4. Windows-приложения с графическим интерфейсом

4.1. Взаимодействие программ и Windows

ОС Windows—многозадачная система. Пользователь может запустить сразу несколько программ, так называемых приложений Windows. Каждая программа должна создать на экране свое главное окно приложения, через которое будет происходить взаимодействие программы с пользователем.

Взаимодействие программ и Windows осуществляется с помощью сообщений. Когда пользователь совершает какие-либо действия в окне приложения, например, выбирает команды меню, нажимает клавиши мыши или клавиатуры, изменяет размеры окна, то ОС Windows сообщает об этом соответствующей программе. Программа ожидает получения сообщения от Windows, когда это происходит, то выполняет некоторые действия. На одни сообщения программа может реагировать, на другие – нет.

Главная особенность программ для ОС Windows - это *необходимость создания своего собственного окна на экране и организации обработки сообщений Windows.*

Windows может посылать сообщения различных типов:

- при нажатии кнопок мыши в области окна;
- при нажатии клавиш на клавиатуре;
- при выборе команд меню;
- при нажатии кнопок на панели инструментов приложения;
- при изменении размеров или восстановлении окна после его сворачивания;
- при закрытии окна и во многих других случаях.

Программа взаимодействует с ОС Windows, если ей необходим доступ к системным ресурсам. Тогда она вызывает *функции API* (*Application Programming Interface*) – интерфейса прикладного программирования. С помощью функций API выполняются все необходимые системные действия, такие как выделение памяти, создание окон и различных элементов пользовательского интерфейса (меню, кнопок, панелей ввода, панелей инструментов), вывод информации в окно и т.п.

Функции API содержатся в библиотеках динамической загрузки (*Dynamic Link Libraries*, или DLL), которые загружаются в память только в тот момент, когда к ним происходит обращение, т.е. при выполнении программы. Одним из подмножеств API является GDI (*Graphics Device Interface* — интерфейс графического устройства). GDI — это та часть ОС Windows, которая обеспечивает поддержку аппаратно -независимой графики.

Поскольку архитектура Windows-приложений основана на принципе сообщений, все эти программы содержат некоторые общие компоненты. Любая Windows-программа должна содержать главную функцию программы WinMain() и функцию окна. На рис.4.1 показано, как программа взаимодействует с системой Windows.

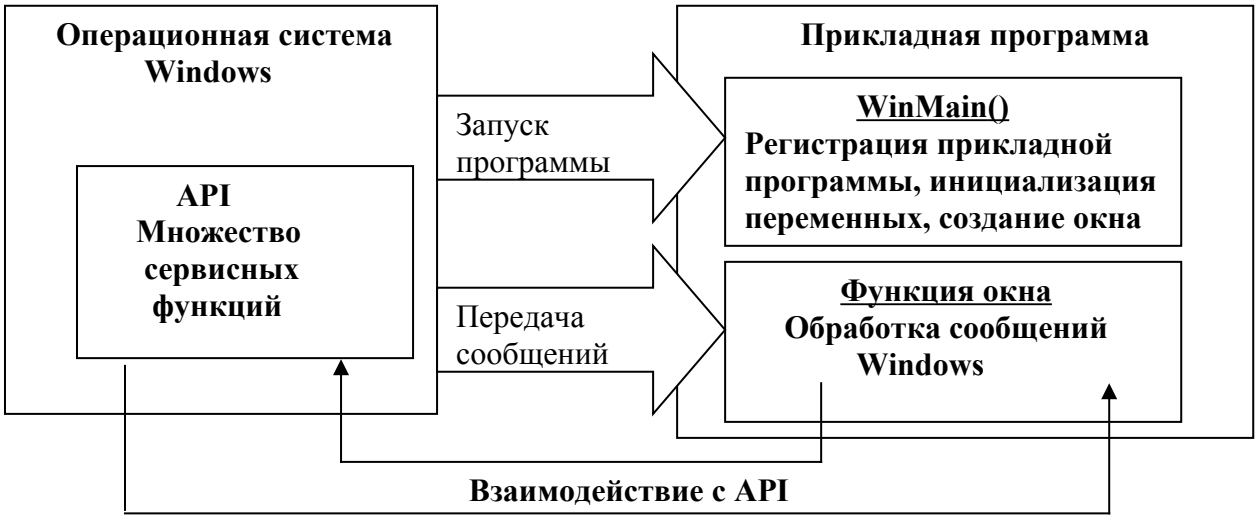

Рис. 4.1. Организация взаимодействия прикладной программы с Windows

4.2. Функция окна

Все Windows-программы должны содержать специальную функцию, которая вызывается не самой программой, а операционной системой, когда Windows передает сообщение программе. Эту функцию называют функцией окна, или процедурой окна. Согласно терминологии Windows, функции, вызываемые системой, называются функциями обратного вызова. Именно через нее осуществляется взаимодействие между программой и системой.

