15. Основы разработки событийно-управляемых программ в среде Windows

Технология создания приложений для современных многозадачных ОС тесно связана с особенностями таких систем. Поскольку в системе одновременно выполняется несколько приложений и все они претендуют на одни и те же ресурсы, то сама операционная система должна отслеживать использование ресурсов. Приложение не должно непосредственно использовать внешние устройства. Любая работа приложения должна контролироваться операционной системой. Поэтому для реализации всех необходимых действий приложению предоставляется богатый набор стандартных системных API-функций, использование которых и составляет первую важнейшую особенность разработки приложений.

В частности, системы семейства Windows предлагают разработчикам около 2000 таких функций, часть которых используется только в узкоспециализированных целях. Набор данных функций часто обозначается термином Win32 API. Важнейшие из этих функций собраны в трех основных модулях:

- В файле krnl32.dll собраны основные системные функции ядра: управление памятью, управление процессами и потоками, управление основными устройствами, управление файлами (около 300 функций).
- В файле user32.dll собраны функции, необходимые для поддержки многооконного пользовательского интерфейса (около 200 функций).
- В файле GDI32.dll собраны функции, отвечающие за графический вывод (GDI: Graphic Device Interface, Интерфейс Графического Устройства), их около 150 штук.

Кроме этих трех базовых модулей существует несколько десятков дополнительных модулей (файлы с расширением .dll), которые реализуют различные дополнительные функции.

Набор API-функций позволяет создать два типа Windows-приложений:

- простейшие **консольные** приложения, не использующие графические окна и ориентированные на ввод с клавиатуры и вывод на текстовый экран; соответствующие программы получаются очень компактными, но использование их весьма ограничено;
- **оконные** приложения с полноценным графическим пользовательским интерфейсом; практически весь дальнейший материал ориентирован именно на эту группу приложений.

Еще одной важной особенностью современных операционных систем является то, что они реализуют **централизованную обработку** возникающих в системе **событий.** Другими словами, сама система следит за возникающими при работе приложений событиями. В силу этого принципиально изменяется структура прикладных программ. Работа прикладной программы управляется потоком поступающих в нее событий, поэтому такие программы принято называть событийно-управляемыми.

В системах Windows наравне с термином **“событие”** (event) широко используется другое понятие – **“сообщение”** (message). Сообщение – это небольшая структура данных, которая содержит основные параметры произошедшего события. Любое сообщение содержит 6 полей, каждое по 4 байта. Наиболее важными являются первые 4 поля:

- целочисленный код (так называемый дескриптор) окна, которому предназначено данное сообщение;
- целочисленный код самого сообщения; все стандартные сообщения пронумерованы целыми числами, которые для удобства программирования заменены символьными константами вида wm_*****; в системе Windows насчитывается несколько сотен стандартных сообщений;
- поле, несущее дополнительную информацию о событии, например, код нажатой клавиши; это поле имеет специальное обозначение wParam.
- еще одно поле с дополнительной информацией о сообщении; это поле обозначается как LParam

Все многообразие событий/сообщений можно разбить на несколько групп, наиболее важными из которых являются следующие:

- оконные сообщения возникают при любых манипуляциях с окном, например:
  - wm_Create – возникает при создании окна;
  - wm_Size – возникает при изменении размеров окна;
  - wm_Move – возникает при перемещении окна;
- клавиатурные события (например, wm_KeyDown или wm_KeyUp связаны с вводом данных с клавиатуры;
- “мышиные” события используются для обработки соответствующих действий пользователя, например:
  - wm_LbuttonDown возникает при нажатии левой кнопки мыши;
  - wm_MouseMove возникает при перемещении мыши.

