Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации и переподготовки

Кафедра «Профессиональная переподготовка»

Е. И. Гридина

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ПРАКТИКУМ
по одноименной дисциплине
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение информационных систем»
заочной формы обучения

Гомель 2018

УДК 004.921(075.8)
ББК 32.973-018.2я73
Г82

Рекомендовано кафедрой «Профессиональная переподготовка» ИПКиП ГГТУ им. П. О. Сухого
(*протокол № 4 от 26.12.2017 г.*)

Рецензент: доц. каф. «Информатика» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *В. И. Мисюткин*

Гридина, Е. И.

Г82 Системное программирование : практикум по одноим. дисциплине для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / Е. И. Гридина. – Гомель : ГГТУ им. П. О. Сухого, 2018. – 71 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

Данный практикум написан в соответствии с требованиями, предъявленными к оформлению методических пособий, доступным языком и содержит множество примеров, позволяющих в кратчайшие сроки овладеть основными принципами системного программирования. Рассмотрены графический интерфейс Windows-приложения, стандартные диалоговые окна, элементы управления, растровая графика, DLL-библиотеки, процессы и потоки. Материал иллюстрирован многочисленными примерами, выполненными в Visual Studio 2013 под управлением Windows 7.

Издание адресовано слушателям ИПКиП специальности 1-40 01 73 «Программное обеспечение информационных систем.

УДК 004.921(075.8)
ББК 32.973-018.2я73

Содержание

Лабораторная работа № 1

Тема работы: Интерфейс Windows-приложения

Цель работы: Научиться организовывать интерфейса приложения на основе меню

Теоретические сведения по изучению темы

Интерфейс прикладной программы (API) – это набор функций, совместно вызываемых из одной программы (или из связанного набора программ), который программист использует при создании приложений.

Win32 API в своем ядре имеет три главных компонента, обеспечивающих его функциональность:

– компонент USER32.DLL отвечает за управление окном (включая сообщения), работу с курсорами, коммуникации, таймеры и множество других функций, не имеющих отношения к отображению окон, но предназначенных управлять окном;

– компонент GDI32.DLL – это интерфейс графических устройств; он отвечает за рисование элементов пользовательского интерфейса и графики, включая файлы Windows, растровые изображения, контексты устройств и шрифты;

– компонент KERNEL32.DLL контролирует все низкоуровневые функции управления памятью, распределения задач и ресурсов, что является сердцевиной Windows.

При программировании для Windows осуществляется работа с объектом, называемым “окном”.

Окно приложения обычно содержит заголовок, меню, рамку и иногда полосы прокрутки. Окна диалога – это дополнительные окна. Больше того, в окне диалога всегда имеется еще несколько окон, называемых “дочерними”. Эти дочерние окна имеют вид кнопок, переключателей, флажков, полей текстовго ввода или редактирования, списков и полос прокрутки.

Окно – это объект. Код – это оконная процедура. Данные – это информация, хранимая оконной процедурой, и информация, хранимая системой Windows для каждой окна и каждого окна и каждого класса окна, которые имеются в системе.

Пользователь рассматривает окна на экране в качестве объектов и непосредственно взаимодействует с этими объектами, нажимая

кнопки и переключатели, передвигая бегунок на полосах прокрутки. Положение программиста аналогично положению пользователя. Окно получает от пользователя информацию в виде оконных “сообщений”. Кроме этого окно обменивается сообщениями с другими окнами.

Программы для Windows также имеют точку входа, и Windows ищет функцию WinMain() именно как начальную точку входа приложения в период выполнения. Фунция WinMain() имеет следующий прототип: **int FAR PASCAL WinMain (HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmdLine, int CmdShow).**

Действия, которые вы программируете в функции WinMain(), могут быть настолько простыми или же настолько сложными, насколько позволяет ваше умение, но в любом случае эта функция должна выполнить следующее:

- инициализировать приложение;
- инициализировать и создать окна приложения;
- войти в цикл сообщений приложения.

Каждое окно, создаваемое Windows, имеет свой собственный цикл сообщений и свою функцию обработки сообщений (оконную процедуру), предназначенную для обработки сообщений именно этого окна. Когда главный цикл сообщений приложения вызывает DispatchMessage(), Windows передает сообщение оконной процедуре приложения. Оконная процедура – это функция, обычно называемая WndProc(), которая обрабатывает сообщения данного окна. Именно в этот момент программа решает, что делать с информацией, содержащейся в структуре MSG обрабатываемого сообщения.

Вот так определяется оконная процедура: **LRESULT CALLBACK WndProc(HWND hWnd, UINT message, PARAM wParam, LPARAM lParam);**

Аргументы функции WndProc() представляют собой первые четыре члена структуры MSG сообщения, которое в данный момент обрабатывается. Эти параметры были извлечены и посланы оконной процедуре благодаря функции DispatchMessage(). Все сообщения приложению передаются из цикла соощений главной оконной процедуре приложения.

Оконная процедура обрабатывает сообщения, поступающие окну. Очень часто эти сообщения передают окну информацию о том, что пользователь осуществил ввод с помощью клавиатуры или мыши. Таким образом, например, кнопки “узнают” о том, что они нажаты.

Другие сообщения говорят окну о том, что необходимо изменить размер окна или о том, что поверхность окна необходимо перерисовать.

Одним из компонентов окна является меню.

Меню представляет собой список команд, позволяющих выполнить любую операцию, реализованную приложением. Различают следующие виды меню:

– главное меню;
– системное меню;
– контекстное меню.

Ресурсы являются составной частью приложений для Windows. В них определяются такие объекты, как пиктограммы, курсоры, растровые образы, таблицы строк, меню, диалоговые окна и многие другие.

Для некоторых видов ресурсов система содержит предопределенные (стандартные) объекты. Все нестандартные ресурсы должны быть определены в файле описания ресурсов (resource script), который является ASCII-файлом с расширением .rc. Хотя теоретически такой файл можно подготовить в обычном текстовом редакторе, подобная технология используется крайне редко, поскольку любая интегрированная среда содержит удобные редакторы ресурсов, максимально упрощающие и автоматизирующие этот процесс.

Файл описания ресурса транслируется компилятором ресурсов (файл rc.exe в составе интегрированной среды). В результате образуется бинарный файл с расширением .res. Затем компоновщик включает полученный файл в выполняемый файл программы вместе с обычным кодом и данными программы из файлов с расширениями .obj и .lib. При работе в среде Visual Studio после выполнения команды Build все эти шаги производятся автоматически.

При загрузке в память исполняемого кода программы Windows обычно оставляет ресурсы на диске. Потом они загружаются в память только по мере необходимости.

Редакторы ресурсов содержат инструменты и интерфейсы для быстрого и удобного обслуживания ресурсов приложения. В составе MS Visual Studio имеется следующий набор редакторов ресурсов:

– графический редактор, который помогает создавать пиктограммы, курсоры, растровые образы;
– редактор меню;
– редактор таблиц быстрых клавиш;
– редактор диалоговых окон;
– редактор таблиц строк;

– редактор панелей инструментов;
– редактор информации о версии;
– редактор ресурсов, определяемых программистом.

Пример выполнения:

Создаем проект в Microsoft Visual Studio, для этого выбираем тип проекта – Проект Win32, (рисунок 1.1).

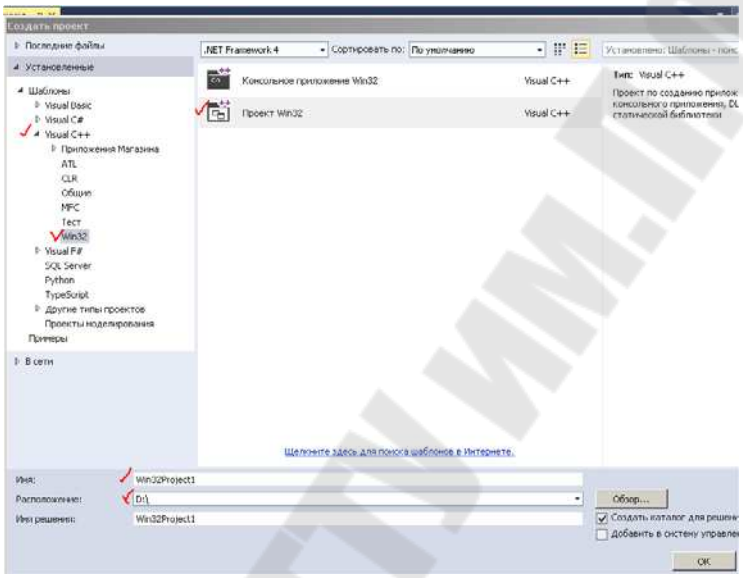

Рисунок 0.1 – Диалоговое окно создания проекта

В результате создан минимальный Windows – проект, (рисунок 1.2).

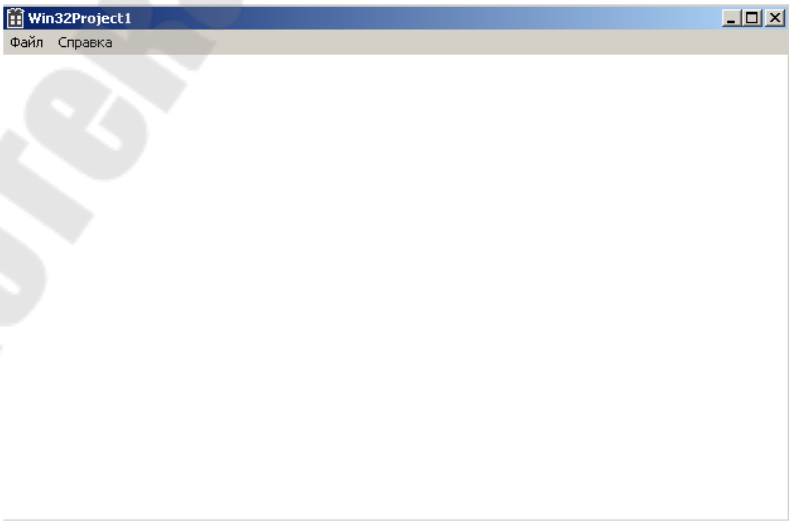

Рисунок 0.2 – Минимальное Windows-приложение

Мастер Visual Studio позволяет автоматически генерировать стандартную заготовку Windows-приложения. Для этого в стартовом окне построителя Win32-приложения достаточно выбрать кнопку Готово. Проект состоит из набора файлов, показанных на рисунке 1.3.

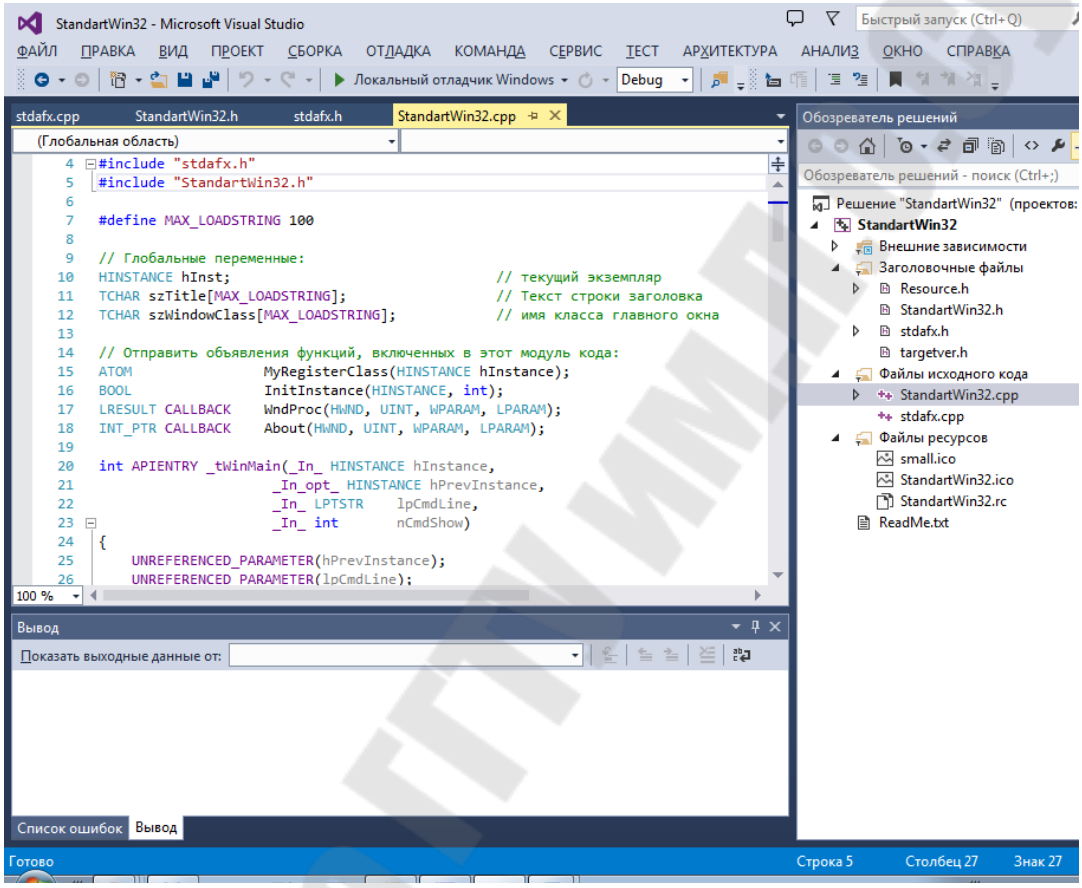

Рисунок 0.3 – Состав стандартной заготовки Win32-приложения

Мастер создает четыре файла включений к стандартной заготовке, а также два файла реализации. Это сделано в целях сокращения времени компиляции приложения. Дело в том, что при внесении изменений в проект происходит лишь частичная компиляция измененных файлов. Поэтому основные библиотеки подключаются файлом stdafx.cpp. В файлах включений stdafx.h и targetver.h размещены основные определения и ключи.