Имя функции окна может быть любое, например, *WindowsFunc*, тип функции – LRESULT CALLBACK. Первое слово LRESULT означает, что функция должна вернуть в качестве результата длинное целое (long) – код завершения, а CALLBACK означает, что это *функция обратного вызова*.

```
LRESULT CALLBACK WindowsFunc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

Функция окна имеет 4 параметра, которые характеризуют передаваемое сообщение: дескриптор окна, тип сообщения, последние два зависят от типа сообщения.

В этой функции должна быть реализована обработка всех сообщений Windows. Обычно она состоит из оператора switch, в котором на каждое сообщение предусмотрена соответствующая реакция.

4.3. Сообщения Windows

Каждое сообщение характеризуется 4 параметрами: дескриптор окна, тип сообщения и два дополнительных параметра, которые зависят от типа сообщения.

Первый параметр *window handle* – это *дескриптор окна*, которому адресовано сообщение. Он представляет собой уникальный номер, идентифицирующий окно.

Второй параметр определяет *тип сообщения* - *message type*. Тип сообщения – это один из идентификаторов, определенных в заголовочных файлах Windows. Идентификаторы начинаются с префикса WM_ (Windows Message). Наиболее часто посылаемые сообщения: WM_DESTROY (при закрытии окна), WM_PAINT (когда окно требует обновления), WM_COMMAND (при выборе команды меню), WM_CHAR (при нажатии клавиши клавиатуры), WM_LBUTTONDOWN (при нажатии левой кнопки мыши), WM_SIZE (при изменении размеров окна) и др.

Последние два параметра содержат дополнительную информацию, необходимую для интерпретации сообщения, например, для сообщений WM_LBUTTONDOWN и WM_RBUTTONDOWN, передаются координаты курсора мыши, для WM_CHAR - код клавиши, WM_COMMAND - идентификатор выбранного пункта меню.

Когда сообщение посылается окну программы, все перечисленные параметры сообщения передаются функции окна.

4.4. Функция WinMain()

Функция WinMain() – это *главная функция программы*, которая вызывается при запуске приложения. Заголовок функции:

```
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
LPSTR lpszArgs, int nWinMode)
```

Первый формальный параметр *hThisInst* задает описатель (дескриптор) данного экземпляра приложения, который идентифицирует программу. Параметр *hPrevInst* использовался в Windows 3.1, а в более поздних версиях всегда равен NULL. Параметр *lpszArgs* – указатель на буфер с аргументами командной строки. Параметр *nWinMode* определяет вид отображения окна при запуске программы (нормальный, развернутый на весь экран или свернутый). Тип функции *WINAPI* задает стандартный для Win32 способ передачи параметров.

Функция WinMain() должна выполнять следующие действия:

1. Определить класс окна.
2. Зарегистрировать класс окна в Windows.
3. Создать окно, определяемое данным классом.
4. Отобразить это окно на экране.
5. Запустить цикл обработки очереди сообщений.

Класс окна должен быть определен и зарегистрирован прежде, чем будет создано окно. Класс окна определяет стиль или тип окна, служит в качестве шаблона, задающего атрибуты окна. В Windows существует несколько предопределенных классов окна, например, WNDCLASS или WNDCLASSEX, но можно определить свои собственные классы.

Для *регистрации класса* окна необходимо объявить структурную переменную определенного класса окна и заполнить элементы структуры, чтобы сообщить ОС Windows имя функции окна, а также какой вид должно иметь окно: стиль окна, вид курсора, иконки, фон окна и др.

```
char szWinName[] = "MyWin";        // Имя "класса" окна
WNDCLASSEX wcl;                    // "класс" окна

  // Определение элементов "класса"  окна
  wcl.hInstance = hThisInst;  // дескриптор данного экземпляра
  wcl.lpszClassName = (LPCWSTR)szWinName;    // имя "класса" окна
  wcl.lpfnWndProc = WindowsFunc;        // функция окна
  wcl.style = 0;                        // стиль по умолчанию
  wcl.cbSize = sizeof(WNDCLASSEX);      // размер структуры
  wcl.hIcon = LoadIcon(NULL,IDI_APPLICATION); // большая иконка
  wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO); // малая иконка
  wcl.hCursor = LoadCursor(NULL, IDC_ARROW); // форма курсора
  wcl.lpszMenuName = NULL;              // меню не используется
  wcl.cbClsExtra = 0;         // дополнительной информации нет
  wcl.cbWndExtra = 0;

  // Регистрация "класса"  окна
  RegisterClassEx(&wcl);
```

При *создании окна* определяется заголовок окна, координаты, размер окна и др.

```
//     Создание окна

  HWND hWnd;                    // дескриптор окна
  hWnd = CreateWindowEx
  (  0,(LPCWSTR)szWinName,  // имя "класса" окна
     (LPCWSTR)"Демонстрационный пример", // заголовок  окна
     WS_OVERLAPPEDWINDOW,   // стиль: стандартное окно
     CW_USEDEFAULT,         // координата X - по умолчанию
     CW_USEDEFAULT,         // координата Y - по умолчанию
     CW_USEDEFAULT,         // ширина       - по умолчанию
     CW_USEDEFAULT,         // высота       - по умолчанию
     HWND_DESKTOP,          // родительского окна нет
     NULL,                  // меню нет
     hThisInst,     // дескриптор данного экземпляра приложения
     NULL                   // дополнительных аргументов нет
  );
  //     Отображение окна
  ShowWindow(hWnd, nWinMode);
```

Функция *CreateWindow*() создает графический образ окна, но не отображает окно на экране. При успешном завершении функция возвращает дескриптор созданного окна, в противном случае возвращается значение NULL.

После отображения окна нужно организовать *цикл обработки сообщений*. В этом цикле постоянно опрашивается очередь сообщений приложения, в которую ОС Windows посылает сообщения о событиях в окне приложения.

