Основы программирования в системе Windows

Урок 1. Общая структура Windows-приложения

Процесс программирования Windows-приложений отличается от способа, ориентированного на обработку последовательностей команд или запросов, как это происходит при программировании консольных приложений.

Обработка приложением Windows-сообщений накладывает на структуру программы жесткие ограничения.

Пример типичного Windows-приложения

Рассмотрим исходный текст приложения, которое создает главное окно, в клиентской области которого выводится текст. При нажатии левой клавишей мыши в клиентской области окна при помощи стандартной диалоговой панели сообщений выдается предупреждение.

Пример 1.

```
// --- Обязательный включаемый файл
#include <windows.h>

// --- Описание функции главного окна
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);

// --- Глобальные переменные
HINSTANCE hInst;                        // Дескриптор экземпляра приложения
char ClassName[] = "Window";            // Название класса окна
char AppTitle[] = "Application Win32";        // Заголовок главного окна

// --- Функция WinMain
int WINAPI WinMain(
        HINSTANCE hInstance,   // Дескриптор экземпляра приложения
        HINSTANCE hPrevInstance, // В Win32 всегда равен NULL
        LPSTR lpCmdLine,       // Указатель на командную строку. Он позволяет
                        // приложению получать данные из командной строки.
        int nCmdShow           // Определяет, как приложение первоначально
                               // отображается на дисплее: пиктограммой
                               // (nCmdShow = SW_SHOWMINNOACTIVE)
                               // или в виде открытого окна
                               //(nCmdShow = SW_SHOWNORMAL).
        )
{
        WNDCLASS wc;    // Структура для информации о класса окна
        HWND hWnd;      // Дескриптор главного окна приложения
        MSG msg;        // Структура для хранения сообщения

        // Сохраняем дескриптор экземпляра приложения в глобальной переменной,
        // чтобы при необходимости воспользоваться им в функции окна.
        hInst = hInstance;

        // --- Проверяем, было ли приложение запущено ранее.
// Воспользуемся функцией FindWindow, которая позволяет найти окно верхнего
```

```
// уровня по имени класса или по заголовку окна:
        //      HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName);
/* Через параметр lpClassName передается указатель на текстовую строку, в кото-
рую необходимо записать имя класса искомого окна. На базе одного и того же клас-
са можно создать несколько окон. Если необходимо найти окно с заданным заголов-
ком, то имя заголовка следует передать через параметр lpWindowName. Если же по-
дойдет любое окно, то параметр lpWindowName может иметь значение NULL.*/
        if((hWnd = FindWindow(ClassName, NULL))!= NULL)
        {
        // Пользователь может не помнить, какие приложения уже запущены,
        // а какие нет. Когда он запускает приложение, то ожидает, что на экране
        // появится его главное окно. Поэтому, если приложение было запущено
        // ранее, целесообразно активизировать и выдвинуть на передний план
        // его главное окно. Это именно то, к чему приготовился пользователь.
                if(IsIconic(hWnd)) ShowWindow(hWnd, SW_RESTORE);
                SetForegroundWindow(hWnd);

        // Найдена работающая копия - работа новой копии прекращается.
        return FALSE;
        }

// --- Работающая копия не найдена - функция WinMain приступает к инициализации.
// Заполнение структуры WNDCLASS для регистрации класса окна.
memset(&wc, 0, sizeof(wc));
wc.lpszClassName = ClassName;                   // Имя класса окон
