МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Невинномысский технологический институт (филиал) СКФУ

Методические указания к проведению лабораторных работ
по дисциплине

«МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ»

для студентов направления подготовки
15.03.04 Автоматизация технологических процессов и производств
Направленность (профиль) - Информационно-управляющие системы

Невинномысск 2022

Методические указания предназначены для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств. Они содержат основы теории, порядок проведения лабораторных работ, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины «Микропроцессорные системы управления». 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 ФГОС ВО в части содержания и уровня подготовки выпускников по направлению подготовки 15.03.04 Автоматизация технологических процессов и производств

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

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

Содержание

Лабораторная работа № 1

Создание простого оконного приложения

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?

Лабораторная работа № 2

Оконное приложение с управляющими элементами

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

Лабораторная 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.

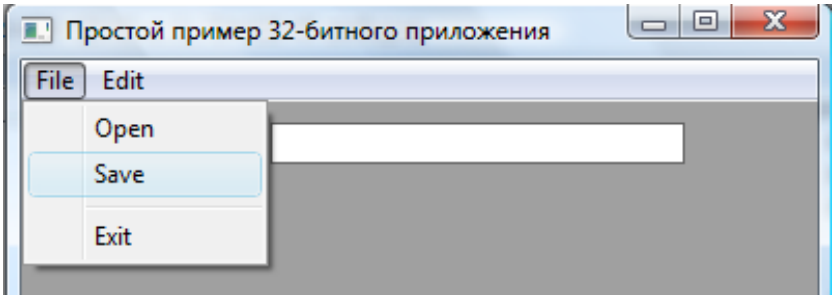

Рисунок 3.1 – Одноуровневое меню

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

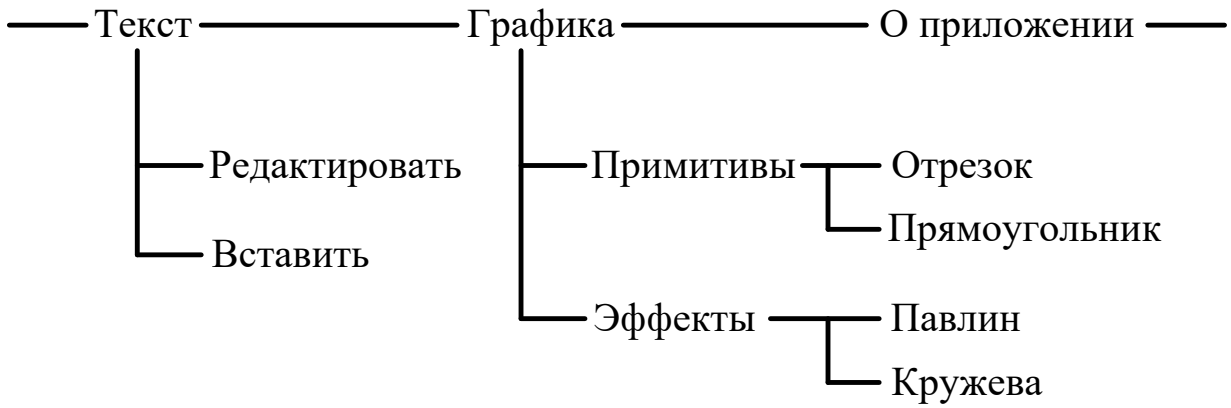

Рисунок 3.2 – Структура меню с двумя уровнями

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
}
```

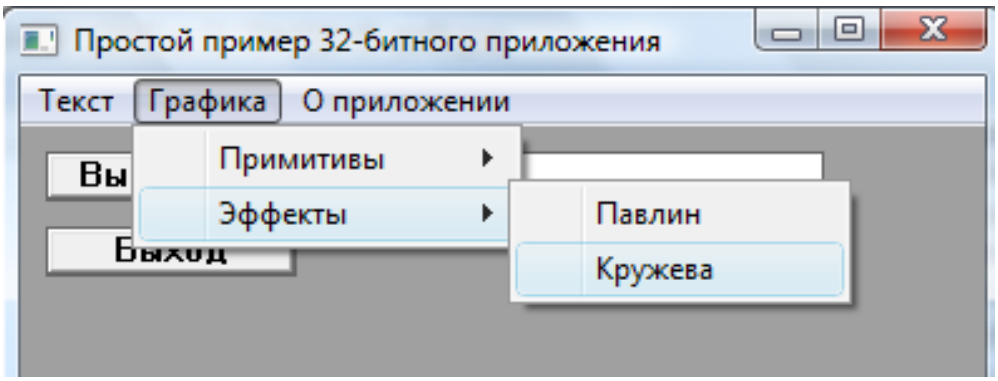

Рисунок 3.3 – Двухуровневое меню

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.

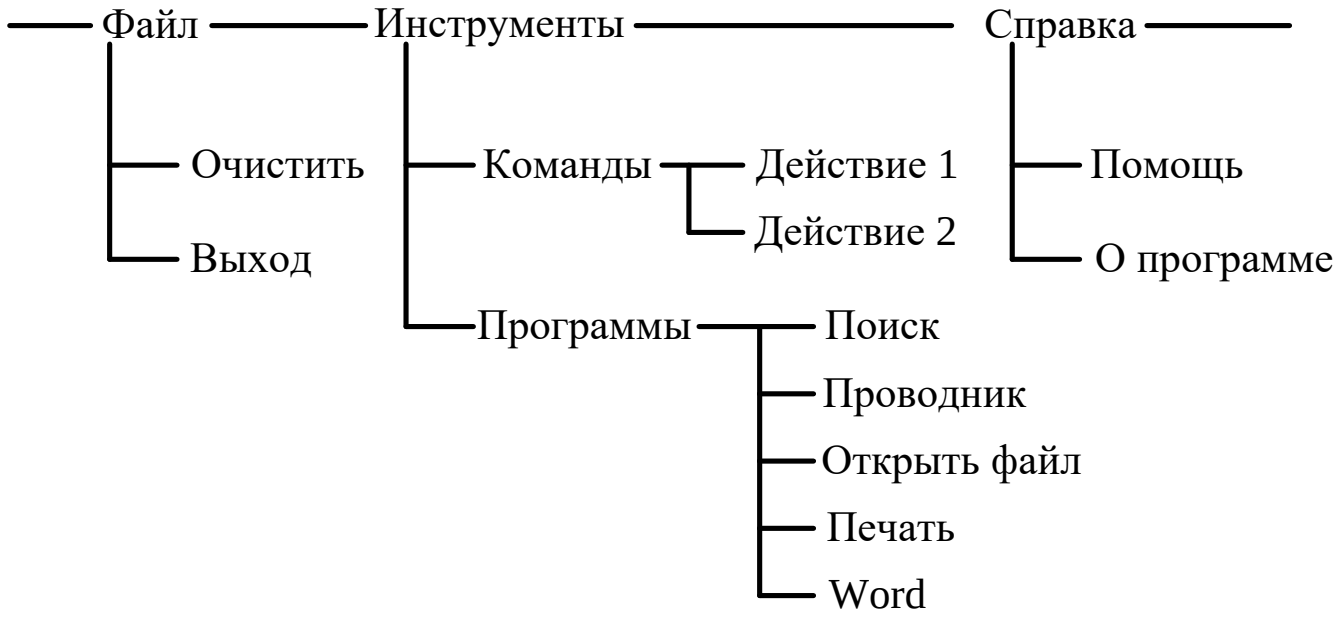

Рисунок 4 – Структура меню задания на лабораторную работу

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?

Лабораторная работа № 4

Разработка DLL-библиотеки

Цель работы: Получить навык создания библиотек динамической компоновки.

Краткие сведения из теории

Использование динамических библиотек (по-другому – библиотек динамической компоновки) – это способ осуществления модульности в период выполнения программы. Динамическая библиотека (Dynamic Link Library – DLL) позволяет упростить разработку программного обеспечения. Вместо того чтобы каждый раз перекомпилировать огромные EXE-программы, достаточно перекомпилировать лишь отдельный динамический модуль. Кроме того, доступ к динамической библиотеке возможен сразу из нескольких исполняемых модулей, что делает многозадачность более гибкой. Использование динамической библиотеки экономит дисковое пространство, т.к. представленная в библиотеке процедура содержится лишь один раз, в отличие от процедур, помещаемых в модули из статических библиотек.

С точки зрения программирования на ассемблере DLL – это самый обычный исполнимый файл формата PE, отличающийся только тем, что

при входе в него в стеке находятся три параметра (идентификатор DLL-модуля, причина вызова процедуры и зарезервированный параметр), которые надо удалить, например командой ret 12. Кроме этой процедуры в DLL входят и другие, часть которых можно вызывать из других программ. Список этих экспортируемых процедур должен быть задан во время компиляции DLL, и поэтому команды для компиляции будут отличаться от обычных.

Рассмотрим различные варианты связывания при трансляции. Во время трансляции связываются имена, указанные в программе как внешние, (EXTERN) с соответствующими именами из библиотек, которые указываются при помощи директивы IMPORTLIB. Такое связывание называется ранним (или статическим). Напротив, в случае с динамической библиотекой связывание происходит во время выполнения модуля. Такое связывание называется поздним (или динамическим). При этом позднее связывание может происходить в автоматическом режиме в начале запуска программы и при помощи специальных API-функций (см. ниже), по желанию программиста. При этом говорят о явном и неявном связывании.

Динамическая библиотека может содержать также ресурсы. Так, файлы шрифтов представляют собой динамические библиотеки, единственным содержимым которых являются ресурсы. Динамическая библиотека как бы становится продолжением программы, загружаясь в адресное пространство процесса. Соответственно, данные процесса доступны из динамической библиотеки, и, наоборот, данные динамической библиотеки доступны для процесса.

В любой динамической библиотеке следует определить точку входа (процедура входа). По умолчанию за точку входа принимают метку, указываемую за директивой END (например, END START). При загрузке ди-

намической библиотеки и выгрузке динамической библиотеки автоматически вызывается процедура входа. Заметим при этом, что каким бы способом ни была загружена динамическая библиотека (явно или неявно), выгрузка динамической библиотеки из памяти будет происходить автоматически при закрытии процесса или потока. В принципе, процедура входа может быть использована для некоторой начальной инициализации переменных. Довольно часто эта процедура остается пустой. При вызове процедуры входа в нее помещаются три параметра:

1. идентификатор DLL-модуля;

2. причина вызова;

3. резерв.

Рассмотрим подробнее второй параметр процедуры входа. Вот четыре возможных значения этого параметра:

```
DLL_PROCESS_DETACH    equ 0
DLL_PROCESS_ATTACH    equ 1
DLL_THREAD_ATTACH     equ 2
DLL_THREAD_DETACH     equ 3
```

DLL_PROCESS_ATTACH – сообщает, что динамическая библиотека загружена в адресное пространство вызывающего процесса.

DLL_THREAD_ATTACH – сообщает, что текущий процесс создает новый поток. Такое сообщение посылается всем динамическим библиотекам, загруженным к этому времени процессом.

DLL_PROCESS_DETACH – сообщает, что динамическая библиотека выгружается из адресного пространства процесса.

DLL_THREAD_DETACH – сообщает, что некий поток, созданный данным процессом, в адресное пространство которого загружена данная динамическая библиотека, уничтожается.

Лабораторная работа включает следующие четыре этапа.

1. Разработка текста DLL-библиотеки.

2. Трансляция и компоновка исходного текста DLL-библиотеки.

3. Сборка приложения с использованием DLL-библиотеки.

4. Проверка работоспособности приложения с использованием DLL-библиотеки.

Разработка текста DLL-библиотеки

В качестве примера рассмотрим библиотеку, содержащую функцию вычисления остатка от целочисленного деления. Функции необходимо передать два параметра (делимое и делитель) и возвращает один параметр в регистре EAX. Программа, которая использует данную функцию, будет выводить результат в стандартном окне-сообщении.

Ниже приводится код DDL-библиотеки.

```
.386P
.model flat, stdcall
public mod_asm       ;разработанная функция
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
; [EBP+10H]    резервный параметр
; [EBP+0CH]    причина вызова
; [EBP+8]      идентификатор DLL-модуля
_start@12:
MOV EAX,1
RET 12
mod_asm proc EXPORT
pop ecx                ; обратный адрес в ecx
pop eax                ; делимое
pop ebx                ; делитель
```

```
push ecx                ; обратный адрес вернуть в стек для RET
xor edx, edx
div ebx
mov eax, edx
ret
mod_asm endp
_TEXT ends
end _start@12
```

Обратите внимание, что за процедурой, вызываемой из другого модуля, мы указали ключевое слово EXPORT. Это слово необходимо для правильной трансляции в MASM.

Трансляция и компоновка исходного текста DLL-библиотеки

```
ml /c /coff /DMASM mod.asm
link /subsystem:windows /DLL /ENTRY:_start@12 mod.obj
```

MASM помещает в динамическую библиотеку вместо mod_asm имя _mod_asm@0, которое нужно учесть в программе.

В результате трансляции получены mod.dll, mod.lib, mod.exp.

Сборка приложения с использованием DLL-библиотеки

Рассмотрим приложение, которое использует явное связывание. Библиотека должна быть вначале загружена при помощи функции LoadLibrary. Затем определяется адрес процедуры с помощью функции GetProcAddress, после чего можно осуществлять вызов. Также необходимо учесть возможность ошибки при вызове функций LoadLibrary и GetProcAddress. В этой связи укажем, как (в какой последовательности) ищет библиотеку функция LoadLibrary:

1. Поиск в каталоге, откуда была запущена программа.

2. Поиск в текущем каталоге.

3. В системном директории (GetSystemDirectory).

4. В директории Windows (GetWindowsDirectory).

5. В каталогах, указанных в окружении (PATH).

В конце программы мы выгружаем из памяти динамическую библиотеку, что, кстати, могли бы и не делать, т.к. по выходе из программы эта процедура выполняется автоматически.

```
.386P
.model flat, stdcall
EXTERN      GetProcAddress@8:NEAR
EXTERN      LoadLibraryA@4:NEAR
EXTERN      FreeLibrary@4:NEAR
EXTERN      ExitProcess@4:NEAR
EXTERN      MessageBoxA@16:NEAR
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib
_DATA SEGMENT DWORD PUBLIC USE32 'DATA'
TXT    DB 'Ошибка динамической библиотеки',0
MS     DB 'Сообщение',0
LIBR   DB 'mod.dll',0
HLIB   DD ?
TXT1 DB 0, 0
NAMEPROC DB '_mod_asm@0',0
_DATA ENDS
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; загрузить библиотеку
    PUSH OFFSET LIBR
```

```
    CALL LoadLibraryA@4
    CMP EAX,0
    JE _ERR
    MOV HLIB,EAX
; получить адрес процедуры
    PUSH OFFSET NAMEPROC
    PUSH HLIB
    CALL GetProcAddress@8
    CMP EAX,0
    JNE YES_NAME
; сообщение об ошибке
_ERR:
    PUSH  0
    PUSH  OFFSET MS
    PUSH  OFFSET TXT
    PUSH  0
    CALL  MessageBoxA@16
    JMP  _EXIT
YES_NAME:
    PUSH 7 ; делитель
    PUSH 112 ; делимое
    CALL EAX
; вывод результата деления в сообщении
   xor eax, 30h
   mov byte ptr [TXT1], al
    PUSH  0
    PUSH  OFFSET MS
```

```
    PUSH  OFFSET TXT1
    PUSH  0
    CALL  MessageBoxA@16
; закрыть библиотеку
    PUSH HLIB
    CALL FreeLibrary@4
; библиотека автоматически закрывается также
; при выходе из программы
; выход
_EXIT:
    PUSH 0
    CALL ExitProcess@4
_TEXT ENDS
END START
```

Теперь рассмотрим программу, которая использует неявное связывание. Во-первых, необходимо объявить вызываемую из динамической библиотеки процедуру как внешнюю, а, во-вторых, подключить статическую библиотеку mod.dll.

```
.386P
.model flat, stdcall
includelib mod.lib
EXTERN      mod_asm@0: NEAR
EXTERN      ExitProcess@4: NEAR
EXTERN      MessageBoxA@16: NEAR
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib
_DATA SEGMENT DWORD PUBLIC USE32 'DATA'
```

