Лекция 1

Въведение

Целта на дисциплината Програмни Системи (ПС) е изучаването на технологии за създаване на програмно осигуряване (пр. системи) чрез създаване на приложения под Операционната Система (ОС) Windows и Microsoft Visual C++. **Тези приложения имат трислойна архитектура, включваща потребителски диалог, бизнес логика и Бази Данни (БД).**

Потребителският диалог включва: *управление на събития*; *управление на прозорци*, *управление на ресурси* – памет и устройства за вход, ресурси в Windows; *обмен на данни между приложенията*; *технологии при големи приложения* – DLL (Dynamic Link Libraries), COM (Component Object Model), MFC Microsoft Foundation Class Library ATL (Active Template Library), БД- ODBC.

Основните концепции в ОС Windows могат да се формулират като:

- ***преместваемост*** на изпълнимите приложения;
- ***обектна ориентация*** на програмирането на приложенията.

ИЗПОЛЗВАНИ СЪКРАЩЕНИЯ

Active Template Library (**ATL**) – библиотека от шаблони

Microsoft Foundation Classes Library (**MFC**) - библиотека от базови класове обекти на фирмата Microsoft

Application Programming Interface (**API)** динамична библиотека с функции,които реализират всички необходими системни действия

Graphic Device Interface (**GDI**) подмножество на API, интерфейс на графични устройства

Component Object Model (**COM**)

Run Time Libraries (**RTL**)

Object Linking Embedding (**OLE**) – Свързване и вграждане на обекти

Dynamic Link Libraries (**DLL**) динамична библиотека

Microsoft Foundation Class Library (**MFC**) – библиотека от базови класове обекти на фирмата Microsoft Microsoft Developer Network (**MSDN**) библиотеката знания

Интегрираната развойна среда MS Visual C++

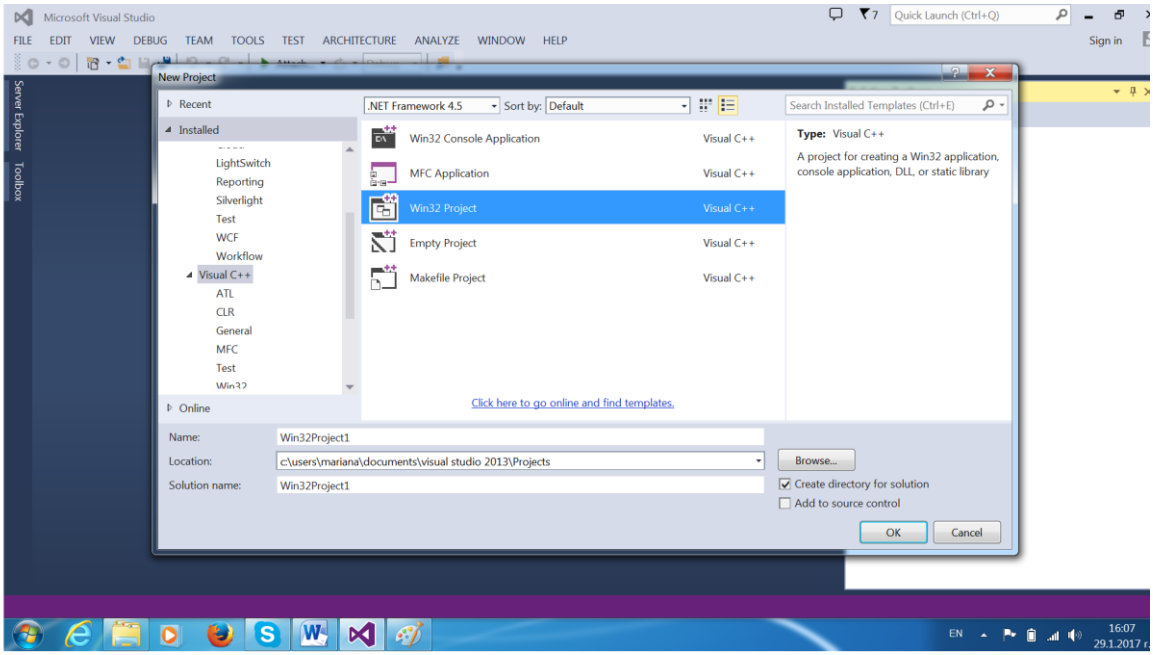
Средата за разработка на програми Microsoft Visual C++ **Интегрирана развойна среда MS Visual C++** (Integrated Development Environment – (IDE)) е предназначена за създаването на приложения – програми, снабдени с всичко необходимо за тяхната работа. В Microsoft Visual C++ приложенията често се разработват на основата на **Microsoft Foundation Class Library** (MFC) – библиотека от базови класове обекти на фирмата Microsoft, макар че има възможност и за създаването и на други типове програми. Такива приложения представляват съвкупност от обекти, които самото приложение и неговите компоненти: документи, вид на документите, диалози и т.н.

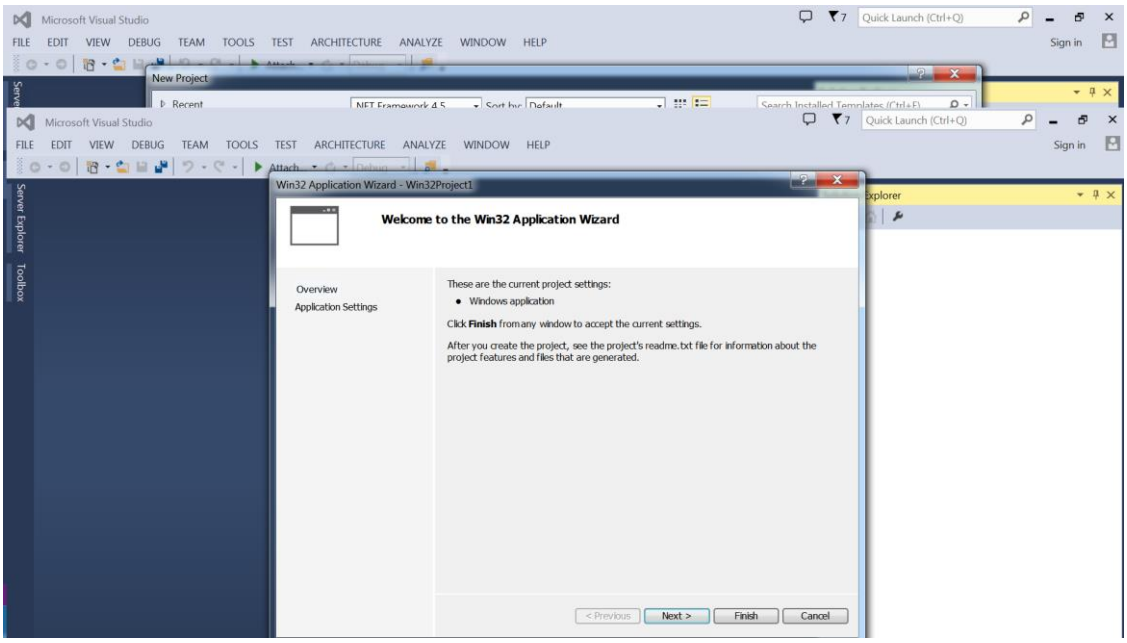
Всяко приложение взаимодейства с Windows чрез **Application Programming Interface** (API). API съдържа няколко стотици функции, които реализират всички необходими системни действия, такива като отделяне на памет, създаване на прозорци, извеждане на екран и т.н. Функциите API се съдържат в **динамични библиотеки Dynamic Link Libraries** (DLL), които се зареждат в паметта само в този момент, когато към тях има обръщение. Едно от подмножествата на API е **Graphic Device Interface** (GDI) – **интерфейса на графични устройства**. Задачата на GDI е да обезпечи апаратната независимост на графичния изход . На GDI се дължи това, че приложенията могат да се изпълняват на различни компютърни конфигурации.

Интерфейсът на Microsoft Visual C++ включва удобни средства за създаването и редактирането на всички типове ресурси. Такива са мощното средство за скелетни приложения (AppWizard), а така също средството за създаване и управление на класовете на приложението (ClassWizard). Създаването на просто приложение, което едновременно ще съдържа пълноценен интерфейс и което ще създава стандартен прозорец със стандартните елементи на управление става с инструменталното средство на Visual C++ AppWizard. То позволява значително да се упрости процеса на създаване на приложение на основата на MFC и Win32. При работа с този инструмент се появява последователност от диалогови прозорци, в които AppWizard задава на програмиста въпроси. В процеса на диалога потребителя определя тип и характеристиките на проекта, който той иска да разработи. Определяйки какви класове от библиотеката MFC са необходими за този проект приложения и

строи основите на всички нужни производни класове. По-нататък програмиста определя свойствата и поведението на обектите на тези класове. Проектът на приложенията обединява в себе си описанията на класовете, описанията на ресурсите и т.н.

Неоценима помощ при разработката на програми може да укаже библиотеката знания Microsoft Developer Network (MSDN). В тази библиотека могат да се намери информация не само за API и MFC , но и много примери.



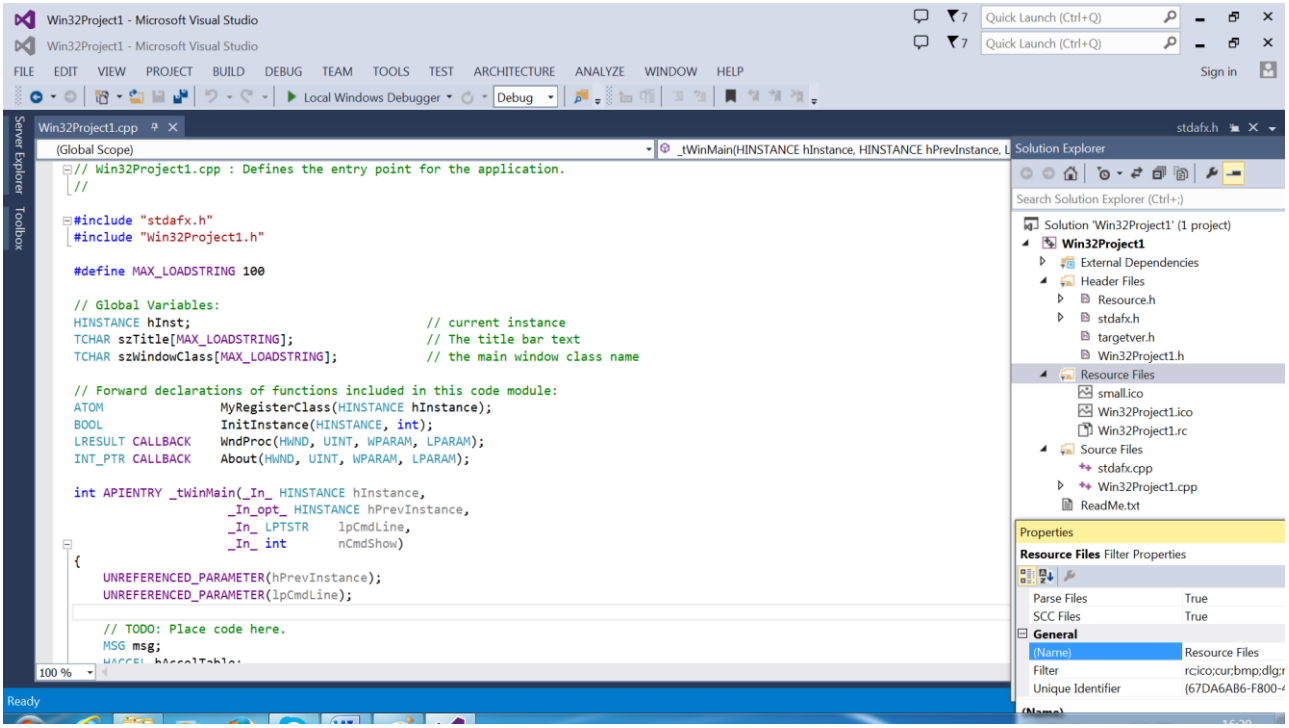


Developer Studio е IDE (Integrated Development Environment) е интегрирана развойна среда, която се споделя от Visual C++, Visual Basic, Visual C++, C# и

няколко други продукта. Средата има два редактора: AppWizard и ClassWizard. ***AppWizard*** е генератор на код, създаващ работещ скелет на едно приложение. Чрез диалогови прозорци се указват характеристики, имена на класове и имена на файлове с изходен код. ***ClassWizard*** е програма (реализирана като DLL), която спестява рутинната работа по писането на класовете във Visual C++. Той може да генерира нови класове, виртуални или нови функции за обработка на съобщения.

Интегрираната развойна среда MS Visual C++ (Integrated Development Environment – (IDE)) ***предоставя:***

- Текстов редактор за създаване на първичен (source) код на програмите;
- Средство отстраняване на програмните дефекти (debugger);
- Компилатор и свързващ редактор (linker);
- Помощна система (Microsoft Developer Network (MSDN))**;**
- Инструменти за оптимизация на програмния код;
- Class Wizard средство за създаване и управление на класовете на приложението;
- AppWizard (AFC) средство за създаване и редактиране на скелетни приложения;
- библиотеки: Object Link Embedding (OLE), Runtime Libraries (RTL); Active Template Library (ATL); *.dll (Dynamic Link Library)



Файлове, създавани в средата на Microsoft Visual C++

Средата използва следните папки и файлове:

При създаване на нов проект се създава папка със ***същото име*** като това на проекта. Папката, съдържа папка с име **Debug** и може да съдържа следните типове файлове:

*.sln Solution File –описание на проекта

(В по-малки версии *.dsw – Project Workspace – файл на работна област на проекта;

*.dsp – Project File – проектен файл;)

*.cpp – C++ source file – файл с първичен код (source) на програмите;

*.h – header file - файл с първичен код на използваните библиотеки;

resource.h – съдържа имената на ресурси и диалоговите контроли със съответните #define константи;

*.rc - Resource Template – текстов файл с шаблон на ресурс;

ReadMe.txt – Text Document – текстов файл;

*.aps – APS File – Active Pages Server

*.ico ; smal.ico – Icon File – съдържа изображения на иконите на приложението;

*.ncb – NCB File (Network Control Block) Представя мрежовия контролен блок. Съдържа информация за командите на представяне. *.html – HTML Documents

*.cur - изображение на курсор;

*.exe – Файл- изпълнимо преместваемо програма – приложение;

*.dll - динамични програмни библиотеки.

Модел на програмирането под Windows

Когато Windows стартира приложна програма, той извиква нейната функция **WinMain**. Най-важната задача на WinMain е да създаде главния прозорец на приложението, който трябва да има свой собствен код за ***обработка на съобщенията***, изпращани от Windows. Една от съществените разлики между програма написана за MS-DOS, и програма написана за Windows, се състои в това, че MS-DOS програмите се позовават на операционната система за да получат вход от потребителя, докато една Windows програма ***обработва потребителския вход*** чрез ***съобщения*** от операционната система. Повечето съобщения в Windows са стриктно дефинирани и се отнасят за всички програми. За да могат да се обработват съобщенията към програмите се поставя изискването за „структурност".

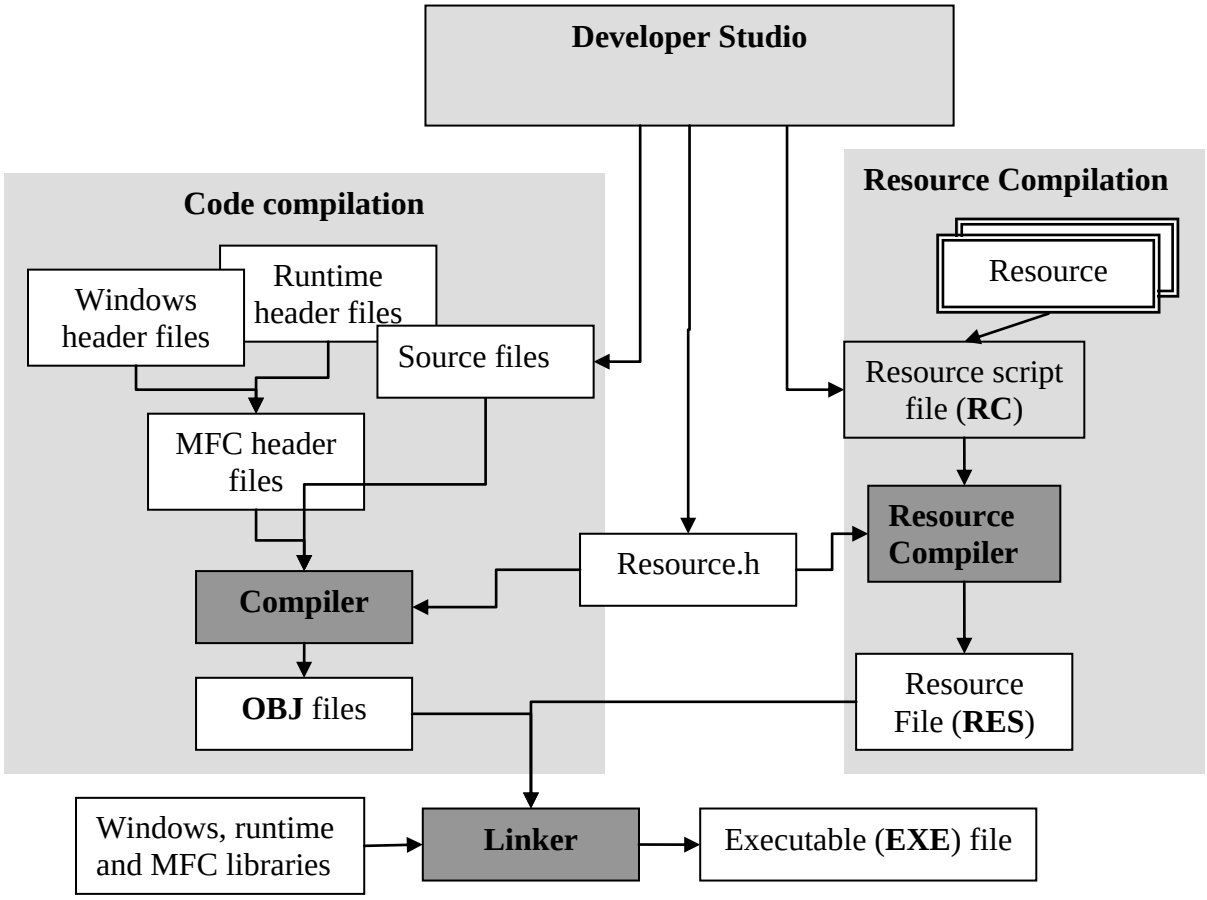
В Windows е въведено едно абстрактно междинно ниво, наречено интерфейс на ***графично устройство или GDI (graphical device interface)***. ОС Windows предоставя драйверите за принтера и видео-драйверите. Вместо да адресира директно хардуера, програмата извиква GDI функции, които се обръщат към една структура от данни, наречена ***контекст на устройство*** (device context). ОС Windows асоциира тази структура с физическото устройство и издава подходящите входно - изходни инструкции. GDI е почти толкова бърз, колкото и директния достъп до видео – паметта, а също така позволява различни приложения, написани за Windows, да използват споделено дисплея.

Когато се програмира под Windows част от данните се съхраняват в ***ресурсен файл*** (*.RES), използвайки определен брой ***установени формати***. Свързващата програма (***linker***) събира този двоичен ресурсен файл с кода, генериран от C++ ***компилатора*** и създава изпълнимата програма.

В Windows има ***библиотеки за динамично свързване*** DLL (Dynamic Link Library), които могат да се зареждат и свързвани по време на изпълнение на програмите. Няколко приложения могат да споделят една библиотека за динамично свързване, което спестява памет и дисково пространство. Динамичното свързване повишава възможността за модифициране на

програмата, защото DLL могат да бъдат компилирани и тествани самостоятелно. Класовете на приложната среда от готовите DLL могат да се свързват към класовете за приложението както статично така и динамично. Потребителят може да създава свои собствени DLL.

Процесът на ***изграждане (компилиране и свързване)*** на едно Visual C++ приложение е илюстриран на следващата схема (Фиг.1.):



Фиг.1. Процес на изграждане на едно приложение

C/ C++ компилаторът обработва както C така и C++ код. Кодът ***Компилаторът на ресурси*** чете ASCIIфайл с описанието на ресурсите (файл RC), създаден от ресурсните редактори и го записва в двоичен ресурсен файл (RES) за свързващата програма. ***Свързващата програма*** (Linker) чете OBJ и RES файлове, създадени от C /C++ компилатора и ресурсния компилатор, и отваря LIB файлове, за да включи MFC код, код от runtime библиотеките и Windows код, след което създава EXE файла на проекта. За да се проследи обработката на данните в една програма се използва ***трасиращата програма*** (Debugger).