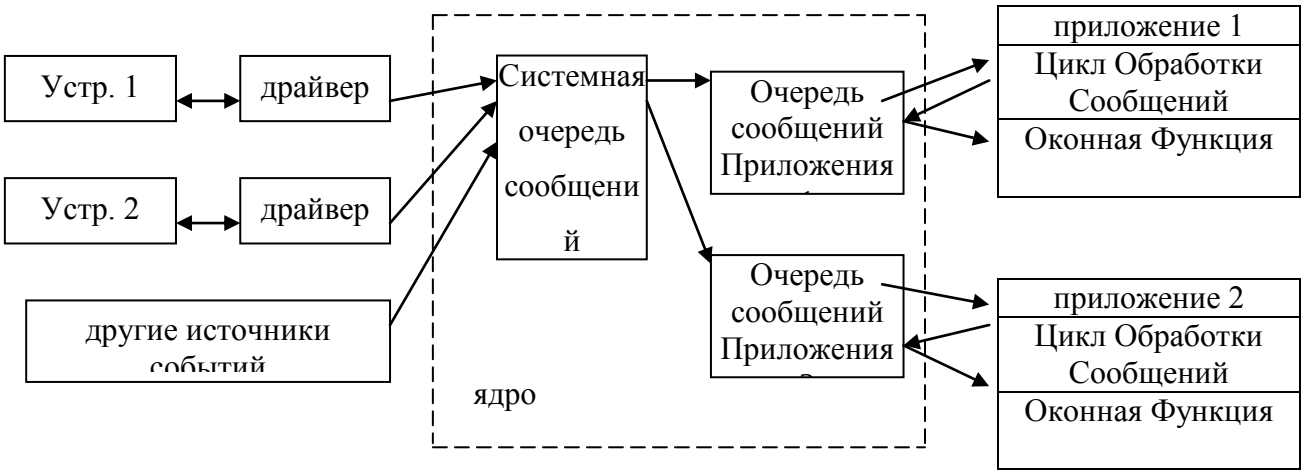
Взаимодействие Windows-приложений с операционной системой основано на использовании механизма событий/сообщений, реализованного на уровне ядра системы. Работа этого механизма может быть описана следующими основными шагами.

- шаги 1 и 2 выполняются системой и включают в себя:
  - инициированное внешним устройством (клавиатура, мышь) событие обрабатывается драйвером и оформляется в виде сообщения, которое поступает в общую системную очередь сообщений, поддерживаемую самой системой;
  - из общей очереди сообщений операционная система распределяет эти сообщения по очередям отдельных приложений;
- шаг 3 выполняется приложением и включает в себя выбор сообщения из начала своей очереди; для этого в приложении создается специальный программный фрагмент – **цикл обработки сообщений** (ЦОС); этот фрагмент является обязательным для любого оконного Windows-приложения; ЦОС запускается при старте приложения и завершает работу при закрытии приложения; выбор сообщения из

очереди говорит лишь о намерении приложения обработать это сообщение, но далеко не всегда приводит к немедленной обработке данного сообщения; о своем намерении приложение в лице ЦОС сообщает системе, которая запускает эту обработку, когда сочтет нужным (если нет более срочной работы), поэтому следующий шаг 4 выполняется опять системой;

- шаг 4: для обработки выбранного приложением сообщения система вызывает еще один специальный и очень важный фрагмент приложения – так называемую **оконную функцию** (ОФ), передавая ей параметры сообщения; особенностью ОФ для программиста является то, что в приложении она обязательно должна быть реализована как подпрограмма, но нигде в приложении эта подпрограмма **явно не вызывается;** поскольку ОФ вызывается системой, то она называется функцией обратного вызова (callback function), в отличие от “прямых” вызовов системных функций приложением;
- шаг 5: после того, как ОФ приложения получает управление от ОС, она наконец приступает к обработке сообщения; ОФ является главным компонентом при написании Windows-программы; она представляет собой набор **обработчиков** тех **событий**, на которые должно реагировать приложение; тем самым ОФ во многом определяет функциональность приложения; что касается обработки сообщений, то она, как правило, включает в себя вызов одной или нескольких API-функций.

Перечисленные шаги можно проиллюстрировать следующей схемой.



Из сказанного выше можно сделать важный вывод: все оконные Windows-приложения должны иметь одинаковую базовую структуру. Любое оконное Windows-приложение включает два основных фрагмента:

- основную, или главную, функцию, с которой начинается выполнение приложения; ее принято называть именем WinMain;
- оконную функцию для обработки сообщений приложения.

Основная функция WinMain. должна выполнять следующие действия:

- описание одного или нескольких оконных классов;
- регистрацию описанных оконных классов;
- создание главного окна приложения и, если это необходимо, создание нескольких дополнительных окон;
- отображение на экране главного окна приложения;
- запуск программного цикла обработки сообщений.

Более подробно все эти действия описываются ниже.

1. Описание оконного класса.