```
//Цикл опроса очереди сообщений
while (GetMessage(&msg, NULL, 0, 0))
  {
   TranslateMessage(&msg); // разрешает использование клавиатуры
   DispatchMessage (&msg); // возвращает управление Windows
  }
```

Функция *GetMessage*() извлекает очередное необработанное сообщение из очереди сообщений приложения и копирует его в структуру msg. Параметр NULL указывает на то, что функция должна принимать любые сообщения, поступающие любому окну приложения. Два последних параметра, равные 0, означают, что не нужно фильтровать сообщения. Фильтры сообщений можно использовать для того, чтобы ограничить прием только теми сообщениями, которые попадают в определенную категорию, например сообщения от клавиатуры или сообщения от мыши.

Функция GetMessage() только в одном случае возвращает значение FALSE, когда она получает сообщение WM_QUIT о завершении программы. Это приводит к завершению цикла обработки сообщений.

Функция *DispatchMessage*() возвращает сообщение ОС Windows. ОС в удобное для нее время вызывает функцию окна приложения, аргументом вызова служит сообщение. Функция окна, получив сообщение, выполняет соответствующие действия.

Функция *TranslateMessage*() необходима в тех программах, которые обрабатывают ввод с клавиатуры. Эта функция преобразует сообщения о нажатии клавиш (WM_KEYDONN и WM_KEYUP) в символьные сообщения (WM_CHAR), содержащие информацию об ASCII- коде нажатой клавиши.

4.5. Каркас Windows приложения

Архитектура Windows-приложений основана на принципе взаимодействия программ и ОС через механизм передачи сообщений. Поэтому все графические Windows-приложения имеют общий каркас, на основе которого можно создавать различные приложения. Каркас минимального Windows- приложения позволяет создать стандартное окно с заголовком, системным меню, тремя размерными кнопками. Его можно перемещать и менять размеры. В ОС Windows, версий 2005 и выше, каркас позволяет создать главное окно приложения, имеющее больше возможностей, например меню с пунктами Exit и About. Мы будем рассматривать каркас минимального Windows- приложения.

Каркас Windows- приложения содержит две функции WinMain () и функцию окна WinFunc(). Главная функция выполняет все действия, описанные в пункте 4.4. Функция окна в каркасе Window-приложения обрабатывает только одно сообщение WM_DESTROY. Сообщение WM_DESTROY посылается при закрытии пользователем окна приложения. При обработке этого сообщения вызывается функция PostQuitMessage(), которая посылает приложению сообщение WM_QUIT, что приводит к

завершению цикла обработки сообщений в главной функции программы и завершению выполнения всей программы.

Все сообщения, не обрабатываемые функцией окна, передаются для обработки в функцию *DefWindowProc*(). Эта функция очищает очередь сообщений от всех ненужных сообщений.

Исходный текст каркаса минимального Windows-приложения приведен в Приложении 2.

4.6. Вывод текстовой и графической информации в окно приложения

4.6.1. Вывод панелей с сообщениями

Для вывода в приложении панели сообщений предназначена функция *MessageBox*():

```
int MessageBoxA(HWND hWnd, LPCTSTR lpText,  LPCTSTR lpCaption,
UINT uType );
```

Параметр *hWnd* – дескриптор окна, параметр *lpText* – адрес строки сообщения, *lpCaption* – адрес строки заголовка панели сообщения, *uType* – стиль панели сообщения.

Параметр *uType* определяет, какие кнопки и значки (icon) будет иметь панель сообщения. Этот параметр задается комбинацией констант, которые имеют префикс MB_ (от слов Message Box). Например, чтобы панель имела кнопку OK и иконку с восклицательным знаком, нужно параметр uType задать в виде MB_OK | MB_ICONEXCLAMATION . Если в качестве этого параметра указать MB_OKCANCEL, то появятся две кнопки: OK и Отмена, а значка не будет. Некоторые из возможных значений параметра uType: MB_ICONQUESTION, MB_ICONINFORMATION, MB_RETRY, MB_YESNO и др. Функция MessageBox() возвращает идентификатор нажатой кнопки, например, если нажата кнопка OK, будет получено значение IDOK.

В следующем фрагменте функции окна обрабатывается сообщение о нажатии правой кнопки мыши, выводится панель с сообщением о координатах курсора мыши (рис.4.2).

```
LRESULT CALLBACK WindowFunc (HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{ char str[40];             // выводимое сообщение
  int xPos,yPos;            // координаты курсора мыши
...
 case WM_RBUTTONDOWN:      // нажата правая клавиша мыши
  xPos = LOWORD(lParam);     // координаты курсора по X
  yPos = HIWORD(lParam);     // координаты курсора по Y
  wsprintfA(
    str,"Координаты курсора мыши:\n x=%d;  y=%d",xPos,yPos);
  MessageBoxA(
    hWnd,str,"правая кнопка", MB_OK|MB_ICONINFORMATION);
```

```
break;
```

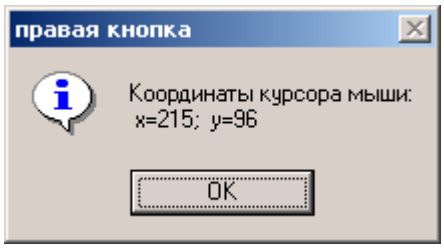

Рис.4.2. Панель с сообщением о действиях пользователя

Через параметры функции окна передается информация о сообщении:

- для сообщения WM_CHAR через параметр *wParam* передается код нажатой клавиши;

- для сообщений WM_LBUTTONDOWN и WM_RBUTTONDOWN параметр *lParam* содержит координаты *x,y* курсора мыши, а *wParam* - флаг, определяющий, какие управляющие клавиши Ctrl или Shift были нажаты одновременно с кнопкой мыши;

- для сообщения WM_COMMAND параметр *wParam* определяет идентификатор выбранного пункта меню.

4.6.2. Вывод информации в окно приложения

4.6.2.1. Контекст устройства

Для вывода информации в окно приложения программа должна получить контекст устройства (*device context*).

Контекст устройства – это структура данных, связывающая программу с драйвером устройства и определяющая состояние драйвера и способ вывода информации. Контексты устройств задают область или устройство графического вывода, например, рабочую область окна приложения, битовую матрицу в памяти, принтер.

Контекст устройства окна можно создать и получить, вызвав функцию *GetDC*(). По окончании вывода программа должна освободить контекст устройства, вызвав функцию *ReleaseDC*().