Пример на resouce.h файл:

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by Win32Project1.rc

#define IDS_APP_TITLE                   103
#define IDR_MAINFRAME                   128
#define IDD_WIN32PROJECT1_DIALOG        102
#define IDD_ABOUTBOX                    103
#define IDM_ABOUT                       104
#define IDM_EXIT                        105
#define IDI_WIN32PROJECT1               107
#define IDI_SMALL                       108
#define IDC_WIN32PROJECT1               109
#define IDC_MYICON                      2
#ifndef IDC_STATIC
#define IDC_STATIC                      -1
#endif
```

Пример за връзка на кода с идентификаторите на ресурси и контроли:

```
....
....
        int wmId = LOWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
        case IDM_ABOUT:
           DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
           break;
        case IDM_EXIT:
           DestroyWindow(hWnd);
           break;
```

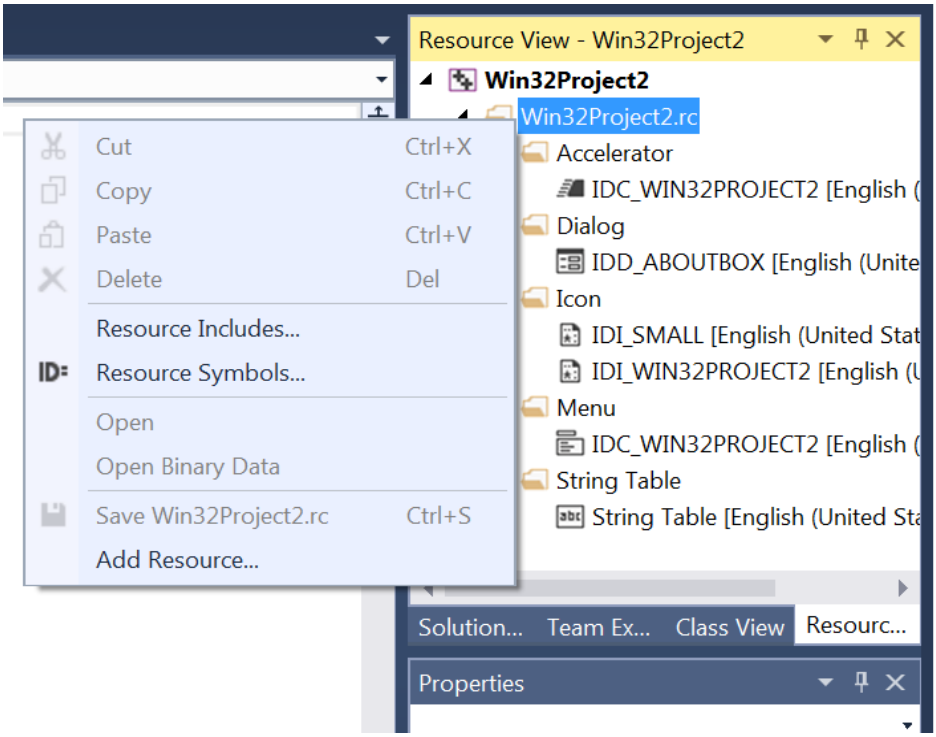
Лекция 2

Ресурси и контроли

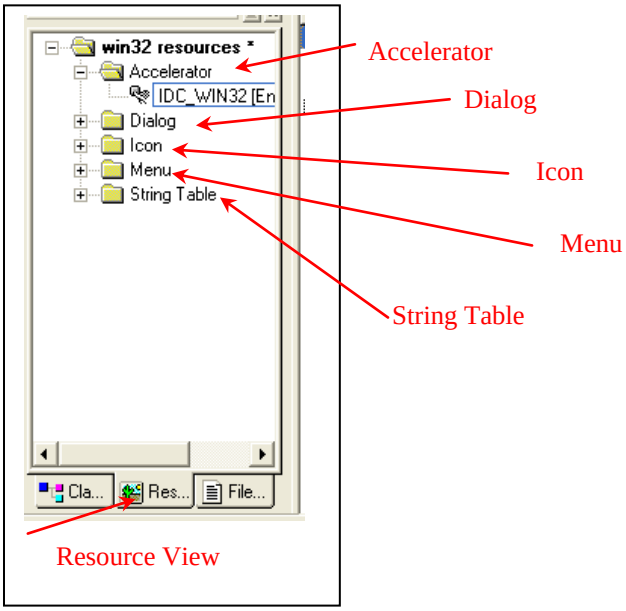
MS Visual C++ предоставя удобни средства за създаване и редактиране на ресурси от всички типове. ***Видове ресурси*** (предефинирани в Windows, ObjectWindows OWL –допълнително помощни ресурси (обекти)): Меню (menu); Таблици с Комбинации от клавиши (accelerators); Диалогови кутии (dialog boxes); Шрифтове (fonts); Курсори за мишката (cursors); Икони (icons); информация за версията на програмата; HTML – WEB страница; Байтови изображения (bitmaps).

Ресурсите са част от изпълняемата програма, но не се помещават в данновите сегменти. Те се зареждат в паметта само когато е необходимо. Ресурсите се описват отделно в текстов файл с разширение RC (Resource Script) с помощта на ***език за описание на ресурсите.*** Тези файлове могат да съдържат и редове с #include директиви за включване на ресурси от други поддиректории. Някои от тези ресурси включват специфични за даден проект обекти, като бит-мап изображения (BMP), икони (ICO), символни низове, съдържащи например съобщения за грешки. Редакторът на ресурси, добавя и съответните на ресурсите и контролите #define константи във файла resource.h.

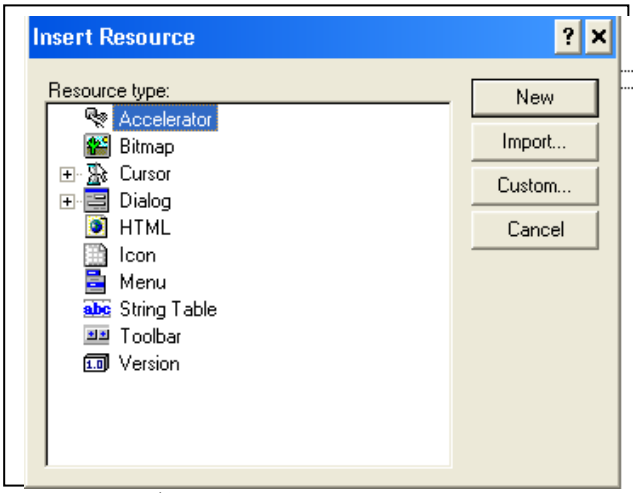
Създаване и редактиране на ресурсите на едно приложение



При създаване на приложения **AppWizard** автоматично окомплектова програмата чрез стандартен набор от ресурси. Те са в компонентата Ресурси (ResourceView) на работното пространство (Workspace) (Фиг.2. а/).



Фиг.2. а/. Редакция на ресурси (Resource View)



Фиг. 2. б/ Добавяне на ресурс.

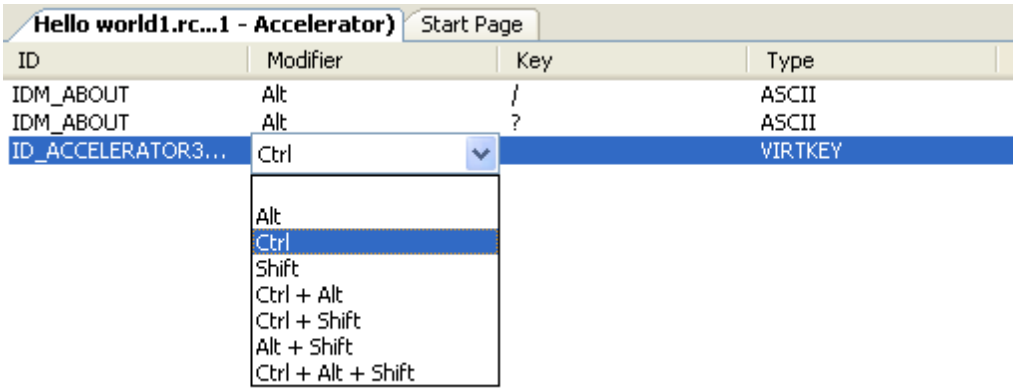
<u>Добавянето на нов ресурс</u> става чрез следните инструкции:

Позициониране на курсора върху папката с името на ресурсите на приложението -**win32 resource** и натискане на десен бутон. Пада менюто за избор на функция за редактиране на ресурсите

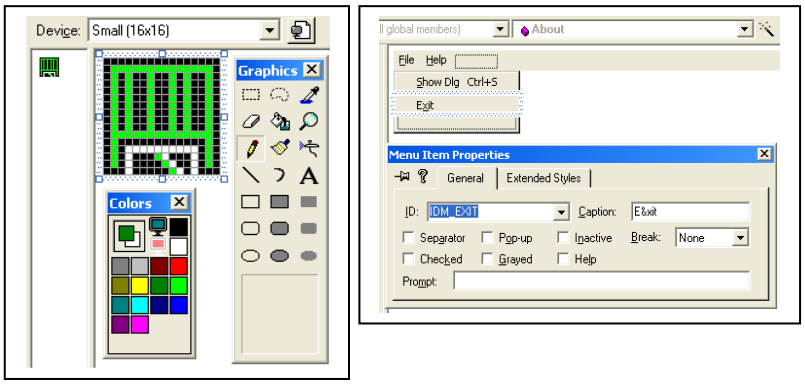
Избираме функция **Insert** чрез натискане на ляв бутон. Пада менюто **Insert resource** (Фиг.2 б/) за да се избере вида на новия ресурс.

Редакция на различните видове ресурси:

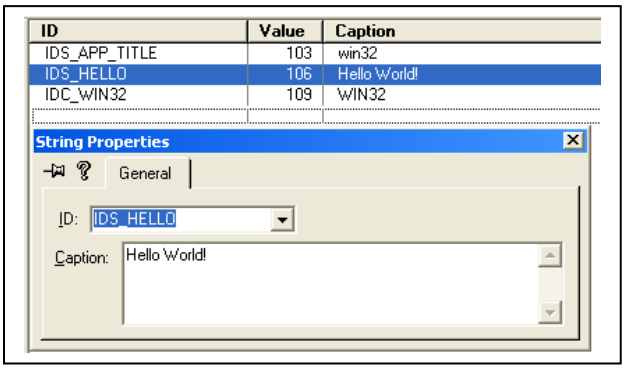
1. Accelerators - Таблици с Комбинации от клавиши



2. Icons – Икони. (Фиг. 4) За създаването и редактирането на иконите се използват средства за рисуване групирани в секциите Colors и Graphics.

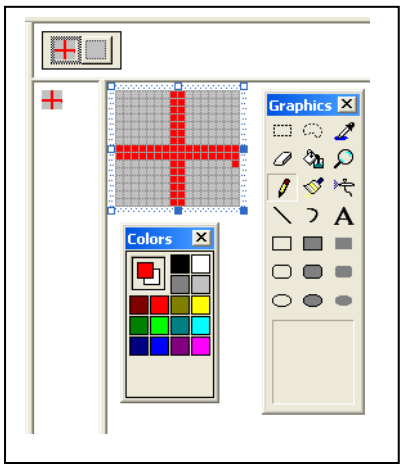


3. Menu – Меню. (Фиг. 5) За създаването и редактирането на менютата се използват Клавишите Insert, Delete и секцията Menu Item Properties.
4. String Table – Таблица със текстове

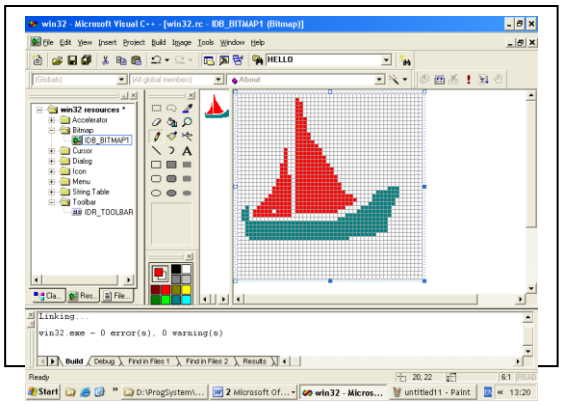


Фиг.6 Редактиране на ресурса – таблица със текстове

5. ToolBar – (Фиг.7) Инструментална лента



Фиг. 7 Редакция на ресурса ToolBar

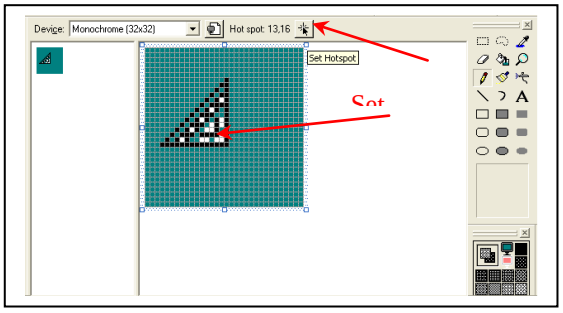


Фиг. 8 Редактиране на ресурса изображение

6. BitMap - Байтови изображения (Фиг. 8). За създаването и редактирането на изображенията се използват средства за рисуване групирани в секциите Colors и Graphics.
7. Cursor - Курсори за мишката

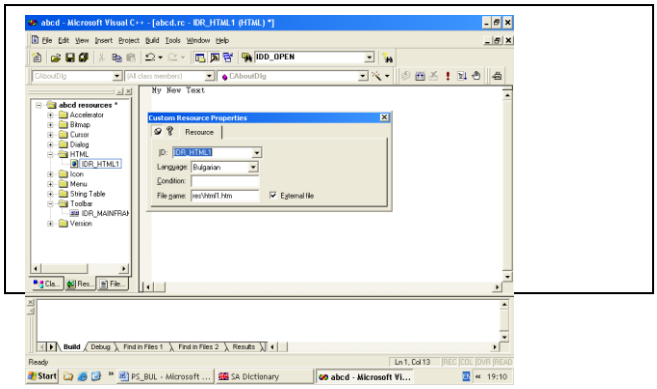
Освен традиционните средства за смяна на цвета, избора на начин и размер на изобразяване съществува възможност да се определи точка от курсора (pixel), HotSpot който ще определя местоположението на мишката. Последователността за това е следната: Натиска се бутона **HotSpot** и после с ляв бутон на мишката се посочва точката, която ще определя местоположението на курсора. За установяване на курсора може да се използва API функцията SetCursor(), като ОС обновява курсора всеки път, когато става преместване на мишката. При това изображението на курсора се заменя със зададеното при регистрацията на класа на прозореца. Един от начините за контрол на формата на курсора се заключава в обработката

на съобщението WM_SETCURSOR. Формата на курсора се установява и в методите CPainterView::OnMouseMove()



Фиг. 9. Създаване или редактиране на курсор

8. Version -информация за версията на програмата
9. HTML – създава се използвайки текстово редактиране и средствата на секцията



Фиг.10 Създаване или редактиране на HTML страница.

Ресурсите се описват във файловете *.RC ->*.RES като се използва език за описание на ресурсите.

В програмния код ***ресурсите*** се указват със своя ***идентификатор***. Начинът на образуване на идентификаторите на ресурсите е указан по-долу:

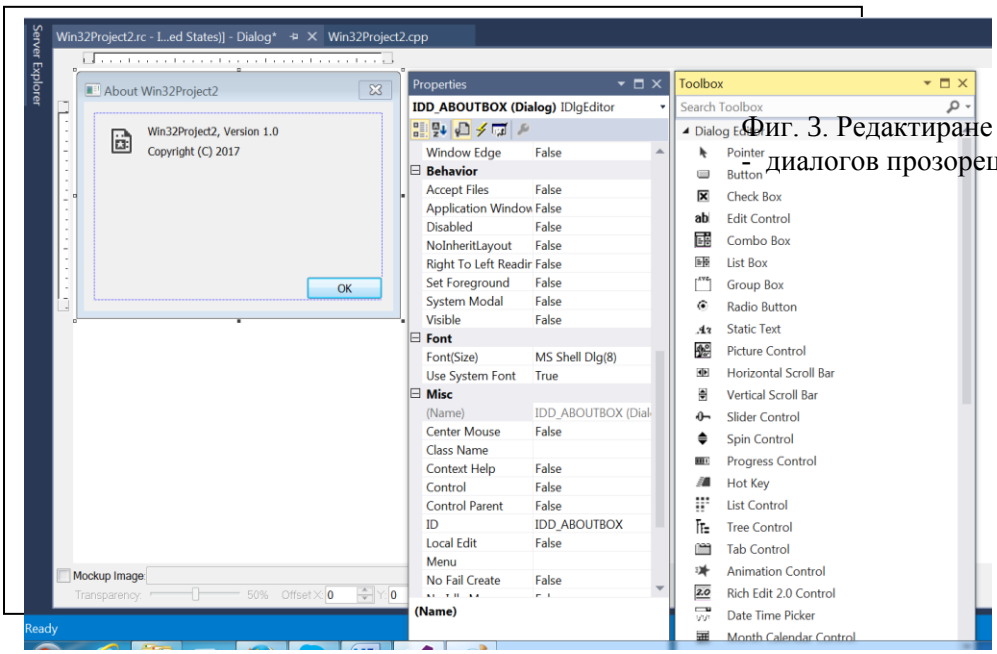
ID{ **1 символ** указва вида на ресурса като: **M** (menu)/ **D** (dialog)/ **I**(icon) / **A** (accelerator) / (string table)/…. }_username

Примери: идентификатор на диалогов прозорец: IDD_ABOUTBOX, идентификатор на меню: IDM_WIN32_8 идентификатор на малка икона: IDI_SMALL.

10. Dialog boxes - Диалогови кутии (Диалогов прозорец)

Създаването и редактирането на диалоговите кутии използва средства групирани в секцията Controls и Dialog Properties.

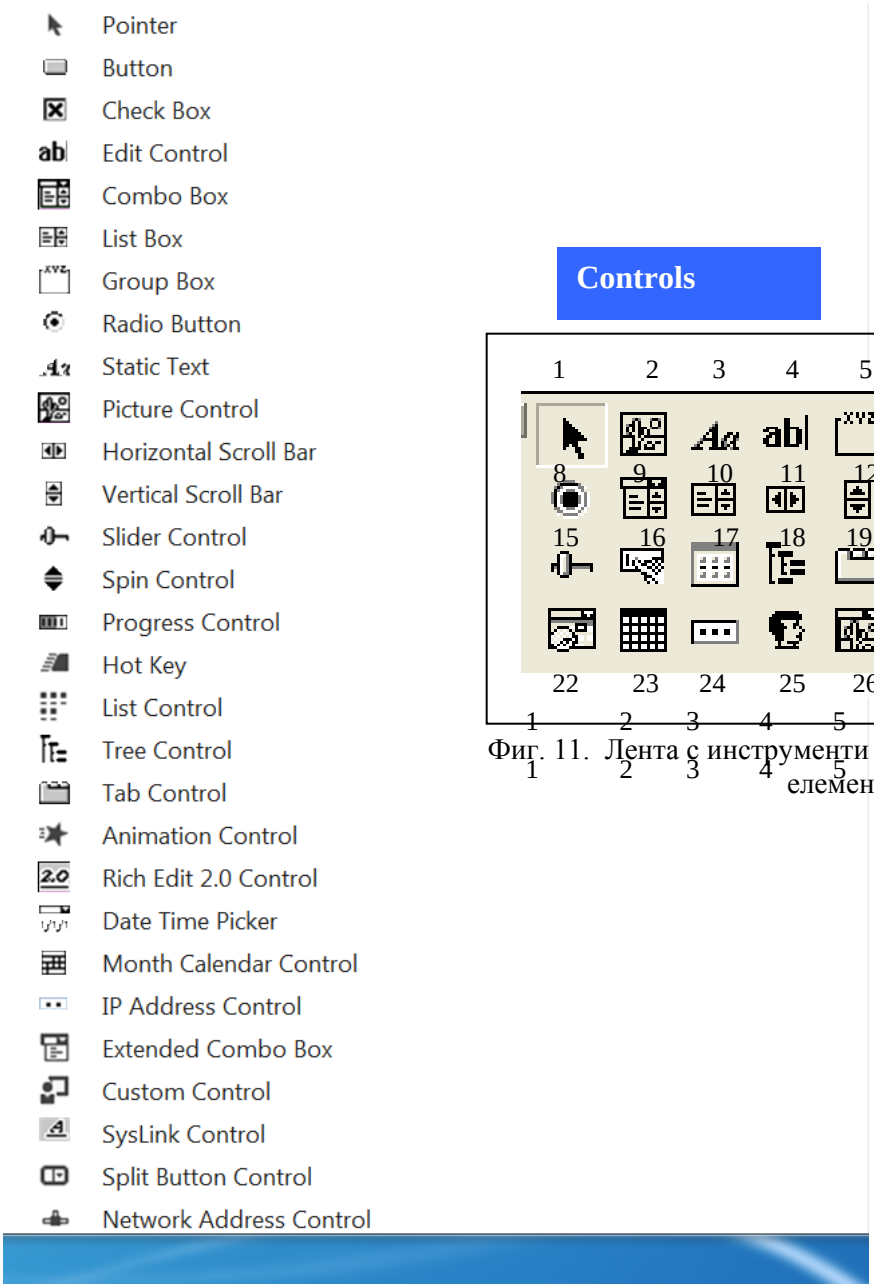
Един диалог съдържа определен брой елементи, наречени **контроли**.



Фиг. 3. Редактиране на ресурс диалогов прозорец

Диалогови Елементи (Диалогови Контроли)

Диалоговите елементи, наречени контроли са достъпни за добавяне в диалоговата кутия чрез Toolbox, част от тях са представени по-долу. Описанието на контролите е достъпно



Фиг. 11. Лента с инструменти (Controls) на диалоговите елементи

Възможностите на някои от диалогови елементи са описани по долу, използвайки въведената номерация:

1. Select –позволява да се кликне върху нещо и да го промените или преместите;
2. Picture - позволява да се покаже изображение.
3. Static Text – позволява да се въвежда текст, който да се появява върху диалоговия прозорец. Потребителят не може да промени този текст. Използва се за надписи.
4. Edit Box - позволява да се добави към диалоговия прозорец контроли за редактиране с цел въвеждане на текст.

5. Group Box – позволява да се постави празно поле около няколко контроли и те да образуват група.
6. Button – позволява да се добави бутон към диалогов прозорец;
7. Check Box - позволява към диалоговия прозорец да се добави контролно поле. То позволява да се определи дали този елемент има или не отметка. Ако има няколко такива контролни полета в една група, потребителят може да отметне повече от един елемент на групата.
8. Radio Button - позволява към диалоговия прозорец да добавяте друг вид контролен бутон (радио-бутон). Тези бутони се използват когато в даден момент само един елемент от групата може да бъде избран. Той позволява да се маркира или не, но като единствен избор от дадена група.
9. Combo Box - позволява да се добави падащ списък към диалогов прозорец. Падащите полета са комбинация от контроли за редактиране и списъчни полета. Може да се избира или въвежда стойност.
10. List Box – позволява да добавите списъчно поле към диалогов прозорец. Списъчните полета дават на потребителя набор от възможности за избор.
11. Horizontal Scroll Box – Инструмент – хоризонтален плъзгач.
12. Vertical Scroll Box - Инструмент – вертикален плъзгач.
13. Spin Bar – инструмент за указване на посока на превъртане
14. Progress Bar - Инструмент за визуализация на процеси
15. Slider - – инструмент за визуализация на стойност
16. Hot Key – поле за г. ключ
17. List Control – полета за списъчен контрол
18. Tree Control - дървовиден контрол
19. Tab Control – таб. контрол
20. Animate - анимационен инструмент
21. Rich Edit - позволява да се добави към диалоговия прозорец контроли за редактиране с цел въвеждане на текст
22. Data Time Piker -
23. Month Calendar – месечен календар
24. IP Address – IP адрес
25. Custom Control – потребителски контрол
26. Extended Combo box – разширен Combo box

Методи за управление на диалози.

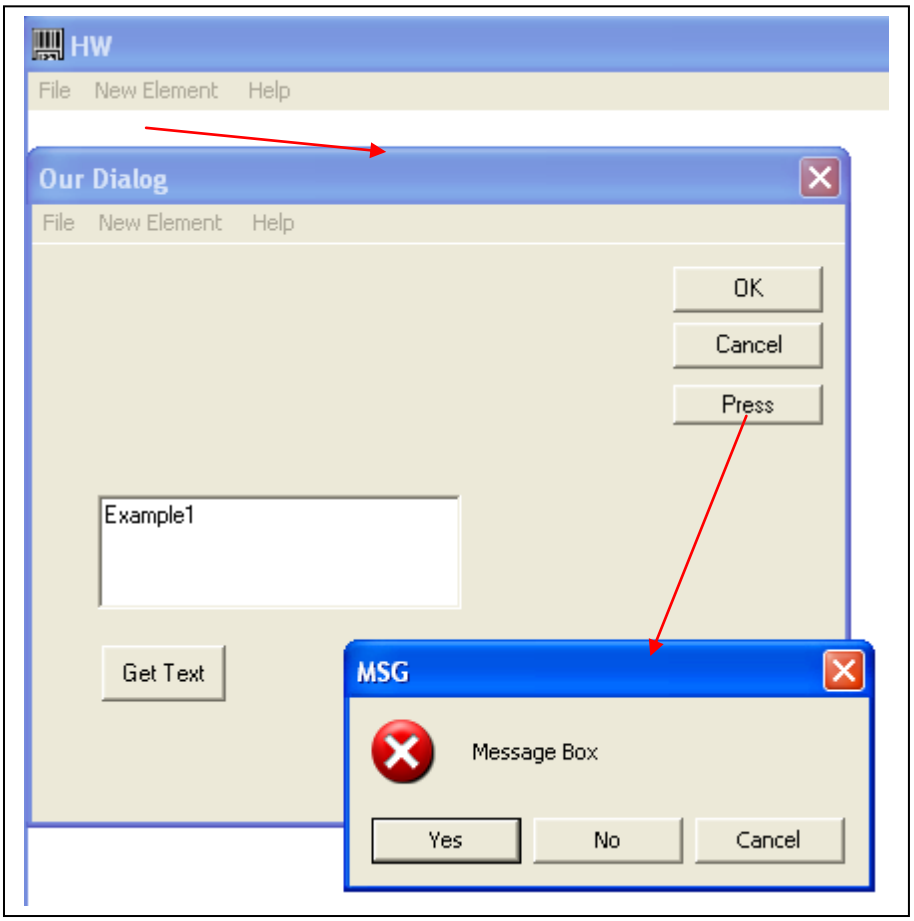
1. Активиране на диалог, указан с идентификатора на диалоговата му кутия и програмата, която го обработва, дъщерен прозорец на дадената инстанция.

DialogBox(**hInst,** MAKEINTRESOURCE(**IDD_ABOUTBOX**), **hWnd**, **About**);

2. Затваряне на диалога.

EndDialog(hDlg, LOWORD(wParam));

Пример:
... case IDM_NA:
DialogBox(hInst, (LPCTSTR)IDD_OUR, hWnd, (DLGPROC)FirstOurProg);
break;



3 Създаване на диалог от тип съобщение

int MessageBox(hDlg, "Message","Title of box", Buttons| Standard Windows)

- **Buttons :** {MB_OK, MB_OKCANCEL, MB_YESNO, MB_YESNOCANCEL, MB_RETRYCANCEL, MB_ICONEXCLAMATION }
- **Standard Windows:** { MB_ICONERROR,MB_ICONSTOP, MB_ICONINFORMATION, MB_ICONQUESTION }

for example - MessageBox(hDlg,"Message Box","MSG",MB_YESNOCANCEL | MB_ICONERROR));
MessageBox(hDlg,"No","MSG",MB_OK);
int x; x = MessageBox(hDlg, "Liub4oslav","Popivanoslav", MB_YESNOCANCEL | MB_ICONQUESTION);

Примери:

```
....      switch(MessageBox(hDlg,"Message Box","MSG",MB_YESNOCANCEL |
MB_ICONERROR ))
{
        case IDYES: MessageBox(hDlg,"Yes","MSG",MB_OK); break;
        case IDNO: MessageBox(hDlg,"No","MSG",MB_OK); break;
        case IDCANCEL: MessageBox(hDlg,"Cancel","MSG",MB_OK); break;
}
```

4. Четене на текст от едит контрол.

GetDlgItemText(hDlg, Control Identification ,text variable , text size)

Пример

```
 char buf[100];  GetDlgItemText(hDlg,IDC_EDIT1,buf,100);
```

5. Запис на текстова променлива или константа в едит контрол

SetDlgItemText(hDlg, Control Identification ,"some text")-

Пример - SetDlgItemText(hDlg, IDC_EDIT1,"Example1");

Общ пример Lab2_D1.cpp

```
#include "stdafx.h"
#include "Lab2_D1.h"
#define MAX_LOADSTRING 100
// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];        // the main window class name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    FirstOurProg(HWND, UINT, WPARAM, LPARAM);

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

```
	// TODO: Place code here.
	MSG msg;
	HACCEL hAccelTable;

	// Initialize global strings
	LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
	LoadString(hInstance, IDC_LAB2_D1, szWindowClass, MAX_LOADSTRING);
	MyRegisterClass(hInstance);

	// Perform application initialization:
	if (!InitInstance (hInstance, nCmdShow))
	{
		return FALSE;
	}
	hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LAB2_D1));
	// Main message loop:
	while (GetMessage(&msg, NULL, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return (int) msg.wParam;
}
//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;
	wcex.cbSize = sizeof(WNDCLASSEX);
	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon			= LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_LAB2_D1));
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= MAKEINTRESOURCE(IDC_LAB2_D1);
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance,
```

```
MAKEINTRESOURCE(IDI_SMALL));

        return RegisterClassEx(&wcex);
}
//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;
   hInst = hInstance; // Store instance handle in our global variable
   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);
   if (!hWnd)
   {
      return FALSE;
   }
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   return TRUE;
}
//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND - process the application menu
//  WM_PAINT        - Paint the main window
//  WM_DESTROY  - post a quit message and return/
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;

switch (message)
        {
case WM_COMMAND:
                wmId    = LOWORD(wParam);
                wmEvent = HIWORD(wParam);
                // Parse the menu selections:
                switch (wmId)
```

```
            {
        case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX),
            hWnd,     About);
                break;

        case IDM_NA:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_NEWDIALOG1), hWnd,
            FirstOurProg);
                break;

case IDM_EXIT:
        DestroyWindow(hWnd);
                    break;
default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
        case WM_PAINT:
                hdc = BeginPaint(hWnd, &ps);
                // TODO: Add any drawing code here...
                EndPaint(hWnd, &ps);
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam)
{
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
                return (INT_PTR)TRUE;

        case WM_COMMAND:
                if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                {
                        EndDialog(hDlg, LOWORD(wParam));
                        return (INT_PTR)TRUE;
                }
                break;
        }
        return (INT_PTR)FALSE;}
```

```
INT_PTR CALLBACK FirstOurProg(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
{
	UNREFERENCED_PARAMETER(lParam);
	switch (message)
	{
		case WM_INITDIALOG:
			SetDlgItemText(hDlg,IDC_EDIT1,"Example1");
				return TRUE;
		case WM_COMMAND:
			if (LOWORD(wParam) == IDOK || LOWORD(wParam) ==
IDCANCEL)
			{
				EndDialog(hDlg, LOWORD(wParam));
				return TRUE;
			}
			else if(LOWORD(wParam)== IDC_PRESS)
			{
			switch(MessageBox(hDlg,"Message Box","MSG",
		MB_YESNOCANCEL | MB_ICONERROR ))
				{
			case IDYES: MessageBox(hDlg,"Yes","MSG",MB_OK); break;
			case IDNO: MessageBox(hDlg,"No","MSG",MB_OK); break;
			case IDCANCEL: MessageBox(hDlg,"Cancel","MSG",MB_OK);
			break;}  }

			else if(LOWORD(wParam)== IDC_GET)
			{
				char buf[100];
				GetDlgItemText(hDlg,IDC_EDIT1,buf,100);
				MessageBox(hDlg, buf, "MSG",MB_OK);
				} break; }
   return FALSE;
}
```

Лекция 3

Особености на програмирането под Windows

Под Операционната система Win32 се има предвид 32-разрядните разновидности на ОС Windows. **Основното и различие** от предшествениците е въвеждането на механизма на ***„стесняващата се многозадачност***" и възможността за пряка адресация на ОП до 4 Г байта. „Стесняващата се многозадачност" означава, че ОС сама решава, на коя от програмите да предостави на разположение процесора. Всяка програма (приложение) след като поработи известно време автоматически се извежда от системата и управлението се предава на друга задача. Тази организация позволява да се създаде илюзията за едновременно изпълнение на няколко програми и не допуска ресурсите на компютъра да се ангажират от някоя сбойна задача. По този начин системата се предпазва от „***зависване***". **Втората особеност** на Win32 е ***32-разрядната адресация*** в пълното виртуално пространство, т.е. всяко приложение (програма на Win32), независимо от това колко приложения са стартирани може да се разполага с ***4 Г байта ОП***. При това отделните приложения не могат да повредят данните един на друг, т.к. всяко приложение отделя свое адресно пространство. Получавайки на свое разположение толкова голям ресурс от памет, програмистите могат да се чуствт комфортно. Освен отбелязаните особености, може да се отбележи и съществуването на :

- ***подръжка на ниво процеси и потоци;***
- ***възможност за синхронизация на потоците***;
- ***съществуването на файлове, които са проектируеми в паметта;***
- ***наличие на механизъм за обмен на данните между процесите***.

В Win32 ***процес*** се нарича всяка изпълняваща се програма. Един процес притежава своя памет (4 GB-байтово виртуално адресно пространство, файлови манипулатори и други системни ресурси. Ако една програма е стартирана два пъти (без да се спира изпълнението на първо-стартираната) са налични два процеса, работещи едновременно. Всеки процес има като минимум един ***поток*** на изпълнението. Терминът „***поток***" (**tread)** или „**нишка**" може да се определи като логическа посока на на изпълнение на програмата. Ако програмата се сравнява с река, то потоците могат да се сравняват с нейни ръкави. Тези потоци могат да текат паралелно. Например: когато програмата

трябва да изпълни някоя продължителна операция е целесъобразно тя да се отдели в отделен поток, като в същото време основният поток на програмата може да продължава да общува с потребителя. Всички потоци на един процес се изпълняват в едно адресно пространство и имат общ код, ресурси и глобални променливи. Пример за един процес, които има няколко главни прозореца, всеки от които се поддържа от своя собствена нишка е изпълнението на Wndows Explrer.

За да могат няколко потока на един процес да не си пречат при използването на общи ресурси се прилага ***синхронизация*** на потоците.Тя може да потрябва в случаите когато един поток трябва да изчака операция от друг поток или ако използват ресурс, които може да обслужи само един от тях. Тъй като потоците се изпълняват в стесняваща се многозадачност, функцията за тяхната синхронизация се изпълнява от операционната среда. За управление на потоците в Windows се използват специални флагове, на които е основано действието на няколко механизма на синхронизация: ***светофари (semaphores), изключващи светофари (mutex), събития (event), критически секции (critical section).***

Всеки процес има възможност за пряка индексация на ОП до **4 GB** , но много по-практично е програмите да работят с ***виртуални адреси***, а не с физически. За да може ОС да предостави голям обем памет в разположение на приложенията се използва специален ***механизъм на свързване*** на оперативната памет с твърдия диск. Този механизъм се нарича ***странична организация*** на паметта (paging). При тази организация логическото адресно пространство на всеки процес се разбива на отделни блокове, наричани ***страници.*** Страниците могат да се разполагат както в оперативнта памет така и на твърд диск. ОС оптимизира отделената ОП по такъв начин, че процесите да могат да получат бърз достъп до често използваните данни. ***Диспечерът*** на Виртуалната памет отбелязва отдавна неизползваните страници и ги изпраща на твъдия диск.

Видимо, подобен механизъм се използва и при създаването на файловете, проектируеми в паметта (memory-mapped files), наричани още отразени файлове. Проектируемият файл като че ли изобразява файл от диска в диапазона на адресите в паметта. Това се постига като се асоциира една адресна област директно с файла. Това позволява да се изпълняват операции

с файлове точно така, както ако работата се извършва в паметта. ОС сама организира обмена на данни между паметта и диска. Използването на файлове, проектируеми в паметта, позволява значително да се опрости работата с файлове, а така също дава възможност да се разделят данните между процесите. ***Разделението на данните*** става по следния начин: два или повече процеси създават в своята виртуална памет проекции на една и съща физическа област памет – обект „проектируем" файл. Когато един процес записва данни в своя „проекционен" файл, то измененията веднага се отразяват и в „проекциите", създадени в други процеси. За организацията на това взаимодействие всички процеси са длъжни да използват еднакво име за обекта „проектиран „ файл.

В Win32 всеки процес има ***свое адресно пространство***, затова едновременно изпълняваните приложения не могат случайно да повредят данните едно на друго. Обаче често възниква необходимост от обмен на информация между процесите. За тази цел в Win32 са предвидени ***специални способи***, позволяващи да се получи съвместен достъп към някакви данни. Всички те са основани на механизма на «проектирания» файл.

Един такъв механизъм се състои в създаването «Пощенска кутия» (mailslot) – специална структура в паметта, имитираща файл на твърд диск. Приложенията могат да поставят данните си в «кутия» и да ги четат от там. Когато всички приложения, работещи с кутията завършат, пощенската кутия се отделя от паметта.

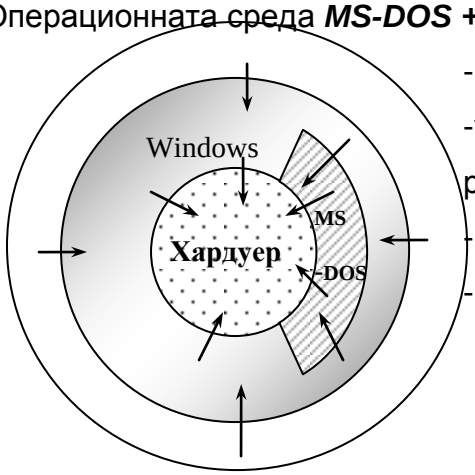
Друг такъв способ се заключава в организацията на комуникационна магистрала – «канал» (pipe), които свързва процесите. Приложение, имащо достъп до канала от единия му край, може да се свързва с приложение, което има достъп от другия му край и да му предаде данни. Каналът може да отделя на приложенията едностранен или двустранен достъп. При едностранния достъп едно приложение може да записва данните в даден канал, а друго да ги чете; при двустранния достъп и двете приложения могат да изпълняват операции четене/запис. Съществува възможност да се организира канал, които комутира едновременно няколко процеса.

Основните концепции в ОС Windows могат да се формулират като:

- ***преместваемост*** на изпълнимите приложения;
- ***обектна ориентация*** на програмирането на приложенията.