Включение меню в приложение требует следующих шагов:

- определение шаблона меню в файле описания ресурсов;
- загрузка меню при создании главного окна;
- обработка событий меню.

Для того чтобы определить меню в файле описания ресурсов необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource…** , (рисунок 1.4).

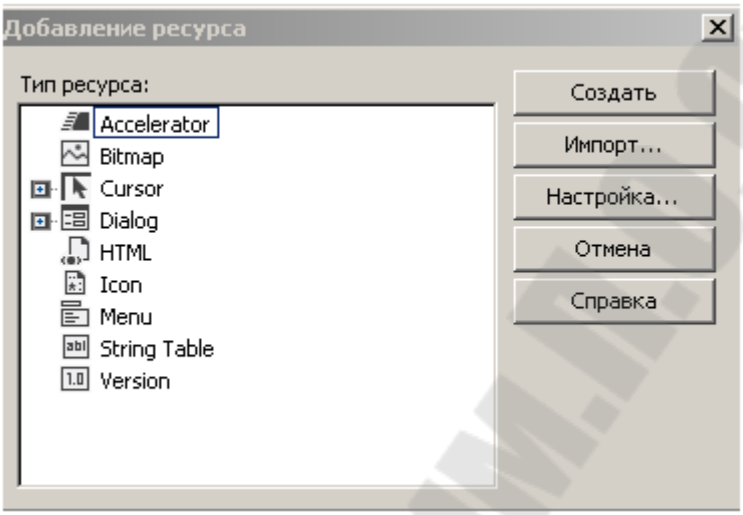

Рисунок 0.4 – Диалоговое окно добавления ресурса

В этом диалоговом окне необходимо выбрать из списка **Menu** и нажать кнопку **New**, в результате чего будет открыто окно редактора меню с заготовкой полосы нового меню, (рисунок 1.5).

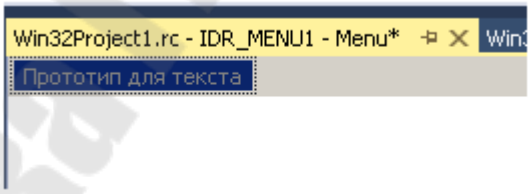

Рисунок 0.5 – Диалоговое окно для работы с меню

По умолчанию редактор присваивает первому из создаваемых шаблонов меню идентификатор **IDR_MENU1**.

Впоследствии этот идентификатор можно изменить на другой идентификатор, отражающий семантику ресурса и выполнить разработку структуру меню, (рисунок 1.6).

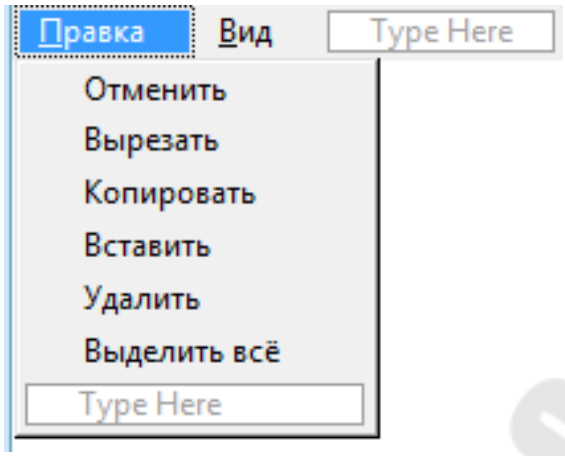

Рисунок 0.6 – Разработанное меню

Свойства меню:

– свойство **Caption** определяет название пункта меню. Если в названии встречается символ «**&**», то следующий за ним символ является мнемоническим и будет выделен подчёркиванием при нажатии клавиши <**Alt**>. Это позволяет определить горячую клавишу (**hotkey**) для быстрого способа выбора пункта при помощи клавиатуры. Например, на представленном выше изображении пункт меню «**Правка**» имеет мнемонический символ <**П**>, что позволяет выбрать пункт, используя комбинацию клавиш <**Alt**><**П**>;

– свойство **ID** определяет идентификатор пункта меню (разумеется, только в том случае, если пункт меню является командой, а не «**popup**»);

– свойство **Checked** определяет, будет ли отмечен пункт меню;

– свойство **Grayed** обуславливает недоступность пункта меню в исходном состоянии;

– свойство **Popup** при значении **True** определяет подменю. В противном случае пункт меню является обычной командой;

– свойство **Break** может принимать одно из трех значений:

o **None** — обычный пункт меню;

o **Column** — для меню верхнего уровня пункт выводится с новой строки, а для подменю — в новом столбце;

o **Bar** — дополнительный столбец подменю отделяется вертикальной линией.

– свойство **Separator** задаёт горизонтальную разделительную линию.

После определения меню в файле описания ресурсов оно ещё не появится в составе главного окна приложения. Для этого необходимо присоединить меню к окну. Один из способов присоединения меню к окну – указание идентификатора меню в свойстве **Menu** диалогового окна.

Существует альтернативный программный способ присоединения меню к главному окну приложения. Для этого сначала необходимо загрузить меню из ресурсов приложения, используя функцию API **LoadMenu**: HMENU LoadMenu(INSTANCE hInstance, LPCTSTR lpMenuName);, где hInstance – дескриптор приложения, lpMenuName – указатель на строку, содержащую имя меню в ресурсах.

После загрузки меню, его необходимо установить с помощью функции API **SetMenu**: BOOL SetMenu(HWND hWnd, HMENU hMenu);, где hWnd – дескриптор окна, к которому присоединяется меню, hMenu – дескриптор меню.

Для присоединения меню к диалоговому окну приложения, указываем: HMENU hMenu = LoadMenu(GetModuleHandle(NULL), MAKEINTRESOURCE(IDR_MENU2)); и присоединяем меню к главному окну приложения: SetMenu(hDialog, hMenu).

Следует также отметить, что Win API предоставляет функции, позволяющие получить дескриптор как для главного меню, так и для любого подменю .

Для получения дескриптора главного меню служит функция API **GetMenu**: HMENU GetMenu(HWND hWnd).

Для получения дескриптора подменю служит функция API **GetSubMenu**: HMENU GetSubMenu (HMENU hMenu, int nPos), где hMenu – дескриптор родительского меню, nPos – позиция пункта-подменю в родительском меню.

Операционная система Windows посылает сообщение **WM_COMMAND** при каждом выборе пункта меню, определяющего команду. При этом **LOWORD(wParam)** содержит идентификатор пункта меню, а **HIWORD(wParam)** и **lParam** содержат нулевые значения.

Чаще всего **WM_COMMAND –** единственное сообщение, обрабатываемое приложением, которое может поступить от меню. При выборе пунктов системного меню вместо указанного сообщения отправляется сообщение **WM_SYSCOMMAND**.

Иногда в программе может потребоваться обработка сообщений **WM_INITMENU** и **WM_INITMENUPOPUP**. Они отправляются непосредственно перед активизацией главного меню или выпадающего меню. Эти сообщения позволяют приложению изменить меню перед тем, как оно будет отображено на экране.

При навигации по меню система отправляет также сообщение **WM_MENUSELECT**. Оно более универсально по сравнению с **WM_COMMAND**, так как инициируется даже тогда, когда выделен недоступный или запрещенный пункт меню. Это сообщение может использоваться для формирования контекстной справки меню, которая отображается в строке состояния приложения.

Сообщение **WM_MENUSELECT** удобно использовать для формирования контекстной справки меню, которая отображается в строке состояния приложения. Для отображения текста контекстной справки в строке состояния может быть использован **ресурс таблицы строк**. Таблица строк представляет собой список строк, связанных с уникальными целочисленными идентификаторами.

Для того чтобы определить таблицу строк в файле описания ресурсов необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource…**

В этом диалоговом окне следует выбрать из списка **String Table** и нажать кнопку **New**, в результате чего будет открыто окно редактора таблицы строк, в котором необходимо определить список строк, указав для каждой из них целочисленный идентификатор.

Для удобства формирования контекстной справки меню каждой строке сопоставлется уже существующий идентификатор одного из пунктов меню. Это позволяет при обработке сообщения **WM_MENUSELECT** загрузить из ресурсов ту строку, которая соответствует выделенному пункту меню. После этого загруженная строка выводится в строку состояния, тем самым, обеспечивая контекстную справку для выделенного пункта меню.

Способ создания меню с помощью средств интегрированной среды разработки приложений не является единственным. Альтернативным подходом является программное (динамическое) создание меню.

Для создания и модификации меню используются следующие функции API.

Функция API **CreateMenu** предназначена для создания главного меню приложения:HMENU CreateMenu(VOID).

Меню, созданное этой функцией, изначально будет пустым, т.е. не будет содержать ни одного пункта. Заполнить меню пунктами можно с помощью функций API **AppendMenu** или **InsertMenu**.

Функция API **CreatePopupMenu** предназначена для создания всплывающего меню приложения: HMENU CreatePopupMenu(VOID);

Всплывающее меню, созданное этой функцией, также изначально будет пустым. Заполнить меню пунктами можно с помощью уже упомянутых функций API **AppendMenu** или **Insert-Menu**.

Функция API **AppendMenu** применяется для добавления пунктов к концу меню. BOOL AppendMenu(HMENU hMenu, UINT uFlags, UINT_PTR uIDNewItem, LPCTSTR lpNewItem), где hMenu – дескриптор меню, к которому добавляется новый пункт, UINT uFlags, – внешний вид и правило поведения добавляемого пункта меню, UINT_PTR uIDNewItem, – идентификатор для нового пункта меню или дескриптор выпадающего меню, LPCTSTR lpNewItem – содержимое нового пункта меню – зависит от второго параметра.

Второй параметр может иметь одно или несколько значений (флагов), приведенных ниже. В последнем случае флаги объединяются с помощью побитовой операции «ИЛИ»:

– **MF_POPUP** – создает всплывающее меню. В этом случае третий параметр функции содержит дескриптор всплывающего меню;

– **MF_CHECKED** – помещает отметку рядом с пунктом меню;

– **MF_DEFAULT** – пункт меню установлен в качестве применяемого по умолчанию. Имя этого пункта выделяется жирным шрифтом. В этом случае при открытии подменю двойным щелчком мыши, Windows автоматически выполнит команду по умолчанию, закрыв при этом подменю;

– **MF_ENABLED –** делает пункт меню доступным для выбора;

– **MF_DISABLED** – делает пункт меню недоступным для выбора;

– **MF_GRAYED** – выделяет серым цветом пункт меню и запрещает его выбор;

– **MF_UNCHECKED** – пункт меню не имеет отметки;

– **MF_SEPARATOR –** указывает, что пункт меню является разделителем (сепаратором);

– **MF_STRING** – отображает пункт меню с использованием текстовой строки, которая задается в четвертом параметре;

Функция API **InsertMenu** позволяет вставить новый пункт в меню. BOOL InsertMenu(HMENU hMenu, UINT uPosition, UINT uFlags, PTR uIDNewItem, LPCTSTR lpNewItem), где hMenu – дескриптор меню, которое модифицируется, uPosition – идентификатор или позиция пункта меню, перед которым происходит вставка нового пункта, uFlags – интерпретация второго параметра, а также внешний вид и правило поведения нового пункта меню, uIDNewItem – идентификатор для нового пункта меню или дескриптор выпадающего меню, lpNewItem – содержимое нового пункта меню – зависит от третьего параметра.

Второму параметру функции **uPosition** передается либо идентификатор пункта меню, либо позиция пункта меню. Выбор варианта интерпретации задается в третьем параметре:

– **MF_BYCOMMAND** – в этом случае второй параметр должен содержать идентификатор пункта меню, перед которым происходит вставка нового пункта;

– **MF_BYPOSITION** – в этом случае второй параметр должен содержать относительную позицию пункта меню с отсчетом от нуля.

Кроме того, в третьем параметре можно дополнительно указать один или несколько флагов, приведенных выше при рассмотрении функции **AppendMenu**.

Функция API **ModifyMenu** применяется для изменения существующего пункта меню: BOOL ModifyMenu(HMENU hMenu, UINT uPosition, UINT uFlags, PTR uIDNewItem, LPCTSTR lpNewItem).

Назначение параметров функции **ModifyMenu** аналогично функции **InsertMenu**.

Функция API **DeleteMenu** применяется для удаления отдельного пункта из указанного меню. BOOL DeleteMenu(HMENU hMenu, UINT uPosition, UINT uFlags).

Следует отметить, что если удаляемым пунктом является выпадающее меню, то оно уничтожается и высвобождается память.

После вызова функций **AppendMenu**, **InsertMenu** или **Delete-Menu** с целью модификации главного меню следует обязательно вызвать рассмотренную ранее функцию API **DrawMenuBar** для повторного отображения изменившейся полосы меню.

Функция API **DestroyMenu** предназначена для уничтожения меню и высвобождения памяти, занимаемой меню: BOOL DestroyMenu(HMENU hMenu).

Контекстное меню – это меню, которое появляется в любой части окна приложения при щелчке правой кнопкой мыши. При этом содержание контекстного меню изменяется в зависимости от «**контекста**» – места, где произведен щелчок правой кнопкой мыши. Обычно контекстное меню содержит пункты, которые дублируют команды основного меню, но сгруппированные иначе, чтобы максимально облегчить пользователю работу с приложением.