```
HDC  GetDC     (HWND hWnd);
int  ReleaseDC (HWND hWnd, HDC hdc);
```

Дескриптор окна вывода задается параметром *hWnd*. Функция GetDC() возвращает дескриптор контекста устройства вывода. Функция ReleaseDC() возвращает ненулевое значение, если освободила контекст устройства, заданного параметром *hdc*.

4.6.2.2. Вывод текста

Для вывода текста в окно приложения предназначена функция *TextOut*():

```
BOOL TextOut (HDC hdc, int x, int y, LPSTR lpstr, int nlengt);
```

Функция TextOut() выводит строку символов, заданную указателем *lpstr* (строка должна завершаться нулевым символом). Длина строки задается параметром *nlengt*, а координаты точки начала текста в окне – параметрами *x, y*.

Координаты текста или изображения в окне не зависят от положения самого окна на экране и всегда указываются относительно начала рабочей области окна. Координаты верхнего левого угла рабочей области окна считаются равными нулю, координата по оси X увеличивается при движении вправо, а по оси Y – при движении вниз.

Например,

```
HWND hWnd;
HDC hdc;

hdc = GetDC(hWnd);
char str[]="HELLO!!!";
TextOut (hdc, 20,25,str,strlen(str));

ReleaseDC(hWnd, hdc);
```

При использовании функции *TextOut*() текст отображается черным цветом на текущем фоне окна. Для задания цвета текста и фона окна можно вызвать соответственно функции *SetTextColor*() и *SetBkColor*() до вызова функции TextOut().

```
COLORREF  SetTextColor   (HDC hdc, COLORREF color);
COLORREF  SetBkColor     (HDC hdc, COLORREF color);
```

Параметр *color* определяет цвет для устройства с контекстом *hdc*. Обе функции возвращают предыдущий цвет, который можно запомнить и при необходимости восстановить.

Цвет задается переменной типа COLORREF, для этого используется *макрос RGB()*, который определен как

```
COLORREF RGB(int red, int green, int blue);
```

Нужный цвет можно получить при смешивании трех компонент – красной (red), зеленой (green) и синей (blue). Каждая компонента задается целым числом от 0 до 255, причем 0 задает минимальную интенсивность, а 255 - максимальную.

Например,

```
COLORREF c1(RGB(255,0,0));              // красный цвет
COLORREF c3 =RGB(200,200,200);          // серый цвет
SetTextColor (hdc,c1);  SetBkColor (hdc,c3);
// текст красными буквами на сером фоне
TextOut (hdc, 20,25,str,10);
```

4.6.2.3. Вывод графических объектов

Графические объекты рисуются с помощью пера (pen) и кисти (brush). *Перья и кисти* являются системными ресурсами. Для получения

дескрипторов стандартных ресурсов используется вызов функции GetStockObject ():

```
HDGIOBJ  GetStockObject (int object);
```

Существует несколько типов системных кистей: BLACK_BRUSH (черная), WHITE_BRUSH (белая), LTGRAY_BRUSH (серая). Перо может быть белое WHITE_PEN, черное BLACK_PEN или прозрачное NULL_PEN.

Можно создавать собственные перья с помощью функции CreatePen() и кисти, например, с помощью функции *CreateSolidBrush*(), которая создает сплошную кисть. Функция *CreateHatchBrush*() создает штриховую кисть, а *CreatePatternBrush*() – кисть с растровым изображением.

```
HPEN  CreatePen (int style, int width, COLORREF color);
HBRUSH  CreateSolidBrush (COLORREF color);
```

Параметр *style* определяет тип линии, создаваемой пером, параметр *width* задает толщину линии в пикселях, а параметр *color* – цвет пера или кисти. Функции возвращают дескриптор созданного объекта (пера или кисти).

Например, вызов функции

```
hRedPen = CreatePen(PS_SOLID, 1, RGB(255,0,0));
```

создает перо толщиной 1 пиксель красного цвета для рисования сплошных линий.

Чтобы использовать для рисования созданные перо и кисть, нужно их выбрать в контекст устройства с помощью функции SelectObject():

```
HGDIOBJ   SelectObject ( HDC hdc, HGDIOBJ hobj);
```

Параметр *hdc* определяет дескриптор контекста устройства, а *hobj* – дескриптор объекта. Функция возвращает дескриптор предыдущего объекта.

Все графические объекты рисуются текущим пером. Замкнутые графические объекты заполняются цветом и способом, соответствующим текущей кисти. По умолчанию используются белая кисть и черное перо толщиной один пиксель.

Созданные перья и кисти необходимо удалять перед завершением программы при обработке сообщения WM_DESTROY, вызвав функцию DeleteObject():