Основни Свойства на ОС

Операционната среда ***MS-DOS +Windows*** притежава следните ***свойства***:



-преносимост на приложенията;

-управление на оперативната памет и ресурсите;

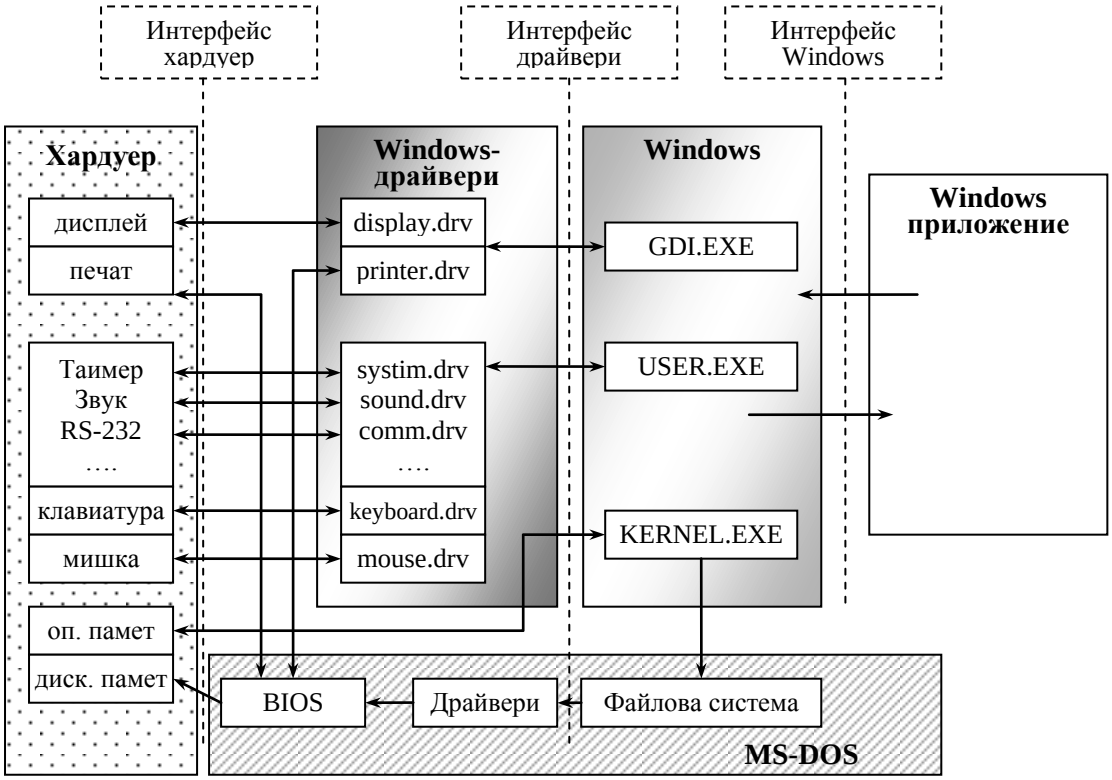
- едновременна работа на много приложения;

- много-прозоречен интерфейс.

Строеж на средата Windows

Фиг. 11. Общ план на операционната среда

Структура на средата, които основните взаимодействащи си компоненти са представени в общ план на Фиг. 11 и в по-подробен план на Фиг. 12 чрез схемите на връзките между компонентите на операционната среда.

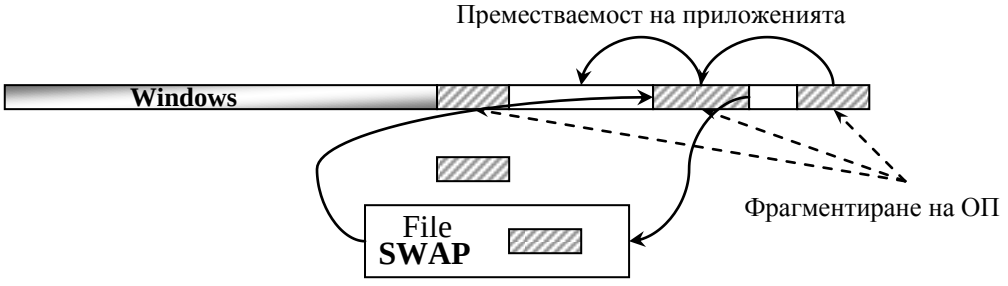


Фиг. 12. Детайлизирана схема на операционната среда

На детайлизираната схема са показани:

- Основните хардуерни устройства
- Използвани ресурси на MS- DOS:
  - Файлова система INT21;
  - Драйвери на файловата система на MS-DOS;
  - BIOS функция за управление на дискове и дискети;
  - BIOS функция за управление на печат (паралелен интерфейс).
- Множеството на Windows драйверите, специфични за всяко конкретно устройство;
- Основни функции на Windows. Ресурси от операционната система Windows:
  - GDI.EXE за управление на дисплей и принтер;
  - USER.EXE за управление на всички останали устройства, клавиатура и мишка;
  - KERNEL.EXE за управление на файлове и ОП.

Фиг. 13. представя примерна схема на разпределение на ОП и „преместваемостта" на приложенията. Показано е, че във фрагментирана ***Оперативната Памет*** са заредени един или няколко приложения. Изпълняваните приложения могат да се зареждат от различни начални адреси и те да се променят по време на изпълнението. (т.е. Приложенията са преместваеми в оперативната памет, като за целта се използва файл за временно съхранение swap „разменяем" файл. ОС Windows определя размера на swap-файла въз основа на наличната RAM памет и свободното дисково пространство.



Фиг. 3. Примерна схема на разпределение на ОП.

Методи за работа с менюта

1. Зареждане на описание на меню на зададен прозорец.

HMENU hMenu = GetMenu(hWnd);

GetMenu – връща в структура от тип ***HMENU*** описанието на менюто на зададения идентификатор на прозорец ***hWnd***;

2. Зареждане на ресурсното меню описание в структура

HMENU hMenu = LoadMenu(hInst, MAKEINTRESOURCE(IDR_POP));

Зарежда зададено със идентификатора си ресурс меню ***IDR_POP,*** в структура от тип ***HMENU.***

3. Извличане на подменю от структура

HMENU hSubMenu = GetSubMenu(hMenu, nPos); -

Извлича под-менюто от задаено описание ***hMenu*** според зададено ниво ***nPos***

HMENU hSubMenu = GetSubMenu(hMenu, 0); -

4. Показване меню за бърз достъп на определено място неговото активиране чрез проследяване избора на елементи на от него.

TrackPopupMenu(hMenu,Flags, x,y, Reserved, hWnd,prcRect);

Flags, – указва центрирането и вида на менюто или при кой натиснат бутон на мишката се появява, x, y – координати на горния ляв ъгъл, прозореца в който се появява, prcRect се игнорира.

TPM_LEFTBUTTON	The user can select menu items with only the left mouse button.
TPM_RIGHTBUTTON	The user can select menu items with both the left and right mouse buttons.

TrackPopupMenu(hSubMenu, TPM_RIGHTBUTTON, 500, 300, hWnd, NULL);

показва и активира указаното описание с начало (500,300) менюто при натискане на десен бутон.

5. Деактивиране на указано меню

DestroyMenu(hMenu);

6. Връщане на флаговете за свойства на отделни елементи от на менюто.

UINT GetMenuState(hMenu,Id, Flags);

В UINT променлива се връща съдържанието на флага за конкретен елемент, зададен с идентификатор ***ID*** от описанието на менюто в структурата ***hMenu***, и начина на задаване на идентификатор на елемент в менюто, зададен чрез ***Flags*** Стойности за Flags MF_BYPOSITION по подразбиране и MF_BYCOMMAND.

Възможни резултати:

MF_CHECKED, MF_UNCHECKED, MF_DISABLED, MF_ENABLED, MF_GRAYED, MF_SEPARATOR и др

Примери

UINT res = GetMenuState(hMenu, IDM_C1, MF_BYCOMMAND);

UINT res = GetMenuState(hMenu, IDM_DI1, MF_BYCOMMAND);

7. **Функции за промяна на свойствата на конкретен елемент от меню**

7.1. Промяна на свойството (Check/UnCheck) за чекнато листо или премахването на чекването

CheckMenuItem(hMenu, ID, Flags|Check)

Променя свойсвото на листо с ***ID*** от меню ***hMenu***, с указано с Параметър ***Check***.

Възможните тойности на Check са: MF_UNCHECKED, MF_CHECKED. Начинът на задаване на идентификатор на елемент в менюто е зададен чрез ***Flags*** Стойности за Flags MF_BYPOSITION по подразбиране и MF_BYCOMMAND.

Пример

CheckMenuItem(hMenu, IDM_CHECK, MF_BYCOMMAND **| MF_CHECKED**)

CheckMenuItem(hMenu, IDM_CHECK, MF_BYCOMMAND | **MF_UNCHECKED**);

7.2. Промяна на свойството (Enable/ Disable) за чекнато листо или премахването на чекването

EnableMenuItem (hMenu, ID, Flags|Enable)

Променя свойсвото на листо с ***ID*** от меню ***hMenu***, с указано с Параметър ***Check***.

Възможните тойности на Enable са: MF_ENABLED, MF_DISABLED, MF_GRAYED.

Начинът на задаване на елемент в менюто е задаен чрез ***Flags*** Стойности за Flags MF_BYPOSITION по подразбиране и MF_BYCOMMAND.

Примери

EnableMenuItem(hMenu, IDM_DISABLE, MF_BYCOMMAND | **MF_ENABLED**);

EnableMenuItem(hMenu, IDM_DISABLE, MF_BYCOMMAND | **MF_GRAYED**);

8 . Добавяне на нов елемент в указано меню

InsertMenuItem(hMenu,Item,ByPosition,MENUITEMINFO lpmii);

Към зададено меню ***hMenu,*** преди елемент с идентификатор ***Item*** се добавя елемент описан с ***MENUITEMINFO lpmii. ByPosition*** определя как се задава ***Item.*** Ако ByPosition=FALSE Item се задава с идентификатор ID, в противен случай със позиция в менюто.

При добавяне на нов елемент в меню се използва ***относително зададен*** идентификатор .

Проверка дали съществува, описан в Resource.h.

If (GetMenuState(hMenu, **IDM_ADD + 4**, MF_BYCOMMAND) == **-1**)

// **отсъства такъв номер в resource.h**

За добавянето се използва структура описваща свойствата на елемента в менюто от типа

```
MENUITEMINFO mii;
ZeroMemory(&mii, sizeof(mii));
mii.cbSize = sizeof(mii);
mii.fMask = MIIM_ID | MIIM_TYPE | MIIM_STATE;
mii.wID = IDM_ADD + 4;
mii.fType = MFT_STRING;   //
```

```
mii.dwTypeData = TEXT("Delete &me");    // надпис на листото
mii.fState = MFS_ENABLED;      // свойство достъпност
```

Пример

```
InsertMenuItem(hMenu, IDM_ADD, FALSE, &mii);
```

9. Премахване на елемент с указания идентификатор в зададено меню

DeleteMenu(hMenu, ID, Flags);

В меню **hMenu** се изтрива елемент с **ID.** Начинът на задаване на идентификатор на елемент в менюто е зададен чрез ***Flags*** Стойности за Flags MF_BYPOSITION по подразбиране и MF_BYCOMMAND.

Пример

```
HMENU hMenu = GetMenu(hWnd);
DeleteMenu(hMenu, IDM_D3, MF_BYCOMMAND);
```

Цялостен пример:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
int wmId, wmEvent;
PAINTSTRUCT ps;
HDC hdc;
LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
switch (message)
{
case WM_COMMAND:
        wmId    = LOWORD(wParam);
        wmEvent = HIWORD(wParam);
   switch (wmId)
        {
        case IDM_OPEN:
        MessageBox(hWnd, _T("Open"), _T("Info"), MB_OK | MB_ICONINFORMATION);
        break;
        case IDM_CHECK:
                HMENU hMenu = GetMenu(hWnd);
                UINT res = GetMenuState(hMenu, IDM_CHECK, MF_BYCOMMAND);
                if(res & MF_CHECKED)
                  CheckMenuItem(hMenu, IDM_CHECK, MF_BYCOMMAND |
MF_UNCHECKED);
```

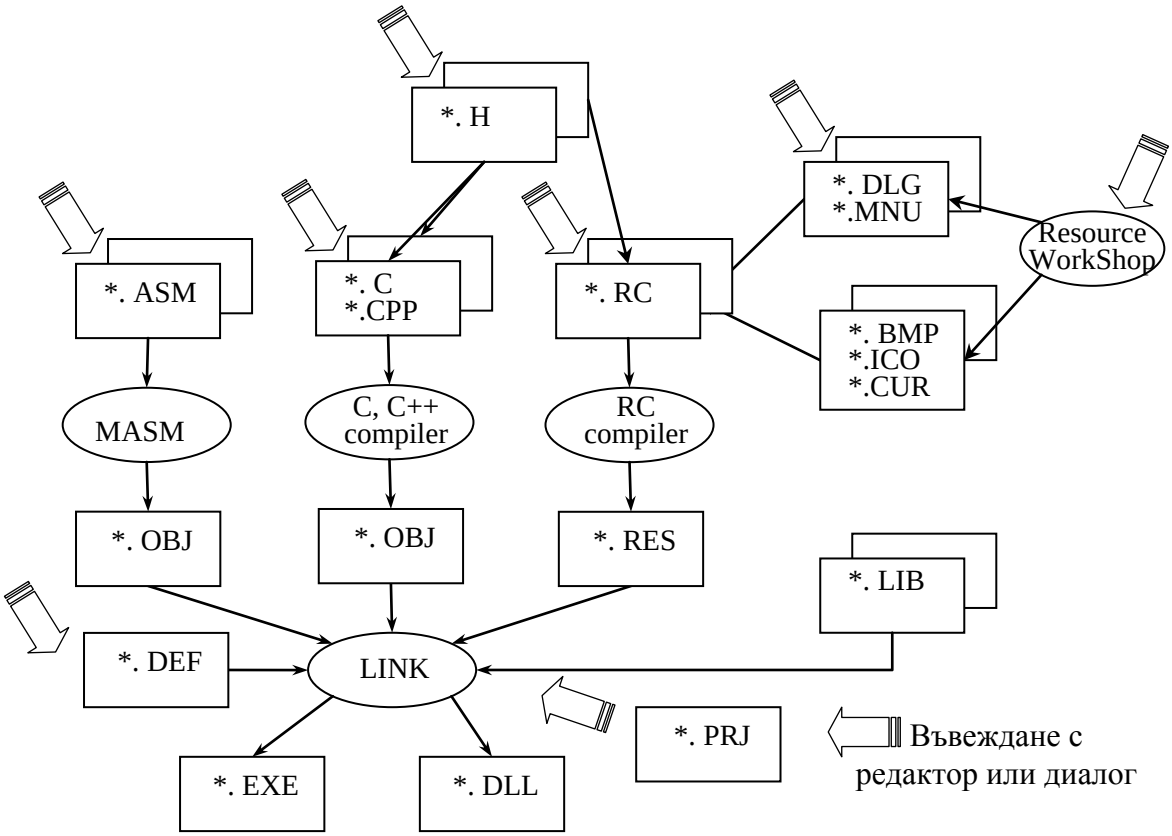
```
                else
                  CheckMenuItem(hMenu, IDM_CHECK, MF_BYCOMMAND |
MF_CHECKED);
                break;
        case IDM_DISABLE:
                HMENU hMenu = GetMenu(hWnd);
                EnableMenuItem(hMenu, IDM_DISABLE, MF_BYCOMMAND |
MF_GRAYED);
                break;
        case IDM_ENABLE:
                HMENU hMenu = GetMenu(hWnd);
                res = GetMenuState(hMenu, IDM_DISABLE, MF_BYCOMMAND);
                if(res & MF_GRAYED)
                  EnableMenuItem(hMenu, IDM_DISABLE, MF_BYCOMMAND |
MF_ENABLED);
                else
                  EnableMenuItem(hMenu, IDM_DISABLE, MF_BYCOMMAND |
MF_GRAYED);
                break;
        case IDM_ADD:
                HMENU hMenu = GetMenu(hWnd);
                if(GetMenuState(hMenu, IDM_ADD + 1, MF_BYCOMMAND) == -1) // or
disable
                        {
                        MENUITEMINFO mii;
                        ZeroMemory(&mii, sizeof(mii));
                        mii.cbSize = sizeof(mii);
                        mii.fMask = MIIM_ID | MIIM_TYPE | MIIM_STATE;
                        mii.wID = IDM_ADD + 1;
                        mii.fType = MFT_STRING;
                        mii.dwTypeData = TEXT("Delete &me");
                        mii.fState = MFS_ENABLED;
                        InsertMenuItem(hMenu, IDM_ADD, FALSE, &mii);
                        }
                break;
        case IDM_ADD + 1:
                HMENU hMenu = GetMenu(hWnd);
                DeleteMenu(hMenu, IDM_ADD + 1, MF_BYCOMMAND);
                break;
        case IDM_DO:
                MessageBox(hWnd, _T("Do Something else"), _T("Title Message"), MB_OK);
                break;
        case IDM_ABOUT:
                DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                break;
        case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
                        break;
case WM_RBUTTONDOWN:
                HMENU hMenu = LoadMenu(hInst, MAKEINTRESOURCE(IDR_POP));
                HMENU hSubMenu = GetSubMenu(hMenu, 0);
```

```
                POINT pt = {LOWORD(lParam), HIWORD(lParam)};
                ClientToScreen(hWnd, &pt);
                TrackPopupMenu(hSubMenu, TPM_RIGHTBUTTON, pt.x, pt.y, 0, hWnd,
NULL);
                DestroyMenu(hMenu);
                        break;
case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        RECT rt;
        GetClientRect(hWnd, &rt);
        EndPaint(hWnd, &ps);
        break;
case WM_DESTROY:
        PostQuitMessage(0);
        break;
default:
        return DefWindowProc(hWnd, message, wParam, lParam);
 }
  return 0;
}
```

Лекция 4

Техника за създаване на приложение

Схемата представяща процеса на създаване на изпълнимо приложение е представена на Фиг. 21.



Фиг. 21. Схема за създаване на приложения.

В схемата са включени следните файлове:

*.ASM - програмен модул написан на Асемблер

.C (.CPP) - програмен модул написан на C или C++

*.H - файлове за включване (Windows.h)

*.RC – описание на ресурсите

*.DLG, *.MNU – Ресурси за диалогови кутии, менюта

.BMP,.ICO,*.CUR Ресурси за Bitmaps, икони, курсори

*.OBJ - обектни модули

*.RES - обектни модули за ресурсите

*.LIB – библиотечен модул (LIBW, LIBCEW)

*.EXE – WINDOWS -приложение

*.DLL – динамична библиотека

*.PRJ, *.DSP – проектен файл

*.DEF – описател на модула, за да се предаде на свързващия редактор linker

Тези файлове се организират в проекти *.sln, които са част от ***работно пространство*** *.DSW.

Алгоритъм за създаване на WINDOWS приложение

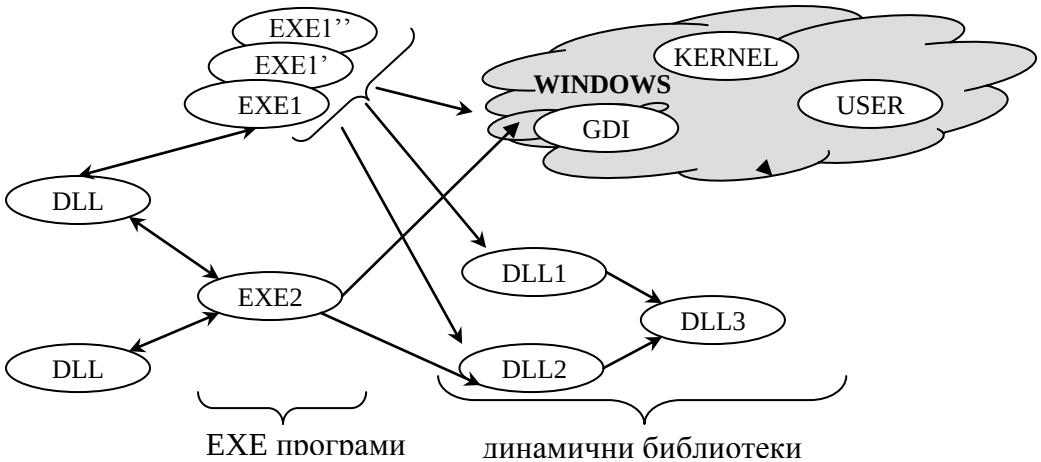
1. Написване и въвеждане на функциите : “WinMain” и “WndProc” на езиците Сили C++ и изнасяне на дефинициите в файлове *.H.
2. Създаване на ресурси (*.DLG, *MNU, *.BMP, *.CUR, *.ICO) чрез Resource Workshop и включването им в файловете *.RC.
3. Написване и въвеждане на файла за описание на модула *.def
4. Компилация на всички програмни модули (*. C, *.CPP)
5. Компилация на ресурсите *.RC.
6. Свързване на обектните модули *.OBJ , ресурсите *.RC и библиотеките *.LIB в изпълним модул (*.EXE или *.DLL)

Файл за описание на модула *.DEF