Оконный класс – это некий шаблон, содержащий наиболее общие параметры целой группы окон. Данные параметры описывают такие общие свойства, как фон окна, используемые курсоры и пиктограммы, и позволяют окнам одного и того же класса выглядеть одинаково.

Для описания оконного класса используется специальная структура данных со стандартным именем WndClass. Эта структура имеет 10 полей, которые **ВСЕ** необходимо заполнить, но, к счастью, многие из них заполняются стандартно. Прежде всего, необходимо ввести переменную данного типа, имеющую любое осмысленное имя, например – MyWndClass. Каждое поле этой структуры имеет свое жестко заданное имя, которое мнемонически определяет смысл данного поля. Порядок установки полей не имеет значения. С практической точки зрения наиболее интересными являются следующие поля:

- поле с именем lpfnWndProc, определяющее адрес оконной функции, связываемой с данным оконным классом:
  MyWndClass.lpfnWndProc := @ имя_оконной_функции;
- поле-указатель lpszClassName на текстовую строку с именем создаваемого оконного класса:
  MyWndClass.lpszClassName := ’MyClass’;
  имя класса может быть практически любым, за исключением нескольких стандартных зарезервированных имен;
- поле-указатель lpszMenuName на текстовую строку с именем меню, используемом в главном окне программы:
  MyWndClass.lpszMenuName := ‘MyMenu’;
  если меню не используется, полю присваивается пустое значение nil или NULL;
- поле hCursor, определяющее тип используемого в приложении курсора; подключение курсора может выполняться с помощью API-функции LoadCursor:
  MyWndClass.hCursor := LoadCursor (0, idc_Arrow);
  здесь второй параметр вызова – это константа, определяющая один из стандартных курсоров, а именно – обычную стрелку; кроме нее, можно использовать, например, такие курсоры, как перекрестие (idc_Cross) или песочные часы (idc_Wait);

- поле hbrBackground определяет шаблон заполнения фона окна с помощью инструмента рисования – кисти (Brush) и может устанавливаться многими способами:

  MyWndClass.hbrBackground := CreateSolidBrush(RGB(0,100,255));

  MyWndClass.hbrBackground := HBrush(Color_Window);

  здесь RGB – системная функция, определяющая цвет кисти для заполнения фона как комбинацию трех базовых составляющих цвета – красного (Red), зеленого (Green) и синего (Blue); интенсивность каждой составляющей задается целым числом от 0 до 255; например, RGB(0, 0, 0) задает черную кисть, а RGB(255, 255, 255) – белую;
- поле hIcon определяет пиктограмму окна при его сворачивании и устанавливается с помощью вызова LoadIcon:

  MyWndClass.hIcon := LoadIcon (0, idi_Application);

  здесь константа idi_Application определяет стандартную пиктограмму приложения.

Кроме перечисленных выше, необходимо установить следующие поля, которые приводятся без комментариев в силу их редкого изменения:

```
MyWndClass.Style := cs_VRedraw OR cs_HRedraw OR cs_DblClks;
MyWndClass.cbClsExtra := 0;
MyWndClass.cbWndExtra := 0;
MyWndClass.hInstance := hInstance;
```

2. После задания **всех 10 полей** структуры WndClass ее надо зарегистрировать в системе с помощью API-вызова RegisterClass с проверкой результата регистрации:

```
if RegisterClass(MyWndClass) = 0 then Exit; // неудачная регистрация
```

необходимость проверки связана с тем, что иногда она может привести к отрицательному результату, и в этом случае дальнейшая работа приложения невозможна.

3. После успешной регистрации оконного класса можно создавать окна данного класса. Как минимум, должно быть создано одно **главное** окно.

Создание одного окна выполняется с помощью одного вызова API–функции CreateWindow. Этот вызов принимает 11 параметров (более поздняя версия – уже 13). Эти параметры конкретизируют создаваемое на основе данного класса окно. Важно понимать, что создание окна – это не отображение окна на экране, а **создание** соответствующей **информационной структуры** с основными параметрами окна. Если создание окна проходит успешно, то вызов CreateWindow возвращает специальный описатель окна, так называемый **дескриптор.** Для работы с дескриптором надо ввести переменную системного типа HWnd. Многие последующие функции работы с окнами требуют задания дескриптора как одного из параметров.