Создание контекстного меню выполняется аналогично созданию главного меню приложения. Для этого используется два подхода:

– определение шаблона меню в файле описания ресурсов;

– программное (динамическое) создание меню.

Создавая шаблон контекстного меню с помощью редактора меню, необходимо определить нулевой пункт меню нулевого уровня как подменю, имеющее какое-нибудь условное имя. Этот пункт нигде не будет отображаться. Он необходим только для получения дескриптора подменю с помощью функции API **GetSubMenu**, рассмотренной ранее.

После определения контекстного меню в файле описания ресурсов необходимо его загрузить с помощью функции API **Load-Menu**. Данная функция вернет дескриптор фиктивного меню нулевого уровня, которое не должно отображаться на экране.

Содержанием контекстного меню является нулевой пункт указанного меню, поэтому окончательное значение дескриптора контекстного меню определяется вызовом функции **GetSubMenu**. Полученный дескриптор контекстного меню затем передается функции **TrackPopupMenu**, которая выводит всплывающее контекстное меню на экран: BOOL TrackPopupMenu(HMENU hMenu, UINT uFlags, int x, int y, int nReserved, HWND hWnd, HWND prcRect), где hMenu – дескриптор контекстного меню, uFlags – флаги, определяющие позиционирование и другие опции меню, x – горизонтальное расположение контекстного меню в экранных координатах, int y – вертикальное расположение контекстного меню в экранных координатах, nReserved – зарезервированное значение; должно быть равно 0, hWnd – дескриптор окна, которому принадлежит контекстное меню, prcRect – параметр игнорируется).

Как известно, при щелчке правой кнопкой мыши Windows отправляет оконной процедуре приложения сообщение **WM_RBUTTONDOWN**, содержащее в **LPARAM** клиентские координаты курсора мыши в момент щелчка. Помимо этого сообщения в оконную процедуру приложения приходит сообщение **WM_CONTEXTMENU**, содержащее в **LPARAM** экранные координаты курсора мыши (в младшей части **LPARAM** находится координата **X** положения курсора мыши, а в старшей части - координата **Y**), а в **WPARAM** дескриптор окна, в котором произведен щелчок, (рисунок 1.7).

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    HMENU hSubMenu, hMenu;
    int xpos, ypos;
    switch (message)
    {
    case WM_RBUTTONDOWN:
        xpos = LOWORD(lParam);
        ypos = HIWORD(lParam);
        // Загрузим меню из ресурсов приложения
        hMenu = LoadMenu(GetModuleHandle(NULL), MAKEINTRESOURCE(IDR_MENU1));
        // Получим дескриптор подменю
        hSubMenu = GetSubMenu(hMenu, 0);
        // Отобразим контекстное меню
        TrackPopupMenu(hSubMenu, TPM_LEFTALIGN, xpos, ypos, 0, hWnd, NULL);
        break;
```

Рисунок 0.7 – Подключение контекстного меню

Для многих пунктов меню определены клавиатурные комбинации (**акселераторы**), предоставляющие альтернативный способ вызова команд меню.

Чтобы добавить в приложение обработку акселераторов, необходимо выполнить следующую последовательность действий:

– модифицировать определение ресурса меню, добавив к имени каждого дублируемого пункта информацию о быстрой клавише;

– определить таблицу акселераторов в файле описания ресурсов;

– обеспечить загрузку таблицы акселераторов в память приложения;

– модифицировать цикл обработки сообщений в функции **WinMain**.

Для того чтобы определить таблицу акселераторов в файле описания ресурсов необходимо воспользоваться редактором таблиц акселераторов. Для этого необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource…** и выбрать **Accelerator.**

В этом диалоговом окне необходимо выбрать из списка **Accelerator** и нажать кнопку **New**, в результате чего будет открыто окно редактора таблицы акселераторов.

По умолчанию редактор присваивает таблице акселераторов имя **IDR_ACCELERATOR1**. Впоследствии этот идентификатор можно изменить на другой идентификатор, отражающий семантику ресурса.

Поскольку акселераторы ставятся в соответствие командам меню, то они должны иметь те же идентификаторы, что и элементы меню, которым они соответствуют. Данное требование не является обязательным при реализации акселераторов, однако если оно не будет соблюдаться, то идея акселераторов как средства быстрого вызова команд меню потеряет смысл.

Во время работы приложения для загрузки таблицы акселераторов в память и получения ее дескриптора используется функция API **LoadAccelerators**, (рисунок 1.8).

```
int APIENTRY _tWinMain(_In_ HINSTANCE hInstance,
                       _In_opt_ HINSTANCE hPrevInstance,
                       _In_ LPTSTR    lpCmdLine,
                       _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: разместите код здесь.
    MSG msg;
    HACCEL hAccelTable;
    hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_WIN32PROJECT1));
```

Рисунок 0.8 – Загрузка таблицы акселераторов

Задание:

Разработать Windows-приложение, согласно варианта, представленого в Приложении А.

Реализовать в приложении:

- наличие меню (главное, контекстное);
- наличие таблицы акселлераторов;
- наличие иконки, курсора.

Контрольные вопросы:

1) Стандартная структура оконного приложения, его основные функции.
2) Окно. Функция создания нового окна и ее параметры.
3) Что такое оконный класс и каким образом он может быть создан.
4) Каким образом для окна можно указать собственные курсор и иконку.
5) За что отвечают функции UpdateWindow() и ShowWindow().
6) Способы отправки сообщений другому окну.
7) Цикл обработки сообщений. Функция GetMessage(), завершение цикла.
8) Функции TranslateMessage(), DispatchMessage().
9) Назначение, логика работы и параметры функции обработки сообщений.
10) Сообщение WM_COMMAND. Данные, передаваемые в wParam и lParam в данном сообщении.
11) Создание собственного пункта меню и обработка нажатия на него.

Лабораторная работа № 2

Тема работы: Работа с файлами
Цель работы: Научиться работать с файлами

Теоретические сведения по изучению темы

Файл – именованная область данных на носителе информации.

Функция **GetOpenFileName** создает стандартное диалоговое окно Открыть, которое дает возможность пользователю определить диск, каталог и имя файла или ряд файлов, чтобы открыть: BOOL GetOpenFileName(LPOPENFILENAME lpofn), где lpofn – адрес структуры с данными инициализации.

Функция **GetSaveFileName** создает стандартное диалоговое окно Сохранить, которое дает возможность пользователю определить диск, каталог и имя файла или ряд файлов, чтобы сохранить.

Для создания нового или открытия существующего файла используется функция **HANDLE CreateFile(**LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECU-RITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreation-Disposition, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile), где lpFileName – указатель на строку, содержащую имя объекта для создания или открытия (имя файла, путь к файлу, имя канала и пр.); dwDesiredAccess – описание желаемого режима доступа к файлу; dwShareMode – определяет режим разделения объекта: 0 – приложение открывает файл для монопольного доступа; lpSecurityAttributes – указатель на структуру SECURI-TY_ATTRIBUTES, которая определяет возможность наследования дескриптора дочерними процессами. Можно передавать NULL – это значит, что дескриптор не может быть наследован (для наших приложений этого достаточно); dwCreationDisposition – определяет то, какие действия необходимо предпринять в случаях, если файл существует и если файл не существует. Этот параметр должен иметь одно из следующих заначений: CREATE_NEW, CREATE_ALWAYS, OPEN_EXISTING, OPEN_ALWAYS, TRUNCATE_EXISTING; dwFlagsAndAttributes – позволяет задавать файловые атрибуты (только для чтения, скрытый, системный и пр.). Также позволяет сообщать операционной системе желаемое поведение при работе с файлами.

Для копирования файлов используется функция **BOOL CopyFile(**LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, BOOL bFailIfExists), где lpExistingFileName – имя существующего файла, lpNewFileName – имя нового файла, bFailIfExists – действие, если файл существует TRUE – ошибка, FALSE – перезаписывать**).** В случае успеха возвращается ненулевое значение

Для переименования файлов и директорий используется функция **BOOL MoveFile** (LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName), где lpExistingFileName – имя файла, lpNewFileName – новое имя файла**).** В случае успеха возвращается ненулевое значение.

Для удаления существующих файлов используется функция **BOOL DeleteFile**(LPCTSTR lpFileName**).** В случае успеха возвращается ненулевое значение

Каждый открытый файл имеет файловый указатель (filepointer), который указывает позицию следующего файла, который будет записан/прочтен. При открытии файла его файловый указатель перемещается на начало файла. После прочтения/записи очередного файла система перемещает файловый указатель. Файловый указатель можно перемещать, используя функцию SetFilePointer.

Для чтения/записи в файл используются функции ReadFile и WriteFile, при этом необходимо, чтобы файл был открыт на чтение и на запись соответственно.

Функция ReadFileчитает из файла указанное количество символов, начиная с позиции, обозначенной файловым указателем. При синхронном (в противоположность асинхронному) чтении файловый указатель сдвигается на фактически прочитанное количество байт.

BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped), где hFile – дескриптор читаемого файла. Должен быть открыт с доступом GENERIC_READ, lpBuffer – указатель на буфер, принимающий данные из файла, nNumberOfBytesToRead – задаёт количество байт, которые необходимо прочитать из файла, lpNumberOfBytesRead – указатель на переменную, которая принимает количество реально прочитанных байт, lpOverlapped – указатель на структуру OVERLAPPED.

Функция WriteFile записывает в файл данные, начиная с позиции, обозначенной файловым указателем. При синхронной (в проти-

воположность асинхронному) записи файловый указатель сдвигается на фактически записанное количество байт.

BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped), где hFile – дескриптор файла, в который производится запись. Должен быть открыт с доступом GENERIC_WRITE, lpBuffer – указатель на буфер, содержащий данные, которые необходимо записать, nNumberOfBytesToWrite – задаёт количество байт, которые необходимо записать в файл, lpNumberOfBytesWritten – указатель на переменную, которая принимает количество реально записанных байт, lpOverlapped – указатель на структуру OVERLAPPED. При не асинхронном доступе следует передавать NULL.

Скроллинг или прокрутку можно установить как свойство окна, добавив флаги горизонтального и вертикального скроллинга WS_VSCROLL и WS_HSCROLL в стиль создаваемого окна функции CreateWindow(). hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW| **WS_VSCROLL | WS_HSCROLL**, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

Для обращения к диалоговому окну выбора шрифта необходимо описать две переменные типа CHOOSEFONT и LOGFONT, где первая обеспечивает взаимодействие с функцией диалога, а вторая требуется для хранения информации о выбранном шрифте.

Обе структуры имеют большое количество полей и определены в файлах включений commdlg.h и wingdi.h.

Порядок выполнения:

1. Выполнить обработку пункта меню «Окрыть файл», в результате которого информация будет считана с файла, выбранного, с использованием стандартного диалогового окна и отображена на экране приложения.

Объявляем необходимые переменные (рисунок 2.1).

Инициализируем переменные структуры OPENFILENAME (рисунок 2.2).

Определяем работу пунктов меню, связанных с работой файлов (рисунок 2.3).

```
const DWORD MaxLength = 250;
]LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    // Переменные для стандартных диалогов "Open", "Save As"
    static OPENFILENAME file;
    static TCHAR name[MAX_PATH];
    static HANDLE hFile;
    BOOL success;
    int i = 0;
    int y = 0;
    DWORD dw = 0;
    static char text[MaxLength];
```

Рисунок 2.1 – Объявление переменных

```
case WM_CREATE:
    // Инициализация структуры ofn
    file.lStructSize = sizeof(OPENFILENAME);
    file.hInstance = hInst;
    file.lpstrFilter = _T("Text\0*.txt");
    file.lpstrFile = name;
    file.nMaxFile = 256;
    file.lpstrInitialDir = _T(".\\");
    file.lpstrDefExt = _T("txt");
    break;
```

Рисунок 2.2 – Инициализация переменных структуры OPENFILENAME