```
BOOL  DeleteObject (HGDIOBJ hObj);
```

где hObj - дескриптор удаляемого объекта.

Ниже приведены прототипы некоторых графических функций. Первый параметр всех функций определяет дескриптор контекста устройства вывода.

Изображение точки

```
COLORREF  SetPixel (HDC hdc, int x, int y, COLORREF color);
```

Функция SetPixel() устанавливает цвет, заданный параметром *color,* для точки с координатами *x,y*. Функция возвращает прежний цвет пикселя.

Установка текущей позиции

```
BOOL  MoveToEx (HDC hdc, int x, int y, LPPOINT lpCoord);
```

Функция MoveToEx() устанавливает текущую графическую позицию в точку с координатами *x,y*. Четвертый параметр – это указатель на структуру, в которой возвращаются координаты старой текущей позиции, если он равен NULL, координаты старой текущей позиции не возвращаются.

Рисование линии

```
BOOL  LineTo (HDC hdc, int x, int y);
```

Функция LineTo() рисует линию от текущей точки до точки с координатами *x,y*, после этого текущая позиция устанавливается в точке *x,y*.

Рисование прямоугольника

```
BOOL  Rectangle (HDC hdc, int upX, int upY, int lowX, int lowY);
```

Функция Rectangle() рисует прямоугольник, координаты верхнего левого и нижнего правого углов должны быть заданы в параметрах *upX, upY* и *lowX, lowY*.

Рисование эллипса

```
BOOL  Ellipse (HDC hdc, int upX, int upY, int lowX, int lowY);
```

Функция Ellipse() рисует эллипс. Эллипс определяется с помощью координат ограничивающего прямоугольника (координаты прямоугольника заданы так же, как для функции Rectangle()). Для того, чтобы нарисовать круг, достаточно определить квадрат.

4.6.3. Обновление содержимого окна

ОС Windows, как правило, не запоминает содержимое окна. В случае восстановления окна после минимизации, или активизации окна, перекрытого другим активным окном, программа должна перерисовать содержимое окна. Для этого Windows посылает программе специальное сообщение WM_PAINT. В ответ на это сообщение программа должна вывести графику в окно заново.

Обработка сообщения WM_PAINT начинается с получения контекста устройства окно при помощи вызова функции BeginPaint():

```
HDC BeginPaint (HWND hWnd, LPPAINTSTRUCT lpPS);
```

Параметр *hWnd* задает дескриптор окна. Параметр *lpPS* – указатель на структуру, содержащую характеристики области перерисовки.

Следующим шагом выводится информация в область перерисовки, затем вызов функции EndPaint() освобождает контекст устройства. Функция EndPaint() имеет такие же аргументы, как и функция BeginPaint().

Для обработки сообщения WM_PAINT в оконную функцию добавляется следующий фрагмент:

```
HWND hWnd;
  HDC hdc;
  PAINTSTRUCT paintstruct;
…
case WM_PAINT:
   hdc=BeginPaint(hWnd, &paintstruct);
     …
   EndPaint (hWnd, &paintstruct);
   break;
```

Программа может сама сообщить ОС Windows о необходимости вывода информации и перерисовки окна, т.е. запросить у Windows сообщение WM_PAINT. Для этого программа готовит информацию для вывода и вызывает функцию InvalidateRect(), а Windows отмечает окно, как подлежащее перерисовке, и посылает программе сообщение WM_PAINT.

```
BOOL InvalidateRect(HWND hWnd, CONST RECT *lpRect, BOOL fErase);
```

В структуре lpRect можно задать координаты прямоугольной области, подлежащей перерисовке. Если второй параметр равен NULL, окно перерисовывается полностью. Если третий параметр равен TRUE, окно перед перерисовкой очищается, если же он равен FALSE, то очистка окна не производится.

Например, вызов *InvalidateRect* (hWnd, NULL, 1); означает полностью перерисовать окно.

4.6.3.1. Проблема обновления окна

Механизм перерисовки окна может быть нескольких видов. В простых случаях можно каждый раз при необходимости перерисовки окна заново выводить всю графическую информацию в окно, т.е. весь вывод в окно выполнять при обработке сообщения WM_PAINT. Однако этот метод неэффективен при больших объемах меняющейся графики.

Наиболее общий метод перерисовки окна – это поддерживать *виртуальное окно в памяти* и копировать его содержимое в окно на экране при получении сообщения WM_PAINT.

Для копирования информации с одного устройства на другое используется функция *BitBlt*():

```
BOOL BitBlt(HDC hdc, int X, int Y, int Width, int Height,
HDC hSourse, int SourseX, int SourseY, DWORD dwRaster);
```

Параметр *hdc* означает дескриптор контекста устройства вывода (окна), параметр *hSourse* – дескриптор исходного контекста устройства (памяти), *X, Y* – координаты вывода растра в окне, *Width, Height* – ширина и высота растра, параметры *SourseX и SourseY* обычно равны 0, параметр *dwRaster* задает способ вывода растра (SRCCOPY означает перерисовку окна с затиранием предыдущего изображения).

Например,

```
maxX = GetSystemMetrics (SM_CXSCREEN);
maxY = GetSystemMetrics (SM_CYSCREEN);
hdc = GetDC (hWnd);
```