wc.lpfnWndProc = (WNDPROC)WndProc;              // Адрес оконной функци
wc.style = CS_HREDRAW|CS_VREDRAW;               // Стиль класса окон
wc.hInstance = hInstance;                       // Экземпляр приложения
wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);      // Пиктограмма для окон
wc.hCursor = LoadCursor(NULL,IDC_ARROW);        // Курсор мыши для окон
wc.hbrBackground =(HBRUSH)GetStockObject(WHITE_BRUSH);// Кисть для окон
        wc.lpszMenuName = NULL;                         // Ресурс меню окон
        wc.cbClsExtra = 0;                              // Дополнительная память
        wc.cbWndExtra = 0;                              // Дополнительная память

        // Регистрация класса окна.
        RegisterClass(&wc);

        // Создаем главное окно приложения.
        hWnd = CreateWindow(
                ClassName,              // Имя класса окон
                AppTitle,               // Заголовок окна
                WS_OVERLAPPEDWINDOW,    // Стиль окна
                CW_USEDEFAULT,          // X-координаты
                CW_USEDEFAULT,          // Y-координаты
                CW_USEDEFAULT,          // Ширина окна
                CW_USEDEFAULT,          // Высота окна
                NULL,                   // Дескриптор окна-родителя
                NULL,                   // Дескриптор меню окна
                hInst,                  // Дескриптор экземпляра приложения
                NULL);                  // Дополнительная информация
        if(!hWnd)
        {
```

```
                // Окно не создано, выдаем предупреждение.
                MessageBox(NULL,"Create: error", AppTitle, MB_OK|MB_ICONSTOP);
                return FALSE;
        }

        // Отображаем окно.
        ShowWindow(hWnd, nCmdShow);

        // Обновляем содержимое клиентской области окна.
        UpdateWindow(hWnd);

// Запускаем цикл обработки очереди сообщений. Функция GetMessage получает
// сообщение из очереди, выдает false при выборке из очереди сообщения WM_QUIT
        while(GetMessage(&msg, NULL, 0, 0))
        {
        // Преобразование некоторых сообщений, полученных с помощью клавиатуры
                TranslateMessage(&msg);

                // Отправляем сообщение оконной процедуре
                DispatchMessage(&msg);
        }

        return msg.wParam;
}

// --- Функция окна
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        char *str = "First Windows aplication";

        switch(msg)
        {
                // Необходимо обновить содержимое клиентской области окна.
                case WM_PAINT:
                {
                HDC hDC;
                PAINTSTRUCT ps;

                hDC = BeginPaint(hWnd, &ps);          // Получить контекст окна
                TextOut(hDC, 20, 20, str, strlen(str));       // Нарисовать текст
                EndPaint(hWnd, &ps);          // Освободить контекст окна
                }; break;

                // Нажата левая клавиша мыши в клиентской области окна.
                case WM_LBUTTONDOWN:
                {
MessageBox(hWnd, "32-bit aplication", "Window", MB_OK|MB_ICONINFORMATION);

                }; break;

                // Пользователь удалил окно.
                case WM_DESTROY:
                {
        // Если данная функция является оконной функцией главного окна, то
        // следует в очередь сообщений приложения послать сообщение WM_QUIT
```

```
                PostQuitMessage(0);
            }; break;

            // Необработанные сообщения передаем в стандартную
            // функцию обработки сообщений по умолчанию.
            default: return DefWindowProc(hWnd, msg, wParam, lParam);
        }
        return 0l;
}
```