```
case IDM_OPEN:
    file.lpstrTitle = _T("Открыть файл для чтения");
    file.Flags = OFN_HIDEREADONLY;
    if (!GetOpenFileName(&file)) return 1;
    hFile = CreateFile(name, GENERIC_READ, 0, NULL,OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    hdc = GetDC(hWnd);
    while (ReadFile(hFile, &text, sizeof(text), &dw, NULL)&& dw!=0)
    {
        TextOutA(hdc, 0, y, text, MaxLength);
        y += 16;
    }
    ReleaseDC(hWnd, hdc);
    break;

case IDM_SAVE:
    success = GetSaveFileName(&file);
    if (success)
    MessageBox(hWnd, file.lpstrFile,TEXT("Файл сохраняется с именем:"), MB_OK);
    else
    MessageBox(hWnd, ESC_OF,TEXT("Отказ от выбора или ошибка"), MB_ICONWARNING);
    break;
```

Рисунок 2.3 – Подключение стандартных окон по работе с файлами

Задание:

На основании приложения, разработанного в лабораторной работе 1, реализовать:

– работу с файлами, используя стандартные окна;

– при выбора пункта меню Открыть файл, осуществить выбор файла с данными и вывести его содержимое на экран приложения.

Контрольные вопросы:

1) Файл. Функции работы с файлом.
2) Формат функций GetOpenFileName(), GetSaveFileName().
3) API-функции для чтения и записи файла.

Лабораторная работа № 3

Тема работы: Окна и элементы управления

Цель работы: Научиться организовывать интерфейса с использованием диалоговых окон и элементов управления

Теоретические сведения по изучению темы

Диалоговые окна, или окна диалога (dialog box), реализуют одну из важнейших составляющих программирования Windows-приложений. Обычно диалоговые окна используются для получения от пользователя дополнительной информации, а также для вывода результатов работы приложения.

Диалоговое окно имеет вид всплывающего окна с одним или несколькими элементами управления (controls), которые являются для него дочерними окнами. Используя элементы управления, пользователь вводит текст, выбирает указанные опции (флажки, переключатели, элементы списка) и нажимает кнопки, вызывающие различные действия приложения.

Диалоговые окна бывают модальные (modal) и немодальные (modeless). Они различаются по влиянию на дальнейшее выполнение программы. Когда в программе вызывается модальное диалоговое окно, оно ожидает выполнения некоторого действия со стороны пользователя, прежде чем программа сможет продолжить свое выполнение. Пользователь не может переключиться между диалоговым окном и другими окнами программы.

Немодальное диалоговое окно не приостанавливает выполнение программы. Оно может получать и терять фокус ввода.

Любое стандартное приложение Windows использует различные органы управления, такие, как кнопки, полосы просмотра, редакторы текстов, меню и т.д. Как правило, эти органы управления используют дочерние окна, созданные на базе предопределенных в Windows классов. Такие дочерние окна имеют вид кнопок (buttons), флажков(check boxes), окон редактирования(edit boxes), списков(list boxes), комбинированных списков(combo boxes), строк текста(text strings) и полос прокрутки(scroll bars)

Обычно элементы управления определяются в шаблоне диалогового окна на языке описания шаблона диалога. В случае создания шаблона с помощью редактора диалоговых окон эти определения элементов генерируются автоматически.

Одним из атрибутов описания элемента управления в шаблоне диалога является идентификатор элемента управления.

Каждый элемент управления, описанный в шаблоне диалога, реализуется Windows в виде окна соответствующего класса. Это окно является дочерним окном по отношению к диалоговому окну. Как всякое окно, оно идентифицируется своим дескриптором типа HWND. Вместо термина «дескриптор окна элемента управления» в документации (MSDN) обычно используется более короткий термин «дескриптор элемента управления».

Создать элементы управления на форме диалога можно двумя способами:

– с помощью средств интегрированной среды разработки Microsoft Visual Studio;

– посредством вызова функции CreateWindowEx.

Обычная кнопка (Button). После размещения кнопки на форме диалога ей назначается идентификатор (например, IDC_BUTTON1), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

При воздействии на элемент управления диалога в диалоговую процедуру DlgProc поступает сообщение WM_COMMAND, в котором LOWORD(wParam) содержит идентификатор элемента управления, HIWORD(wParam) содержит код уведомления, а lParam – дескриптор элемента управления.

Флажок (Check Box). После размещения флажка на форме диалога ему назначается идентификатор (например, IDC_CHECK1), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Для того, чтобы перевести Check Box в некоторое состояние, ему необходимо отправить сообщение BM_SETCHECK, передав в WPARAM одно из следующих значений:

– BST_CHECKED – установить отметку;

– BST_UNCHECKED – снять отметку;

– BST_INDETERMINATE – установить неопределенное состояние.

Существует альтернативный способ программной инициализации состояния элемента управления Check Box. Для этого используется функция API CheckDlgButton: BOOL CheckDlgButton(HWND hDlg, int nIDButton, UINT uCheck), где hDlg – дескриптор диалога, содержащего кнопку (флажок); nIDButton, – идентификатор

элемента управления (флажка); uCheck – состояние флажка – одно из вышеперечисленных значений.

Переключатель (Radio Button). Данный элемент управления обычно применяется для представления в окне множества взаимоисключающих опций, из которых можно выбрать только одну. В отличие от флажков, повторный щелчок на переключателе не меняет его состояние.

После размещения переключателя на форме диалога ему назначается идентификатор (например, IDC_RADIO1), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Для того, чтобы перевести Radio Button в некоторое состояние, ему необходимо отправить сообщение BM_SETCHECK, передав в WPARAM одно из следующих значений:

– BST_CHECKED – установить отметку;

– BST_UNCHECKED – снять отметку;

– BST_INDETERMINATE – установить неопределенное состояние.

Существует альтернативный способ выбора переключателя. Для этого используется функция API CheckRadioButton: BOOL CheckRadioButton(HWND hDlg, int nIDFirstButton, int nIDLastButton, int nIDCheckButton), где hDlg – дескриптор диалога, содержащего кнопку (переключатель), nIDFirstButton – идентификатор первого переключателя в группе, nIDLastButton – идентификатор последнего переключателя в группе, nIDCheckButton – идентификатор выбираемого переключателя.

Данная функция помечает указанный переключатель в группе, удаляя отметку со всех других переключателей этой же группы. Другими словами, функция CheckRadioButton посылает сообщение BM_SETCHECK каждому переключателю указанной группы. При этом выбираемому переключателю в WPARAM передается BST_CHECKED, а остальным – BST_UNCHECKED.

Элемент управления «текстовое поле ввода» (Edit Control). Одним из простейших применений окон редактирования – элементов управления является простое однострочное окно ввода данных, сообщения, посылаемые элементу управления Edit Control, представлены в таблице 3.1.

Таблица 3.1 – Сообщения элементу управления **Edit Control**:

Код сообщения	**wParam**	**lParam**	**Описание**
EM_SETSEL	iStart	iEnd	Выделить текст, начиная с позиции iStart и заканчивая позицией iEnd
EM_GETSEL	&iStart	&iEnd	Получить начальное и конечное положения текущего выделения
WM_CLEAR	0	0	Удалить выделенный текст
WM_CUT	0	0	Удалить выделенный текст и поместить его в буфер обмена
WM_COPY	0	0	Скопировать выделенный текст в буфер обмена Windows
WM_PASTE	0	0	Вставить текст из буфера обмена в месте, соответствующем позиции курсора
EM_CANUNDO	0	0	Определить возможность отмены последнего действия
WM_UNDO	0	0	Отменить последнее действие
WM_GETTEXT	nMax	szBuffer	Скопировать текст (не более nMax символов) из элемента управления в буфер szBuffer
EM_GETLINECOUNT	0	0	Получить число строк для многострочного окна редактирования
EM_LINELENGTH	iLine	0	Получить длину строки iLine
EM_GETLINE	iLine	szBuffer	Скопировать строку iLine в буфер szBuffer
EM_LINEFROMCHAR	-1	0	Получить номер строки, в которой расположен курсор
EM_LINEINDEX	iLine	0	Получить номер первого символа строки iLine

Как известно, при воздействии на элемент управления диалога (например, при вводе текста в **Edit Control**), в диалоговую процедуру **DlgProc** поступает сообщение **WM_COMMAND,** в котором **LOWORD(wParam)** содержит идентификатор элемента управления, **HIWORD(wParam)** содержит код уведомления (например, **EN_CHANGE**), а **lParam** – дескриптор элемента управления.

Таблица 3.2 – Уведомления от текстового поля ввода

Код уведомления	**Интерпретация**
EN_SETFOCUS	Окно получило фокус ввода
EN_KILLFOCUS	Окно потеряло фокус ввода
EN_UPDATE	Содержимое окна будет меняться
EN_CHANGE	Содержимое окна изменилось
EN ERRSPACE	Произошло переполнение буфера редактирования

Элемент управления **List Box** представляет собой список элементов, из которых пользователь может выбрать один или более. Элементы списка могут быть представлены строками, растровыми образами или комбинацией текста и изображения.

После размещения списка на форме диалога ему назначается идентификатор (например, **IDC_LIST1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Следует отметить, что различают списки с единичным выбором, в которых пользователь может выбрать только один пункт списка, и списки с множественным выбором, допускающие выбор более одного пункта списка. Система обычно отображает выбранный элемент в списке, инвертируя цвет текста и цвет фона для символов.

Для выполнения различных операций со списком приложение может отправлять сообщения элементу управления **List Box** (таблица 3.3).

Таблица 3.3 – Сообщения элементу управления **List Box**

Код сообщения	**wParam**	**lParam**	**Описание**
LB_ADDSTRING	0	szString	Добавить в список строку **szString**. Если свойство **Sort** у списка выключено, то строка будет добавлена в конец списка. Если это свойство включено, то после добавления строки производится сортировка списка
LB_FINDSTRING	iStart	szString	Найти строку, начинающуюся префиксом **szString**. Поиск начинается с элемента с индексом **iStart + 1**. Если **wParam** равен **-1**, то поиск начинается с начала списка. **SendMessage** возвращает индекс найденной строки или **LB_ERR**, если поиск завершился неуспешно
LB_GETCURSEL	0	0	Получить индекс текущего выбранного элемента или **LB_ERR**, если такого элемента нет (только для списков с единичным выбором)
LB_GETSELCOUNT	0	0	Получить количество выбранных элементов (для списка с множественным выбором)
LB_GETSELITEMS	nMax	pBuf	Заполнить буфер **pBuf** массивом индексов выделенных элементов для списка с множественным выбором. Параметр **nMax** задает максимальное количество таких элементов

Продолжение таблицы 3.3

Код сообщения	**wParam**	**lParam**	**Описание**
LB_INSERTSTRING	iIndex	szString	Вставить строку **szString** после строки с индексом **iIndex**. Применяется для списка с выключенным свойством **Sort**
LB_SELECTSTRING	iStart	szString	Аналогично сообщению **LB_FINDSTRING**, но дополнительно выделяется найденная строка
LB_SETSEL	wParam	iIndex	Выбрать элемент с индексом **iIndex** (в списке с множественным выбором). Если **wParam** равен **TRUE**, элемент выбирается и выделяется,если **FALSE** — выбор отменяется. Если **lParam** равен -1, то операция применяется ко всем элементам списка
LB_GETTEXT	iIndex	szString	Скопировать строку с индексом **iIndex** в буфер **szString**
LB_DELETESTRING	iIndex	0	Удалить строку с индексом **iIndex**.
LB_GETCOUNT	0	0	Получить количество элементов в списке
LB_GETTEXTLEN	iIndex	0	Получить длину строки с индексом iIndex
LB_RESETCONTENT	0	0	Удалить все элементы из списка
LB_SETCURSEL	iIndex	0	Выбрать элемент с индексом **iIndex** (в списке с единичным выбором)
LB_SETITEMDATA	iIndex	nValue	Установить целочисленное значение **nValue**, ассоциированное с указанным элементом списка
LB_GETITEMDATA	iIndex	0	Получить целочисленное значение, ассоциированное с элементом списка, имеющим индекс **iIndex**

При воздействии на **List Box** (например, при выборе элемента списка), в диалоговую процедуру **DlgProc** поступает сообщение **WM_COMMAND,** в котором **LOWORD (wParam)** содержит идентификатор списка, **HIWORD (wParam)** содержит код уведомления (таблица 3.4**)**, а **lParam** – дескриптор списка.

Таблица 3.4 – Уведомления от списка приведены в таблице

Код уведомления	**Интерпретация**
LBN_SETFOCUS	Окно получило фокус ввода
LBN_KILLFOCUS	Окно потеряло фокус ввода
LBN_SELCHANGE	Текущий выбор был изменен (свойство **Notify** должно иметь значение **True**)
LBN_DBLCLK	На данном пункте списка был двойной щелчок мыши (свойство **Notify** должно иметь значение **True**)
LBN_ERRSPACE	Превышен размер памяти, отведенный для списка

Комбинированный список (**Combo Box**) является комбинацией списка и текстового поля ввода. Обмен сообщениями с комбинированным списком во многом похож на обмен сообщениями с элементами управления **Edit Box** и **List Box**. При этом идентификаторы сообщений, отправляемых элементу управления **ComboBox**, начинаются с префикса **CB_**, а идентификаторы уведомительных сообщений, посылаемых от **ComboBox** своему родительскому окну, имеют префикс **CBN_**.

Элемент управления **Progress Control** обычно используется в приложениях для отображения процесса выполнения некоторой длительной операции.

После размещения индикатора на форме диалога ему назначается идентификатор (например, **IDC_PROGRESS1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Для управления индикатором процесса используются сообщения, приведенные в таблице 3.5.

Таблица 3.5 - Сообщения элементу управления **Progress Control**

Код сообщения	**wParam**	**lParam**	**Описание**
PBM_SETRANGE	0	MAKELPARAM (wMin, wMax)	Установка интервала для индикатора
PBM_SETPOS	nNewPos	0	Установка текущей позиции индикатора
PBM_DELTAPOS	nInc	0	Изменение текущей позиции прибавлением смещения nInc
PBM_SETSTEP	nStepInc	0	Установка шага приращения для индикатора
PBM STEPIT	0	0	Изменение текущей позиции прибавлением шага nStepInc
PBM_SETBARCOLOR	0	(COLORREF) clrBar	Установка цвета заполняемых прямоугольников
PBM_SETBKCOLOR	0	(COLORREF) clrBk	Установка цвета фона индикатора

Элемент управления **Slider Control** (ползунок или регулятор), который ранее назывался **Trackbar**, представляет собой окно с линейкой и перемещаемым по ней ползунком. Часто используемыми сообщениями для управления регулятором, представлены в таблице 3.6.

Таблица 3.6 - Сообщения элементу управления **Slider Control**

Код сообщения	**wParam**	**lParam**	**Описание**
TBM_GETPOS	0	0	Возвращает текущую позицию ползунка
TBM_SETPOS	fRedraw	newPos	Устанавливает новую позицию ползунка. Если **fRedraw** равно TRUE, регулятор перерисовывается после этого.
TBM_SETRANGE	fRedraw	MAKELPARAM (wMin, wMax)	Устанавливает диапазон регулятора
TBM_SETTICFREQ	wFreq	0	Устанавливает шаг меток
TBM_SETLINESIZE	0	nLineSize	Устанавливает размер «строки»
TBM_SETPAGESIZE	0	nPageSize	Устанавливает размер «страницы»

Регулятор уведомляет свое родительское окно (диалог) о действиях пользователя, посылая сообщение **WM_HSCROLL** или **WM_VSCROLL** — в зависимости от ориентации элемента управления (**Horizontal** или **Vertical**). В любом случае в младшем слове параметра **wParam** содержится код уведомления, а параметр **lParam** содержит дескриптор ползунка.

Возможные коды уведомления приведены в таблице 3.7.

Таблица 3.7 – Уведомления от **Slider Control**

Код уведомления	**Интерпретация**
TB_LINEUP	Нажата клавиша «стрелка влево» (**VK_LEFT**) или клавиша «стрелка вверх» (**VK_UP**)
TB_LINEDOWN	Нажата клавиша «стрелка вправо» (**VK_RIGHT**) или клавиша «стрелка вниз» (**VK_DOWN**)
TB_PAGEUP	Нажата клавиша «Page Up» (**VK_PRIOR**) или щелчок мышью на линейке левее или выше ползунка
TB_ PAGEDOWN	Нажата клавиша «Page Down» (**VK_NEXT**) или щелчок мышью на линейке правее или ниже ползунка
TB_THUMBPOSITION	Отпущена кнопка мыши после перемещения ползунка (это сообщение следует всегда после сообщения **TB_THUMBTRACK**)
TB_THUMBTRACK	Ползунок перемещается с помощью мыши
TB_TOP	Нажата клавиша «Home» (**VK_HOME**) — ползунок устанавливается в крайнее левое (верхнее) положение, соответствующее значению **wMin**
TB_BOTTOM	Нажата клавиша «End» (**VK_END**) — ползунок устанавливается в крайнее правое (нижнее) положение, соответствующее значению **wMax**

Порядок выполнения:

Для работы с данными, находящимися в файле, необходимо создать структуру (рисунок 3.1), характеризующую хранимую информацию.