```
NAME            sample                  ; име на приложението
DESCRIPTION     ‘Wndows Application’    ; текстово описание
EXETYPE         WINDOWS                 ; тип на EXE
STUB            ‘Winstub.exe’           ; име на програма, която се стартира в MS-DOS
CADE            MOVEABLE                ; премества сегмент
DATA MOVEABLE MULTIPLE                  ; премества сегмент за многократно използване
HEAPSIZE        1024                    ; размер на динамичната област
STACUSIZE       4096                    ; размер на стека
EXPORT          WndProc@1
                About@2
```

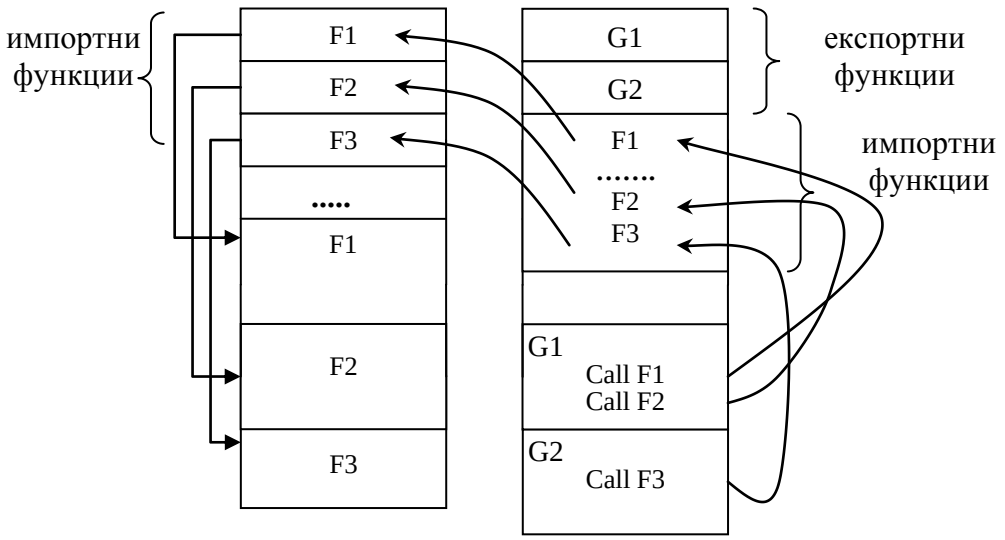
Средства за връзка между приложенията в средата на Windows.

Взаимодействието на програмите в средата на Windows е илюстрирано с примера от на Фиг 22. Показана е една примерна схема, включваща две приложения, едното от които е сегментирано . По време на изпълнението си приложенията взаимодействат с ОС и динамичните библиотеки. Със стрелки са означени входните и или изходните връзки.



Фиг. 22. Взаимодействие на програмите при изпълнение

Илюстрация на взаимодействието на изпълнимите модули чрез експортни и импортни функции е представени на схемата от Фиг 23.



Фиг. 23. Механизъм на взаимодействие между изпълнимите модули

Динамични библиотеки Dynamic Link Library (DLL)

Динамичната библиотека е модул за работа в средата на Windows, който експортира библиотечни функции като:

- библиотечните функции са достъпни за всяко активно приложение и може да се използват паралелно (В случай на статични данни);
- достъпът (връзка) към експортните функции на модула се реализира в ***реално време*** при обръщение към функциите;
- тялото на библиотеката не е част от приложенията, а е обикновено общо за повече приложения;
- името и поредния номер на библиотечните функции на самата среда са записани в *LIBW.LIB (Напр. GetMessage има номер 108);
- при потребителски DLL модули се създава също XXX .LIB с името и номера на функциите.

Събития – съобщения

Съобщението е кодирано като уникално 32-битово цяло значение. Във файла WINDOWS.H за всички съобщения са определени стандартни имена като: WM_COMMAND, WM_INITDIALOG, WM_DESTROY WM_CHAR, WM_PAINT, WM_MOVE, WM_SIZE, WMLBUTTONDOWN, WM_CREATE, WM_CLOSE и др. Често съобщенията се съпровождат с параметри носещи допълнителна информация (координати на курсора, код на натиснатия клавиш и др.) Всеки път когато произтича събитието, касаещо програмата ОС и изпраща съответното съобщение. Не е нужно да се описва реакцията на всяко съобщение. За съобщенията, които няма специална обработка, в MFC – програма се обработват по стандартен начин.

Структура на съобщенията:

32бита – код (хендъл) на обект; 32 бита код на съобщението; 32 бита данни 1; 32 бита данни 2. Пример ≡object1.method2(d1,d2)

Кодът (хендълът) на ***обекта – прозорец*** всъщност е един номер или индекс от 0 до 2^{32}, който идентифицира всеки обект. ***Кодът е еквивалентен номер на зает в оперативната памет блок за обекта.*** Това в същност е референция към обект (Манипулатор на обект). Кодът на съобщението (номера на съобщението) съответства на вътрешна функция метод. Данните

32б. и 32б. са съпровождащи съобщението данни.

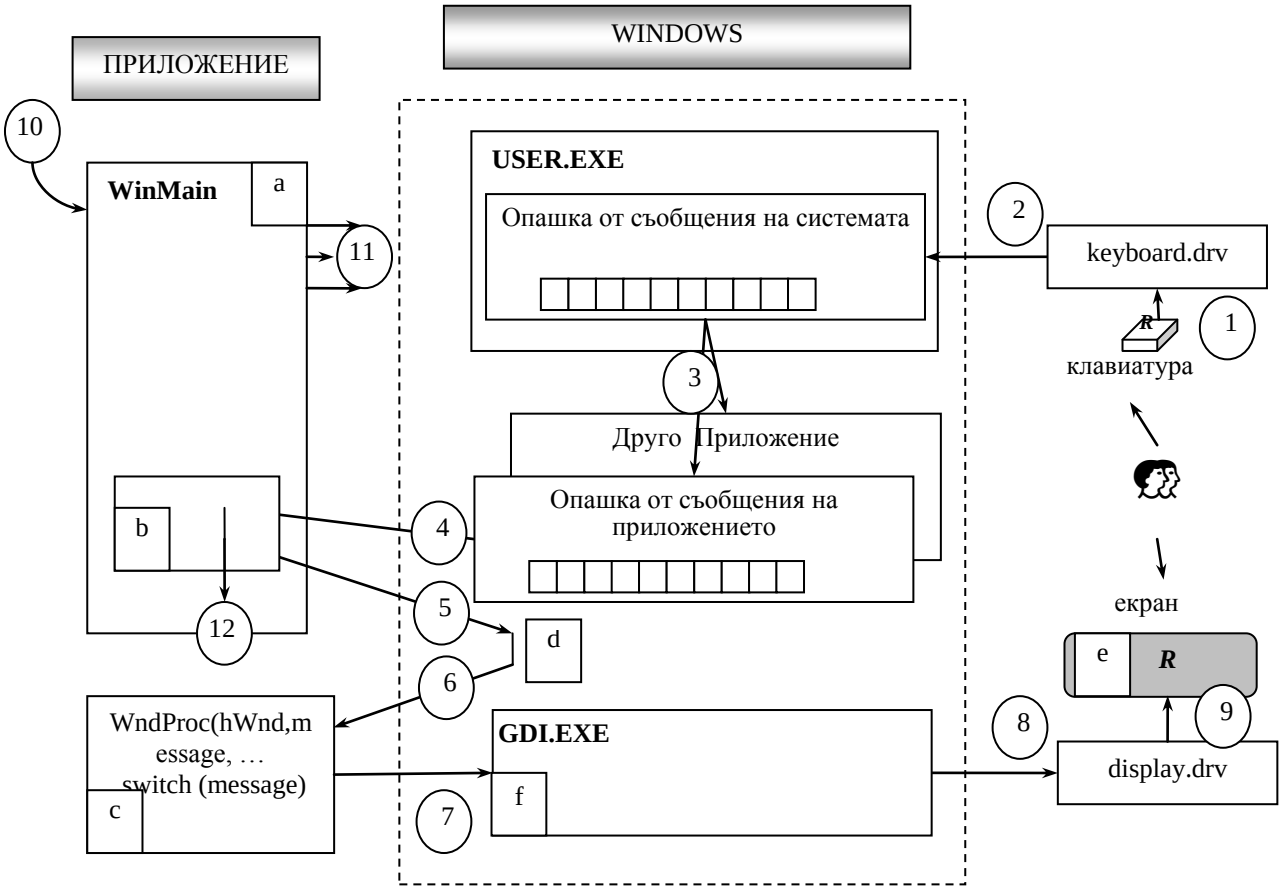
Пример: Ако натиснем ляв бутон на мишката се формира следното съобщение: получател:прозоречен обект (активния прозорец); код на съобщението: WM_LBUTTONDOWN-това е ***32б. число***, данни 1: кодиране от клавиатурата; данни2: старши 16б. X, младша 16б. Y координати на мишката. (Координатите могат да се заредят в променливите x, y чрез : int x = LOWORD(lParam); int y = HIWORD(lParam);

Главната обслужваща функция за всеки обект има един и същ формат:

long WinApi WndProc(hWnd, UINT message, long wParam, long lParam)

Обмен на съобщения. Обработка и изпращане на съобщения. Синхронизация. Диспечер на съобщенията.

Фиг16. представя последователността на обработка на съобщения чрез примера за събитието (съобщението) „натискане на клавиш от клавиатурата”.



Фиг. 16. Последователност на обмен на съобщенията

При изобразяване на обработката на съобщенията са използвани означения с букви заградени в квадрат на главните компоненти от едно приложение и модулите на ОС, а обработките (обмена) са представени със стрелки като последователността е означена с поредни номера заграден в кръг.

Означения на основните компоненти:

a – главна програма на приложението;

b – цикъл на обработка - извличане на съобщения WM_CHAR ;

c – главна функция на прозореца на приложението;

d – WM_CHAR съобщение от драйвера на клавиатурата към главната функция

e – прозорец на приложението (активен прозорец – фокус);

f – извикване на функцията TextOut (GDI – функция).

Обработки:

номера от 1 до 9 представят *основния цикъл на обработката на съобщенията* WM_CHAR*:*

1 – обръщение от клавиатурата към драйвера на клавиатурата;

2 – съобщение (Получател=WM_CHAR на активния прозорец(WM_CHAR – съобщения от клавиатура));

3 – разпределяне на опашките;

4, 5 – извличане на съобщенията WM_CHAR;

6 – Windows извиква WndProc;

7- WndProc извежда текст чрез TextOut;

8 – обръщение към драйвера на дисплея;

9 – от драйвера на дисплея към екрана;

10 – **стартиране** на приложение – Управлението се предава на WinMain;

11 – Функции към Windows (RegisterClass, CreateWindow, …) за начално установяване;

12 – **завършване** на приложението при получаване на съобщението WM_QUIT

Обработка на съобщенията (събитията) от клавиатура, таймер и мишка.

Управление на клавиатурата

Драйверът на клавиатурата е **keyboard.drv, kbdr.drv.**

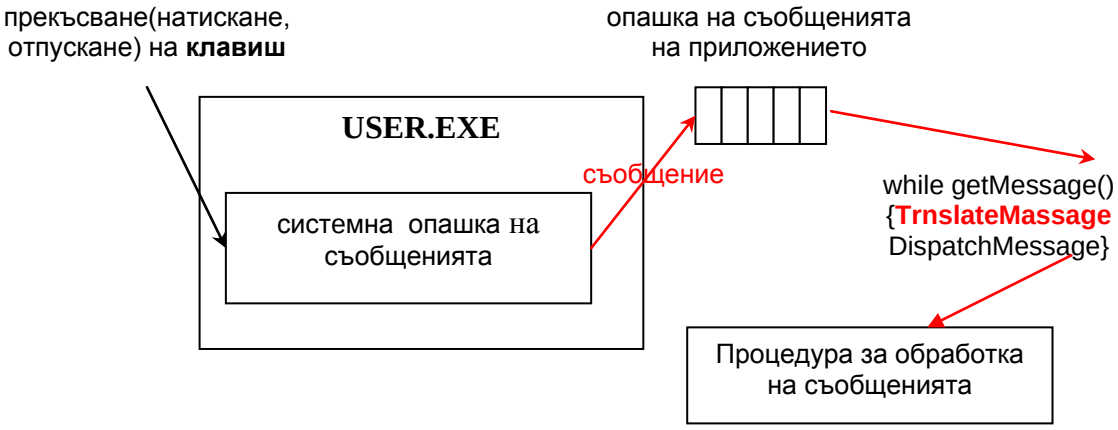
Експортни функции : enable, disable – ToAScii, AnsiToOlm, OemToAnsi.

Общо съществуват 8 съобщения за събития в клавиатурата. Не е нужно да се реагира на всичките. Основното съобщение е WM_COMMAND, което при

работа с диалогови елементи съдържа крайния резултат (например избраната опция от менюто).

Фокус за въвеждане

Клавиатурата е разделяем ресурс и се управлява от фокуса на приложението. Съобщение от клавиатурата получава онова приложение, което има в момента фокуса (или чиито прозорец има в момента фокуса). Активен е прозореца, който има фокуса или ако дъщерния му прозорец има фокуса. Обработваща функция на прозореца получава съобщения WM_SETFOCUS/WM_KILLFOKUS съответно получава или губи фокуса.



Фиг. 29. Схема на процеса на обработка на съобщението

Съобщения от клавиатурата са:

- Системни и несистемни клавиши, системни (ALT_TAB, ALT_ESC).
- Съответните съобщения за натискане на системни клавиши са :WM_SYSKEYDOWN, WM_SYSKEYUP, и несистемни клавиши са: WM_KEYDOWN, WM_KEYUP.
- Информация за натиснатия клавиш е в lParam кодиране на символа.
- Виртуален клавишен код: в lParam се кодира символа . Имената на виртуалните клавиши са в WINDOWS.h , напр. VK_RETURN, VK_HOME, VK_0, VK_A,…..и т.н.
- За предпочитане да не се работи директно с ASCII таблица а с виртуални клавишни кодове.

Типове на данните в Windows

В Windows – програмите се използват типове данни и структури, определени в библиотечни файлове (напр. Window.h). Често използвани типове:

- **HANDLE –** 32 разрядно, цяло число в качеството на дескриптор (числов идентификатор на някои ресурс);
- **HWND – 32** разрядно цяло число, дескриптор на прозорец;
- BYTE – 8 разрядно без-знаково символно значение;
- **WORD** – 16 разрядно, без-знаково цяло число;
- **DWORD, UINT** - 32 разрядно без-знаково цяло число;
- **LONG - 32** разрядно без-знаково цяло число;
- **BOOL** – цяло число, използвано за обозначаване на истина (1True) или лъжа (0 False)
- **LPSTR** -32 разряден указател на ред символи.

Освен тези съществуват още много други типове, обозначаващи дескриптори на различни ресурси, указатели, структури. Различията между тях се състоят главно в това какво обозначават. В програмите могат да се използват и всички традиционни типове данни.

Унгарски запис

Съществува конвенция за включване типа на променливите като префикс към тяхното име. Тя е предложена от Чарлз Симонай (унгарец). Тази конвенция позволява по-добър контрол на типовете на променливите.

Унгарският запис е представен в таблицата:

Prefix	Type	Description	Example
c	char	8 bits	cSymbol
by	BYTE	Unsigned char	byCode
n	short, int	16 bits	nCounter
x	short	X co-ordinate	xParam
y	short	Y co-ordinate	yParam
i	int	16 bits	iCount
b	BOOL	Int	bFlag
w	WORD	Unsigned integer	wParam
h	HANDLE	Unsigned integer	hInstance

l	LONG	32 bits	lParam
dw	DWORD	Unsigned long	dwBufsize
fn	function	function	fnWndProc
s		string	sName
sz		String+0	szName
p	*	index	pP1
lp	for *	Long index	lpPtr
np	Near *	Short index	npStr

Методи за управление на диалогови контроли

Работа с числови данни в диалоговите елементи

1. ***Четене на цяло число*** въведено от контрола edit box

BOOL *LPt=NULL; BOOL SIG=TRUE;

int GetDlgItemInt(hDlg, edit box identification ,LPt, SIG) – връща цяло число, което се въвежда в контрола;

Пример: BOOL *LPt=NULL; BOOL SIG=TRUE;

g1=GetDlgItemInt(hDlg,IDC_EDIT1,LPt, SIG);

2. Записване на цяло число в контрола edit box

SetDlgItemInt(hDlg, edit box identification, INumber,TRUE);

Примери: SetDlgItemInt(hDlg,IDC_EDIT4, g1,TRUE);

Преобразуващи функции:да се добави

получаване на реално число, което е въведено като текст в edit box:

char *stopstring; **double** variable=strtod(**char var**[10],&stopstring);

float val=atof(“10.2”); int val1=atoi(“777”); long val2=atol(“989”);

получаване на символен низ, който е получен от реална стойност.

float d; char txt[10]; **sprintf(txt, “%10.2f”,d);**]; **sprintf_s(txt, “%10.2f”,d);/**

gcvt(d,10,Text);

Примери:

```
char t1[10];   GetDlgItemText(hDlg,IDC_EDIT1,t1,10);
x1=strtod(t1,&stopstring);  gcvt(x1,10,t11);

MessageBox(hDlg, t11,"OK", MB_OK);
SetDlgItemText(hDlg,IDC_EDIT3,t11);
```

3. Изпращане на съобщения към контрол

Слеващата функция ***изпраща съобщения към различните видове диалогови елементи***, което се изразява в различни действия, в зависимост от типа на елемента и параметрите които и се подават. Общия и формат изглежда така:

int SendDlgItemMessage(hDlg,ID_of_DialogControl, Processing_Way, Index,(LPARAM) "Some text in Dialog Control")

Функцията връща номера на реда в списъчните контроли към които тя изпраща съобщение. Ако стойността е по-малка от 0 то контролът е празен.

Параметри:

- **hDlg** –описател на обекта;
- **ID_of_DialogControl** – идентификатор на диалоговия елемент;
- **Processing_Way** – Параметър, които казва как се изпраща съобщението. Зависи от типа на диалоговия елемент по следния начин:
  - за **list box** възможните стандартни константи за този параметър започват с **LB: LB_GETTEXT, LB_SETTEXT, LB_RESETCONTENT, LB_INSERTSTRING, LB_SELECTSTRING, LB_FINDSTRING, LB_GETCURSEL, LB_SETCURSEL**, **LB_DELETESTRING**.

LB_GETTEXT – за четене на текст от контрола с указан номер на реда
LB_SETTEXT – за запис на текст в контрола с указан номер на реда
LB_RESETCONTENT – за изпразване от текст на контрола
LB_INSERTSTRING – за вмъкване на текст
LB_ADDSTRING – за добавяне на текст
LB_SELECTSTRING – за избор на ред от лист контрола
LB_FINDSTRING – за търсене на текст в контрола
LB_GETCURSEL – за връщане на номера на реда в контрола, в който е курсора
LB_SETCURSEL – за установяване на курсора на указан ред в контрола
LB_DELETESTRING – за изтриване на ред в контрола

- за **combo box** стандартни константи започват с възможните стандартни константи за този параметър започват с **CB: CB_SETTEXT, LB_ISERTSTRING, CB_ADDSTRING, CB_SELECTSTRING, CB_GETCURSEL, CB_SETCURSEL, CB_GETLBTEXT, CB_FINDSTRING, CB_DELETESTRING**

CB_GETLBTEXT – за четене на текст от контрола с указан номер на реда
CB_SETTEXT – за запис на текст в контрола с указан номер на реда
CB_RESETCONTENT – за изпразване от текст на контрола
CB_INSERTSTRING – за вмъкване на текст
CB_ADDSTRING – за добавяне на текст
CB_SELECTSTRING – за избор на ред от лист контрола
CB_FINDSTRING – за търсене на текст в контрола
CB_GETCURSEL – за връщане на номера на реда в контрола, в който е курсора
CB_SETCURSEL – за установяване на курсора на указан ред в контрола
CB_DELETESTRING – за изтриване на ред в контрола

- за **radio and check boxes** възможните стандартни константи за този параметър започват с **BM: BM_SETCHECK, BM_GETCHECK**;

BM_SETCHECK – за поставяне на отметка
BM_GETCHECK – за връщане на резултат за отметка

- за **edit box** възможните стандартни константи за този параметър започват с **WM: WM_GETTEXT, WM_SETTEXT**.

WM_GETTEXT – за четене на текст от контрола
WM_SETTEXT. – за запис на текст в контрола

Пример - SendDlgItemMessage(hDlg,IDC_LIST1,LB_GETTEXT,lbIndex,(LPARAM)data);

- Параметър Index :

a/ Указва позицията на изпращания текст в случай на list или combo boxes.

b/ Установява отметка или премахва в случай на radio and check boxes, използвайки константите : BST_CHECKED, BST_UNCHECKED.

c/ Установява дължината на текста в случай на edit box като напр. (LPARAM) "Some text in Dialog Control" –за изпращания текст"Some text in Dialog Control" .

Примери:

// combo box

string val[10]=”{“text1”, “text2”, “text3”, text4”}

for (int i=0; i<4; i++)

c_index=SendDlgItemMessage(hDlg,IDC_COMBO1,**CB_ADDSTRING**,0,(LPARAM)

;val[i].c_str());

SendDlgItemMessage(hDlg,IDC_COMBO1,**CB_RESETCONTENT**,0,-1);

i=SendDlgItemMessage(hDlg,IDC_COMBO1,**CB_FINDSTRING**,0,(LPARAM)**data**)

if (i>-1))

{….// текста data е намерен в к.б. и номерът му е **i**…} else (…// data липсва в к.б.}

cbIndex=SendDlgItemMessage(hDlg,IDC_COMBO1,**CB_GETCURSEL**,0,0);

SendDlgItemMessage(hDlg,IDC_COMBO1,**CB_GETLBTEXT**,cbIndex,(LPARAM)data);

_____

