.exe',NULL,NULL,3).

В следующем примере загружается приложение MS Word для редактирования файла MyDocument.doc, расположенного в папке c:\MyDocs:

nReturn = ShellExecute(0,'open','MyDocument.doc', NULL,'c:\MyDocs',1).

И последний пример, в котором при помощи MS Word выполняется печать файла MyDocument.doc. MS Word загружается в скрытом (Hide) режиме (окно не выводится, индикатор в панели "Пуск" не отображается). Документ распечатывается на принтере, используемом по умолчанию:

nReturn = ShellExecute(0,'print','c:\MyDocs\MyDocument.doc', NULL,NULL,0).

Контрольные вопросы

1. Что такое ресурс и какие ресурсы Вам известны?

2. В чем заключается преимущество использования ресурсов?

3. Как создать двух и трехуровневое меню?

4. Как обрабатываются сообщения от меню?

5. На каком этапе работы программы загружается меню?

6. Как запустить файл с расширением exe с помощью функции Win32 API?

7. Как меню прикрепляется к главному окну?

Литература

Основная литература

1. Новиков Ю.В. Основы микропроцессорной техники. — Электрон. текст. дан.— М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — Режим доступа : http ://www.iprbookshop.ru/52207. — ЭБС «IPRbooks», по паролю.

2. Водовозов А.М. Микроконтроллеры для систем автоматики : Учеб. пособие. — Электрон. текст. дан. — М. : Инфра-Инженерия, 2016. — Режим доступа : http ://www.iprbookshop.ru/51727.— ЭБС «IPRbooks», по паролю

Дополнительная литература

1. Гуров В.В. Архитектура микропроцессоров [Электронный ресурс]/ Гуров В.В.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 115 с.— Режим доступа: http://www.iprbookshop.ru/56313.html.— ЭБС «IPRbooks»

2. Аблязов Р.З. Программирование на ассемблере на платформе x86-64 [Электронный ресурс]/ Аблязов Р.З.— Электрон. текстовые данные.— Саратов: Профобразование, 2017.— 304 с.— Режим доступа: http://www.iprbookshop.ru/63951.html.— ЭБС «IPRbooks»