```
struct mobil{
    TCHAR model[25];
    int price;
    float size;
}ms, mobils[25];
```

Рисунок 3.1 – Пример объявления структуры

Организовать пункт меню, реализующий добавление данных в файл, который будет вызывать диалоговое окно, имеющее необходимые элементы управления, для ввода данных (рисунки 3.2, 3.3).

После того, как будет организован ввод данных, переработать пункт меню Открыть файл, для сохрания данных в массив структур.

```
case IDM_OPEN:
    file.lpstrTitle = _T("Открыть файл для чтения");
    file.Flags = OFN_HIDEREADONLY;
    if (!GetOpenFileName(&file)) return 1;
    hFile = CreateFile(name, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    break;

case IDM_SAVE:
    success = GetSaveFileName(&file);
    if (success)
    MessageBox(hWnd, file.lpstrFile,TEXT("Файл сохраняется с именем:"), MB_OK);
    else
    MessageBox(hWnd, ESC_OF,TEXT("Отказ от выбора или ошибка"), MB_ICONWARNING);
    break;

case IDM_ADD:
    DialogBox(hInst, MAKEINTRESOURCE(IDD_ADD), NULL, DlgProcAdd);
    break;
```

Рисунок 3.2 – Обработка пунктов меню

Также необходимо определить функцию, которая будет добавлять структуру в файл и формировать массив (рисунок 3.4).

Согласно пунктов меню 3-5, организовать диалоговое окно, выполняющее обработку необходимых данных.

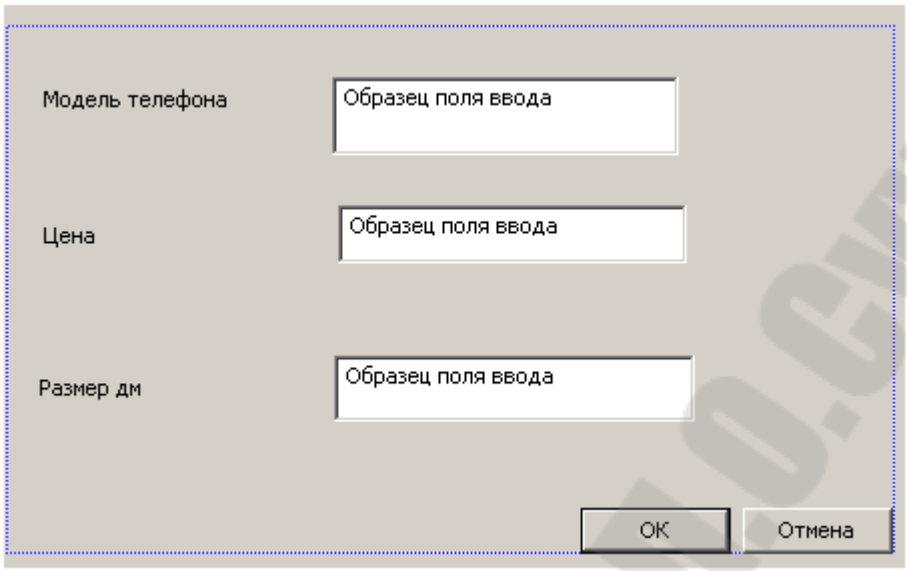

Рисунок 3.3 – Диалоговое окно добавления элементов

```
HWND hModel, hPrice, hSize;
TCHAR str[5];
BOOL CALLBACK DlgProcAdd(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static j = 0;
    switch (message)
    {
    case WM_INITDIALOG:
        hModel = GetDlgItem(hWnd, IDC_EDIT1);
        hPrice = GetDlgItem(hWnd, IDC_EDIT2);
        hSize = GetDlgItem(hWnd, IDC_EDIT3);
        return TRUE;
    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK)
        {
            GetDlgItemText(hWnd, IDC_EDIT1, (LPWSTR)ms.model, sizeof(ms.model));
            ms.price = GetDlgItemInt(hWnd, IDC_EDIT2, 0, 1);
            GetDlgItemText(hWnd, IDC_EDIT3, (LPWSTR)str, sizeof(str));
            ms.size = _wtof(str);
            FlushFileBuffers(hFile);
            SetFilePointer(hFile, 0, 0, FILE_END);
            WriteFile(hFile, &ms, sizeof(ms), &dw, NULL);
            mobils[j++] = ms;
            EndDialog(hWnd, 0);
            break;
        }
        return TRUE;
    case WM_CLOSE:
        EndDialog(hWnd, 0);
        return TRUE;
    }
    return FALSE;
}
```

Рисунок 3.4 – Диалоговая функция

Для этого добавляем соответствующее диалоговое окно (рисунок 3.4), которое будет вызывать из соотвествующего пункта меня и добавлять в список данные из массива структур.

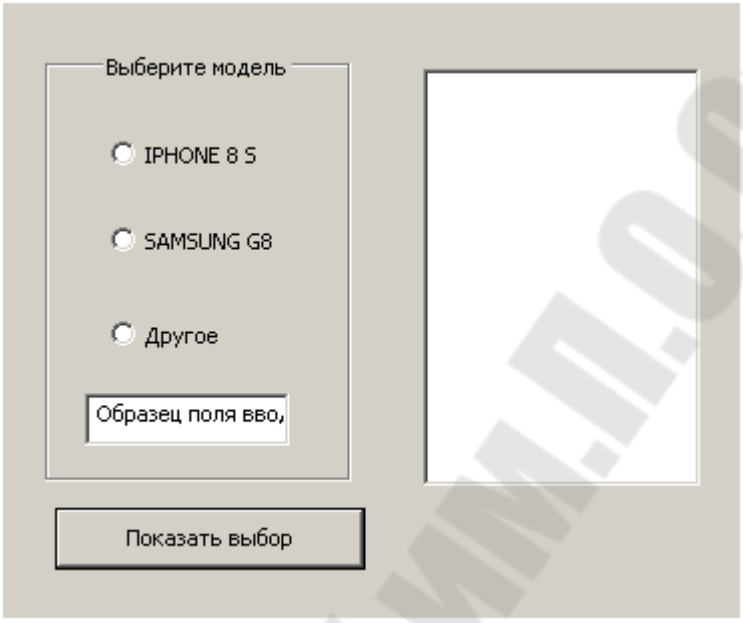

Рисунок 3.5 – Диалоговое окно

Обработку диалогового окна, выполняем в соответствующей диалоговой функцией (рисунок 3.6).

Задание:

В Приложении, разработанном в лабораторных работах 1-2, реализовать:

– добавление данных в файл;

– пункты 3-5 задания, согласно варианта, представленого в Приложении А.

Контрольные вопросы:

1. Техника создания окна, аргументы функции CreateWindow().
2. Назначение функции UpdateWindow()?
3. Создание диалогового окна, функции DialogBox() и CreateDialog().
4. Стандартные и общие элементы управления. В чем заключается особенность
5. общих элементов управления?

6. Как осуществляется обмен данными с элементами управления?

```
BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static HWND road, hList1;
    static TCHAR str[20], roadName[20];
    int size = sizeof(mobils) / sizeof(ms);
    switch (message)
    {
    case  WM_INITDIALOG:
        //получение дескрипторов
        hList1 = GetDlgItem(hWnd, IDC_LIST1);
        //задаем какая радиокнопка будет выделена при запуске
        SendDlgItemMessage(hWnd, IDC_RADIO2, BM_SETCHECK, WPARAM(BST_CHECKED), 0);
        return TRUE;
        //обработка выбора кнопки
    case WM_COMMAND:
        switch (LOWORD(wParam))
        {
        case IDC_RADIO1:wsprintf(str, TEXT("%s"), TEXT("IPHONE 8 S")); break;
        case IDC_RADIO2:wsprintf(str, TEXT("%s"), TEXT("SAMSUNG G8")); break;
        case IDC_RADIO3:
            road = GetDlgItem(hWnd, IDC_EDIT1);
            GetWindowText(road, roadName, 20);
            wsprintf(str, TEXT("%s"),roadName);
            break;
        case IDC_SHOW:
            //заполнение элементов списка
            for (int i = 0; i < size;i++)
                if (wcscmp(str, mobils[i].model)==0)
                    SendMessage(hList1, LB_ADDSTRING, 0, LPARAM(mobils[i].model));
            break;
        }
        return TRUE;
    case WM_CLOSE:
        EndDialog(hWnd, 0);
        return TRUE;
    }
    return FALSE;
}
```

Рисунок 3.6 – Диалоговая функция

Лабораторная работа № 4

Тема работы: Растровая графика

Цель работы: Изучить возможности графической подсистемы. Научиться использовать расширенные возможности графики GDI.

Теоретические сведения по изучению темы

Растровая графика – это графическое изображение на компьютере или в другом цифровом виде, состоящее из массива сетки пикселей, или точек различных цветов, которые имеют одинаковый размер и форму.

В простейшем случае изображение может формироваться по-пиксельно с помощью функции SetPixel() (есть также функция GetPixel() – доступ к отдельным пикселям изображения), но обычно этот путь слишком медленный и неудобный.

GDI предоставляет функции манипулирования целыми фрагментами изображений:

– BitBlt() – копирование прямоугольной области из одного контекста в другой без каких-либо преобразований;

– StretchBlt() – копирование области с масштабированием по обоим осям (обычно с уменьшением, так как при растяжении изображения заметно портится);

– MaskBlt() – перенос с маскированием части изображения;

– PlgBlt() – перенос прямоугольного фрагмента в непрямоугольную область с соответствующим геометрическим искажением.

В ряде случаев формирование всего изображения заново может быть слишком длительным – например, различного рода анимация или изображения из очень большого числа элементов. В этом случае эффективно использование «теневых» контекстов или «контекстов в памяти». Это такие DC, которые не имеют связи с реальным устройством. Вместо него рабочую поверхность для них образует битовая карта (bitmap) или метафайл. (Под метафайлом здесь понимается объект Metafile, фактически хранящий последовательность применявшихся к нему графических примитивов и способный их воспроизвести.)

Контекст создается совместимым с текущим экранным режимом, и с ним ассоциируется «битовая карта»:

```
hShadowDC = CreateCompatibleDC( NULL);
hBitmap = CreateCompatibleBitmap( hPrimDC, width, height);
SelectObject( hShadowDC, hBitmap);
```

Здесь hPrimeDC — «реальный» контекст, в котором необходимо формировать изображение.

После создания контекста в нем формируется необходимое изображение. Далее в процессе работы программы это изображение или его фрагменты переносятся на «основной» (видимый) контекст.

Порядок выполнения:

Создать в графическом редакторе несколько изображений, одинакового размера.

Определить траекторию движения анимированного объекта и реализовать движение, с использованием соответствующих функций (рисунок 4.1).