```
MS     DB 'Сообщение',0
TXT1 DB 0, 0
_DATA ENDS
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
     PUSH 7 ; делитель
     PUSH 112 ; делимое
     CALL mod_asm@0
; вывод результата деления в сообщении
    xor eax, 30h
    mov byte ptr [TXT1], al
     PUSH  0
     PUSH  OFFSET MS
     PUSH  OFFSET TXT1
     PUSH  0
     CALL  MessageBoxA@16
     PUSH 0
     CALL ExitProcess@4
_TEXT ENDS
END START
```

Методика и порядок выполнения работы

1. Изучить принципы разработки библиотек динамической компоновки в операционной системе Windows.

2. Разработать DLL и оконное приложение, использующее функции из разработанной библиотеки. Разработать приложение с явным и приложение с неявным связыванием. Библиотека должна содержать две функции.

3. Защитить лабораторную работу преподавателю.

Таблица 4.1 – Варианты индивидуальных заданий на лабораторню работу № 4

№	Первая функция	Вторая функция
1	**2**	**3**
1	$y1=\begin{cases} a+x,\ if\ x>a \\ 2a-x,\ if\ x\le a \end{cases}$	$y2=\begin{cases} a-x,\ if\ x>a \\ 2ax,\ if\ x\le a \end{cases}$
2	$y1=\begin{cases} x-2,\ if\ x\ge 2 \\ 8,\ if\ x<2 \end{cases}$	$y2=\begin{cases} 4,\ if\ x=0 \\ a-x,\ if\ x\ne 0 \end{cases}$
3	$y1=\begin{cases} x-a,\ if\ x>a \\ 5,\ if\ x\le a \end{cases}$	$y2=\begin{cases} a,\ if\ a>x \\ a\times x,\ if\ a\le x \end{cases}$
4	$y1=\begin{cases} 2-x,\ if\ x<2 \\ a+3,\ if\ x\ge 2 \end{cases}$	$y2=\begin{cases} a-1,\ if\ x<a \\ a\times x-1,\ if\ x\ge a \end{cases}$
5	$y1=\begin{cases} \lvert x\rvert,\ if\ x<0 \\ x-a,\ if\ x\ge 0 \end{cases}$	$y2=\begin{cases} a+x,\ if\ x\bmod 3=1 \\ 7,\ if\ x\bmod 3\ne 1 \end{cases}$
6	$y1=\begin{cases} x\bmod 4,\ if\ x>a \\ a,\ if\ x\le a \end{cases}$	$y2=\begin{cases} a\times x,\ if\ x/a>3 \\ x,\ if\ x/a\le 3 \end{cases}$
7	$y1=\begin{cases} 4-x,\ if\ \lvert x\rvert<3 \\ a+x,\ if\ \lvert x\rvert\ge 3 \end{cases}$	$y2=\begin{cases} 2,\ if\ x-\div\text{åò í î å} \\ 2a-x,\ if\ x-\text{í å}\div\text{åò í î å} \end{cases}$
8	$y1=\begin{cases} 4\times x,\ if\ x\le 4 \\ x-a,\ if\ x>4 \end{cases}$	$y2=\begin{cases} 7,\ if\ x-\text{í å}\div\text{åò í î å} \\ x/2+a,\ if\ x-\div\text{åò í î å} \end{cases}$
9	$y1=\begin{cases} a\times x,\ if\ x\bmod 3=2 \\ 9,\ if\ x\bmod 3\ne 2 \end{cases}$	$y2=\begin{cases} a-x,\ if\ a>x \\ a+2,\ if\ a\le x \end{cases}$
10	$y1=\begin{cases} a+\lvert x\rvert,\ if\ x>a \\ a-7,\ if\ x\le a \end{cases}$	$y2=\begin{cases} a\times 3,\ if\ a>3 \\ 11,\ if\ a\le 3 \end{cases}$
11	$y1=\begin{cases} 10+x,\ if\ x>1 \\ \lvert x\rvert+a,\ if\ x\le 1 \end{cases}$	$y2=\begin{cases} 2,\ if\ x>4 \\ x,\ if\ x\le 4 \end{cases}$

Таблица 4.1 (продолжение)

1	2	3
12	$y1=\begin{cases}15+x,\ if\ \ x>7\\ \lvert a\rvert-9,\ if\ \ x\le 7\end{cases}$	$y2=\begin{cases}3,\ if\ \ x>2\\ \lvert x\rvert-5,\ if\ \ x\le 2\end{cases}$
13	$y1=\begin{cases}3+x,\ if\ \ x=a\\ a-x,\ if\ \ x\ne a\end{cases}$	$y2=\begin{cases}\lvert a\rvert,\ if\ \ x>10\\ \lvert a\rvert-x,\ if\ \ x\le 10\end{cases}$
14	$y1=\begin{cases}2x+a,\ if\ \ x>2\\ 2x+1,\ if\ \ x\le 2\end{cases}$	$y2=\begin{cases}\lvert x\rvert+1,\ if\ \ x>0\\ a-1,\ if\ \ x\le 0\end{cases}$
15	$y1=\begin{cases}8+\lvert x\rvert,\ if\ \ x<1\\ \lvert a\rvert\times 2,\ if\ \ x\ge 1\end{cases}$	$y2=\begin{cases}3,\ if\ \ x=a\\ a+1,\ if\ \ x\ne a\end{cases}$
16	$y1=\begin{cases}4+x,\ if\ \ x\le 3\\ a\times x,\ if\ \ x>3\end{cases}$	$y2=\begin{cases}\lvert a\rvert-2,\ if\ \ x>a\\ x,\ if\ \ x\le a\end{cases}$
17	$y1=\begin{cases}a+\lvert x\rvert,\ if\ \ x<0\\ x-a,\ if\ \ x\ge 0\end{cases}$	$y2=\begin{cases}7,\ if\ \ x<3\\ a,\ if\ \ x\ge 3\end{cases}$
18	$y1=\begin{cases}7+x,\ if\ \ x<3\\ \lvert a\rvert+x,\ if\ \ x\ge 3\end{cases}$	$y2=\begin{cases}1,\ if\ \ x>5\\ a+x,\ if\ \ x\le 5\end{cases}$
19	$y1=\begin{cases}-5,\ if\ \ x>4\\ x-a,\ if\ \ x\le 4\end{cases}$	$y2=\begin{cases}\lvert a\rvert,\ if\ \ x>a\\ 9,\ if\ \ x\le a\end{cases}$
20	$y1=\begin{cases}2\times x,\ if\ \ x<5\\ \lvert a\rvert+x,\ if\ \ x\ge 5\end{cases}$	$y2=\begin{cases}3,\ if\ \ x<0\\ a+x,\ if\ \ x\ge 0\end{cases}$
21	$y1=\begin{cases}3,\ if\ \ x\bmod 3=1\\ x-a,\ if\ \ x\bmod 3\ne 1\end{cases}$	$y2=\begin{cases}a/x,\ if\ \ x\ne 0\\ 4,\ if\ \ x=0\end{cases}$
22	$y1=\begin{cases}\lvert a\rvert+\lvert x\rvert,\ if\ \ x<0\\ a\times x,\ if\ \ x\ge 0\end{cases}$	$y2=\begin{cases}3,\ if\ \ x=a\\ a-x,\ if\ \ x\ne a\end{cases}$
23	$y1=\begin{cases}2\times x,\ if\ \ \lvert x\rvert>4\\ 4+a,\ if\ \ \lvert x\rvert\le 4\end{cases}$	$y2=\begin{cases}9,\ if\ \ x=0\\ a/x,\ if\ \ x\ne a\end{cases}$
24	$y1=\begin{cases}x,\ if\ \ x\bmod 4\ne 2\\ a+x,\ if\ \ x\bmod 4=2\end{cases}$	$y2=\begin{cases}a-x,\ if\ \ x<a\\ a\times x,\ if\ \ x\ge a\end{cases}$
25	$y1=\begin{cases}12,\ if\ \ x<12\\ x+1,\ if\ \ x\ge 12\end{cases}$	$y2=\begin{cases}2,\ if\ \ x>2\\ a+x,\ if\ \ x\le 2\end{cases}$

Контрольные вопросы

1. В чем заключается преимущество использования библиотек динамической компоновки?

2. Чем отличается явное связывание от неявного связывания?

3. Этапы разработки DLL.

4. Из каких частей состоит библиотека динамической компоновки?

5. Структура сообщения передаваемого библиотеке.

6. В какую область адресного пространства процесса загружается DLL.

7. Адресное пространство процесса.

8. Причины вызова DLL-модуля.

Лабораторная работа № 5

Использование таймера

Цель работы: Получить навык написания программ, использующих таймер.

Краткие сведения из теории

Таймер является одним из мощных инструментов, предоставляемых операционной системой и позволяющих решать самые разнообразные задачи. Для работы с таймером в консольных приложениях используются функциии timeSetEvent и timeKillEvent. В оконных приложениях чаще используют функции SetTimer и KillTimer. Особенность таймера, создаваемого функцией SetTimer, заключается в том, что сообщение WM_TIMER, которое начинает посылать система приложению после выполнения функции SetTimer, приходит со всеми другими сообщениями наравне, на общих основаниях. Следовательно, интервал между двумя

приходами сообщения WM_TIMER может несколько варьироваться. В большинстве случаев это не существенно.

Если система посылает сообщение приложению, а предыдущее сообщение еще стоит в очереди, то система объединяет эти два сообщения. Таким образом, «вынужденный простой» не приводит к приходу на приложение подряд нескольких сообщений таймера.

Перечислим те задачи, которые можно решить с помощью таймера.

- Отслеживание времени: секундомер, часы и т.д. Нарушение периодичности здесь не имеет значения, так как по приходу сообщения время можно отследить, вызвав функцию получения системного времени.
- Таймер – один из способов осуществления многозадачности. Можно установить сразу несколько таймеров на разные функции, в результате периодически будет исполняться то одна, то другая функция.
- Периодический вывод на экран обновленной информации.
- Автосохранение – функция особенно полезная для редакторов.
- Задание темпа изменения каких-либо объектов на экране.
- Мультипликация – по приходе сообщения от таймера обновляется графическое содержимое экрана или окна, так что возникает эффект мультипликации.

Параметры функции setTimer:

1-й параметр – дескриптор окна, с которым ассоциируется таймер. Если этот параметр сделать равным NULL (0), то будет проигнорирован и второй параметр;

2-й параметр – определяет идентификатор таймера;

3-й параметр – определяет интервал посылки сообщения WM_TIMER;

4-й параметр – определяет адрес функции, на которую будет приходить сообщение WM_TIMER. Если параметр равен NULL, то сообщение будет приходить на функцию окна.

Если функция выполнилась успешно, то возвращаемым значением будет являться идентификатор таймера, который, естественно, будет совпадать со вторым параметром, если первый параметр будет отличным от NULL. В случае неудачи функция возвратит нуль.

Из сказанного следует, что функция может быть вызвана тремя способами: задан дескриптор окна, а четвертый параметр задается равным нулю; задан дескриптор окна, а четвертый параметр определяет функцию, на которую будет приходить сообщение WM_TIMER; дескриптор окна равен NULL, а четвертый параметр определяет функцию, на которую будет приходить сообщение WM_TIMER(идентификатор таймера в этом случае будет определяться по возвращаемому функцией значению).

Функция, на которую приходит сообщение WM_TIMER, имеет следующие параметры:

1-й параметр – дескриптор окна, с которым ассоциирован таймер;

2-й параметр – сообщение WM_TIMER;

3-й параметр – идентификатор таймера;

4-й параметр – время в миллисекундах, прошедшее с момента запуска Windows.

Функция KillTimer удаляет созданный параметр и имеет следующие параметры:

1-й параметр – дескриптор окна;

2-й параметр – идентификатор таймера.

Для работы с таймером необходимо определить константу

```
WM_TIMER      equ 113h
```

и прототипы внешних процедур:

```
EXTERN     SetTimer@16:NEAR
EXTERN     KillTimer@8:NEAR
```

Для компоновщика понадобится следующая библиотека:

```
includelib \masm32\lib\wmvcore.lib
```

Установка таймера, реализуется в оконной процедуре:

```
PUSH 0      ;Параметр = NULL
PUSH 1000   ;Интервал  1 с
PUSH 1      ;Идентификатор таймера
PUSH DWORD  PTR  [EBP+08H]
CALL    SetTimer@16
```

Удаление таймера

```
PUSH 1   ;Идентификатор таймера
PUSH DWORD  PTR  [EBP+08H]
CALL KillTimer@8
```

Обнаружение сообщения

```
CMP     DWORD PTR [EBP+0CH],WM_TIMER
```

Рассмотрим пример, в котором реализовано два таймера. Можно считать, что запускаются одновременно две задачи. Одна задача с периодичностью 0,5 сек. получает системное время и формирует строку для вывода (STRCOPY). Эта задача имеет свою собственную функцию, на которую приходит сообщение WM_TIMER. Вторая задача работает в рамках функции окна. Эта задача с периодичностью 1 сек. выводит время и дату в окно редактирования, расположенное на диалоговом окне. Таким образом, две задачи взаимодействуют друг с другом посредством глобальной переменной STRCOPY. Поскольку на функцию таймера приходит сообщение, в котором указан идентификатор таймера, мы можем на базе одной функции реализовать любое количество таймеров.