```
// создание совместимого контекста устройства памяти
memdc = CreateCompatibleDC (hdc);
hbit = CreateCompatibleBitmap (hdc, maxX, maxY);
PAINTSTRUCT paintStruct;
     case WM_PAINT:
          hdc=BeginPaint(hWnd, &paintStruct);
          BitBlt(hdc, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
          EndPaint (hWnd, &paintstruct);
          break;
```

Для использования виртуального окна необходимо создать контекст устройства памяти, совместимый с контекстом реального окна и совместимый растр изображения. Растровое изображение содержит побитовое представление рисунка, который должен быть изображен в окне. Устройство памяти используется для создания изображения перед выводом в окно.

При создании окна оконная функция получает сообщение WM_CREATE, при обработке этого сообщения и необходимо создать виртуальное окно, выполнив следующую последовательность действий:

1. получить контекст устройства окна, вызвав функцию *GetDC*();
2. создать совместимый контекст устройства памяти, вызвав функцию *CreateCompatibleDC*();
3. создать совместимый растр изображения, вызвав функцию *CreateCompatibleBitmap* ();
4. выбрать растровое изображение в контекст устройства памяти, вызвав функцию *SelectObject*();
5. получить дескриптор кисти функцией *GetStockObject*();
6. выбрать кисть как текущую для контекста устройства памяти функцией *SelectObject*()
7. зарисовать этой кистью виртуальное окно с помощью функции *PatBlt*();
8. освободить контекст устройства окна, используя функцию *ReleaseDC*();

Созданный виртуальный контекст устройства памяти будет существовать до окончания выполнения программы. Его необходимо удалить перед завершением программы вызовом функцией *DeleteDC*().

Прототипы перечисленных функций и фрагмент кода для создания виртуального окна приведены в приложении 5.

Приложение 2. Каркас стандартного Windows -приложения

```
#include “windows.h”
     /* объявление функции окна       */
LRESULT CALLBACK WindowFunc (HWND, UINT, WPARAM, LPARAM);

char szWinName[] = "MyWin";           // Имя "класса" окна
    /*       Главная функция          */
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
                  LPSTR lpszArgs, int nWinMode)
{ HWND hWnd;                          // дескриптор окна
  MSG msg;                            // сообщение
  WNDCLASSEX wcl;                     // "класс" окна
      /*  Определение "класса" (стиля) окна */
  wcl.hInstance = hThisInst; /*дескриптор данного экземпляра*/
  wcl.lpszClassName = (LPCWSTR)szWinName;   // имя "класса" окна
  wcl.lpfnWndProc = WindowFunc;       // функция окна
  wcl.style = 0;                      // стиль по умолчанию
  wcl.cbSize = sizeof(WNDCLASSEX);    // размер структуры
  wcl.hIcon = LoadIcon(NULL,IDI_APPLICATION); //больш.иконка
  wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO); // малая иконка
  wcl.hCursor = LoadCursor(NULL, IDC_ARROW); //форма курсора
  wcl.lpszMenuName = NULL;      // меню не используется
  wcl.cbClsExtra = 0;           // дополнит. информации нет
  wcl.cbWndExtra = 0;
      /* Фон окна задается белым   */
  wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
      /* Регистрация "класса" окна */
  if (!RegisterClassEx(&wcl))  return 0;
      /*     Создание окна         */
  hWnd = CreateWindowEx
  (  0, (LPCWSTR)szWinName,       // имя "класса" окна
     (LPCWSTR)"",                 // заголовок
     WS_OVERLAPPEDWINDOW,   // стандартное окно
     CW_USEDEFAULT,         // координата X - по умолчанию
     CW_USEDEFAULT,         // координата Y - по умолчанию
     CW_USEDEFAULT,         // ширина       - по умолчанию
     CW_USEDEFAULT,         // высота       - по умолчанию
     HWND_DESKTOP,          // родительского окна нет
     NULL,                  // меню нет
     hThisInst, // дескриптор данного экземпляра приложения
     NULL                  // дополнительных аргументов нет
  );
      /*     Отображение окна       */
  ShowWindow(hWnd, nWinMode);

      /*     Цикл сообщений        */
  while (GetMessage(&msg, NULL, 0, 0))
  { TranslateMessage(&msg);
    DispatchMessage (&msg);
  }
  return msg.wParam;
```

```
}
   /*-----------------------------------------------*/
   /*                Функция  окна                  */
   /*-----------------------------------------------*/
   /* Эту функцию вызывает Windows и передает ей  */
   /* на обработку сообщения из очереди сообщений */
   /*-----------------------------------------------*/
LRESULT CALLBACK WindowFunc (HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
  switch(message)
  { case WM_DESTROY:         /* "завершить программу" */
      PostQuitMessage(0);
      break;

    default:
      /* Остальные сообщения обрабатывать  */
      /* операционной системе              */
      return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

Приложение 3. Константы Windows - приложений

Таблица 4. Пиктограммы

Константа	Вид пиктограммы
IDI_APPLICATION	Стандартная системная пиктограмма
IDI_ASTERISK	Информационная пиктограмма с буквой "I"
IDI_EXCLAMATION	Пиктограмма с восклицательным знаком
IDI_HAND	Пиктограмма с надписью "Stop"
IDI_QUESTION	Пиктограмма с вопросительным знаком
IDI_WINLOGO	Программа с логотипом Windows

Таблица 5. Некоторые из встроенных курсоров Windows

Константа	Форма курсора
IDC_ARROW	Стандартный указатель типа стрелка
IDC_CROSS	Перекрестие
IDC_WAIT	Песочные часы

Таблица 6. Стили масштабируемых окон

Константа стиля	Элемент окна
WS_OVERLAPPED	Стандартное окно с рамкой
WS_MAXIMIZEBOX	Кнопка максимизации
WS_MINIMIZEBOX	Кнопка минимизации
WS_SYSMENU	Системное меню
WS_HSCROLL	Горизонтальная полоса прокрутки
WS_VSCROLL	Вертикальная полоса прокрутки