```
case WM_CREATE:
    hdc = GetDC(hWnd);
    memDC = CreateCompatibleDC(hdc);
    SelectObject(memDC, hCircle);
    ReleaseDC(hWnd, hdc);
    break;

case WM_TIMER:
    hdc = GetDC(hWnd);
    BitBlt(hdc, x, y, 32, 32, NULL, 0, 0, PATCOPY);
    xt += vx * 10;//координаты шарика
    yt += vy * 10;
    x = int(xt + 0.5);
    y = int(yt + 0.5);
    BitBlt(hdc, x, y, 32, 32, memDC, 0, 0, SRCCOPY);
    ReleaseDC(hWnd, hdc);
    break;
```

Рисунок 4.1 – Обработка сообщения WM_CREATE и WM_TIMER

Задание:

Релизовать графическую анимацию, предусмотрев использование различных изображений, выполняющих имитацию движения, в соотвествии с вариантом (Приложение А).

Контрольные вопросы:

1) Использование таймера.
2) Создание графических контекстов.
3) Формат BMP файла.
4) Отображение BMP картинки.
5) Анимация движения.

Лабораторная работа № 5

Тема работы: Библиотеки динамической компоновки DLL

Цель работы: Научиться использовать библиотеки динамической компоновки при разработке приложений

Теоретические сведения по изучению темы

Библиотеки динамической компоновки (dynamic link libraries – DLL) являются исполняемыми файлами особого формата, которые содержат функции, данные или ресурсы, доступные для других приложений.

Особый формат модулей DLL предполагает наличие в них разделов импорта и экспорта. Раздел экспорта указывает те идентификаторы объектов (функций, классов, переменных), доступ к которым разрешен для клиентов.

Подавляющее большинство DLL (за исключением DLL, содержащих только ресурсы) импортирует функции из системных DLL - kernel32.dll, user32.dll, gdi32.dll и других библиотек.

Применение DLL может дать ряд преимуществ:

– расширение функциональности приложения. DLL можно загружать в адресное пространство процесса на этапе выполнения, что позволит программе, определив, какие действия от нее требуются, подгружать нужный код;

– более простое управление проектом. Использование DLL упрощает отладку, тестирование и сопровождение проекта;

– экономия памяти. Если одну и ту же DLL используют несколько приложений, то в оперативной памяти хранится только один ее экземпляр, доступный этим приложениям;

– разделение ресурсов. DLL могут содержать такие ресурсы, как строки, растровые изображения, шаблоны диалоговых окон. Этими ресурсами может воспользоваться любое приложение;

– возможность использования разных языков программирования.

Исполняемый код в DLL не предполагает автономного использования. Содержимое каждого DLL-файла загружается приложением и проецируется на адресное пространство вызывающего процесса. Это достигается либо за счет неявного связывания при загрузке, либо за счет явного связывания в период выполнения.

Важно понимать, что процесс, загрузивший DLL, получает собственную копию глобальных данных, используемых этой библиотекой. Это защищает DLL от ошибок приложений, а процессы, использующие DLL, от взаимного влияния друг на друга.

Вызов функций из DLL

Существует три способа загрузки DLL:

– неявная;

– явная;

– отложенная.

Рассмотрим неявную загрузку DLL. Для построения приложения, рассчитанного на неявную загрузку DLL, необходимо иметь:

– библиотечный включаемый файл с описаниями используемых объектов из DLL (прототипы функций, объявления классов и типов). Этот файл используется компилятором;

– LIB-файл со списком импортируемых идентификаторов. Этот файл нужно добавить в настройки проекта (в список библиотек, используемых компоновщиком).

Компиляция проекта осуществляется обычным образом. Используя объектные модули и LIB-файл, а также учитывая ссылки на импортируемые идентификаторы, компоновщик (линкер, редактор связей) формирует загрузочный EXE-модуль. В этом модуле компоновщик помещает также раздел импорта, где перечисляются имена всех необходимых DLL-модулей. Для каждой DLL в разделе импорта указывается, на какие имена функций и переменных встречаются ссылки в коде исполняемого файла. Эти сведения будет использовать загрузчик операционной системы.

Что же происходит на этапе выполнения клиентского приложения? После запуска EXE-модуля загрузчик операционной системы выполняет следующие операции:

– создает виртуальное адресное пространство для нового процесса и проецирует на него исполняемый модуль;

– анализирует раздел импорта, определяя все необходимые DLL-модули и тоже проецируя их на адресное пространство процесса.

DLL может импортировать функции и переменные из другой DLL. А значит, у нее может быть собственный раздел импорта, для которого необходимо повторить те же действия. В результате на инициализацию процесса может уйти довольно длительное время.

После отображения EXE-модуля и всех DLL-модулей на адресное пространство процесса его первичный поток готов к выполнению, и приложение начинает работу.

Недостатками неявной загрузки являются обязательная загрузка библиотеки, даже если обращение к ее функциям не будет происходить, и, соответственно, обязательное требование к наличию библиотеки при компоновке.

Явная загрузка DLL устраняет отмеченные выше недостатки за счет некоторого усложнения кода. Программисту приходится самому заботиться о загрузке DLL и подключении экспортируемых функций. Зато явная загрузка позволяет подгружать DLL по мере необходимости и дает возможность программе обрабатывать ситуации, возникающие при отсутствии DLL.

В случае явной загрузки процесс работы с DLL происходит в три этапа:

– загрузка DLL с помощью функции **LoadLibrary** (или ее расширенного аналога **LoadLibraryEx).** В случае успешной загрузки функция возвращает дескриптор hLib типа HMODULE , что позволяет в дальнейшем обращаться к этой DLL;

– вызовы функции **GetProcAddress** для получения указателей на требуемые функции или другие объекты. В качестве первого параметра функция **GetProcAddress** получает дескриптор hLib, в качестве второго параметра -адрес строки с именем импортируемой функции. Далее полученный указатель используется клиентом. Например, если это указатель на функцию, то осуществляется вызов нужной функции;

– когда загруженная динамическая библиотека больше не нужна, рекомендуется ее освободить, вызвав функцию **FreeLibrary.** Освобождение библиотеки не означает, что операционная система немедленно удалит ее из памяти. Задержка выгрузки предусмотрена на тот случай, когда эта же DLL через некоторое время вновь понадобится какому-то процессу. Но если возникнут проблемы с оперативной памятью, Windows в первую очередь удаляет из памяти освобожденные библиотеки.

Рассмотрим отложенную загрузку DLL. DLL отложенной загрузки (delay-load DLL) – это неявно связываемая DLL, которая не загружается до тех пор, пока код не обратится к какому-нибудь экспортируемому из нее идентификатору. Такие DLL могут быть полезны в следующих ситуациях:

– если приложение использует несколько DLL, его инициализация может занимать длительное время, требуемое загрузчику для проецирования всех DLL на адресное пространство процесса. DLL отложенной загрузки позволяют решить эту проблему, распределяя загрузку DLL в ходе выполнения приложения;

– если приложение предназначено для работы в различных версиях ОС, то часть функций может появиться лишь в поздних версиях ОС и не использоваться в текущей версии. Но если программа не вызывает конкретной функции, то DLL ей не нужна, и она может спокойно продолжать работу. При обращении же к несуществующей функции можно предусмотреть выдачу пользователю соответствующего предупреждения.

Для реализации метода отложенной загрузки требуется дополнительно в список библиотек компоновщика добавить не только нужную библиотеку импорта MyLib.lib, но еще и системную библиотеку импорта delayimp.lib. Кроме этого, требуется добавить в опциях компоновщика флаг / delayload:MyLib.dll.

Перечисленные настройки заставляют компоновщик выполнить следующие операции:

– внедрить в EXE-модуль специальную функцию delayLoadHelper;

– удалить MyLib.dll из раздела импорта исполняемого модуля, чтобы загрузчик операционной системы не пытался выполнить неявную загрузку этой библиотеки на этапе загрузки приложения;

– добавить в EXE-файл новый раздел отложенного импорта со списком функций, импортируемых из MyLib.dll;

– преобразовать вызовы функций из DLL к вызовам delayLoadhelper.

На этапе выполнения приложения вызов функции из DLL реализуется обращением к delayLoadHelper. Эта функция, используя информацию из раздела отложенного импорта, вызывает сначала LoadLibrary, а затем GetprocAddress. Получив адрес DLL-функции, delayLoadHelper делает так, чтобы в дальнейшем эта DLL-функция вызывалась напрямую. Отметим, что каждая функция в DLL настраивается индивидуально при первом ее вызове.

Функция входа/выхода DLL. Для решения указанных проблем вы можете включить в состав DLL специальную функцию точки входа DllMain. Эта функция вызывается операционной системой в следующих случаях:

– когда DLL проецируется на адресное пространство процесса (подключение DLL);когда процессом, загрузившим DLL, вызывается новый поток;

– когда завершается поток, принадлежащий процессу, который связан с DLL;

– когда процесс освобождает DLL (отключение DLL**).**

В момент вызова функция получает информацию от операционной системы через свои параметры.

Первый параметр, hinstDLL, принимает значение дескриптора модуля DLL, являющееся, по сути, виртуальным адресом загрузки DLL. Если в библиотеке имеются вызовы функций, которым нужен данный дескриптор, необходимо сохранить значение hinstDLL в глобальной переменной.

Второй параметр, fdwReason, может принимать одно из следующих значений:

– DLLPROCESSATTACH – уведомление о том, что DLL загружена в адресное пространство процесса либо в результате его старта, либо в результате вызова функции LoadLibrary;

– DLLTHREADATTACH – уведомление о том, что текущий процесс создал новый поток. Это уведомление посылается всем DLL, подключенным к процессу. Вызов DllMain происходит в контексте нового потока;

– DLLTHREADDETACH – уведомление о том, что поток корректно завершается. Вызов DllMain происходит в контексте завершающегося потока;

– DLLPROCESSDETACH – уведомление о том, что DLL отключается от адресного пространства процесса в результате одного из трех событий: а) неудачное завершение загрузки DLL; б) вызов функции FreeLibrary; в) завершение процесса.

Если при вызове функции используется первый параметр со значением DLLPROCESSATTACH, то по значению третьего параметра можно выяснить, каким способом загружается DLL. При явной загрузки параметр IvlReserved равен нулю, а при неявной загрузке принимает ненулевое значение.

Следует отметить следующие моменты работы с функцией входа-выхода DLL:

1. Поток, вызвавший DllMain со значением DLLPROCESSATTACH, не вызывает повторно DllMain со значением DLLTHREADATTACH.

2. Когда DLL загружается вызовом функции LoadLibrary, существующие потоки не вызывают DllMain для вновь загруженной библиотеки.

3. Функция DllMain не вызывается, если поток или процесс завершаются по причине вызова функции TerminateThread или TerminateProcess.

Поскольку функция DllMain должна обрабатывать все возможные причины своего вызова, то в ее коде обьгано анализируется dwReason и выполняются необходимые действия по инициализации или освобождению ресурсов.

Порядок выполнения:

При создании нового Win32-проекта выбрать тип DLL. Будет создан файл реализации dlmain.cpp.

При разработке библиотеки, содержащей ресурсы, необходимо их добавить в библиотеку динамической компоновки (рисунок 5.1). DLL-библиотека должна быть создана с использованием def-файла: EXPORTS hIcon.

```
#include "stdafx.h"
#include "resource.h"
__declspec(dllexport) HICON hIcon;
BOOL APIENTRY DllMain( HMODULE hModule,//дескриптор библиотеки, прис
                       DWORD  ul_reason_for_call,//код уведомления
                       LPVOID lpReserved //зарезервировано
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        hIcon = LoadIcon(hModule, MAKEINTRESOURCE(IDI_ICON1));
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        DestroyIcon(hIcon);
        break;
    }
    return TRUE;
}
```

Рисунок 5.1 – Реализация функции DllMain

Библиотеку нужно скомпилировать и убедиться, что папка проекта содержит lib- и dll-файлы. Для явной загрузки библиотеки в созданный проект добавим dll-файл созданной библиотеки. В

сообщении WM_CREATE необходимо получить дескриптор библиотеки: hDll = LoadLibrary(_T("DllIcon")), передавая ей в качестве параметра имя DLL-файла. Далее, функцией GetProcAddress() находим дескриптор иконки, уже загруженной в библиотеке, передавая ей дескриптор иконки как текстовую строку: hIcon = *((HICON*)GetProcAddress(hDll, "hIcon")). Переменная hIcon описана как HICON.

После этого можем изменить малую иконку класса окна SetClassLong(hWnd, GCL_HICONSM, (LONG)hIcon).

Для неявной загрузки библиотеки динамической компоновки. В DLL определяем функции (рисунок 5.2).