Что касается входных параметров вызова CreateWindow, то из 11 параметров наиболее важными являются первые 9:

- первый параметр определяет имя оконного класса, на основе которого создается данное окно; это имя чаще всего совпадает с именем, заданным в качестве значения поля lpszClassName структуры оконного класса;
- второй параметр определяет текст в заголовке окна (если текст не нужен, надо задать 0);
- третий параметр задает стиль создаваемого окна; для этого используется набор специальных стилевых констант, среди которых чаще всего используются:
    - константа ws_Overlapped определяет стиль главного окна;
    - константа ws_Child определяет стиль дочернего окна, которое обязательно имеет родительское окно, одинаковый с ним вид и поведение; родителем может быть как главное окно, так и любое ранее созданное дочернее окно; тем самым окна могут образовывать вложенную иерархию;
    - константа ws_PopUp определяет еще одну разновидность подчиненных окон, так называемые всплывающие окна, которые также должны иметь родителя, но ведут себя немного по-

другому по сравнению с дочерними окнами; перечисленные выше стили не должны использоваться совместно при создании одного окна;

- константа ws_Visible определяет видимость создаваемого окна;
- константа Ws_Border определяет наличие рамки у окна;
- константа Ws_Caption определяет наличие у окна заголовка;
- константы ws_MaximizeBox и ws_MinimizeBox определяют, имеет окно кнопки максимизации или минимизации;

при необходимости некоторые перечисленные выше стили можно объединять вместе, используя логическую операцию ИЛИ (например – OR):

WsVisible OR ws_Caption OR ws_Border

очевидно, что не все стили можно комбинировать вместе, например – стили главного окна, дочернего и всплывающего противоречат друг другу;

- четвертый и пятый параметры определяют координаты левого верхнего угла окна относительно клиентской части его родителя, а для главного окна - относительно рабочего стола (клиентская часть окна образуется за вычетом заголовка и рамки); координаты задаются в пикселах;
- шестой и седьмой параметры определяют ширину и высоту создаваемого окна в пикселах:
- восьмой параметр определяет для создаваемого окна его родителя и задается соответствующим дескриптором; отсюда видно, что порядок создания окон должен быть вполне конкретным: сначала родитель, потом только подчиненное окно; первое по порядку создаваемое окно является главным, и для него восьмой параметр надо задать нулевым;
- девятый параметр в основном используется для присваивания окнам уникальных идентификаторов; использование этого параметра будет объяснено позже, а пока будем считать его нулевым.

Пример программного кода, создающего два окна, может выглядеть так:

```
var  MyMainWin, MyChildWin : HWnd;
MyMainWin := CreateWindow('MyClass', 'Это главное окно',
    ws_OverlappedWindow, 100, 100, 400, 300, 0, 0, hInstance, nil);
MyChildWin := CreateWindow('MyClass', 'А это дочернее', ws_Child OR
    ws_Visible OR ws_Caption OR ws_Border, 50, 50, 200, 100,
    MyMainWin, 0, hInstance, nil);
```

4. После создания окон можно выполнить их отображение на экране. При этом достаточно отобразить лишь главное окно, все подчиненные окна отобразятся автоматически. Для отображения главного окна в целом и его клиентской части используются два системных вызова:

```
ShowWindow (MyMainWin, cmdShow);
UpdateWindow (MyMainWin);
```

5. Последним шагом в главной программе является запуск цикла обработки сообщений. В простейшем случае этот цикл реализуется двумя API функциями:

```
while GetMessage (MyMessage, 0, 0, 0)
        DispatchMessage (MyMessage);
```

Здесь параметр MyMessage - это переменная системного типа Msg, используемая для хранения извлекаемого из очереди сообщения. Вызов GetMessage извлекает из очереди очередное сообщение, а вызов DispatchMessage передает это сообщение в оконную функцию на обработку. Цикл while закончит свою работу, когда вызов GetMessage вернет значение 0, а это произойдет в ответ на появление в очереди сообщений специального сообщения wm_Quit, помещаемого в очередь в ответ на закрытие главного окна приложения. Практически всегда перед вызовом DispatchMessage выполняется вызов функции TranslateMessage, используемой для обработки клавиатурных сообщений.