```
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
```

```
;Получить дескриптор приложения.
    PUSH  0
    CALL  GetModuleHandleA@4
    MOV   [HINST], EAX
;Создать диалоговое окно.
    PUSH  0
    PUSH  OFFSET WNDPROC
    PUSH  0
    PUSH  OFFSET PA
    PUSH  [HINST]
    CALL  DialogBoxParamA@20
    CMP   EAX,-1
    JNE   KOL
;Здесь можно разместить сообщение об ошибке.
KOL:
    PUSH  0
    CALL  ExitProcess@4
WNDPROC PROC
      PUSH EBP
      MOV  EBP, ESP
      PUSH EBX
      PUSH ESI
      PUSH EDI
      CMP  DWORD PTR [EBP+0CH],WM_CLOSE
      JNE  L1
;Удалить таймер 1
      PUSH 1   ;Идентификатор таймера.
      PUSH DWORD  PTR  [EBP+08H]
      CALL KillTimer@8
;Удалить таймер 2
      PUSH 2   ;Идентификатор таймера.
      PUSH DWORD  PTR  [EBP+08H]
      CALL KillTimer@8
;Закрыть диалог
      PUSH 0
      PUSH DWORD PTR [EBP+08H]
      CALL EndDialog@8
      JMP  FINISH
L1:
      CMP  DWORD PTR [EBP+0CH],WM_INITDIALOG
      JNE  L2
```

```
;Установить таймер 1
        PUSH 0      ;Параметр = NULL
        PUSH 1000   ;Интервал  1 с
        PUSH 1      ;Идентификатор таймера
        PUSH DWORD  PTR  [EBP+08H]
        CALL SetTimer@16
;Установить таймер 2
        PUSH OFFSET TIMPROC  ;Параметр = NULL
        PUSH 500             ;Интервал 0.5 с
        PUSH 2               ;Идентификатор таймера
        PUSH DWORD  PTR  [EBP+08H]
        CALL SetTimer@16
        JMP  FINISH
L2:
        CMP  DWORD PTR [EBP+0CH],WM_TIMER
        JNE  FINISH
;Отправить строку в окно
        PUSH OFFSET  STRCOPY
        PUSH 0
        PUSH WM_SETTEXT
        PUSH 1  ;Идентификатор элемента.
        PUSH DWORD PTR [EBP+08H]
        CALL SendDlgItemMessageA@20
FINISH:
        POP  EDI
        POP  ESI
        POP  EBX
        POP  EBP
        MOV  EAX,0
        RET  16
WNDPROC ENDP
;-------------------------------------
;Процедура таймера
;Расположение параметров в стеке.
;[EBP+014H]   LPARAM - промежуток запуска Windows.
;[EBP+10H]    WAPARAM - идентификатор таймера.
;[EBP+0CH]    WM_TIMER
;[EBP+8]      HWND
TIMPROC PROC
        PUSH EBP
        MOV  EBP,ESP
```

```
;Получить локальное время
        PUSH OFFSET DATA
        CALL GetLocalTime@4
;Получить  строку для вывода даты и времени
        MOVZX EAX, DATA.sec
        PUSH EAX
        MOVZX EAX, DATA.min
        PUSH EAX
        MOVZX EAX,DATA.hour
        PUSH EAX
        MOVZX EAX,DATA.year
        PUSH EAX
        MOVZX EAX,DATA.month
        PUSH EAX
        MOVZX EAX,DATA.day
        PUSH EAX
        PUSH OFFSET TIM
        PUSH OFFSET STRCOPY
        CALL  wsprintfA
;Восстановить стек
        ADD  ESP,32
        POP  EBP
        RET  16
TIMPROC ENDP
_TEXT   ENDS
        END START
```

Обратите внимание на весьма полезную функцию GetLocalTime. Информация, полученная с помощью этой функции, легко может быть использована для самых разных целей, в том числе и для вывода на экран. Аналогично, с помощью функции SetLocalTime, вы сможете установить текущее время. Для получения времени по Гринвичу используется функция GetSystemTime. Соответственно, с помощью SetSystemTime используется для установки времени в Гринвичском выражении.

Таблица 6.1 – Варианты заданий на лабораторную работу № 6

№	Задание на лабораторную работу

пп.	Т 1, сек.	Действие	Т 2, сек.	Действие
1	2	Открытие/закрытие CD-ROM	0,5	Счетчик сообщений WM_TIMER
2	3	Изменение текста в окне приложения	1	Отображение времени
3	1	Изменение размера окна	4	Инкремент даты
4	5	Создание/удаление файла	1,4	Изменение цвета окна
5	1,5	Изменение текста окна сообщения	3	Изменение цвета шрифта
6	0,8	Последовательный вывод текста	1,7	Счетчик сообщений WM_SIZE
7	1,7	Исчезновение и появление текста	2,5	Свернуть и развернуть окно
8	4,2	Отключение съемного устройства	0,9	Состояние съемного накопителя
9	2,5	Информация об изменении размера диска C:\	0,4	Счетчик сообщений WM_CHAR
10	4	Информация об изменении размера flash-диска	2,2	Список логических дисков
11	1,1	Вывод элементов арифметической прогрессии	5,1	Обновление содержимого файла с элементами
12	3,2	Вывод элементов геометрической прогрессии	1,2	Счетчик сообщений WM_RBUTTONDOWN
13	1,8	Счетчик сообщений WM_KEYDOWN	3,9	Центрирование курсора мыши
14	6	Отправка сообщения с помощью net send	0,3	Отображение координат положения курсора
15	3,5	Создание файла с именем равным текущему времени	1,5	Счетчик сообщений WM_KEYUP
16	2,3	Вывод элементов ряда Фибоначчи	4,4	Декремент 100 до 0 и сброс в 100
17	0,7	Вывод времени последнего изменения размера диска C:\	2,7	Обнуление счетчика WM_RBUTTONDOWN
18	4,5	Обнуление счетчика WM_MBUTTONDBLCLK	10	Создание MessageBox

Методика и порядок выполнения работы

1. Изучить принципы использования системного таймера.
2. Разработать оконное приложение, использующее таймер.
3. Защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Что такое системный таймер?
2. Для каких целей используют системный таймер?
3. Как взаимодействует приложение с системным таймером?
4. Как организовать работу двух таймеров в одном приложении?
5. Особенности использования двух и более таймеров.

Лабораторная работа № 6

Создание и использование потоков

Цель работы: Получить навык написания программ, использующих потоки.

Краткие сведения из теории

Поток может быть создан при помощи функции CreateThread. Рассмотрим параметры этой функции.

1-й параметр – указатель на структуру атрибутов доступа. Имеет значение только для Windows NT. Обычно полагается NULL.

2-й параметр – размер стека потока. Если параметр равен нулю, то берется размер стека по умолчанию, равный размеру стека родительского потока.

3-й параметр – указатель на потоковую функцию, с вызова которой начинается исполнение потока.

4-й параметр – Параметр для потоковой функции.

5-й параметр – флаг, определяющий состояние потока. Если флаг равен 0, то выполнение потока начинается немедленно. Если значение флага потока равно CREATE_SUSPENDED (4H), то поток находится в состоянии ожидания и запускается по выполнению функции ResumeThread.

6-й параметр – Указатель на переменную, куда будет помещен дескриптор потока.

Выполнение потока начинается с потоковой функции. Окончание работы этой функции приводит к естественному окончанию работы потока. Поток также может закончить свою работу, выполнив функцию ExitThread с указанием кода выхода. Наконец, порождающий поток может закончить работу порожденного потока при помощи функции TerminateThread. В нижеприведенном примере запускаемый процесс не может сам закончить свою работу и прекращает свою работу вместе с приложением по команде TerminateThread. Надо сказать, что такое завершение является аварийным и не рекомендуется к обычному употреблению. Связано это с тем, что при таком завершении не выполняется никаких действий по освобождению занятых ресурсов (блоки памяти, открытые файлы и т. п.). Поэтому разработайте свое приложение так, чтобы поток завершался по выходу из потоковой процедуры.

Идеальной является ситуация, когда функция окна берет на себя только реакцию на события, происходящие с элементами, а всю трудоемкую работу (сложные вычисления, файловая обработка) должны взять на себя потоки. Поток может создавать новые потоки, так что в результате может возникнуть целое дерево.

Ниже представлена программа, использующая поток для вычисления и вывода в окно редактирования текущей даты и времени. Заметим в этой связи, что если бы такая обработка была реализована в оконной функции, вы бы сразу почувствовали разницу – окно почти бы перестало реагировать на внешнее воздействие.