```
// dllmain.cpp: определяет точку входа для приложения DLL.
#include "stdafx.h"
__declspec(dllexport) BOOL WINAPI Sum(int *, int);
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

BOOL WINAPI Sum(int *a, int n)
{
    int sum=0;
    for (int i = 0; i <n; i++)
    {
        sum += a[i];
    }
    return (sum);
}
```

Рисунок 5.2 – Реализация функции DllMain

Библиотеку нужно скомпилировать и убедиться, что папка проекта содержит lib- и dll-файлы. К проекту, подключаем lib-файл динамически подключаемой библиотеки (рисунок 5.3).

Объявляем прототип функции : WINGDIAPI BOOL WINAPI Sum(int*, int).

И вызываем данную функцию, согласно алгоритма программы int k=Sum(mas, 10).

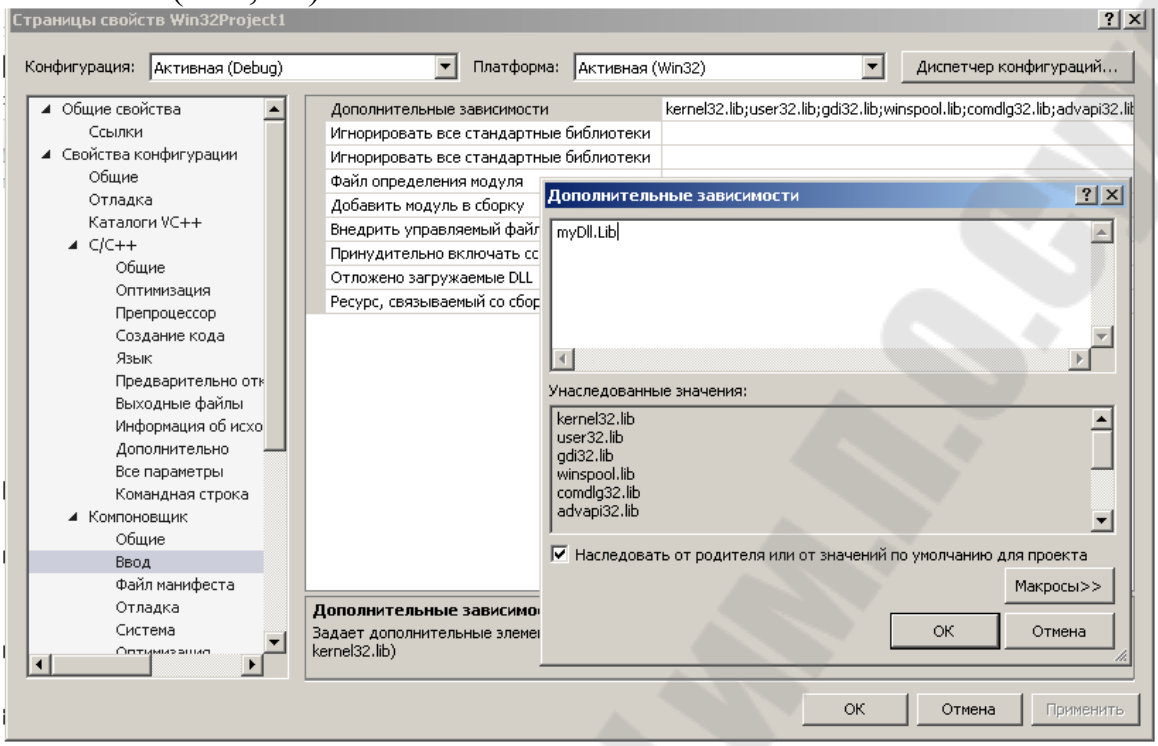

Рисунок 5.3 – Подключение файла lib к проекту

Задание:

В приложение, разработанном в лабораторных работах 1-4 использовать библиотеки динамической компоновки:

– создайте библиотеку содержащую набор ресурсов: иконку, курсор, растровое изображение, используемых в приложении. Библиотеку подключить явно.

– создайте библиотеку, позволяющую выполнять осуществлять пункт 4 задания (приложение А). Библиотеки подключить неявно.

Контрольные вопросы:

1.Что такое DLL?

2.Зачем нужен раздел экспорта?

3.Какие преимущества использования DLL?

4.Как защищаются данные DLL от ошибок приложений?

5.Какие способы загрузки DLL Вы знаете?

6.Как осуществляется неявная загрузка DLL?

7.Как выполняется загрузка exe-файла, использующего DLL при неявном связывании?

8.Каковы недостатки неявной загрузки?
9.Чем отличается явная загрузка от неявной?
10.Каково назначение функции **LoadLibrary?**
11.Зачем используется функция **GetProcAdress?**
12.Перечислите этапы работы с DLL при явной загрузке.
13.Что такое отложенная загрузка DLL?
14.Зачем используется отложенная загрузка?
15.Как организуется отложенная загрузка?
16.Зачем нужна функция входа-выхода DLL?
17.В каких случаях вызывается DLLMain?
18.Какие параметры у функции входа-выхода?
19.Какие причины вызова DLLMain Вы знаете?
20.Как узнать, каким способом загружается DLL?

Лабораторная работа № 6

Тема работы: Процессы и потоки

Цель работы: Научиться использовать при разработке приложений управление процессами и потоками

Теоретические сведения по изучению темы

Поток (Thread) – системный объект, соответствующий последовательности («потоку») команд, выполняемой независимо и асинхронно по отношению к другим подобным последовательностям. Поток – базовый объект, которому планировщик задач ОС распределяет время центрального процессора. Каждый процесс имеет как минимум один главный (первичный – primary) поток. Главный поток может порождать подчиненные (вторичные) потоки, которые будут выполняться одновременно с ним, равно как и с потоками прочих процессов.

Многопоточность, свойственная Win 32 и ряду других современных ОС удачно дополняет их многозадачность и существенно повышает гибкость системы. Процесс в Win 32 играет роль владельца ресурсов и «контейнера потоков» планирования ресурсов. Ресурсы процесса доступны всем его потокам, и все потоки одного процесса совместно используют его виртуальное адресное пространство (но каждый поток имеет собственный стек). В то же время процессорное время распределяется именно между потоками, а не процессами.

Поток может находиться в нескольких состояниях, начиная от его инициализации до завершения, но наибольший интерес представляют следующие:

– выполнение (running) или активность (active) – поток обладает всеми необходимыми ресурсами, включая процессорное время, и выполняется;

– готовность (ready) – поток обладает ресурсами для выполнения, но время ему не выделено, и он ожидает активизации в очереди планировщика;

– ожидание (wait), или заблокированное (blocked), или «спящее» (sleep) – поток не обладает требуемыми ресурсами и не может выполняться, он ожидает выполнения необходимых условий или наступления каких-либо событий.

Например, активный поток переходит в состояние ожидания после обращения к блокирующему системному вызову до его заверше-

ния, затем перемещается в очередь планировщика, откуда выбирается для следующей активизации. В простейшем случае, переход в состояние ожидания произойдет при выполнении вызова Sleep().

Планировщик задач ОС выделяет, в рамках принятой в Win 32 модели многозадачности, процессорное время отдельным потокам квантами. Переключение потоков и изменение их состояний происходит по мере израсходования выделенных квантов и в соответствии с приоритетами потоков.

Потоки идентифицируются их описателями (handle), уникальными в рамках одного процесса, и идентификаторами (уникальны в системе). Однако большинство функций работают именно с описателями. Поток всегда сам является обладателем специального локального описателя, поэтому закрытие «внешнего» его описателя не приводит к удалению объекта. Одновременно, поток как объект системы не может быть удален даже после прекращения его выполнения, если «внешний» описатель не был закрыт. Неверная работа с описателями может приводить при интенсивном создании и разрушении потоков к заметному накоплению «мусора» и дальнейшим ошибкам.

К основным функциям Win32 API по управлению процессами и потоками относятся:

- CreateProcess – создать новый процесс;
- ExitProcess – завершить текущий процесс и все его потоки (вызывается из какого-либо потока данного процесса);
- TerminateProcess – завершить текущий процесс и все его потоки (вызывается из любого потока любого процесса);
- CreateThread – создать новый поток в существующем процессе;
- ExitThread – завершить этот поток (вызывается из завершаемого потока);
- TerminateThread – завершить поток (вызывается из любого потока любого процесса);
- SetPriorityClass – задать класс приоритета для процесса;
- GetPriorityClass – получить класс приоритета для процесса;
- SetThreadPriority – задать приоритет для потока;
- GetThreadPriority – получить приоритет для потока.

Функция CreateProcess имеет следующий прототип:

BOOL CreateProcess(LPCTSTR lpApplicationName, LPCTSTR lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes, LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL

bInheritHandles, DWORD dwCreationFlags, LPVOID lpEnvironment, LPTSTR lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo, LPPROCESS_INFORMATION lpProcessInformation), где lpApplicationName – указатель на строку имени исполняемого файла, lpCommandLine – указатель на командную строку, lpProcessAttributes – указатель на атрибуты защиты процесса, lpThreadAttributes – указатель на атрибуты защиты первичного потока, bInheritHandles – признак передачи процессу описателей наследуемых объектов, dwCreationFlags – определяет флаги, воздействующие на способ создания нового процесса, lpEnvironment – указатель на блок памяти, хранящий строки переменных окружения, которыми будет пользоваться данный процесс, lpCurrentDirectory – указатель на имя текущего каталога для вновь создаваемого процесса (если NULL, то наследуется от родительского процесса), lpStartupInfo – указатель на структуру, описывающую параметры окна, связанного с создаваемым процессом, lpProcessInformation – указатель на структуру, в которую функция возвращает идентификаторы (глобальный идентификатор и идентификатор, служащий для обращения к объектам в функциях) процесса и первичного потока. Если lpszImageName=NULL, то параметр lpszCommandLine трактуется как имя исполняемого файла, вместе с которым может быть передана и командная строка.

Функция ExitProcess имеет следующий прототип:

VOID ExitProcess(UINT uExitCode). Параметр uExitCode служит для возврата кода выхода из процесса.

Функция TerminateProcess имеет следующий прототип:

BOOL TerminateProcess(HANDLE hProcess, UINT uExitCode).

Функция CreateThread имеет следующий прототип:

HANDLE CreateThread (LPSECURITY_ATTRIBUTES lpThreadAttributes, DWORD dwStackSize, LPTHREAD_START_ ROUTINE lpStartAddress, LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId), где lpThreadAttributes – указатель на атрибуты защиты создаваемого потока, dwStackSize – начальный размер стека потока в байтах, lpStartAddress – указатель на функцию потока, то есть на функцию с которой начинается исполнение потока, lpParameter – аргумент, передаваемый в функцию потока, dwCreationFlags – флаги, управляющие созданием потока, lpThreadId – указатель на идентификатор создаваемого потока.

При успешном выполнении функция CreateThread возвращает указатель на идентификатор описателя вновь созданного потока. Функция потока имеет следующий единый прототип:

DWORD WINAPI ThreadFunc(LPVOID lpvThreadParm){ DWORD dwResult = 0;…Return (dwResult);}.

Функция ExitThread имеет следующий прототип:

VOID ExitThread(DWORD dwExitCode). Параметр dwExitCode служит для возврата кода выхода из потока.

Функция TerminateThread имеет следующий прототип:

BOOL TerminateThread(HANDLE hThread, UINT uExitCode).

Функция SetPriorityClass имеет следующий прототип:

BOOL SetPriorityClass(HANDLE hProcess, DWORD dwPriorityClass);

Возможны следующие значения параметра dwPriorityClass:

– IDLE_PRIORITY_CLASS – простаивающий – потоки процесса выполняются, когда система простаивает (уровень 4);

– NORMAL_PRIORITY_CLASS – нормальный;

– HIGH_PRIORITY_CLAS – высокий приоритет;

– REALTIME_PRIORITY_CLASS – приоритет реального времени.

Функция GetPriorityClass имеет следующий прототип:

WORD GetPriorityClass(HANDLE hProcess);

Функция GetPriorityClass возвращает класс приоритета процесса.

Функция SetThreadPriority имеет следующий прототип:

BOOL SetThreadPriority(HANDLE hThread, int nPriority);

Параметр nPriority может принимать следующие значения:

– THREAD_PRIORITY_ABOVE_NORMAL – приоритет потока должен быть на 1 единицу больше класса приоритета процесса;

– THREAD_PRIORITY_BELOW_NORMAL – приоритет потока должен быть на 1 единицу меньше класса приоритета процесса;

– THREAD_PRIORITY_HIGHEST – приоритет потока должен быть на 2 единицы больше класса приоритета процесса;

– THREAD_PRIORITY_LOWEST – приоритет потока должен быть на 2 единицы меньше класса приоритета процесса;

– THREAD_PRIORITY_NORMAL – приоритет потока должен соответствовать классу приоритета процесса;

– THREAD_PRIORITY_TIME_CRITICAL – указывает уровень базового приоритета 15 для классов приоритета процесса

IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, и уровень базового приоритета 31 для класса приоритета REALTIME_PRIORITY_CLASS;

– THREAD_PRIORITY_IDLE – указывает базовый уровень приоритета 1 для классов приоритета процесса IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, и уровень базового приоритета 16 для класса приоритета процесса REALTIME_PRIORITY_CLASS процесса.

Функция GetThreadPriority имеет следующий прототип:

int GetThreadPriority(HANDLE hThread). Функция GetThreadPriority возвращает уровень приоритета потока.

Взаимодействие между процессами может быть организовано посредством посылки сообщений их окнам. Для этого служит функция SendMessage. Функция SendMessage имеет следующий прототип:LRESULT SendMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam), где hWnd – идентификатор дескриптора окна, которому посылается сообщение, Msg – посылаемое сообщение, wParam – первый параметр сообщения, lParam – второй параметр сообщения.

Чтобы осуществить передачу данных из одного процесса другому, нужно послать из первого процесса окну второго процесса сообщение WM_COPYDATA: SendMessage(hwndReceiver, WM_COPYDATA, (WPARAM) hwndSender, (LPARAM)&cds). Здесь hwndReceiver – идентификатор дескриптора окна второго процесса; hwndSender – идентификатор дескриптора окна первого процесса; cds – переменная типа COPYDATASTRUCT, определенная следующим образом: typedef struct tagCOPYDATASTRUCT {DWORD dwData; DWORD cbData; PVOID lpDATA; } COPYDATASTRUCT;

Чтобы переслать данные другому потоку нужно вначале инициировать переменную cds данного типа. Параметр dwData резервируется для использования в программе. Параметр cbData задает число байт, пересылаемых в другой процесс, а параметр lpData указывает на первый байт данных.

Для определения идентификаторов окон может быть использована функция HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName).

Если функция выполнилась успешно, то она возвращает идентификатор дескриптора окна, в противном случае – NULL.

Пример выполнения:

Для выполнения задания, необходимо объявить переменную объекта синхронизации и прототипы функции (рисунок 6.1).