Таблица 7. Константы отображения

Константа отображения	Действие
SW_HIDE	Окно закрывается
SW_MAXIMIZE	Окно раскрывается на весь экран
SW_MINIMIZE	Окно сворачивается в пиктограмму
SW_RESTORE	Окно приводится к нормальному размеру

Таблица 8. Стили пера

Стиль	Описание
PS_DASH	штрих
PS_DOT	пунктир
PS_SOLID	сплошной
PS_DASHDOT	штрих - пунктир
PS_NULL	нет пера

Таблица 9. Стандартные кисти

Константа	Тип фона
BLACK_BRUSH	Черный
DKGRAY_BRUSH	Темно-серый
HOLLOW_BRUSH	"Прозрачный"
LTGRAY_BRUSH	Светло-серый
WHITE_BRUSH	Белый

Приложение 4. Сообщения о событиях мыши

Для обработки нажатий клавиш мыши предусмотрены такие сообщения как:

WM_LBUTTONDOWN –нажата левая клавиша мыши;
WM_LBUTTONDBLCLK двойной щелчок на левой клавише мыши;
WM_RBUTTONDOWN – нажата правая клавиша мыши;
WM_RBUTTONDBLCLK - двойной щелчоке на правой клавише мыши;
WM_LBUTTONUP – пользователь отпускает левую клавишу мыши;
WM_RBUTTONUP - пользователь отпускает правую клавишу мыши;
WM_MOUSEMOVE - сообщение посылается при передвижении курсора.

Параметры вышеуказанных сообщений:

fwKeys = *wParam*; // флаги
xPos = LOWORD(*lParam*); // горизонтальная позиция курсора
yPos = HIWORD(*lParam*); // вертикальная позиция курсора

Флаги *fwKeys* – значение параметра *wParam* может быть комбинацией следующих значений:

MK_CONTROL - флаг установлен, если нажата клавиша CTRL;
MK_LBUTTON - флаг установлен, если нажата левая клавиша мыши;
MK_MBUTTON - флаг установлен, если нажата средняя клавиша мыши;
MK_RBUTTON - флаг установлен, если нажата правая клавиша мыши;
MK_SHIFT - флаг установлен, если нажата клавиша SHIFT.

Нижнее слово параметра *lParam – xPos* - горизонтальная позиция курсора, относительно левого верхнего угла клиентской области. Верхнее слово параметра *lParam – yPo*s - горизонтальная позиция курсора,

относительно левого верхнего угла клиентской области.

Для обработки сообщений от мыши, надо добавить обработчик сообщения в функцию окна.

Приложение 5. Создание виртуального окна вывода

Виртуальное окно создается при обработке сообщения WM_CREATE о создании основного окна вывода:

```
case WM_CREATE:
/* получение размеров окна */
maxX = GetSystemMetrics (SM_CXSCREEN);
maxY = GetSystemMetrics (SM_CYSCREEN);
/* создание контекста устройства окна */
hdc = GetDC (hWnd);
/*создание совместимого контекста устройства памяти */
memdc = CreateCompatibleDC (hdc);
/* создание совместимого растра */
hbit = CreateCompatibleBitmap (hdc, maxX, maxY);
/* выбор растра в контекст устройства памяти */
SelectObject (memdc, hbit);
/* создание белой кисти */
hbrush = GetStockObject (WHITE_BRUSH);
/* выбор белой кисти */
SelectObject (memdc, hbrush);
/* закраска окна белой кистью */
PatBlt (memdc, 0, 0, maxX, maxY, PATCOPY);
/* освобождение контекста устройства окна */
ReleaseDC (hWnd, hdc);
break;
```

Описание функций, используемых при создании виртуального окна:

```
HDC CreateCompatibleDC(HDC hdc);
```

Функция возвращает дескриптор контекста памяти, совместимый с контекстом устройства, дескриптор которого задан в качестве параметра.

```
HBITMAP CreateCompatibleBitmap(HDC hdc, int width, int height);
```

Функция создает в памяти растровое изображение, совместимое с устройством, заданным первым параметром, размер растра задают два последних параметра.

```
BOOL PatBlt (HDC hdcmem, int X, int Y, int Width, int Height,
DWORD dwRaster);
```

Функция заполняет прямоугольную область, заданную контекстом устройства (первый параметр), с помощью текущей кисти, способом, определенным последним параметром.