```
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
;Получить дескриптор приложения.
```

```
    PUSH  0
    CALL  GetModuleHandleA@4
    MOV   [HINST], EAX
;Создать диалоговое окно.
    PUSH  0
    PUSH  OFFSET WNDPROC
    PUSH  0
    PUSH  OFFSET PA
    PUSH  [HINST]
    CALL  DialogBoxParamA@20
    CMP   EAX,-1
    JNE   KOL
;Здесь можно разместить сообщение об ошибке.
KOL:
    PUSH  0
    CALL  ExitProcess@4
;Процедура окна.
WNDPROC PROC
      PUSH EBP
      MOV  EBP, ESP
      PUSH EBX
      PUSH ESI
      PUSH EDI
;Сообщение при закрытии окна.
      CMP  DWORD PTR [EBP+0CH],WM_CLOSE
      JNE  L1
L3:
;Здесь реакция на закрытие окна.
;Удалить поток.
      PUSH 0
      PUSH HTHR
      CALL TerminateThread@8
;Закрыть диалог.
      PUSH 0
      PUSH DWORD PTR [EBP+08H]
      CALL EndDialog@8
      JMP  FINISH
L1:
      CMP  DWORD PTR [EBP+0CH],WM_INITDIALOG
      JNE  L2
;Здесь начальная инициализация.
```

```
;Получить дескриптор окна редактирования.
        PUSH 1
        PUSH DWORD PTR [EBP+08H]
        CALL GetDlgItem@8
;Создать поток.
        PUSH OFFSET HTHR  ;Сюда дескриптор потока.
        PUSH 0
        PUSH EAX     ;Параметр.
        PUSH OFFSET GETTIME ;Адрес процедуры.
        PUSH 0
        PUSH 0
        CALL CreateThread@24
        JMP  FINISH
L2:
        CMP  DWORD PTR [EBP+0CH],WM_COMMAND
        JNE  FINISH
;Кнопка выхода?
        CMP  WORD PTR [EBP+10H],2
        JE   L3
FINISH:
        POP  EDI
        POP  ESI
        POP  EBX
        POP  EBP
        MOV  EAX,0
        RET  16
WNDPROC ENDP
;-------------------------------------
;Потоковая функция.
; [EBP+8]  параметр=дескриптор окна редактирования
GETTIME PROC
        PUSH EBP
        MOV  EBP,ESP
L0:
;Задержка в 1 секунду.
        PUSH 1000
        CALL Sleep@4
;Получить локальное время.
        PUSH OFFSET DATA
        CALL GetLocalTime@4
;Получить строку для вывода даты и времени.
```

```
        MOVZX EAX,DATA.sec
        PUSH EAX
        MOVZX EAX,DATA.min
        PUSH EAX
        MOVZX EAX,DATA.hour
        PUSH EAX
        MOVZX EAX,DATA.year
        PUSH EAX
        MOVZX EAX,DATA.month
        PUSH EAX
        MOVZX EAX,DATA.day
        PUSH EAX
        PUSH OFFSET TIM
        PUSH OFFSET STRCOPY
        CALL wsprintfA
;Отправить строку в окно редактирования.
        PUSH OFFSET STRCOPY
        PUSH 0
        PUSH WM_SETTEXT
        PUSH DWORD PTR [EBP+08H]
        CALL SendMessageA@16
        JMP  L0 ;Бесконечный цикл.
        POP  EBP
        RET  4
GETTIME ENDP
_TEXT   ENDS
        END START
```

Возьмите на вооружение весьма полезную функцию Sleep. Эта функция особенно часто используется именно в потоках, с тем чтобы высвободить процессорное время.

Методика и порядок выполнения работы

1. Изучить принципы создания и использования потоков.

2. Разработать оконное приложение, использующее потоки. Задание на лабораторную работу взять из таблицы 6.1.

3. Защитить лабораторную работу преподавателю.

Контрольные вопросы

1. Что такое поток?
2. В чем преимущество использования потоков?
3. Как создаются потоки?
4. Как правильно завершать работу потоков?
5. Как создается дерево потоков?

Лабораторная работа № 7

Многопотоковое программирование

Цель работы: Получить навык создания многопотоковых приложений.

Краткие сведения из теории

Семафор представляет собой глобальный объект, позволяющий синхронизировать работу двух или нескольких процессов или потоков. Для программиста семафор – это просто счетчик (хотя манипулировать им можно только при помощи специальных функций). Если счетчик равен N, это означает, что к ресурсу имеют доступ N процессов. Рассмотрим функции для работы с семафорами.

CreateSemaphor – создает глобальный объект-семафор. Возвращает дескриптор семафора. Параметры функции:

1-й параметр. Указатель на структуру, определяющую атрибуты доступа. Может иметь значение для Windows NT. Обычно данный параметр полагается равным NULL.

2-й параметр. Начальное значение счетчика семафора. Определяет, сколько задач имеют доступ к ресурсу вначале.

3-й параметр. Количество задач, которые имеют одновременный доступ к ресурсу.

4-й параметр. Указатель на строку, содержащую имя семафора.

OpenSemaphor – открыть уже созданный семафор. Возвращает дескриптор семафора. Данную функцию используют не так часто. Обычно создают семафор и присваивают его дескриптор глобальной переменной, а потом используют этот дескриптор в порождаемых потоках. Параметры функции:

1-й параметр. Определяет желаемый уровень доступа к семафору. Возможные значения:

SEMAPHORE_MODIFY_STATE (2h) – разрешить использование функции ReleaseSemaphore,

SYNCHRONIZE (100000h) – разрешить использование любой функции ожидания, только для Windows NT,

SEMAPHORE_ALL_ACCESS (0F0000h + SYNCHRONIZE + 3h) – специфицирует все возможные флаги доступа к семафору.

WaitForSingleObject – ожидать открытия семафора. При успешном завершении, т.е. открытии доступа к объекту, функция возвращает 0. Значение 102h говорит о том, что заданный период ожидания закончился. Параметры функции:

1-й параметр. Дескриптор семафора.

2-й параметр. Время ожидания в миллисекундах. Если параметр равен INFINITE (0FFFFFFFFh), то время ожидания не ограничено.

ReleaseSemaphor – освоб дить семафор и тем самым открыть доступ к ресурсу другим процессам. Параметры функции:

1-й параметр. Дескриптор семафора.

2-й параметр. Определяет, какое значение должно быть добавлено к счетчику семафора. Чаще всего этот параметр равен единице.

3-й параметр. Указатель на переменную, куда должно быть помещено предыдущее значение счетчика.

Рассмотрим алгоритм работы с семафором. Сначала при помощи функции CreateSemaphor создается семафор и его дескриптор присваивается глобальной переменной. Перед попыткой обращения к ресурсам, доступ к которым необходимо ограничить, поток должен вызвать функцию WaitForSingleObject. При открытии доступа функция возвращает 0. По окончании работы с ресурсом следует вызвать функцию ReleaseSemaphor. Тем самым увеличивается счетчик доступа на 1. С помощью семафора можно регулировать количество потоков, которые одновременно могут иметь доступ к ресурсу. Максимальное значение счетчика как раз и определяет, сколько потоков могут получить доступ к ресурсу одновременно. Но обычно максимальное значение полагают равным 1.

Событие является объектом, очень похожим на семафор, но в несколько видоизмененном виде. Рассмотрим функции для работы с событиями.

CreateEvent - сздает объект-событие. Параметры функции:

1-й параметр. Имеет тот же смысл, что и первый параметр функции CreateSemaphor. Обычно полагается равным NULL.

2-параметр. Если параметр не равен нулю, то событие может быть сброшено при помощи функции ResetEvent. Иначе событие сбрасывается при доступе к нему какого-либо процесса.

3-й параметр. Если параметр равен 0, то событие инициализируется как сброшенное, в противном случае сразу же подается сигнал о наступлении соответствующей ситуации.

4-й параметр. Указатель на строку, которая содержит имя события.

Ожидание события осуществляется, как и в случае с семафором, функцией WaitForSingleObject.

Функция OpenEvent аналогична функции OpenSemaphor, и на ней мы останавливаться не будем.

SetEvent – подать сигнал о наступлении события. Параметры функции:

1-й параметр. Дескриптор события.

Критическая секция – это фрагмент программы, защищенный от одновременного выполнения несколькими потоками. Критическую секцию в данный момент может выполнять только один поток. Рассмотрим функции для работы с критической секцией.

InitializeCriticaiSection – данная функция создает объект под названием критическая секция. Параметры функции:

1-й параметр. Указатель на структуру, указанную ниже. Поля данной структуры используются только внутренними процедурами, и смысл безразличен.