Теперь рассмотрим структуру оконной функции. Уже было отмечено, что ОФ – это обработчик сообщений, поступающих в приложение. Оконная функция оформляется в соответствии со следующими правилами:

- ОФ может иметь любое имя, но это имя обязательно должно совпадать со значением, заданным в поле lpfnWndProc структуры данного оконного класса.
- ОФ всегда принимает 4 входных параметра, которые соответствуют первым 4 полям структуры сообщения.
- Заголовок ОФ оформляется особым образом с использованием специальных директив, зависящих от используемого языка программирования
- Тело ОФ представляет собой инструкцию множественного выбора типа case_of, в которой в качестве селектора выступает код сообщения (второй входной параметр), а значениями селектора – текстовые имена-константы конкретных сообщений:

  case Message of

  wm_Create : “код обработки сообщения wm_Create”

  wm_Move : “код обработки сообщения wm_Move”;

  wm_KeyDown : “код обработки сообщения wm_KeyDown”;

  - - - - - - - - - - - - - - - -

  wm_Destroy : “код обработки сообщения wm_Destroy”;

  end;
- Набор сообщений может быть любым, но, как минимум, обязательно должна присутствовать обработка сообщения wm_Destroy, которое возникает при попытке закрытия главного окна приложения; при этом для правильного завершения работы приложения обработчик сообщения wm_Destroy должен с помощью системного вызова PostQuitMessage поместить в очередь сообщение wm_Quit, в ответ на которое цикл обработки сообщений завершит свою работу.

- После оператора выбора в конце ОФ обязательно должна вызываться специальная системная оконная функция, назначением которой является обработка всех тех сообщений, которые не обрабатываются данной оконной функцией. Такая оконная функция называется **умалчиваемой** и вызывается .с помощью функции DefWindowProc.

В итоге, стандартная базовая структура Windows-приложения выглядит следующим образом (использован язык Object Pascal системы Delphi).

```
program  Project1;
  uses Windows, Messages;   // подключение интерфейсных модулей

var
    MyWin : HWnd;                  // переменная-дескриптор главного окна
    MyChildWin, MyPopWin : HWnd;   // переменные-дескрипторы
подчиненных окон

                // начало описания оконной функции
  function WinProc(Win: HWnd; Mess, wParam, lParam : longint):longint; export;
                      stdcall;
    begin
      WinProc := 0;
      case  Mess  of
        wm_Destroy : begin             // обработчик сообщения
                          PostQuitMessage(0);
                          Exit;
                       end;

        wm_Create : begin              // обработчик сообщения
                          // действия, выполняемые при создании окна
                           Exit;
                      end;

        wm_***** : begin               // еще один обработчик
                         // обработка некоторого сообщения wm_*****
                         Exit;
                      end;
          . . . . . .  // другие обработчики
       end;                  //  для основного оператора case
      WinProc := DefWindowProc(Win, Mess, wParam, lParam);      // вызов
                                                          стандартного системного
                                                          обработчика
   end;                      // конец оконной функции
```

```
  procedure  WinMain;                  // основная функция приложения
    var
       Mess : TMsg;                    // переменная-сообщение
       MyWndClass : TWndClass;   // переменная для описания оконного
класса

    begin
       MyWndClass.Style := . . . .;    // здесь идет формирование BCEX 10
полей
       . . . . . .
        if  RegisterClass(MyWndClass) = 0  then  Exit;
                    // создание главного окна
        MyWin := CreateWindow('MyClass',  'Ура!Мое первое окно!',
                          ws_OverlappedWindow, 100, 100, 300, 200, 0, 0,
                          hInstance, nil);
        MyChildWin := CreateWindow( . . . . . );     // создание дочернего окна
        MyPopWin := CreateWindow( . . . . . );     // создание всплывающего окна

        ShowWindow(MyWin, cmdShow);         // отображение окна (только
главного!)
        UpdateWindow(MyWin);

        while  GetMessage(Mess, 0, 0, 0) do            // цикл обработки
сообщений
        begin
           TranslateMessage(Mess);
           DispatchMessage(Mess);
        end;
        Halt(Mess.wParam);
     end;                                 // конец основной функции

begin
   WinMain                               // запуск приложения
end.
```

Такое приложение имеет минимальную функциональность: после его запуска создаются главное окно и два подчиненных окна, которые можно перемещать по экрану, можно изменять размеры окон и, наконец, можно закрыть главное окно с завершением приложения. Вся эта стандартная функциональность с точки зрения обработки сообщений обеспечивается системной оконной функцией по умолчанию. Для придания приложению

полезных свойств необходимо, прежде всего, уметь реализовывать интерфейс с пользователем на основе стандартных управляющих элементов.