Приложение 7. Программа, демонстрирующая вывод графики и работу с меню

```
#include “windows.h”
#include “resource.h”
     /*         объявление функции окна                */
LRESULT CALLBACK WindowFunc (HWND, UINT, WPARAM, LPARAM);
char szWinName[] = "MyWin";       // Имя "класса" окна
/*-------------------------------------------------------*/
/*                  Главная функция                      */
/*-------------------------------------------------------*/
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
                  LPSTR lpszArgs, int nWinMode)
{ HWND hWnd;                        // дескриптор окна
  MSG msg;                          //  сообщение
  WNDCLASSEX wcl;                   // "класс" окна
      /*  Определение элементов "класса"  окна */
  wcl.hInstance = hThisInst;      //дескриптор данного экземпляра
  wcl.lpszClassName = szWinName;  // имя "класса" окна
              */
  wcl.lpfnWndProc = WindowFunc;   // функция окна
              */
  wcl.style = 0;                                    */
  wcl.cbSize = sizeof(WNDCLASSEX);
  wcl.hIcon = LoadIcon(NULL,IDI_APPLICATION);
  wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO);
  wcl.hCursor = LoadCursor(NULL, IDC_ARROW);
  wcl.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1); // меню
  wcl.cbClsExtra = 0; wcl.cbWndExtra = 0;
      /* Фон окна задается белым   */
  wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
      /*         Регистрация "класса" окна         */
  if (!RegisterClassEx(&wcl))  return 0;
      /*         Создание окна                     */
  hWnd = CreateWindow
  (  szWinName,
     "Демонстрация вывода графики",
     WS_OVERLAPPEDWINDOW,
     CW_USEDEFAULT,  CW_USEDEFAULT,
     CW_USEDEFAULT,  CW_USEDEFAULT,
     HWND_DESKTOP,   NULL,
     hThisInst,  NULL);
      /*              Отображение окна             */
  ShowWindow(hWnd, nWinMode);
  UpdateWindow(hWnd);
      /*              Цикл сообщений               */
  while (GetMessage(&msg, NULL, 0, 0))
  { TranslateMessage(&msg);
    DispatchMessage (&msg);
  }
  return msg.wParam;
}
/*-------------------------------------------------------*/
```

```
HDC memdc;          // контекст устройства памяти
HBITMAP hbit;       // растр изображения в окне
HBRUSH hbrush;      // дескриптор текущей кисти
HBRUSH hOldbrush;          // дескриптор прежней кисти
HPEN hRedpen, hBluepen;    // дескрипторы перьев
HPEN hOldpen;              // дескриптор прежнего пера
int maxX, maxY;            // размеры экрана
 /*   массивы точек  - координаты вершин треугольников     */
 POINT mp1[3]={{150,50},{50,150},{150,150}};// 1-й треуг-к
 POINT mp2[3]={{250,150},{350,50},{450,150}};// 2-й треуг-к
 COLORREF color= RGB(255,255,0);   //  цвет кисти
 /*-------------------------------------------------------*/
 /*                      Функция  окна                    */
 /*-------------------------------------------------------*/
   LRESULT CALLBACK WindowFunc (HWND hWnd, UINT message,
                        WPARAM wParam, LPARAM lParam)
{HDC hdc;                  // контекст устройства окна
 PAINTSTRUCT paintstruct;  // характеристики области перерисовки
 switch(message)
 { case WM_CREATE:
       /* получение размеров окна */
       maxX = GetSystemMetrics (SM_CXSCREEN);
       maxY = GetSystemMetrics (SM_CYSCREEN);
       /* создание контекста устройства окна                */
       hdc = GetDC (hWnd);
       /* создание совместимого контекста устройства памяти */
       memdc = CreateCompatibleDC (hdc);
       /* создание совместимого растра                      */
       hbit = CreateCompatibleBitmap (hdc, maxX, maxY);
/* выбор растрового изображения в контекст устройства памяти */
       SelectObject (memdc, hbit);
       /* заполнение окна белой кистью                      */
       PatBlt (memdc, 0, 0, maxX, maxY, PATCOPY);
       /* создание красного и синего перьев                 */
       hRedpen = CreatePen (PS_SOLID, 2, RGB (200, 0, 0));
       hBluepen = CreatePen (PS_SOLID, 4, RGB (0,0,255));
       /* освобождение контекста устройства окна            */
       ReleaseDC (hWnd, hdc);
       break;
 case WM_COMMAND:                  // выбрана команда меню
  switch (wParam)
  {
    case ID_RECT:
       PatBlt (memdc, 0, 0, maxX, maxY, PATCOPY);
       hbrush = CreateSolidBrush (color);
       hOldbrush = (HBRUSH)SelectObject (memdc,hbrush );
       hOldpen = (HPEN)SelectObject (memdc,hRedpen );
       Rectangle (memdc, 50, 50, 200, 200);
       SelectObject (memdc, hBluepen);
       RoundRect (memdc, 250,50, 400, 150, 30, 30);
       SelectObject (memdc, hOldpen);
       SelectObject (memdc, hOldbrush);
       DeleteObject (hbrush);
```

```
        Polygon(memdc,(CONST POINT *) 3,mp1);
        InvalidateRect (hWnd, NULL, 0);
        break;
      case ID_RED:
        color = RGB(250,0,0);
        break;
   }
   break;
 case WM_PAINT:                  //обновление окна
        hdc=BeginPaint (hWnd, &paintstruct);
        BitBlt (hdc, 0, 0, maxX, maxY, memdc, 0, 0, SRCCOPY);
        EndPaint (hWnd, &paintstruct);
        break;
   case  WM_DESTROY:              // "завершить программу"
        /* удаление перьев */
        DeleteObject (hRedpen);
        DeleteObject (hBluepen);
        DeleteDC (memdc);          // удаление контекста памяти
        PostQuitMessage(0);
        break;
   default:
        return DefWindowProc ( hWnd, message, wParam, lParam );
   }
   return 0;
 }
```

Файл lab_5.rc

```
#include "resource.h"
IDR_MENU1 MENU DISCARDABLE
BEGIN
    POPUP "&Фигуры"
    BEGIN
        MENUITEM "&Треугольники",       ID_TREUG
    END
    POPUP "&Цвет"
    BEGIN
        MENUITEM "Красный",             ID_RED
    END
END
```

Файл resource.h

```
#define IDR_MENU1          101
#define ID_TREUG           40003
#define ID_RED             40004
```