```
case WM_COMMAND:
                if(mID==IDCANCEL)
                        EndDialog(hDlg,mID);
        else if(mID==IDC_BUTTON3)
                {
        if(!(IsDlgButtonChecked(hDlg,IDC_CHECK1)+IsDlgButtonChecked(hDlg,IDC_CHEC
K2)))
                        MessageBox(hDlg,"no checked added","asd",MB_OK);
                        else
                        GetDlgItemText(hDlg,IDC_ToAdd,data,sizeof(data));

        if(IsDlgButtonChecked(hDlg,IDC_CHECK1)&&!(SendDlgItemMessage(hDlg,IDC_CO
MBO1,CB_FINDSTRING,0,(LPARAM)data)>-1))
                        {
```

```
        c_index=SendDlgItemMessage(hDlg,IDC_COMBO1,CB_ADDSTRING,0,(LPARAM)d
ata);
                SendDlgItemMessage(hDlg,IDC_COMBO1,CB_SETCURSEL,c_index,0);
        MessageBox(hDlg,"added in combo box","asd",MB_OK);
                        }

// list box
l_index=SendDlgItemMessage(hDlg,IDC_LIST1,LB_ADDSTRING,0,(LPARAM)"List
row 1");
SendDlgItemMessage(hDlg,IDC_COMBO1,CB_SELECTSTRING,-1,(LPARAM)
"Row 2 in Combo Box");
SendDlgItemMessage(hDlg,IDC_LIST1,LB_SETCURSEL,0,0);
I= SendDlgItemMessage(hDlg,IDC_LIST1,LB_FINDSTRING,0,(LPARAM)data);
SendDlgItemMessage(hDlg,IDC_LIST1,LB_GETTEXT,lbIndex,(LPARAM)data);
// check box
I= SendDlgItemMessage(hDlg,IDC_CHECK1, BM_GETCHECK,0,0);
```

4. ***Проверка и смяна статуса на елемента като отметнат или не.***

4.1.Получаване на стойност съобразно това дали елемента е отметната или не

IsDlgButtonChecked(hDlg, ID)

връща лог. стойност за указания с ID check box или radio box. в зависимост от това дали е избран или не.

Пример: IsDlgButtonChecked(hDlg, ID_ZNAK)

4.2. *Смяна статуса на елемента като отметнат или не*

ChechDlgButton(hDlg, ID, (BST_CHECKED or BST_UNCHECKED))

Функцията сменя статуса на указания в ID контрол в указания като трети параметър.

Функцията освен това връща състоянието преди функцията да бъде извикана. Ако не върне 0 функцията е била недостъпна преди извикването. Ако върне 0 е била достъпна.

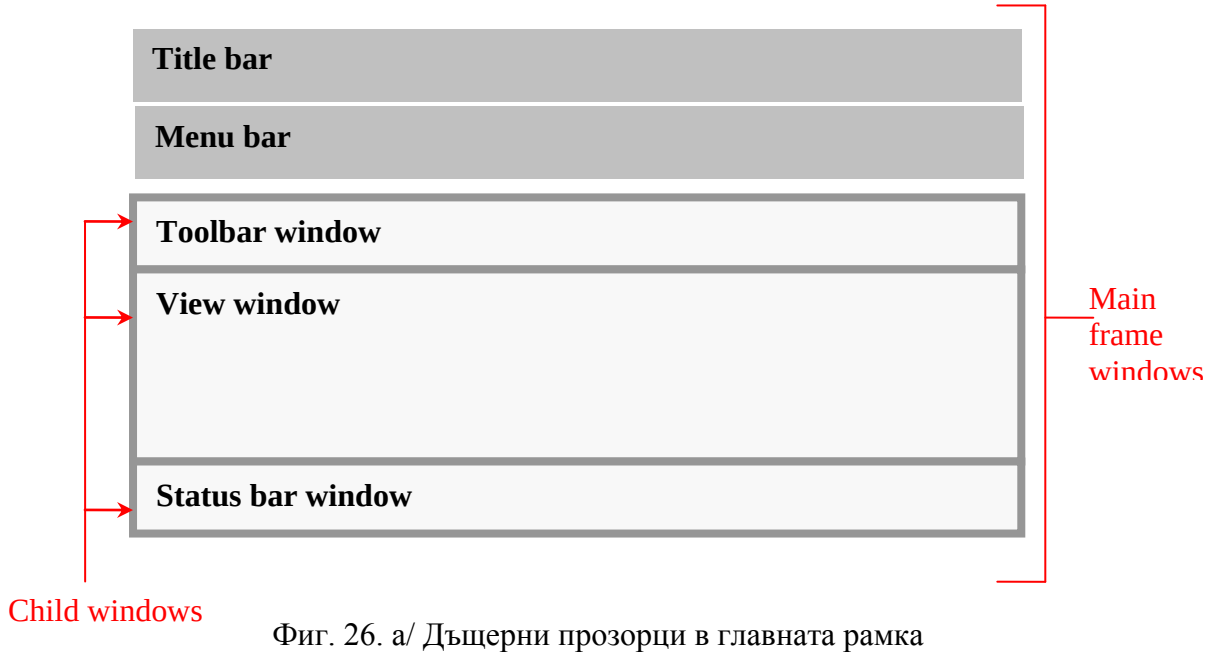
Пример: ChechDlgButton(hDlg, ID_C, BST_CHECKED

Лекция 5

Управление на прозорци в средата на Windows

Типове прозорци

В едно приложение в ***прозорецът на главната рамка*** (main frame window) се намира ***прозореца на изгледа***. Прозорецът на главната рамка съдържа заглавието и менюто. Различните дъщерни прозорци, между които тези на панела с инструменти, прозорецът на изгледа и прозореца на лентата за статус, заемат клиентската област на прозореца на рамката, както е показано на Фиг. 26. а/ . Приложната среда контролира взаимодействието между рамката и изгледа, като пренасочва съобщения от нея към него.

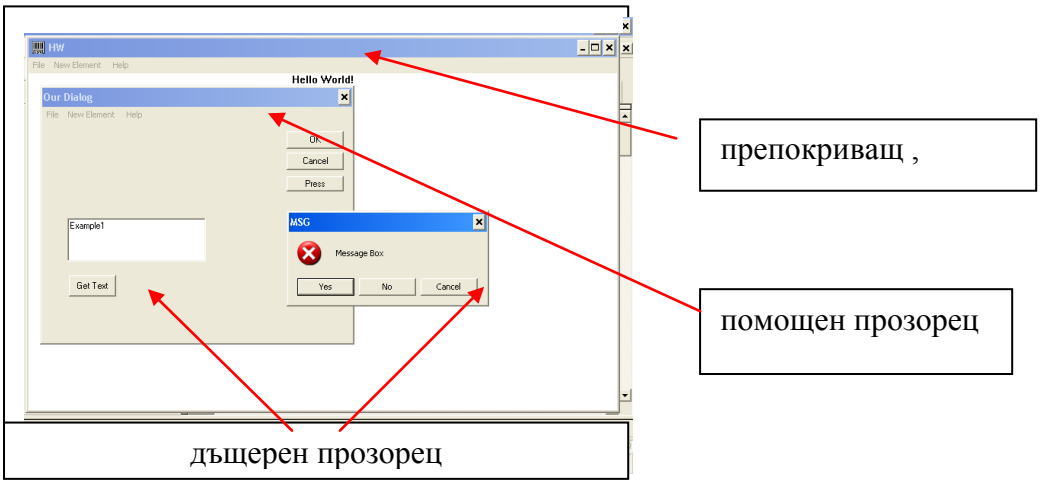


Фиг. 26. а/ Дъщерни прозорци в главната рамка

Задава се в заглавния параметър на CREATE_WINDOW с 32-бита определител описан в WINDOWS.H. Фиг. 26 б/ представя примери на типовете прозорци.

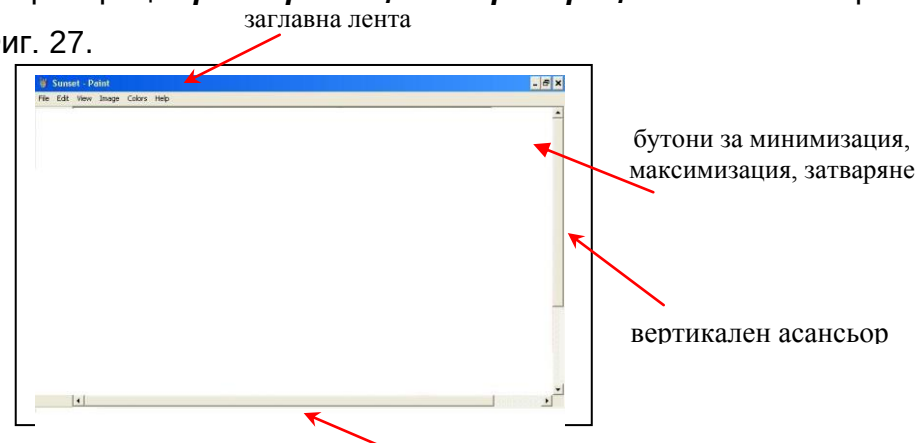
Съществуват 3-типа прозорци:

- ***WS_OVERLAPED*** : основен тип, ***препокриващ*** прозорец прикрива другите, ако е активен (т.е. получил фокуса). Такъв е прозорецът на главната рамка;
- ***WS_POPUP*** – помощен прозорец, („изскачащ” , ***продължаващ***, ***междинен)*** прозорец, изобразява се винаги най-отгоре, използва се като диалогов прозорец, панелен прозорец;
- ***WS_CHILD*** – ***дъщерен*** прозорец, подчинен на родителския прозорец.



Фиг.26. Илюстрация на различни видове прозорци

Елементи на прозорец ***препокриващия прозорец*** на главната рамка са показани на Фиг. 27.



Фиг. 27. Схема на основен препокриващ прозорец

Свойства на прозорците

Те се задават в 32-битов определител, който е дефиниран в WINDOWS.H. WS_SYSMENU|WS_VSCROLL|.

Разпределение на битовете в определителя:

- Битовете 16÷21 (WS_MAXIMIZEROX, WS_MINIMIZEBOX, WS_THICKFRAME, WS_SYSMENU, WS_HSCROLL, WS_VSCROLL) описват елементите на прозореца;
- Битовете 22÷-23 (WS_DLGFRAME, WS_BORDER, WS_CAPTION) – дефинират вида на рамката – тънка, дебела рамка и заглавие (00, 01, 10, 11);

- Бит 24 WS_MAXIMIZE максимално изобразяване;
- Битове 25, 26 (WS_CLIPCHILDREN, WS_CLIPSIBLINGS) – обработка на дъщерни прозорци;
- Бит 27 (WS_DISABLED) – неактивен прозорец;
- Битовете 28,29 (WS_VISIBLE, WS_MINIMAZE)-видим прозорец и икона ;
- Битовете 30, 31 (WS_OVERLAPED, WS_CHILD, WS_POPUP) - определят типа на прозореца.

Помощни прозорци (WS_POPUP).

Използва се за временно извеждане на информация или временен диалог (съдържа съобщения, диалог, бутони “Yes”, “No” и други елементи).

Особености:

- всеки прозорец има собствена ***главна обработваща функция***;
- имат заглавие на прозореца и рамка;
- обикновено имат фиксиран размер, но не е задължително;
- може да имат или не родителски прозорец. Ако нямат са самостоятелни препокриващи прозорци, а ако имат това не означава, че са дъщерни прозорци. Ако родителския прозорец се минимизира, помощният прозорец остава винаги на върха на изображенията.

Дъщерни прозорци (WS_CHILD)

Използват се за диалогови елементи: текстово поле, поле за редактиране, бутон, списъчна кутия и останалите ресурси на диалоговите кутии.

Особености:

- тясно се свързани с родителския прозорец;
- не излизат извън размера на родителския прозорец;
- координатите са им относителни спрямо горния ляв ъгъл на работното поле на родителския прозорец;
- при преместване на родителския прозорец дъщерните прозорци се местят с него;
- дъщерните прозорци не могат никога да бъдат активни;
- дъщерните прозорци обикновено ***нямат главни обслужващи функции***;
- съобщенията от тях се подават през параметъра **wParam** на обслужващата функция на родителския прозорец при съобщението

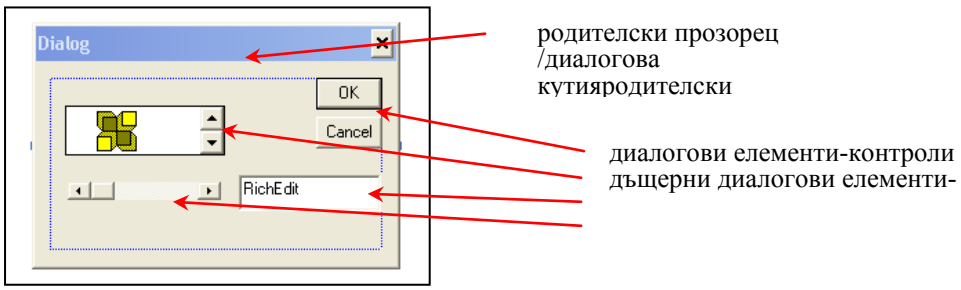
WM_COMMAND и чрез програмната структура switch по wParam се обработват.

Модален диалог и контролите на Windows с общо предназначение

Диалогова кутия и Диалогови елементи

Почти всяка Windows програма използва диалогов прозорец, за да взаимодейства с потребителя. Той може да бъде обикновен прозорец за съобщение, с един бутон ОК, но може да бъде сложна форма за въвеждане на данни. Съществуват два типа диалози: модален (modal) и не-модален (modeless). С ***модален*** диалог като File Open за отваряне на файлове, потребителят не може да работи на друго място в същото приложение (по-точно в същата нишкса от потребителския интерфейс), докато диалога не бъде затворен. С ***немодалния*** диалог потребителят може да работи на друго място в същото приложение, докато диалога остава на екрана. Пример за немодален диалог е диалогът Find and Replace. Той може да стои на екрана докато документа се редактира. Изборът на типа на диалога зависи от приложението, но модалните диалози са по-лесни за програмиране.

Най-често използваните диалози са ***модалните***. Действие от страна на потребителя (избор от меню) извежда диалога на екрана, след което потребителят въвежда данни и затваря диалога.



Фиг. 30. Пример с диалогови елементи

Диалоговите кутии са прозорци с обработващи функции или помощни прозорци . Прозорците са свързани йерархично, като един родителски прозорец може да съдържа няколко дъщерни прозореца. Диалоговите елементи са дъщерни прозорци на помощен прозорец и един такъв пример с диалоговите елементи е представен на Фиг. 30. Съобщенията от дъщерните прозорци се групират в съобщението: WM_COMMAND . В WINDOWS има предефинирани стандартни класове дъщерни прозорци като диалогови елементи (контроли). Например:

“button”, “edit”, “scroolbar”, “listbox” и др., подробно описани в началото на текста. Създават се опростено чрез CreateWindow и указване на стандартния клас.

Диалоговите елементи биват:

- ***Прости*** като бутони, превключватели, групова рамка, текстова рамка;
- ***Сложни*** като асансьори, поле за редактиране, списъчен прозорец за избор, комбинирани прозорци.

Собствен диалогови елемент се създава като:

- се регистрира клас;
- създава се дъщерен прозорец;
- обработват се съобщенията към дъщерния прозорец;
- изпраща се резултат към родителския прозорец чрез SendDlgItemMessage към ->WM_COMMAND

ОС Windows управлява диалоговите контроли използвайки специална логика за групиране и смяна на логиката, което значително облекчава програмирането. Може да се реферира към диалоговия елемент или чрез указател или индексен номер (с асоциирана #define константа), зададен в ресурса.

Съобщенията от дъщерните прозорци се групират в съобщението WM_COMMAND. (WM_COMMAND: wParam - индекс на дъщерния прозорец (диалоговия контролен елемент); lParam - специфична информация). Изпращането на съобщения към родителския прозорец:

SendDlgItemMessage(hDlg,…….

Получаване на избора:

c_index=SendDlgItemMessage(hDlg,IDC_COMBO1,**CB_ADDSTRING**,0,(LPARAM)”hh’’);

Работна област на прозорците

Приложенията извеждат информация в работната област на прозорците. Екранът е разделяем между приложенията ресурс.

Особености:

- Приложенията не работят с фиксиран размер на прозорците (Размерът на прозореца се определя от потребителя при всеки извод).
- Приложенията не знаят, в кой точно момент ще се извърши извеждане върху екрана на прозореца. Извеждането се извършва асинхронно чрез

съобщения. Програмният фрагмент, който извежда в прозореца, трябва да може да се извиква по всяко време.

- При прикриване от друг прозорец работната област не се съхранява в паметта или файл. При откриване на закрита област изображението се пречертава наново (обновява се).
- Обновяването се управлява със съобщението WM_PAINT за целия прозорец или за част от него.

WM_PAINT не се изпраща генерално към всяка главна обслужваща функция на изведените прозорци, защото ще се получат голям брой излишни пречертавания.

ОС определя автоматично само видимите части и поддържа за всеки прозорец "PAINT STRUCTURE", което съдържа координатите на областта за обновяване „invalid rectangle", което се поддържа автоматично и се променя текущо.

WM_PAINT се премества в опашката на съобщенията с нисък приоритет: когато „invalid rectangle" е непразен и в опашката няма друго съобщение. Действията на потребителя са с по-голям приоритет. Забавеното обновяване се постига с функциите с функциите: InvalidateRectangle(hwnd, &rect, True/False)

WM_PAINT се подава непосредствено към обслужващата функция, ако е необходимо непосредствено обновяване, напр. при първично изчертаване. Използва се функция UpdateWindow()

Контекст на устройствата. (GDI)

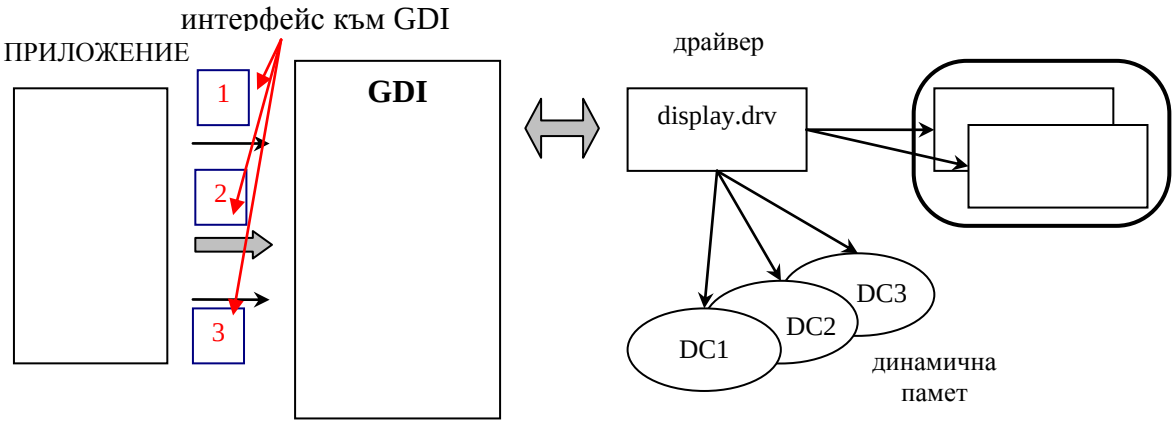
При извеждане на информация към екрана, принтера и др. тя не се извежда директно към хардуера, а се използва универсален интерфейс, наречено интерфейс на ***графично устройство или GDI (graphical device interface)*** . Windows предоставя драйверите за принтера и видео-драйверите. Вместо да адресира директно хардуера, програмата извиква GDI функции, които се обръщат към една структура от данни, наречена ***контекст на устройство*** (device context). Windows асоциира тази структура с физическото устройство и издава подходящите входно - изходни инструкции. GDI е почти толкова бърз, колкото и директния достъп до видео – паметта, а също така позволява различни приложения, написани за Windows, да използват споделено дисплея.

Всеки път когато програмата чертае на дисплея или на принтера, тя използва GDI функциите. Това са функции които чертаят точки, линии, правоъгълници, многоъгълници, битмапи, текст. В GDI ключовият елемент е ***контекст на***

устройството (device context), който представлява дадено физическо устройство. Всеки C++ обект (от тип контекст на устройството) има асоцииран с него Windows – контекст на устройство. Той е асоцииран посредством 32-битов манипулатор (handle) от тип HDC.

Текущо „състояние” на контекста на устройствата включва:

- Прикачени GDI обекти за чертане, като писалки, четки и шрифтове;
- Режим на съпоставяне, който определя мащаба на елементите при тяхното изчертаване;
- Различни детайли, като параметри за подравняване на текста и режими за запълване на полигони.



Фиг. 28. Схемна илюстрация на извеждане на прозорци

Контекстът на устройствата (Device Context DC) включва структура от данни – описваща специфични параметри на устройствата и се създава за всеки прозорец

Процесът на извеждане на информация към екрана е представен на Фиг. 28. Процесите при извеждане към екрана или принтера са означени с цифри и са съответно:

① Създаване на контекст на устройството.

② Операции по извеждане в работната област.

③Освобождаване от контекста.

Има два метода за извеждане в прозореца, представени по долу:

a) PAINTSTRUCT pc
HDC hdc
//////////……