```
using namespace std;
CRITICAL_SECTION cs;
//переменная для использования критической секции

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

DWORD WINAPI Write(LPVOID);
DWORD WINAPI Read(LPVOID);
```

Рисунок 6.1 – Объявление прототипов функций

Определить функции, которые будут выполнять чтение/запись данных из файла. И в диалоговой функции создавать потоки, передавая выполнение функциям чтения/записи (Рисунок 6.2).

```
BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_INITDIALOG:
        //инициализация критической секции
        InitializeCriticalSection(&cs);
        return TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDC_BUTTON1)
        {
            HANDLE hThread = CreateThread(NULL, 0, Write, 0, 0, NULL);
            CloseHandle(hThread);
            hThread = CreateThread(NULL, 0, Read, 0, 0, NULL);
            CloseHandle(hThread);
        }
        return TRUE;


    case WM_CLOSE:
        DeleteCriticalSection(&cs);
        EndDialog(hWnd, 0);
        return TRUE;

    }
    return FALSE;
}
```

Рисунок 6.2 – Диалоговая функция

При реализации функций чтения/записи, учитываем моменты вхождения в объект синхронизации EnterCriticalSection(&cs) и выход LeaveCriticalSection(&cs).

Задание:

Предусмотреть в приложении, разработанном в лабораторных работах 1-5, работу с тремя рабочими потоками. Первый поток выполняет добавление данных в файл. Второй поток считывает данные из файла. А третий поток, выполняет вывод данных на экран, в соответствии с критериями поиска. При этом необходимо использовать объект синхронизации: Критическая секция (нечетный вариант) или Мьютекс (четный вариант).

Контрольные вопросы:

1. Что такое поток.
2. Функция создания потока и ее параметры.
3. Что такое функция потока, ее параметры и их использование.
4. Функции приостановки и возобновления потока.
5. Функция Sleep.
6. Какие функции по управлению процессами и потоками Вы знаете? Опишите назначение их параметров.
7. Какие классы приоритета процессов Вы знаете?
8. Какая функция позволяет изменить класс приоритета процесса? Опишите ее параметры.
9. Какая функция позволяет узнать текущий класс приоритета процесса?
10. Какие уровни приоритета потоков Вы знаете?
11. Какая функция позволяет изменить уровень приоритета потока? Опишите ее параметры.
12. Какая функция позволяет узнать текущий уровень приоритета потока?
13. Какая функция позволяет переслать данные из одного процесса в другой? Опишите ее параметры.
14. Какая функция позволяет определить дескриптор окна приложения по его имени? Опишите ее.

Приложение А
(справочное)

Варианты заданий

Вариант 1

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Тип процессора
     -Объём памяти
   - Help
     - About
2. Для всех пунктов меню создать горячие клавиши.
3. В случае выбора элемента меню "Тип процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Тип процессора" и со значениями "Celeron", "Pentium-II", "Pentium-III", "Pentium-4", "AMD K7 Athlon", "AMD K7 Duron", "Other:" и поле для ввода другого значения, кнопку OK.
4. В случае выбора элемента меню "Объём памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём памяти" и со значениями "32Mb", "64Mb", "128Mb","256Mb","512Mb", "Other:" и поле для ввода другого значения, кнопку OK.
5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 2

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File

- Open base
- Exit
- информация
-Тип памяти
-Объём винчестера
- Help
- About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Тип памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Тип памяти" и со значениями "DIMM", "RIMM", "DDR", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Объём винчестера" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём винчестера" и со значениями "10", "20", "30", "40", "60", "80", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 3

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
- File
- Open base
- Exit
- информация
-Цена
-Фирма
- Help
- About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Цена" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Цена" и со значениями "(0,250)", "[250,500), "[500,750)",

"[750,1000)", "[1000,+inf)", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Фирма" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Фирма" и со значениями "РиМ", "РиК", "ASCOD", "СВЕГА-ПЛЮС", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 4

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Станция Метро
     -Частота процессора
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Станция Метро" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Станция Метро" и со значениями "Горьк.", "Петр.", "Техн.и-т.", "Пл.Восст.", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Частота процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Частота процессора" и со значениями "500", "1000", "1500", "2000", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 5

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Модель видеокарты
     -CD-ROM
   - Help
     - About
2. Для всех пунктов меню должны быть горячие клавиши.
3. В случае выбора элемента меню "Модель видеокарты" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Модель видеокарты" и со значениями "Riva TNT", "Riva TNT2", "ATI Radeon", "GeForce", "GeForce2", "GeForce4", "Other:" и поле для ввода другого значения, кнопку OK.
4. В случае выбора элемента меню "CD-ROM" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "CD-ROM" и со значениями "48x", "52x", "CD-RW", "DVD", "Other:" и поле для ввода другого значения, кнопку OK.
5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 6

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Объём видеопамяти
     -Гарантия
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Объём видеопамяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём видеопамяти" и со значениями "8Mb", "16Mb", "32Mb", "64Mb", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Гарантия" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Гарантия" и со значениями "1г.", "2г.", "3г.", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 7

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Тип процессора
     -Гарантия
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Тип процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Тип процессора" и со значениями "Celeron", "Pentium-II", "Pentium-III", "Pentium-4", "AMD K7 Athlon", "AMD K7 Duron", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Гарантия" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Гарантия" и со значениями "1г.", "2г.", "3г.", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 8

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Объём памяти
     -Объём видеопамяти
   - Help
     - About
2. Для всех пунктов меню должны быть горячие клавиши.
3. В случае выбора элемента меню "Объём памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём памяти" и со значениями "32Mb", "64Mb", "128Mb", "256Mb","512Mb", "Other:" и поле для ввода другого значения, кнопку OK.
4. В случае выбора элемента меню "Объём видеопамяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём видеопамяти" и со значениями "8Mb", "16Mb", "32Mb", "64Mb", "Other:" и поле для ввода другого значения, кнопку OK.
5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 9

Составить программу, которая выдает справочную информацию о продаваемых компьютерах (данные о рынке компьютеров приведены в файле pc.txt).

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Тип памяти
     -CD-ROM

- Help
  - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Тип памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Тип памяти" и со значениями "DIMM", "RIMM", "DDR", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "CD-ROM" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "CD-ROM" и со значениями "48x", "52x", "CD-RW", "DVD", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 10

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
- File
  - Open base
  - Exit
- информация
  -Объём винчестера
  -Модель видеокарты
- Help
  - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Объём винчестера" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём винчестера" и со значениями "10", "20", "30", "40", "60", "80", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Модель видеокарты" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Модель видеокарты" и со значениями "Riva TNT", "Riva TNT2", "ATI Radeon", "GeForce",

"GeForce2", "GeForce4", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 11

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Цена
     -Частота процессора
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Цена" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Цена" и со значениями "(0,250)", "[250,500), "[500,750)", "[750,1000)", "[1000,+inf)", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Частота процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой"Частота процессора" и со значениями "500", "1000", "1500", "2000", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 12

Составить программу, которая выдает справочную информацию о продаваемых компьютерах.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base

- Exit
- информация
-Фирма
-Станция Метро
- Help
- About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Фирма" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Фирма" и со значениями "РиМ", "РиК", "ASCOD", "СВЕГА-ПЛЮС", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Станция Метро" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Станция Метро" и со значениями "Горьк.", "Петр.", "Техн.и-т.", "Пл.Восст.", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список компьютеров.

Вариант 13

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
- File
- Open base
- Exit
- информация
-Тип процессора
-Цена
- Help
- About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Тип процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Тип процессора" и со значениями "Pentium", "AMD K7", "Celeron", "Pentium-II", "Pentium-III", "Pentium-4",

"Transmeta Crusoe", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Цена" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Цена" и со значениями "(0,750)","[750,1500)","[1500,2250)", "[2250,3000)","[3000,+inf)", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 14

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Объём памяти
     -Объём винчестера
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Объём памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём памяти" и со значениями "32Mb", "64Mb", "128Mb", "256Mb","512Mb", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Объём винчестера" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём винчестера" и со значениями "10", "20", "30", "40", "60", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 15

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Производитель
     -Объём памяти
   - Help
     - About
2. Для всех пунктов меню должны быть горячие клавиши.
3. В случае выбора элемента меню "Производитель" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Производитель" и со значениями "Toshiba", "ASUS", "Sony", "DELL", "Other:" и поле для ввода другого значения, кнопку OK.
4. В случае выбора элемента меню "Объём памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём памяти" и со значениями "32Mb", "64Mb", "128Mb","256Mb","512Mb", "Other:" и поле для ввода другого значения, кнопку OK.
5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 16

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Частота процессора
     -CD-ROM
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Частота процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Частота процессора" и со значениями "500", "1000", "1500", "2000", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "CD-ROM" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "CD-ROM" и со значениями "DVD-ROM", "CD-ROM", "DVD/CD-RW", "DVD-RW/CD-RW", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 17

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Монитор
     -Максимальное разрешение
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Монитор" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Монитор" и со значениями "12", "13.3", "14.1", "15.1", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Максимальное разрешение" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Максимальное разрешение" и со значениями "640x480", "800x600", "1024x768", "1280x1024", "1600x1200" , "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 18

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Производитель
     -Тип процессора
   - Help
     - About
2. Для всех пунктов меню должны быть горячие клавиши.
3. В случае выбора элемента меню "Производитель" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Производитель" и со значениями "Toshiba", "ASUS", "Sony", "DELL", "Other:" и поле для ввода другого значения, кнопку OK.
4. В случае выбора элемента меню "Тип процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Тип процессора" и со значениями "Pentium", "AMD K7", "Celeron", "Pentium-II", "Pentium-III", "Pentium-4", "Trans-meta Crusoe", "Other:" и поле для ввода другого значения, кнопку OK.
5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 19

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация

-Цена
-Максимальное разрешение
- Help
- About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Цена" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Цена" и со значениями "(0,750)","[750,1500)","[1500,2250)", "[2250,3000)","[3000,+inf)", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Максимальное разрешение" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Максимальное разрешение" и со значениями "640x480", "800x600", "1024x768", "1280x1024", "1600x1200", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 20

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
- File
- Open base
- Exit
- информация
-Объём памяти
-Монитор
- Help
- About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Объём памяти" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём памяти" и со значениями "32Mb", "64Mb", "128Mb", "256Mb","512Mb", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Монитор" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Монитор" и со значениями "12", "13.3", "14.1", "15.1", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 21

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
   - File
     - Open base
     - Exit
   - информация
     -Объём винчестера
     -CD-ROM
   - Help
     - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Объём винчестера" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Объём винчестера" и со значениями "10", "20", "30", "40", "60", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "CD-ROM" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "CD-ROM" и со значениями "DVD-ROM", "CD-ROM", "DVD/CD-RW", "DVD-RW/CD-RW", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 22

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:

- File
  - Open base
  - Exit
- информация
  -Фирма
  -Частота процессора
- Help
  - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Фирма" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Фирма" и со значениями "Микробит", "Диалектика", "ДВМ-Нева", "МикроМатикс", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Частота процессора" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Частота процессора" и со значениями "500", "1000", "1500", "2000", "Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Вариант 23

Составить программу, которая выдает справочную информацию о продаваемых ноутбуках.

Программа должна:

1. Сформировать строку меню, включающую элементы выбора:
- File
  - Open base
  - Exit
- информация
  -Станция метро
  -Производитель
- Help
  - About

2. Для всех пунктов меню должны быть горячие клавиши.

3. В случае выбора элемента меню "Станция метро" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Станция метро" и со значениями "Горьк.",

"Петр.", "Техн.и-т.", "Пл.Восст.", "Other:" и поле для ввода другого значения, кнопку OK.

4. В случае выбора элемента меню "Производитель" программа создает окно диалога, содержащее группу взаимоисключающих радио-кнопок с меткой "Производитель" и со значениями "Toshiba", "ASUS", "Sony", "DELL","Other:" и поле для ввода другого значения, кнопку OK.

5. На основании выбранных в п.3 или в п.4 значений программа должна вывести отсортированный в порядке возрастания цены список ноутбуков.

Литература

1. Литвиненко, Н. А. Технология программирования на C++. Win32 API-приложения. – СПб.: БХВ-Петербург, 2010. – 288 с.: ил. – (Учебное пособие)
2. Румянцев, П. В. Работа с файлами в Win 32 / П. В. Румянцев. – 2-е изд.. – Москва : Горячая линия – Телеком, 2009. – 214 с. – (Бархатный путь)
3. Финогенов К. Г. Win32. Основы программирования. – Изд. 2-е, испр. и доп. – Москва : ДИАЛОГ-МИФИ, 2006. – 416 с
4. Щупак Ю. А. Win 32 API. Разработка приложений для Windows. – Санкт-Петербург : Питер, 2008. – 592 с.. – (Библиотека программиста)
5. Щупак Ю. А. Win32 API. Эффективная разработка приложений. – Санкт-Петербург : Питер, 2007. – 571с.. – (Для профессионалов)

Гридина Елена Ивановна

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Практикум
по одноименной дисциплине
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение информационных систем»
заочной формы обучения

Подписано в печать 13.06.18.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 4,18. Уч.-изд. л. 4,44.
Изд. № 1.
http://www.gstu.by

Отпечатано на цифровом дуплекаторе
с макета оригинала авторского для внутреннего использования.
Учреждение образования «Гомельский государственный
технический университет имени П.О. Сухого».
246746, г. Гомель, пр. Октября, 48.