Точка входа в программу

При запуске приложения в Windows ОС вызывает в программе функцию WinMain. Самая важная задача этой функции – создание основного окна программы, с которым должен быть связан код, способный обрабатывать ***сообщения***, передаваемые ОС этому окну.

Существенное различие между консольной и Windows-программами заключается в том, что первая для получения данных, введенных пользователем, вызывает ОС, а вторая делает это через **сообщения**, передаваемые программе операционной системой.

Точкой входа программы для Windows является функция WinMain, которая всегда определяется следующим образом:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow)
```

Эта функция использует последовательность вызовов WinAPI и, по своему завершению, возвращает системе Windows целое число.

Назначение параметров функции WinMain.

- Параметр hInstance называется **дескриптором экземпляра приложения**. Дескриптор экземпляра приложения - это уникальное число, идентифицирующее программу, когда она работает под Windows. Каждая копия одной и той же запущенной несколько раз программы называется «экземпляром» и у каждой свое значение hInstance.
- Параметр hPrevInstance в настоящее время устарел и в Win32 всегда равен NULL.
- Параметр lpCmdLine является указателем на оканчивающуюся нулем строку, в которой содержаться параметры, переданные программе из командной строки.
- Параметр nCmdShow определяет, как приложение первоначально отображается на экране: пиктограммой (nCmdShow = SW_SHOWMINNOACTIVE) или в виде открытого окна (nCmdShow = SW_SHOWNORMAL).

Действия, обычно выполняемые функцией WinMain:

Если найдена работающая копия приложения, то работа новой копии прекращается.

Иначе выполняются следующие действия:

1) Сохранение дескриптора экземпляра приложения в глобальной переменной.
2) Регистрация класса окна приложения ( ) и другие инициализации.
3) Создание главного окна приложения (CreateWindow( ) или CreateWindowEx( )) и, возможно, других, подчиненных окон.
4) Отображение созданного окна и отрисовка содержимого его внутренней части (ShowWindow( )).
5) Запуск цикла обработки сообщений (WM_), помещаемых в очередь приложения.
6) Завершение работы приложения при извлечении из очереди сообщения **WM_QUIT**.

Поиск работающей копии приложения

При запуске любого приложения пользователь может не помнить, какие приложения уже запущены, а какие нет. Когда он запускает приложение, то ожидает, что на экране появится его главное окно. Поэтому, если приложение было запущено ранее, целесообразно активизировать и выдвинуть на передний план его главное окно. Это именно то, к чему приготовился пользователь.

Для проверки того было ли приложение запущено ранее, можно воспользоваться функцией FindWindow, которая позволяет найти окно верхнего уровня по имени класса или по заголовку окна:

```
HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName);
```

Через параметр lpClassName передается указатель на текстовую строку, в которую необходимо записать имя класса искомого окна. На базе одного и того же класса может быть создано несколько окон. Если необходимо найти окно с заданным заголовком, то имя заголовка следует передать через параметр lpWindowName. Если же подойдет любое окно, то параметр lpWindowName может иметь значение NULL.

Регистрация класса окна

Обычно приложение создает окно за два шага. Сначала с помощью функции RegisterClass( ) регистрируется класс окна, а затем создается само окно ***зарегистрированного*** класса с помощью функции CreateWindow (CreateWindowEx).

Класс окна определяет общее поведение нового типа окон, включая адрес новой оконной процедуры. Такие параметры, как размер, расположение и внешний вид окна определяются при его создании.

Новый класс окна регистрируется при вызове приложением следующей функции:

```
ATOM  RegisterClass(const WNDCLASS *lpwc);
```

Единственный параметр этой функции lpwc указывает на структуру типа WNDCLASS, описывающую **тип** (см. MSDN) нового окна.

Возвращаемое значение является ***атомом*** Windows – 16-разрядным значением, идентифицирующим уникальную символьную строку в таблице Windows.

Приведем фрагмент кода функции WinMain, в котором осуществляется заполнение полей структуры, описывающей класс окна, и регистрация класса окна:

```
        . . .
// Сохраняем дескриптор экземпляра приложения в глобальной переменной hInst типа
// HINSTANCE, чтобы при необходимости воспользоваться им в функции окна.

hInst = hInstance;

// Заполнение структуры WNDCLASS для регистрации класса окна.
WNDCLASS wc;
memset(&wc, 0, sizeof(wc));                    // Очистка полей структуры
wc.lpszClassName = "MyWindows";        // Имя класса окон - строка
wc.lpfnWndProc =(WNDPROC)WndProc;              // Адрес оконной функции
wc.style = CS_HREDRAW | CS_VREDRAW;            // Стиль класса окон
wc.hInstance = hInst;                          // Экземпляр приложения
wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);     // Пиктограмма для окон
```