```
CRITICAL_SECTION STRUCT
        DebugInfo      DWORD ?
        LockCount      LONG ?
        RecursionCount LONG ?
        OwningThread   HANDLE ?
        LockSemaphore  HANDLE ?
        SpinCount      DWORD ?
CRITICAL_SECTION ENDS
```

EnterCriticalSection – войти в критическую секцию. После выполнения этой функции данный поток становится владельцем данной секции. Следующий поток, вызвав данную функцию, будет находиться в состоя-

нии ожидания. Параметр функции такой же, что и в предыдущей функции.

LeaveCriticalSection – покинуть критическую секцию. После этого второй поток, который был остановлен функцией EnterCriticalSection, станет владельцем критической секции. Параметр функции LeaveCriticalSection такой же, как и у предыдущих функций.

DeleteCriticalSection - удалить объект «критическая секция». Параметр аналогичен предыдущим.

Программно можно определить несколько объектов критической секции, с которыми будут работать несколько потоков. Мы не зря, говоря о критических секциях, упоминаем только потоки. Разные процессы не могут использовать данный объект синхронизации.

Мьютексы (Mutex) это объекты ядра, которые создаются функцией CreateMutex(). Мьютекс бывает в двух состояниях - занятом и свободном. Мьютексом хорошо защищать единичный ресурс от одновременного обращения к нему разными потоками.

Методика и порядок выполнения работы

1. Изучить принципы создания и использования потоков.

2. Написать программу, которая создаёт несколько потоков и синхронизирует их работу с использованием инструментов Critical Section, mutex и semaphore. Для управления работой потоков использовать только функции синхронизации.

3. Защитить лабораторную работу преподавателю.

Таблица 8.1 – Задания на лабораторную работу № 8.

№	Индивидуальное задание
1	Поток считает сумму ряда по 100 элементов, после чего он переходит в режим ожидания, пока в основном потоке пользователь не нажмет

	любую клавишу. На экран выводить текущее значение суммы ряда. Использовать mutex.
2	Создать 2 потока, которые по очереди считают значение определенного интеграла на заданном интервале, причем каждый поток считает только значение только на своём интервале. Оба потока используют одни и те же глобальные переменные для расчётов. Текущее значение интеграла и номер активного потока выводиться на экран. Использовать Critical section.
3	Создать 2 потока. Первый поток уменьшает значение какой либо глобальной переменной на фиксированное значение, второй – увеличивает то же значение. При нажатии на клавиши поток блокируется/активизируется. Значение величины и состояние каждого потока выводится на экран. Использовать mutex.
4	Создать 4 потока, которые увеличивают значение целой глобальной переменной. При достижении определенного значения переменная обнуляется. Пользователь с клавиатуры задаёт число одновременно работающих потоков. Текущее значение переменной выводится на экран. Использовать semaphore.
5	Создать 3 потока, которые через случайное время (50 – 1500 мс) меняют значение глобальной переменной на случайную величину (-10 … 10). После чего блокируют изменение этой переменной на это же время. Использовать функцию sleep и critical section. Отображать на экране значение этой переменной, и номер потока, изменивший её значение в последний раз.
6	Создать 4 потока, которые через равные интервалы времени (1 сек) меняют в строке местами 2 произвольных символа. Пользователь с клавиатуры задаёт количество одновременно работающих потоков. На экране отображать текущее состояние строки.
7	Создать 3 потока, которые по очереди рассчитывают значения функции на заданном диапазоне с заданным шагом. Каждый поток рассчитывает только по одному значению. На экран выводить текущее значение аргумента и функции, номер потока, рассчитавшего последнее значение. Использовать mutex.
8	Создать 2 потока. При нажатии на клавиши клавиатуры активизируется первый поток на какое-то случайное время (300-2000 мс). Каждые 200 мс он разрешает/запрещает расчет суммы числового ряда, которая считается вторым потоком. На экран выводить текущее значение суммы ряда. Использовать mutex.
9	Создать 2 потока. Первый через произвольное время (200-1000 мс) удаляет из начала строки символ (если строка не пуста) и блокирует

	изменение строки на 200 мс. Второй поток – добавляет символ в конец строки через 300-1200 мс, и блокирует изменение строки на 300мс. На экране отображать текущую строку, и номер потока, последним менявший строку. Использовать critical section.
10	Создать 2 потока, которые меняют значения 2х элементов массива в произвольной позиции по следующим правилам. Первый поток к значению большего элемента прибавляет 1, а от значения меньшего – отнимает 1. Второй поток – к меньшему прибавляет 1, от большего элемента – отнимает 1. Пользователь при нажатии клавиши моет блокировать/разрешать работу каждого потока. На экран выводить текущие значения элементов массива.
11	Создать 2 потока. Первый поток удаляет первый символ строки каждые 500 мс. Второй поток добавляет символ в конец строки через случайные промежутки времени (100 – 1000 мс). При нажатии на клавиши блокируется/разрешается работа каждого потока в отдельности. Для управления работой потоков использовать critical section.
12	Создать 3 потока. Первый поток добавляет символ «1» в конец строки через 250 – 2000 мс. Второй поток добавляет символ «2» в начало строки через 250 – 2000 мс. Третий поток удаляет символ из середины строки через каждые 250 мс. Пользователь с клавиатуры задает количество одновременно работающих потоков (0-3). Использовать semaphore.
13	Создать 2 потока. Первый поток циклически сдвигает символы текста вправо через случайные промежутки времени (100 – 2000 мс), второй поток – сдвигает влево. После сдвига изменение текста блокируется на время 500 мс. Использовать critical section и функцию sleep.
14	Создать 3 потока. Каждый поток считает сумму 1000 значений функции y=sin(x) с шагом 0.0001. Для вычислений использовать глобальные переменные. Для синхронизации вычислений использовать critical section.
15	Создать 3 потока, каждый из которых через произвольное время (100 – 3000 мс) удаляет из начала строки символ и добавляет в конец строки свой номер. После чего блокирует строку от изменений на 250 мс. Использовать critical section
16	Создать 2 потока. Первый поток генерирует через 100-500 мс случайное число. Второй поток считает среднее арифметическое этих случайных чисел. Значения выводятся на экран. Использовать mutex.
17	Создать 3 потока. Первый поток циклически сдвигает текст влево, второй – сдвигает циклически вправо, третий – добавляет/удаляет символ в/из середины строки. Потоки выполняют свои действия через

	100 – 2000 мс, после чего блокируют изменение строки на 200 мс. Если строка блокирована, то время до выполнения потоком следующей итерации удваивается. Использовать функцию sleep и critical section.
18	Создать 2 потока. Первый поток считает значение функции y=cos(x), начиная от 0 с шагом 0.001. Второй поток считает сумму значений этой функции. Использовать mutex.
19	Создать 2 потока, которые производят циклический сдвиг цифр числа через каждые 200 мс. Пользователь, нажимая цифровые на клавиши, задаёт количество итераций, выполняемых потоками. Использовать semaphore.
20	Создать 2 потока. Первый поток считает сумму положительных чисел, введенных пользователем с клавиатуры, второй поток – сумму отрицательных. Текущие значения выводить на экран. Использовать mutex.

Контрольные вопросы

1. Чем отличаются процессы и потоки?
2. Средства синхронизации потоков в WinAPI.
3. Преимущества многопоточных приложений.
4. В каких случаях необходимо использовать критические секции?
5. Отличие семафора от события.
6. Алгоритм использования семафора.
7. Для чего необходима синхронизация потоков?

Рекомендуемая литература

1. Пятибратов А.П. Вычислительные системы, сети и телекоммуникации. – М.: Финансы и статистика, 2005.

2. Головин Ю. А. Информационные сети:учебник.- М.: Академия, 2013.

3. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы : учебное.- М ; Спб. : Питер, 2009.

4. Переяслова, И. Г. Информационные технологии в экономике : учебное пособие - М. ; Ростов н/Д : Дашков и К : Академцентр, 2009.

5. Мельников В.П. Информационные технологии: учебник.- М.: Академия, 2012.

6. Черников Б. В. Информационные технологии управления : учебник. – М. : ИНФРА-М, 2011.

Приложение А

```
.386P
.MODEL FLAT, stdcall

; сообщение приходит при закрытии окна
WM_DESTROY        equ 2
; сообщение приходит при создании окна
WM_CREATE         equ 1
; сообщение при щелчке левой кнопкой мыши в области окна
WM_LBUTTONDOWN    equ 201h
; сообщение при щелчке правой кнопкой мыши в области окна
WM_RBUTTONDOWN    equ 204h

; свойства 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

; идентификатор стандартной иконки
IDI_APPLICATION   equ 32512
; идентификатор курсора
IDC_CROSS         equ 32515

; режим показа окна – нормальный
SW_SHOWNORMAL     equ 1

; прототипы внешних процедур
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

; директивы компоновщику для подключения библиотек
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib
;-------------------------------------------------
; структура сообщения
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
;---------
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

; сегмент данных
_DATA SEGMENT  DWORD PUBLIC USE32 'DATA'
    NEWHWND   DD 0
    MSG       MSGSTRUCT <?>
    WC        WNDCLASS  <?>
```

```
    HINST     DD 0 ; здесь хранится дескриптор приложения
    TITLENAME DB 'Простой пример 32-битного приложения',0
    CLASSNAME DB 'CLASS32',0
    CAP       DB 'Сообщение',0
    MES1      DB 'Вы нажали левую кнопку мыши',0
    MES2      DB 'Выход из программы. Пока!',0
_DATA ENDS

; сегмент кода
_TEXT SEGMENT DWORD PUBLIC USE32 'CODE'
START:
; получить дескриптор приложения
    PUSH     0
    CALL     GetModuleHandleA@4
    MOV      [HINST], EAX

REG_CLASS:
; заполнить структуру окна стиль
    MOV [WC.CLSSTYLE], style
    MOV [WC.CLWNDPROC], OFFSET WNDPROC
    MOV [WC.CLSCSEXTRA], 0
    MOV [WC.CLWNDEXTRA], 0
    MOV EAX, [HINST]
    MOV [WC.CLSHINSTANCE], EAX

;--------- иконка окна
    PUSH IDI_APPLICATION
    PUSH 0
    CALL LoadIconA@8
    MOV  [WC.CLSHICON], EAX

;---------- курсор окна
    PUSH IDC_CROSS
    PUSH 0
    CALL LoadCursorA@8
    MOV  [WC.CLSHCURSOR], EAX
;---------регистрируем класс
    MOV  [WC.CLBKGROUND], 17 ; цвет окна
    MOV DWORD PTR [WC.CLMENUNAME], 0
    MOV DWORD PTR [WC.CLNAME], OFFSET CLASSNAME
    PUSH OFFSET WC
```

```
    CALL RegisterClassA@4

; создать окно зарегистрированного класса
    PUSH 0
    PUSH [HINST]
    PUSH 0
    PUSH 0
    PUSH 400 ; DY – высота окна
    PUSH 400 ; DX – ширина окна
    PUSH 100 ; Y – координата левого верхнего угла
    PUSH 100 ; X – координата левого верхнего угла
    PUSH WS_OVERLAPPEDWINDOW
    PUSH OFFSET TITLENAME ; имя окна
    PUSH OFFSET CLASSNAME ; имя класса
    PUSH 0
    CALL CreateWindowExA@48