```
case WM_PAINT:
hDC=BeginPaint(hwnd, &pc);

    ///…… графични функции

EndPaint(hwnd, &pc);
break;
```

b) HDC hdc

//////……

```
case WM_DRAW:
hDC=GetDC(hwnd);
     ///…… графични функции
ReleaseDC(hwnd, hdc);
break;
```

Относно hDC можем да получим информация за спецификата на устройството чрез функции като:

GetTextMetrix(hdc, &tm), където tm е размера на символите, примрно default font:

xChar=tm.tmAvecharWidth; yChar=tm.tmHeight;

или да изпълним графичните функции:

TextOut(hDC,ixPos,iYPos,szText,strlen(szText)); като например:

TextOut(hdc, 150, 100, bla, strlen(bla));

Език за дефиниране на ресурси

Ресурсите са данни , които се присъединяват към края на изпълняемия файл. Те се съхраняват на диска и се зареждат в паметта при необходимост. Стандартните ресурси представляват данни от различен тип.

Видове ресурси (предефинирани в Windows, ObjectWindows (OWL) – допълнително по-мощтни ресурси (обекти)):

- Меню (menu);
- Таблици с Комбинации от клавиши (accelerators);
- Диалогови кутии (dialogboxes);
- Шрифтове (fonts);
- Курсори за мишката (cursors);
- Икони (icons);
- информация за версията на програмата;
- HTML – WEB страница;

- Байтови изображения (bitmaps).

Ресурсите са част от изпълняемата програма, но не се помещават в данновите сегменти. Те се зареждат в паметта само когато е необходимо. Ресурсите се описват отделно в ресурсен - скрипт файл с помощта на език за описание на ресурсите.

Ресурсите се описват във файловете *.RC като се използва ***език за описание на ресурсите***. От тези файлове компилаторът на ресурси получава *.RES, които участват при свързването на приложението***.**

Следва пример за описание на ресурс „menu”

- **Описание на МЕНЮ (*.MNU)**

```
Demonstration MENU
BEGIN
    POPUP “&File”
    BEGIN
            MENUITEM “&NEW”, IDM_NEW
            MENUITEM “&Open”,IDM_OPEN
    - -
    END
    POPUP “&EDIT”
    BEGIN
    - - -
    END
END
```

под-меню начало

Връзка с обработващата функция.
Изпраща се в пром. на WM_COMMAND
или в *.h файл

- **Описание на диалогова кутия (*.DLF)**

```
About Box DIALOG 22, 17 144, 75
STYLE WS_POPUP|WS_DLGFRAME
BEGIN
        CTEXT “MicroSoft Windows”, - 1, 0, 5, 144, 8
        - - -
        DEFPUSHBUTTON  “OK”  IDOK, 53,59,32,14, WS_GROUP
END
```

Друг пример на фрагмент от *.RC файл,* описващ на диалог с идентификатор включващ различни контроли:
IDD_NewDiag,

```
IDD_NewDiag DIALOG DISCARDABLE  0, 0, 219, 130
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
  PUSHBUTTON      "Edit",IDC_EDIT1,120,7,50,14
  CONTROL         "Insert data in Combo",IDC_CHECK1,"Button",
                  BS_AUTOCHECKBOX | WS_TABSTOP,7,31,81,10
  CONTROL         "Insert Data in List",IDC_CHECK2,"Button",
                  BS_AUTOCHECKBOX | WS_TABSTOP,123,28,71,10
```

```
    COMBOBOX        IDC_COMBO1,7,50,98,66,CBS_DROPDOWN | CBS_SORT |
                    WS_VSCROLL | WS_TABSTOP
    LISTBOX         IDC_LIST1,124,42,81,41,LBS_SORT |
LBS_NOINTEGRALHEIGHT |
                    WS_VSCROLL | WS_TABSTOP
    EDITTEXT        IDC_ToAdd,49,7,64,14,ES_AUTOHSCROLL
    PUSHBUTTON      "Select from Combo",IDC_SELECTED,7,70,96,14
    EDITTEXT        IDC_COMBO_SEL,7,94,98,13,ES_AUTOHSCROLL
    EDITTEXT        IDC_LIST_SEL,127,105,75,14,ES_AUTOHSCROLL
    PUSHBUTTON      "Select from List",IDC_BUTTON1,131,88,70,13
END
```

Методи за работа с ProgressBar контрол и таймери

1. Функция, изпращаща съобщения към диалогов елемент ProgressBar

SendDlgItemMessage(hDlg,ID_of_DialogControl, Processing_Way, Index,(LPARAM) "Some text in Dialog Control")

Processing_Way - възможните стандартни константи за този параметър започват с **PBM: PBM_SETRANGE, PBM_GETRANGE, PBM_DELTAPOS, PBM_SETSTEP, PBM_SETPOS, PBM_STEPIT, PBM_GETPOS, PBM_GETSTEP, PBM_SETBARCOLOR, PBM_SETBKCOLOR;**

PBM_SETRANGE – задава диапазона (мин. и макс.) на които съответства началото и края на прогрес бара.

PBM_GETRANGE – връща диапазона, съответстващ на началото и края на бара

PBM_DELTAPOS – премества текущата позиция в п.б. на зададеното отстояние

PBM_SETPOS - премества текущата позиция в п.б. на зададена позиция

PBM_SETSTEP – задава стъпка за преместване

PBM_GETSTEP – връща стъпката за преместване на курсора

PBM_STEPIT – мести текущата позиция една стъпка

PBM_SETBARCOLOR задава цвят на бара (от 0 до COLORREF)

PBM_SETBKCOLOR – задава цвят на фона на п.б (от 0 до COLORREF)

COLORREF SetBarColor(COLORREF clrBar);

Примери:

SendDlgItemMessage(hDlg, IDC_PROGBAR, PBM_SETRANGE, 0, MAKELPARAM(0, 1000));

```
SendDlgItemMessage(hDlg, IDC_PROGBAR, PBM_DELTAPOS, 100, 0);
r= SendDlgItemMessage(hDlg, IDC_PROGBAR, PBM_GETPOS, 0, 0)
step = -20;
SendDlgItemMessage(hDlg, IDC_PROGBAR, PBM_SETSTEP, step,0);
SendDlgItemMessage(hDlg, IDC_PROGBAR, PBM_STEPIT,0,0);
```

2. Методи за работа с таймери

2.1. Създаване на таймери и установяване на стартовата стойност и стъпката за промяна

SetTimer (hWnd, nIDEvent, uElapse, Timerproc);
nIDEvent –идентификатор на таймера

uElapse – Време на изчакване в милисекунди до издаване на съобщениение

TimerProc Указател към функция, която за да бъде извикана, когато времето за изчакване стойност изтече.

Пример:

```
# include <commctrl.h>
#define TIMER1 500   // дефиниция на името на  таймера
SetTimer(hDlg, TIMER1, 20, NULL)
```

Приложението може да обработва съобщението WM_TIMER чрез включване като случай при обработката на съобщенията или чрез определяне на функция за обратно извикване на TimerProc, указана при създаването на таймера. Когато се зададе функцията TimerProc, подразбиращата се процедурата, обработваща прозореца извиква функцията когато се обработва WM_TIMER. Следователно, трябва да се изпраща съобщения в извикващия поток, дори когато се използва TimerProc вместо обработка WM_TIMER.

Параметърът wParam на съобщение WM_TIMER съдържа стойността на параметъра за nIDEvent. Това може да се използва за да се различат отделните таймери в случай, че се ползват повече от един в приложението.

Идентификаторът на таймер, nIDEvent, е специфичен за асоциране с прозореца. Друг прозорец може да има свой собствен таймер, който има същия идентификатор като този собственост на друг прозорец, но таймерите са различни.

Идентификаторът на таймер, nIDEvent, е специфичен за свързаните с прозореца.

2.2. Спиране работата на указан таймер

KillTimer(hWnd, nIDEvent) – спиране на таймер nIDEvent, асоцииран с hWnd

Пример: KillTimer(hDlg, TIMER1) – спира TIMER1.

4. Връщане идентификатора (хендела) на обекта в прозотрец hDlg, в който е указания с control ID контрол.

HWND GetDlgItem(hDlg, control ID);

5. Установяване на състояние на достъпност или недостъпност на прозорец

EnableWindow(hWnd, bEnable);

hWnd – handle на прозореца , който ще сменя състоянието си

bEnable е променлива (bool), указваща как ще се смени статуса на прозореца

6. Показване на прозорец или контрол

ShowWindow((hWnd, bEnable);

hWnd – handle на прозореца , който ще сменя състоянивто си

bEnable е променлива (bool), указваща как ще се смени статуса на прозореца

Примери:

CheckDlgButton(hDlg, IDC_A, BST_CHECKED); // => CheckRadioButton(...)

{UINT state = IsDlgButtonChecked(hDlg, IDC_CHECK_CTRL);

EnableWindow(GetDlgItem(hDlg, IDC_B), state); // прави контрола достъпен ако state=TRUE

Бутон с идентификатор IDC_A - има свойството visible=false

if (strcmp(psw,”password”)==0)

ShowWindow(GetDlgItem(hDlg, IDC_A), TRUE); // прави видим бутона IDC_A

6. Координати на мишката

int **LOWORD**(lParam) – връща x- координата на на позицията на мишката of mouse;int y = **HIWORD**(lParam) връща the y- координата на на позицията на мишката;

Лекция 6

Структура на WIN32-приложение.

Примерна програма за управление на прозорец

Основни правила и ограничения при програмиране под WINDOWS

1. Да не се използват стандартни функции за извод към екрана като: printf, scamf, getchar, putchar.
2. Да не се работи пряко с ОП или периферните устройства (клавиатура, мишка, таймер, дисплай).
3. Да не се присвоява процесора само към една задача. Така се избягват зациклиянията.
4. Да не се използват библиотечни функции за работа с паралелния или серийния интерфейс като с файлове. Да се използват функциите за комуникация на WINDOWS.
5. Да не се използват стандартни функции за работа с паметта malloc, calloc, realloc, free. Те се заменят автоматично от функции на WINDOWS.
6. Стандартните функции за работа с файлове са допустими, но използването на Windows- функциите за работа с файлове е препоръчително (имат по-големи възможности).

Структура на Windows – приложения

Работата в Windows-програмите се основава на обработка на съобщения, които постъпват от потребителя, ОС и други програми. Структурата на приложението непременно включва в себе си главната функция **WinMain**, еднакво устроена за всички приложения. С нея започва изпълнението на всички Windows – програми. **WinMain** трябва да изпълни следните действия:

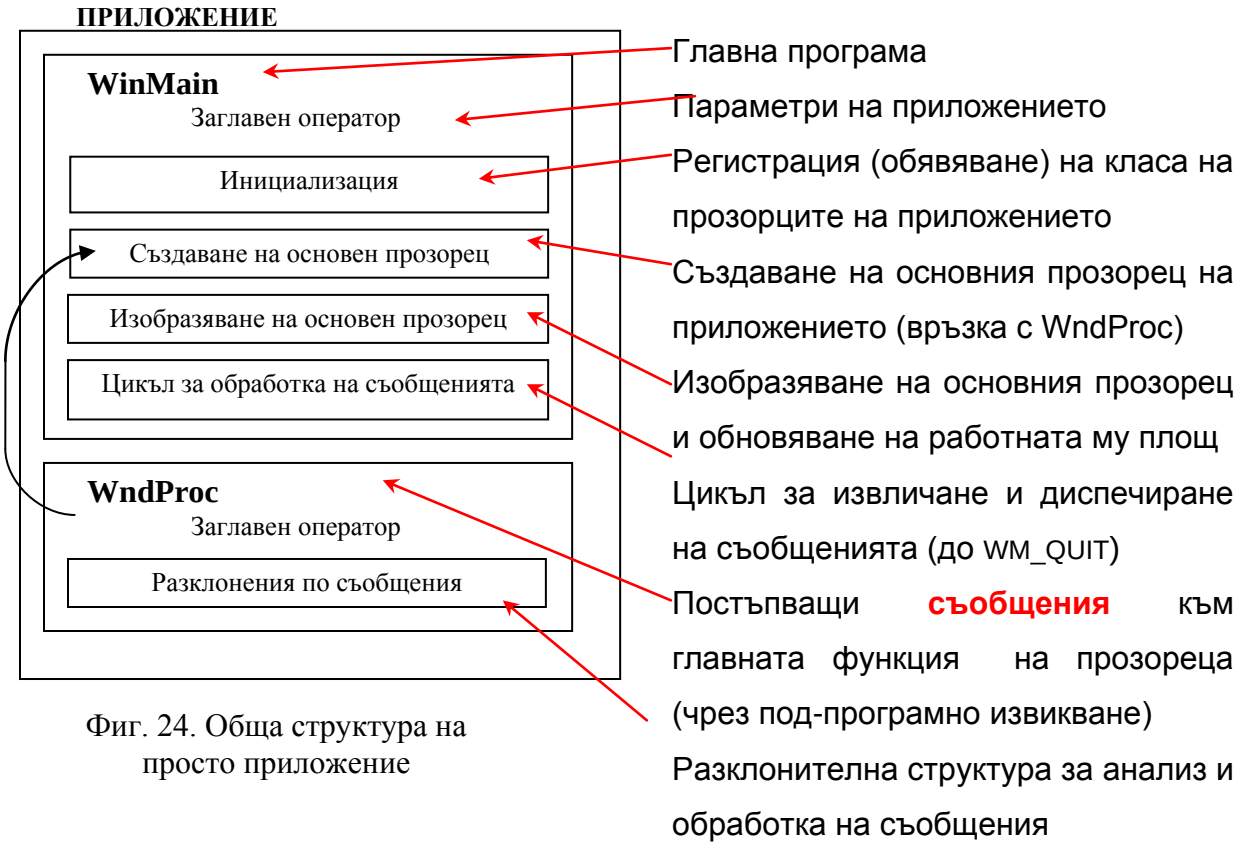
1. Да определи и регистрира клас –прозорец в Windows.
2. Да създаде и изобрази прозорец, определен от данните на класа.
3. Да организира цикъл за обработка на съобщенията.

При определяне на класа на прозореца се указва специална функция на прозореца, която трябва да реагира на постъпващите в прозореца съобщения. Всяко приложение има своя опашка от съобщения, която създава Windows и помества там всички съобщения, адресирани към прозореца на приложението.

След създаването и изобразяването на прозореца се организира основния ***цикъл за обработка*** на съобщенията, в който съобщенията преминават ***предварителна обработка*** и се връщат в Windows обратно. След това Windows извиква ***функцията на прозореца*** в програмата с опашката от съобщения в качеството на аргументи. Анализирайки съобщенията, функцията на прозореца инициира съответните операции. Съобщенията, чиято обработка не е предвидена от функцията на прозореца, се предават на Windows, която изпълнява обработка по „подразбиране".

Обща структура на просто приложение е представено на Фиг.24

По-долу е приведена минимална програма за Windows.



Фиг. 24. Обща структура на просто приложение

Процес на обработка на съобщенията

Като правило, програмите, написани за Windows, използват прозорци за организация на интерфейса и основно се използва механизмът за обмяна на съобщения. Съобщенията могат да се предават на прозоречната процедура веднага, а може да се поставят и в опашката от съобщения. При стартиране на приложение Windows създава за него опашка на съобщенията. Такава опашка се създава за всеки поток (thread), или най-малко една опашка за всеки процес. При всяко действие с прозореца (изменение на размерите и др.) Windows

поставя съответното съобщение в опашката. Форматът на съообщението съответства на структурата MSG.

Структура MSG:

```
typedef struct tagMSG
{
   HWND hwnd;
   UINT message;
   WPARAM wParam;
   LPARAM lParam;
    DWORD time;
   POINT pt;
} MSG;
```

- hwnd – хендъл на прозореца на получателя на съобщението;
- message - код на съобщението;
- wParam - параметър на съобщението;
- lParam – допълнителен параметър на съобщението;
- time – времето на изпращане на съобщението;
- pt – координатите на курсора на мишката в момент на отправяне на съобщението.

Хендъл на прозорец - число идентифициращо прозореца. Всеки прозорец в Windows има своя свой хендъл. Практически във всички функции на API се изисква да се укаже хендъл на прозореца.

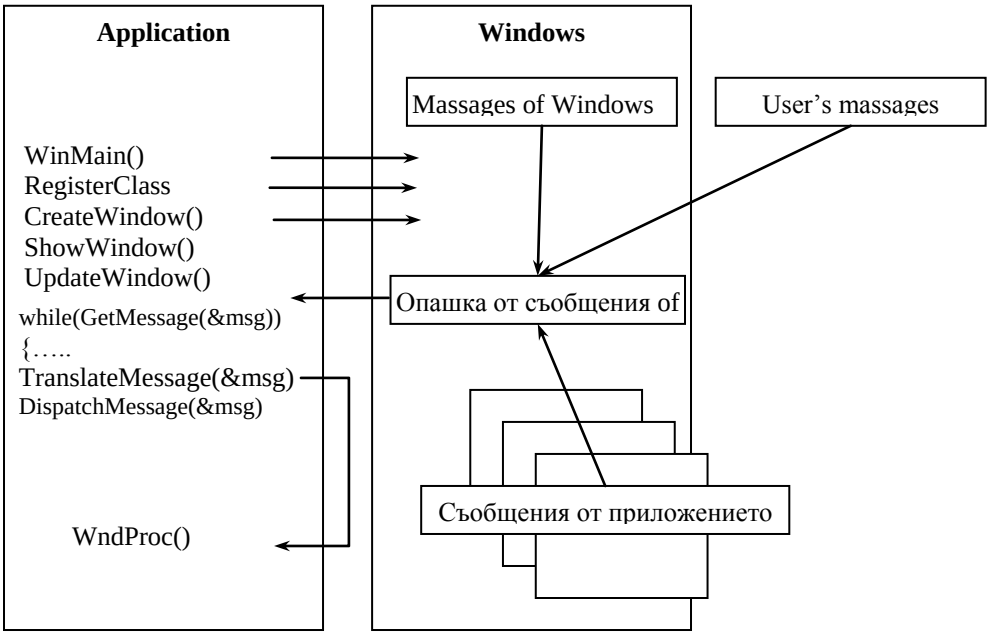
Съобщенията, поставени в опашката на съобщения, трябва да се извличат и обработват правилно. За целта е предвиден цикъл за обработка на съобщенията.

Схематично ***процеса на обработка на съобщенията*** на приложението е представена с структурата на Фиг. 25.

Всички програми за Windows трябва да включват библиотечния файл ***WINDOWS.H.*** Този файл ***съдържа*** прототипи на API – функциите, а така също определянето на различи типове данни, макроси и константи. Функцията на главния прозорец, използвана в програмата се нарича : **WinProc(),** която е

обявена като функция на обратното извикване т.к. тя се извиква от ОС за обработка на съобщенията.

Изпълнението на приложението започва с извикването на **WinMain().**Тя първо определя класа на прозореца по пътя на запълването на полетата на структурата **WNDCLASSEX**. С помощта на дадената структура програмата указва на WINDOWS функцията и стила на прозореца. Регистрацията на класа на прозореца се изпълнява с помощта на API функцията **RegisterClassEx.**



Фиг. 25. Схема на процеса на обработка на съобщенията на приложението

Следва създаването на прозореца с помощта на **API** функцията **CreateWindow()** , параметрите на която определят външния вид на прозореца. В края на функцията WinMain() е разположен цикъла на съобщенията. Той е задължителна част от всяко приложение. Докато приложението се изпълнява, то непрекъснато получава съобщения и ги извлича от опашката с помощта на API функцията **GetMessage().** Съобщенията постъпват чрез параметър на функцията от тип ***структура MSG***. Ако опашката е празна, то извикването на **GetMessage()** предава управлението на WINDOWS. При завършване на работа с програмата функцията **GetMessage()** връща нула. След като съобщението е прочетено, то се предава към API функцията **TranslateMessage(&msg)** и **DispatchMessage().** WINDOWS съхранява съобщението до тогава, докато не го предаде на програмата в качеството на параметър на функцията на прозореца. Когато цикълът на съобщенията завърши, **WinMain()** също завършва своето изпълнение

и предава на ОС значението на кода на завършването. Този начин на обработка на съобщенията дава възможност за:

- филтриране на получените съобщения, задавайки нужните параметри на функцията GetMessage();
- да се транслират виртуалните кодове на клавишите в клавиатурни съобщения с помощта на функцията TranslateMessage(&msg);
- да се транслират кодовете на натискане на «горещи» клавиши в съобщения на командите с помощта на функцията TranslateAccelerator(hwnd, hAccel, &msg).

Цикъл за обработка на съобщенията:

```
MSG msg;
while(GetMessage(&msg,0,0,0))
{
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
```

Функция GetMessage():

```
BOOL GetMessage(
   LPMSG lpMsg,
   HWND hWnd,
   UINT wMsgFilterMin,
   UINT wMsgFilterMax);
```

Функцията GetMessage() извлича поредното съобщение от опашката на съобщения и го поставя в структурата lpMsg типа MSG.Ако липсва в опашката съобщение то текущия поток се привежда в състояние на очакване и ОС предава управението на друг поток.

Връща значенията: Ако функцията извлича съобщение различно от WM_QUIT, то функция връща неенулево значение.

Ако функцията извлича съобщението WM_QUIT, то тя връща 0.

Ако има грешка, то тя връща -1.

- lpMsg - указател на структурата, където ще се постави съобщението;

- hWnd - хендъл на прозореца, за който се извлича съобщението, ако се укаже 0 то функцията GetMessage() ще извлече съобщения за всички прозорци;
- wMsgFilterMin –долната граница на диапазона на извличане, например, ако се укаже 10, то ще се извличат съобщения с кодове започващи от 10, ако се укаже 0 то тази граница отсъства.;
- wMsgFilterMax – горна граница на диапазона на изличаните съобщения, например ако е 300, то ще се извличат съобщения с кодове до 300, ако е зададено 0 то тази граница отсъства.

Функция TranslateMessage():

BOOL TranslateMessage(CONST MSG *lpMsg);

Функцията TranslateMessage() транслира съобщения от клавиатурата. Тя превежда съобщенията на виртуалните клавиши (Virtual-key) в символни съобщения.

Функция DispatchMessage():

LONG DispatchMessage(CONST MSG *lpmsg);

Функцията DispatchMessage() изпраща съобщение в прозоречната процедура. По същество тази функция ***извиква прозоречната процедура***.

Прозоречна процедура

Прозоречната процедура е функция за обработка на събития (съобщения). Тя извиква ОС Windows и получава параметри: хендъл на прозореца - hWnd, код на съобщението - msg и два параметъра lParam и wParam. Процедурата може да има всякакво име, но обикновено е **WndProc**.

/* Прототип */

```
LRESULT CALLBACK WndProc(
    HWND hWnd,
    UINT msg,
    WPARAM wParam,
    LPARAM lParam );
```

Прозоречната процедура трябва да умее да обработва всички съобщения, но в Windows такива съобщения са няколко хиляди. Всичко се свежда до

обработката само на тези съобщения, които са необходими на конкретното приложение, а всички останали ще се обработват по подразбиране с функцията - **DefWindowProc();**

```
/* Пример за прозоречна процедура */
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam,
LPARAM lParam)
{.......   switch(msg)
  {

    case WM_DESTROY:
    // да се постави съобщението WM_QUIT в опашката на съобщенията
    PostQuitMessage(0);
    break;

    default:
    // прозоречна роцедура по подразбиране
    return DefWindowProc(hWnd,msg,wParam,lParam);
  }
  return 0;}
```

Тази процедура обработва само едно съобщение - WM_DESTROY, а всички други се обработват от процедурата по подразбиране. Съобщението WM_DESTROY се изпраща при опит да се затвори прозореца.
Процедурата обработва съобщенията за прозорци от определени класове.

Класове на прозорците, регистрация на нов клас

Прозоречният клас е този, на чиято основа се създават прозорци. На базата на един прозоречен клас могат да се създават няколко прозореца. В ОС Windows има пре-определени класове прозорци. За да се създаде прозорец на основата на даден клас, то той трябва да се регистрира в системата с функцията **RegisterClass()** или **RegisterClassEx().**
Необходимо да се запълни структурата WNDCLASSEX.

Структура WNDCLASSEX:

```
typedef struct _WNDCLASSEX {
    UINT            cbSize;
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
    HICON hIconSm;
} WNDCLASSEX;
```

- cbSize - размер на структурата WNDCLASSEX;
- style – битови флагове , определящи стила на прозореца ;
- lpfnWndProc - адрес на прозоречната процедура;
- cbClsExtra - размер за допълнителна памет на класа;
- cbWndExtra - размер на доп. памет;
- hInstance - хендъл на екземпляра на приложението, на което принадлежи прозоречната процедура;
- hIcon - хендъл на икона (32x32 пиксела);
- hCursor хендъл на курсора на прозореца по подразбиране;
- hbrBackground хендъл на цвета по подразбиране;
- lpszMenuName – менюто на прозореца (името на ресурса);
- lpszClassName – име на класса;
- hIconSm – хендъл на малката икона (16x16 пикселей).

След запълване на структурата **WNDCLASSEX**, класът трябва да се регистрира в системата. Това става с помощта на **RegisterClassEx();**

```
WNDCLASSEX wc;
// попълване на структурата WNDCLASS
// ....
// Регистриране на нов клас
RegisterClassEx(&wc);
```

Създаване на прозорец

Това става с функцията **CreateWindow()** или **CreateWindowEx()**.

В случай на успех на функцията CreateWindow() тя връща хендъл на новия създаден прозорец, в противен случай връща NULL.

```
HWND CreateWindow(
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    HMENU hMenu,
    HANDLE hInstance,
    LPVOID lpParam );
```

- lpClassName – името на регистрирания клас на прозореца;
- lpWindowName – заглавието на прозореца;
- dwStyle –стил на прозореца (комбинация от флагове);
- x - начално положение по X;
- y - начално положение по Y;
- nWidth - ширина на прозореца;
- nHeight – височина на прозореца ;
- hWndParent - хендъл на прозореца на родителя ;
- hMenu - хендъл на менюто на прозореца;
- hInstance хендъл на екземпляра на приложението, създаващ прозореца;
- lpParam - указател на параметрите за WM_CREATE.

След създаването на прозореца е необходимо той да се изобрази на екрана. За показване на екрана се извиква функцията ShowWindow().

```
ShowWindow(hWnd, SW_SHOW);
```

Шаблон на програма за windows

```
#include <windows.h>
// Обявяване на прозоречната процедура
LRESULT CALLBACK WndProc(HWND hWnd,UINT msg,
                                        WPARAM wParam,LPARAM lParam);
int WINAPI WinMain(HINSTANCE hInstance,
                                    HINSTANCE hPrevInstance,
                                   LPSTR lpCmdLine, int iCmdShow)
{
const char szClassName[]="MyApp"; // Име на прозоречния клас
WNDCLASSEX wc;
MSG msg;
HWND hWnd;

// Запълване на полетата на структурата wc
wc.cbSize=sizeof(wc); // Размер на структурата wc
wc.cbClsExtra=0;        // допълнителна памет за класа
wc.cbWndExtra=0;     // допълнителна памет за прозореца
wc.hbrBackground=CreateSolidBrush(RGB(135,163,187)); // Цвят на фона
wc.hCursor=LoadCursor(0,IDC_ARROW); // Вид на курсора на мишката
wc.hIcon=LoadIcon(hInstance,IDI_APPLICATION); // Икона
wc.hIconSm=LoadIcon(hInstance,IDI_APPLICATION); // Малка икона
wc.hInstance=hInstance; // хендъл на екземпляра на приложението
wc.lpfnWndProc=WndProc; // Име на прозоречната процедура
wc.lpszClassName=szClassName; // Име на прозоречния клас
wc.lpszMenuName=NULL; // Име на менюто
wc.style=CS_HREDRAW | CS_VREDRAW; // Стил на прозореца

// Регистрация на нов прозоречен клас
if(!RegisterClassEx(&wc))
{
MessageBox(NULL,"Не е възможно да се регистрира клас","Грешка",MB_OK |
MB_ICONSTOP);
    return -1;}
```

```
// Създаване на прозорец
hWnd=CreateWindow( szClassName, // Име на проречния клас
                   "МоЁ Окно", // Заглавие на прозореца
                   WS_OVERLAPPEDWINDOW, // Стил на прозореца.
                   CW_USEDEFAULT, // Координата x
                   CW_USEDEFAULT, // Координата y
                   CW_USEDEFAULT, // Ширина на прозореца
                   CW_USEDEFAULT, //Височина на прозореца
                   0, // хендъл на родителския прозорец
                   NULL, // хендъл на менюто
                   hInstance, // хендъл на екземпляра на приложението
                   NULL); // Допълнителни параметри

if(hWnd==NULL)
{
   MessageBox(NULL,"Не може да се създаде нов прозорец ","Грешка",MB_OK |
MB_ICONSTOP);
   return -1;
}

ShowWindow(hWnd,SW_SHOW); // Показване на прозореца
UpdateWindow(hWnd); // Прозоречната процедура изпраща съобщението
WM_PAINT

// Цикъл за обработка на съобщенията
while (GetMessage(&msg,0,0,0))
{
   TranslateMessage(&msg); // за съобщения от клавиатурата
   DispatchMessage(&msg); // Стартиране на прозоречната процедура
}
   return 0;
}
// Прозоречна процедура
LRESULT CALLBACK WndProc(HWND hWnd,
```

```
                        UINT msg,
                        WPARAM wParam,
                        LPARAM lParam)
{
   HDC hDC;
   PAINTSTRUCT ps;
   switch(msg)
   {
      case WM_PAINT:
      {          hDC=BeginPaint(hWnd,&ps);
         // Нещо се рисува
         EndPaint(hWnd,&ps);
      }
      break;
      case WM_DESTROY:
      PostQuitMessage(0);
      break;
      default:
      return DefWindowProc(hWnd,msg,wParam,lParam);
   }
   return 0;}
```

Примерен програмен код, съответстващ на структура на просто приложение:

Листинг на минимално приложение

```
# include <windows.h>
# include “sample.h”
// Прототип на функция на прозорец
LRESULT, CALLBACK WndProc (
                HWND hvnd,  // указател на прозореца
                UINT uint,  // идентификатор на съобщения
                WPARAM  wparam, // параметри на съобщенията
                LPARAM lparam);// координати на мишката
                char szWindowName[]=”myClass”;  // име на класа на прозореца

// Функция WinMain ①
int  WINAPI WinMain(
                HINSTANCE hInstance,       // Дескриптор на екземпляра на приложението
                HINSTANCE hPrevInstance,   // NULL
```

```
                LPSTR lpCmdLine,          // указател на ред с аргументи
                int nCmdShow)             // Способ за визуализация

{
// Initialization ②
    if(!Init(hInstance)) return NULL;
    // указател на на главния прозорец на програмата
    HWND hwnd;
    // Структура- буфер за съхранение на съобщенията
    MSG msg;
    // Структура за определяне на класа на прозореца
            WNDCLASSEX wc1;
            wc1.hInstance-hThisInst;                    //Дескриптор на приложнието
            wc1.lpszClassName = szWindowName;  // Име на класа на прозореца
            wc1.style=0;                                // Стил по подразбиране
            wc1.lpfnWnfProc=(WNDPROC) WndProc;          // обработваща функция
            wc1.hIcon=LoadIcon(NULL, IDI_APPLICATION);      // Икона
            wc1.hCursor=LoadCursor(NULL, IDC_ARROW);    //  Курсор
            wc1.lpzMenueName=NULL;      // Без меню
            wc1.cbClsExtra=0;   // Без доп. информация
            wc1.cbWndExtra=0;   // Без доп. информация
            wc1.hbrBackground=(HBRUSH)GetStockObject(WHILE_BRUSH);   // Фон

    //Регистрация на класа на прозореца
    IF (!RegisterClassEx(&wc1)) RETURN 0;

//Създаване на прозореца ③
    hwnd=CreateWindow (
            szWindowName,  // Име на класа на прозореца
            “Minimal Program”, // Заглавие
            WS_OVERRLAPPEDWINDOW, // Стил
            CW_USERDEFAULT,   // Координати x, y
            CW_USERDEFAULT,   // горен ляв ъгъл
            CW_USERDEFAULT,   //ширина
            CW_USERDEFAULT,   // височина
            HWND_DESKTOP,   // родителски прозорец
            NULL, /дескриптор на меню
            hInstance, //дескриптор на приложение
            NULL);   // без доп. информация

// Изобразяване на прозореца ④
    ShowWindow (hwnd, nCmdShow)

// Прерисуване на прозореца ⑤
    UpdateWindow(hwnd)

// Цикъл на обработка извличане на съобщенията  ⑥
    while (GetMessage())
        &msg,  //  Буфер за съобщения
        NULL, // Да се получават съобщения от всички прозорци на приложението
        0,0))  // Диапазон на получаванните съобщения-всички
        {
        // връщане на управлението на Windows
        TranslateMessage(&msg);
```

```
        DispatchMessage(&msg);
        }
        return msg.wParam;  // Значение, връщано на програмата
}
// Функция на прозореца извиквана от ОС за обработка на съобщенята ⑦

LRESULT CALLBACK WndProc  (
                        HWND hwnd, //  Дескриптор на прозорец
                        UINT message,   //  Идентификатор на съобщения
                        WPARAM  wParam.//    Параметри
                        LPARAM lParam)
{    int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
switch (message)     // многократен разклонител за действия с използването на lParam, WParam
  {
      // case  WM_COMMAND:--------:   // ⑧ Обработка на  съобщение

                wmId    = LOWORD(wParam);
                wmEvent = HIWORD(wParam);
                switch (wmId)
                {
                        case IDM_ABOUT:
                        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd,
                        About);  break;
                        case IDM_EXIT:
                        DestroyWindow(hWnd);   break;
                        default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
        break;

//     ---------
        case WM_PAINT: // рисуване или обновяване на прозореца
                hdc = BeginPaint(hWnd, &ps);
                // TODO: Add any drawing code here...
                EndPaint(hWnd, &ps);
                break;
        case  WM_DESTROY:       //  ⑨  Затваряне  на  прозореца  .  WM_QUITкъм
      WINDOWS
                PostQuitMessage(0);
                break;
        default:   //      ⑩ Предаване на обработваните съобщения към WINDOWS обратно за обработка по подразбиране
        return DefWindowProc(hwnd, message, wParam, lParam);
  }
 return  0;
};
```

Подробно описание на функциите и техните параметри

- ① Главна програма

hInstance - Дескриптор на активното копие на приложението;

hPrevInstace - Дескриптор на последното активното копие на приложението, ако е нула, това е единственото;

- if(!Init(hPrevInstance)) return NULL; ако е възможно пускането на повече копия;

- if(!Init(hInstance)) return NULL; Инициализира само когато е първо копие създаване на глобални ресурси (клас на прозорците);

lpCmdLine – дълъг указател към команден ред, анализ за параметри от приложението;

nCmdShow – как се изобразява първоначално приложението.

- ② Инициализация

```
BOOL Init(Instance)  // обявяване на клас прозорци
HANDLE hInstance; // дескриптор на текущото копие
{
HANDLE hMem;  // дескриптор на динамичен блок памет
PWNDCLASS BWndClass;  // указател към класа прозорец
BOOL bResult      // резултат от обявяването на класа
hMem=LocalAlloc(LPTR, sizeof(WNDCLASS);   // заявка за памет
pWndClass= (PWNDCLASS) LocalLock(hMem) // фиксиране на блока и получаване на физическия адрес;
pWndClass->style                  = NULL;    // свойства на прозореца;
pWndClass->lpfnWndProc         =WndProc;   // главна обслужваща функция
pWndClass->hInstance             = hInstance;   // дескриптор на копието
pWndClass->hIcon                  =LoadIcon(NULL,IDI_APPLICATION);    // икона
pWndClass->hCursor               = LoadCursor(NUUL,IDC_ARROW);  //курсор
pWndClass->hbrBackground      = GetStockObject(WHITE_BRUSH);  // основа
pWndClass->lpszMenuName       = (LPSTR) NULL;   // ако има меню
pWndClass->lpszClassName      = (LPsTR) “sample”;  // глобално име на класа
bRes = RegisterClass(pWndClass);
LocalUnlock(hMem);
LocalFree(hMem);
return bRes;
}
```

- Всяко приложение извежда информация в прозорец на екрана. Екрана е разделяем ресурс.
- Всеки прозорец е инстанция на определен клас.
- Всеки клас трябва да се обяви предварително, преди да се създаде прозорец от този клас.
- Класът определя общите характеристики на прозорците наследници: вид на курсора, цвят на основата, ..
- Обявяване на клас: Запълване на структура на Windows с параметри. Обръщение към функцията RegisterClass. Използване на Динамичната памет на класа
- NULL :Windows поставя подразбиращи се стойности (defaults)
- lpfnWndProc – указател за динамично свързване на главната обслужваща функция на обекта прозорец.
- LoadIcon, LoadCursor може да зареждат икони или курсори, създадени предварително като ресурси

• ③ Създаване на прозорец

```
hWnd = CreateWindow (     // връща дескриптор на създадения прозорец
“sample”,                 // име на класа
“Sample Application”      // заглавие на прозореца
WS_OVERLAPPEDWINDOW,  // тип на прозореца  (ws_... from WINDOWS.H)
100, 80                   // X,Y координати на горния ляв ъгъл
300, 200,                 //  размери на прозореца (WS_USERDEFAULT)
NULL,                     // ДЕСКРИПТОР НА РОДИТЕЛСКИЯ ПРОЗОРЕЦ
NULL                      // дескриптор на меню
hInstance,                // допълнителен указател
NULL          );
IF (!HwND)  return NULL          // при невъзможност да се създаде прозореца
```

• ④ Изобразяване на и ⑤ обновяване на прозорец

```
hwnd                      // деескриптор на създадения прозорец
nCmdShow                  // сочи как да се изобрази- нормално или като икона
```

UpdateWindow(hwnd) – генерира съобщението WM_PAINT. WM_PAINT се интерпретира в главната функция на обекта прозорец и предизвиква пречертаване съдържанието на прозореца.

• ⑤ Прерисуване на прозореца.

• ⑥ Цикъл за извличане и обработка на съобщения.

Този цикъл извършва главната задача на WinMain

GetMessage - Извлича съобщението и го записва във структурата msg

DispatchMessage - Windows го предава на съответната главна обработваща функция на прозорец.

TranslateMessage - Преобразуване на съобщения от клавиатурата в стандарта ANS
WM_QUIT- връща се NULL и приложението завършва . Излъчва се от обработващата функция на главния прозорец. PostQuitMessage

- ⑦ Цикъл за извличане и обработка на съобщения
- ⑧ Обработва различни съобщения (WM_ например WM_PAINT)
- ⑨ Затваря главния прозорец и край на приложението (WM_DESTROY)
- ⑩ Съобщенията, които не се интерпретират от текущата функция се предават за обработка към Windows обратно.