; проверка на ошибку
    CMP EAX, 0
    JZ  _ERR
    MOV [NEWHWND], EAX ; дескриптор окна
; ------------------------------------------------
    PUSH SW_SHOWNORMAL
    PUSH [NEWHWND]
    CALL ShowWindow@8; показать созданное окно
; ------------------------------------------------
    PUSH [NEWHWND]
    CALL UpdateWindow@4  ; команда перерисовать видимую
                         ; часть окна, сообщение WM_PAINT
; петля обработки сообщений
MSG_LOOP:
    PUSH 0
    PUSH 0
    PUSH 0
    PUSH OFFSET MSG
    CALL GetMessageA@16
    CMP EAX, 0
    JE  END_LOOP
    PUSH OFFSET MSG
    CALL TranslateMessage@4
    PUSH OFFSET MSG
```

```
    CALL DispatchMessageA@4
    JMP MSG_LOOP

END_LOOP:
; выход из программы (закрыть процесс)
    PUSH [MSG.MSWPARAM]
    CALL ExitProcess@4

_ERR:
    JMP END_LOOP
; ---------------------------------------------
; процедура окна
; расположение параметров в стеке
; [EBP+014H] LPARAM
; [EBP+10H] WPARAM
; [EBP+0CH] MES
; [EBP+8] HWND
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_LBUTTONDOWN   ;левая кнопка
    JE  LBUTTON
CMP DWORD PTR [EBP+0CH] ,WM_RBUTTONDOWN   ;правая кнопка
    JE  RBUTTON
    JMP DEFWNDPROC

; нажатие правой кнопки приводит к закрытию окна
RBUTTON:
    JMP WMDESTROY

; нажатие левой кнопки мыши
LBUTTON:
; выводим сообщение
    PUSH 0 ; MB_OK
```

```
    PUSH OFFSET CAP
    PUSH OFFSET MES1
    PUSH DWORD PTR [EBP+08H]
    CALL MessageBoxA@16
    MOV EAX, 0
    JMP FINISH

WMCREATE:
    MOV EAX, 0
    JMP FINISH

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 MES2
    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
```

Приложение Б

Индивидуальное задание на лабораторную работу №1

№	Обрабатываемые сообщения	Свойства окна	Режим показа окна	Цвет фона
1	WM_DESTROY WM_CREATE WM_MOVE WM_CHAR(ввод «г»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_HIDE	COLOR_SCROLLBAR
2	WM_DESTROY WM_CREATE WM_SIZE WM_CHAR(ввод «b»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWNORMAL	COLOR_BACKGROUND
3	WM_DESTROY WM_CREATE WM_MOUSEMOVE WM_CHAR(ввод «ю»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMINIMIZED	COLOR_ACTIVECAPTION
4	WM_DESTROY WM_CREATE WM_LBUTTONDOWN WM_CHAR(ввод «d»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMAXIMIZED	COLOR_INACTIVECAPTION
5	WM_DESTROY WM_CREATE WM_LBUTTONUP WM_CHAR(ввод «я»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_MAXIMIZE	COLOR_MENU
6	WM_DESTROY WM_CREATE WM_RBUTTONDOWN WM_CHAR(ввод «f»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWNOACTIVATE	COLOR_WINDOW

7	WM_DESTROY WM_CREATE WM_RBUTTONUP WM_CHAR(ввод «g»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOW	COLOR_WINDOWFRAME
8	WM_DESTROY WM_CREATE WM_MBUTTONDOWN WM_CHAR(ввод «h»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_MINIMIZE	COLOR_MENUTEXT
9	WM_DESTROY WM_CREATE WM_MBUTTONUP WM_CHAR(ввод «I»)	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMINNOACTIVE	COLOR_WINDOWTEXT
10	WM_DESTROY WM_CREATE WM_LBUTTONDBLCLK WM_CHAR(ввод «J»)	CS_VREDRAW CS_HREDRAW CS_DBLCLKS	SW_SHOWNA	COLOR_CAPTIONTEXT
11	WM_DESTROY WM_CREATE WM_RBUTTONDBLCLK WM_CHAR(ввод «Д»)	CS_VREDRAW CS_HREDRAW CS_DBLCLKS	SW_RESTORE	COLOR_ACTIVEBORDER
12	WM_DESTROY WM_CREATE WM_MBUTTONDBLCLK WM_CHAR(ввод «L»)	CS_VREDRAW CS_HREDRAW CS_DBLCLKS	SW_SHOWDEFAULT	COLOR_INACTIVEBORDER
13	WM_DESTROY WM_CREATE WM_CHAR(ввод «Ф») SIZE_MAXHIDE	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_FORCEMINIMIZE	COLOR_APPWORKSPACE

14	WM_DESTROY WM_CREATE WM_CHAR(ввод «N») SIZE_MAXSHOW	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_HIDE	COLOR_HIGHLIGHT
15	WM_DESTROY WM_CREATE WM_CHAR(ввод «ц») SIZE_MAXIMIZED	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWNORMAL	COLOR_HIGHLIGHTTEXT
16	WM_DESTROY WM_CREATE WM_CHAR(ввод «Ш») SIZE_MINIMIZED	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWMINIMIZED	COLOR_BTNFACE
17	WM_DESTROY WM_CREATE WM_CHAR(ввод «Q») WM_MENUSELECT	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWMAXIMIZED	COLOR_BTNSHADOW
18	WM_DESTROY WM_CREATE WM_CHAR(ввод «S») WM_SETCURSOR	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_MAXIMIZE	COLOR_GRAYTEXT
19	WM_DESTROY WM_CREATE WM_CHAR(ввод «п») WM_SHOWWINDOW	CS_VREDRAW CS_HREDRAW CS_NOCLOSE	SW_SHOWNOACTIVATE	COLOR_BTNTEXT
20	WM_DESTROY WM_CREATE WM_CHAR(ввод «U») WM_KEYUP	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOW	COLOR_INACTIVECAPTIONTEXT

21	WM_DESTROY WM_CREATE WM_CHAR(ввод «W») WM_KEYDOWN	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_MINIMIZE	COLOR_BTNHIGHLIGHT
22	WM_DESTROY WM_CREATE WM_CHAR(ввод «X») WM_SYSKEYUP	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWMINNOACTIVE	COLOR_3DDKSHADOW
23	WM_DESTROY WM_CREATE WM_CHAR(ввод «Y») WM_SYSKEYDOWN	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWNA	COLOR_3DLIGHT
24	WM_DESTROY WM_CREATE WM_CHAR(ввод «Z») WM_SYSDEADCHAR	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_RESTORE	COLOR_INFOTEXT
25	WM_DESTROY WM_CREATE WM_CHAR(ввод «q») WM_SYSCHAR	CS_VREDRAW CS_HREDRAW CS_GLOBALCLASS	SW_SHOWDEFAULT	COLOR_INFOBK

Приложение В

Таблица В.1 – Сообщения операционной системы Windows

Сообщение системы	Назначение
WM_ACTIVATE	Посылается функции окна перед активизацией и деактивизацией этого окна.
WM_ACTIVATEAPP	Посылается функции окна перед активизацией окна другого приложения.
WM_CHAR	Сообщение, возникающее при трансляции сообщения WM_KEYDOWN функцией TranslateMessage.
WM_CLOSE	Сообщение, приходящее на процедуру окна при его закрытии. Приходит до WM_DESTROY. Дальнейшее выполнение DefWindowProc, EndDialog или WindowsDestroy и вызывает появление сообщения WM_DESTROY.
WM_COMMAND	Сообщение, приходящее на функцию окна, при наступлении события с управляющим элементом, пунктом меню, а также от акселератора.
WM_CREATE	Первое сообщение, приходящее на функцию окна при его создании. Приходит один раз.
WM_DEADCHAR	Сообщение, возникающее при трансляции сообщения WM_KEYUP функцией TranslateMessage.
WM_DESTROY	Сообщение, приходящее на функцию окна при его уничтожении.
WM_GETTEXT	Посылается окну для получения текстовой строки, ассоциированной с данным окном (строка редактирования, заголовок окна и т.д.).
WM_HOTKEY	Генерируется при нажатии горячей клавиши.
WM_INITDIALOG	Сообщение, приходящее на функцию диалогового окна вместо сообщения WM_CREATE.
WM_KEYDOWN	Сообщение, генерируемое при нажатии клавиши клавиатуры и посылаемое окну, имеющему фокус ввода.
WM_KEYUP	Сообщение, генерируемое при отпускании клавиши клавиатуры и посылаемое окну, имеющему фокус ввода.
WM_LBUTTONDOWN	Сообщение генерируется при нажатии левой кнопки мыши.
WM_MENUSELECT	Посылается окну, содержащему меню, при выборе пункта меню.
WM_PAINT	Сообщение посылается окну перед его перерисовкой.
WM_QUIT	Сообщение, приходящее приложению (не окну) при выполнении функции PostQuitMessage. При получе-

Сообщение системы	Назначение
	нии этого сообщения происходит выход из цикла ожидания и, как следствие, выход из программы.
WM_RBUTTONDOWN	Сообщение генерируется при нажатии правой кнопки мыши.
WM_SETFOCUS	Сообщение, посылаемое окну, после того, как оно получило фокус.
WM_SETICON	Приложение посылает окну данное сообщение, чтобы ассоциировать с ним новую иконку (значок).
WM_SETTEXT	Сообщение, используемое приложением для посылки текстовой строки окну и интерпретируемое в зависимости от типа окна (обычное окно - заголовок, кнопка — надпись на кнопке, окно редактирования - содержимое этого окна и т.д.).
WM_SIZE	Посылается функции окна после изменения его размера.
WM_SYSCHAR	Сообщение, возникающее при трансляции сообщения WM_SYSKEYDOWN функцией TranslateMessage.
WM_SYSCOMMAND	Генерируется при выборе пунктов системного меню или меню окна.
WM_SYSDEADCHAR	Сообщение, возникающее при трансляции сообщения WM_SYSKEYUP функцией TranslateMessage.
WM_SYSKEYDOWN	Сообщение аналогично WM_KEYDOWN, но генерируется, когда нажата и удерживается еще и клавиша Alt.
WM_SYSKEYUP	Сообщение аналогично WM_SYSKEYDOWN, но генерируется при отпускании клавиши.
WM_TIMER	Сообщение, приходящее на функцию окна или специально определенную таймерную процедуру после определения интервала таймера при помощи функции SetTimer.
WM_VKEYTOITEM	Сообщение окну приложения, когда нажимается какая-либо клавиша при наличии фокуса на данном списке. Список должен иметь свойство LBS_WANTKEYBOARDINPUT.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

Методические указания к самостоятельным работам

Направление подготовки 15.03.04 Автоматизация технологических процессов и производств

Направленность (профиль) «Информационно-управляющие системы»

Квалификация выпускника – бакалавр

Невинномысск 2021

Методические указания предназначены для студентов направления подготовки 15.03.04 Автоматизация технологических процессов и производств и других технических специальностей. Они содержат рекомендации по организации самостоятельных работ студента для дисциплины «Микропроцессорные системы управления».

Методические указания разработаны в соответствии с требованиями ФГОС ВО в части содержания и уровня подготовки выпускников направления 15.03.04 Автоматизация технологических процессов и производств

Содержание

1 Подготовка к лекциям

Главное в период подготовки к лекционным занятиям – научиться методам самостоятельного умственного труда, сознательно развивать свои творческие способности и овладевать навыками творческой работы. Для этого необходимо строго соблюдать дисциплину учебы и поведения. Четкое планирование своего рабочего времени и отдыха является необходимым условием для успешной самостоятельной работы. В основу его нужно положить рабочие программы изучаемых в семестре дисциплин.

Каждому студенту следует составлять еженедельный и семестровый планы работы, а также план на каждый рабочий день. С вечера всегда надо распределять работу на завтрашний день. В конце каждого дня целесообразно подводить итог работы: тщательно проверить, все ли выполнено по намеченному плану, не было ли каких-либо отступлений, а если были, по какой причине это произошло. Нужно осуществлять самоконтроль, который является необходимым условием успешной учебы. Если что-то осталось невыполненным, необходимо изыскать время для завершения этой части работы, не уменьшая объема недельного плана.

Слушание и запись лекций – сложный вид вузовской аудиторной работы. Внимательное слушание и конспектирование лекций предполагает интенсивную умственную деятельность студента. Краткие записи лекций, их конспектирование помогает усвоить учебный материал. Конспект является полезным тогда, когда записано самое существенное, основное и сделано это самим студентом. Не надо стремиться записать дословно всю лекцию. Такое «конспектирование» приносит больше вреда, чем пользы. Запись лекций рекомендуется вести по возможности собственными формулировками. Желательно запись осуществлять на одной странице, а следующую оставлять для проработки учебного материала самостоятельно в домашних условиях.

Конспект лекций лучше подразделять на пункты, параграфы, соблюдая красную строку. Этому в большой степени будут способствовать пункты плана лекции, предложенные преподавателям. Принципиальные места, опре-

деления, формулы и другое следует сопровождать замечаниями «важно», «особо важно», «хорошо запомнить» и т.п. Можно делать это и с помощью разноцветных маркеров или ручек. Лучше если они будут собственными, чтобы не приходилось присить их у однокурсников и тем самым не отвлекать их во время лекции. Целесообразно разработать собственную «маркографию» (значки, символы), сокращения слов. Не лишним будет и изучение основ стенографии. Работая над конспектом лекций, всегда необходимо использовать не только учебник, но и ту литературу, которую дополнительно рекомендовал лектор. Именно такая серьезная, кропотливая работа с лекционным материалом позволит глубоко овладеть знаниями.

2 Подготовка к лабораторным работам

Подготовку к каждому практическому занятию студент должен начать с ознакомления с методическими указаниями, которые включают содержание работы. Тщательное продумывание и изучение вопросов основывается на проработке текущего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованную к данной теме. На основе индивидуальных предпочтений студенту необходимо самостоятельно выбрать тему доклада по проблеме и по возможности подготовить по нему презентацию.

Если программой дисциплины предусмотрено выполнение практического задания, то его необходимо выполнить с учетом предложенной инструкции (устно или письменно). Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы семинара, его выступлении и участии в коллективном обсуждении вопросов изучаемой темы, правильном выполнении практических заданий и контрольных работ.

В зависимости от содержания и количества отведенного времени на изучение каждой темы практическое занятие может состоять из четырех-пяти частей:

1. Обсуждение теоретических вопросов, определенных программой дисциплины.

2. Доклад и/ или выступление с презентациями по выбранной проблеме.

3. Обсуждение выступлений по теме – дискуссия.

4. Выполнение практического задания с последующим разбором полученных результатов или обсуждение практического задания.

5. Подведение итогов занятия.

Первая часть – обсуждение теоретических вопросов – проводится в виде фронтальной беседы со всей группой и включает выборочную проверку преподавателем теоретических знаний студентов. Примерная продолжительность — до 15 минут. Вторая часть — выступление студентов с докладами, которые должны сопровождаться презентациями с целью усиления наглядности восприятия, по одному из вопросов практического занятия. Обязательный элемент доклада – представление и анализ статистических данных, обоснование социальных последствий любого экономического факта, явления или процесса. Примерная продолжительность — 20-25 минут. После докладов следует их обсуждение – дискуссия. В ходе этого этапа практического занятия могут быть заданы уточняющие вопросы к докладчикам. Примерная продолжительность – до 15-20 минут. Если программой предусмотрено выполнение практического задания в рамках конкретной темы, то преподавателями определяется его содержание и дается время на его выполнение, а замет идет обсуждение результатов. Подведением итогов заканчивается практическое занятие.

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на самостоятельное изучение рекомендованной учебно-методической (а также научной и популярной) литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

3 Самостоятельное изучение темы. Конспект

Конспект – наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «обзор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место издания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник – непременное правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответсвтии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становиться малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об изложении содержания работы, текст конспекта может быть сплошным, с

выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу или дополняют конспект соответствующими материалами их других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект – это расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры, таблицы и схемы, взятые из книги. Следует помнить, что работа над конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из другими источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, писать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает, что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содержанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.

4 Подготовка к экзамену

Экзаменационная сессия – очень тяжелый период работы для студентов и ответственный труд для преподавателей. Главная задача экзаменов – проверка качества усвоения содержания дисциплины.

На основе такой проверки оценивается учебная работа не только студентов, но и преподавателей: по результатам экзаменов можно судить и о качестве всего учебного процесса. При подготовке к экзамену студенты повторяют материал курсов, которые они слушали и изучали в течение семестра, обобщают полученные знания, выделяют главное в предмете, воспроизводят общую картину для того, чтобы яснее понять связь между отдельными элементами дисциплины.

При подготовке к экзаменам основное направление дают программы курса и конспект, которые указывают, что в курсе наиболее важно. Основной материал должен прорабатываться по учебнику, поскольку конспекта недостаточно для изучения дисциплины. Учебник должен быть проработан в течение семестра, а перед экзаменом важно сосредоточить внимание на основных, наиболее сложных разделах. Подготовку по каждому разделу следует заканчивать восстановлением в памяти его краткого содержания в логической последовательности.

До экзамена обычно проводится консультация, но она не может возместить отсутствия систематической работы в течение семестра и помочь за несколько часов освоить материал, требующийся к экзамену. На консультации студент получает лишь ответы на трудные или оставшиеся неясными вопросы. Польза от консультации будет только в том случае, если студент до нее проработает весь материал. Надо учиться задавать вопросы, вырабатывать привычку пользоваться справочниками, энциклопедиями, а не быть на иждивении у преподавателей, который не всегда может тут же, «с ходу» назвать какой-либо факт, имя, событие. На экзамене нужно показать не только знание предмета, но и умение логически связно построить устный ответ.

Получив билет, надо вдуматься в поставленные вопросы для того, чтобы правильно понять их. Нередко студент отвечает не на тот вопрос, который поставлен, или в простом вопросе ищет скрытого смысла. Не поняв вопроса и не обдумав план ответа, не следует начинать писать. Конспект своего ответа надо рассматривать как план краткого сообщения на данную тему и составлять ответ нужно кратко. При этом необходимо показать умение выражать мысль четко и доходчиво.

Отвечать нужно спокойно, четко, продуманно, без торопливости, придерживаясь записи своего ответа. На экзаменах студент показывает не только свои знания, но и учится владеть собой. После ответа на билет могут следовать вопросы, которые имеют целью выяснить понимание других разделов курса, не вошедших в билет. Как правило, на них можно ответить кратко, достаточно показать знание сути вопроса. Часто студенты при ответе на дополнительные вопросы проявляют поспешность: не поняв смысла того, что у них спрашивают, начинают отвечать и нередко говорят не по сути.

Следует помнить, что необходимым условием правильного режима работы в период экзаменационной сессии является нормальный сон, поэтому подготовка к экзаменам не должна быть в ущерб сну. Установлено, что сильное эмоциональное напряжение во время экзаменов неблагоприятно отражается на нервной системе и многие студенты из-за волнений не спят ночи перед экзаменами. Обычно в сессию студенту не до болезни, так как весь организм озабочен одним - сдать экзамены. Но это еще не значит, что последствия неправильно организованного труда и чрезмерной занятости не скажутся потом. Поэтому каждый студент помнить о важности рационального распорядка рабочего дня и о своевременности снятия или уменьшения умственного напряжения.