```
wc.hCursor = LoadCursor(NULL,IDC_ARROW);              // Курсор мыши для окон
wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);      // Кисть для окон
wc.lpszMenuName = NULL;                          // Ресурс меню окон
wc.cbClsExtra = 0;                               // Дополнительная память
wc.cbWndExtra = 0;                               // Дополнительная память
// Регистрация класса окна
       RegisterClass( &wc );
       . . .
```

Поля структуры WNDCLASS

Название нового регистрируемого класса окон задается при помощи параметра – строки lpszClassName.

Наиболее важными параметрами структуры WNDCLASS являются style и lpfnWndProc.

Основная часть того, что представляет окно как уникальный объект системы Windows, управляется через **стиль класса окна** и **оконную процедуру**.

Параметр lpfnWndProc указывает адрес функции оконной процедуры. Эта функция отвечает за обработку всех ***сообщений***, получаемых окном. Она может обрабатывать эти сообщения сама или вызывать оконную процедуру по умолчанию DefWindowProc.

Сообщения могут быть разнообразными: изменение размера и перемещение окна, события от мыши, клавиатуры, команды меню, запросы на перерисовку, события от таймера и другого аппаратного обеспечения и т.д.

Некоторые глобальные характеристики окна управляются через параметр стиля окна – style (WS_). Для этого параметра можно установить комбинацию значений, используя операцию поразрядного ИЛИ (|).

Например, значение CS_DBLCLKS указывает Windows генерировать события о двойном щелчке кнопкой мыши. Значения CS_HREDRAW | CS_VREDRAW указывают, что окно должно перерисовываться полностью каждый раз при изменении горизонтального или вертикального размера.

Приведем назначение остальных, менее важных для функционирования окна, полей структуры WNDCLASS.

- поле hIcon – дескриптор пиктограммы, используемой для представления минимизированных окон этого класса;
- поле hCursor – дескриптор стандартного указателя мыши для окон этого класса;
- hbrBackground – дескриптор кисти интерфейса GDI, используемой для рисования фона окна;
- строка lpszMenuName определяет ресурс меню (по имени или с помощью макроса MAKEINTRESOURCE по целому идентификатору), который используется для стандартного меню этого класса;
- параметры cbClsExtra и cbWndExtra используются, чтобы выделить дополнительную память для класса окна и для отдельных окон, через них задается размер этой дополнительной памяти. Приложения могут использовать эту дополнительную память для хранения специальной информации приложения, относящейся к классу окна или к отдельным окнам.

Создание окна

Регистрация нового класса является первым шагом в создании окна. Затем приложение должно создать само окно с помощью функции CreateWindow, которая возвращает дескриптор созданного окна типа HWND:

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

У каждого окна в Windows имеется уникальный дескриптор типа HWND. Дескриптор окна – это один из важнейших описателей, которыми оперирует программа для Windows. Для многих функций Windows требуется дескриптор окна, благодаря которому Windows «знает», к какому окну применить функцию.

Пример фрагмента кода функции WinMain, в котором осуществляется создание окна зарегистрированного ранее класса с именем "MyWindows":

```
        . . .
// Создаем главное окно приложения.
HWND hWndMain = CreateWindow(
        "MyWindows",           // Имя класса окон
        "My Window",           // Заголовок окна
        WS_OVERLAPPEDWINDOW,           // Стиль окна
        CW_USEDEFAULT,CW_USEDEFAULT,   // X- и Y-координаты
        CW_USEDEFAULT,CW_USEDEFAULT,   // Ширина и высота окна
        NULL,                          // Дескриптор окна-родителя
        NULL,                          // Дескриптор меню окна
        hInst,                         // Дескриптор экземпляра приложения
        NULL);                         // Дополнительная информация
        if(!hWndMain)
        {
                       // Окно не создано, выдаем предупреждение.
        MessageBox(NULL,"Create:error", "My Window", MB_OK | MB_ICONSTOP);
        return FALSE;
                }
        . . .
```

Параметры функции CreateWindow

Первый параметр lpClassName указывает имя класса, поведение которого наследует данное окно. Этот класс должен быть зарегистрирован с помощью функции RegisterClass или быть одним из предопределенных классов элементов управления. Эти предопределенные классы включают в себя стандартные классы элементов управления с именами “button”, “combobox”, “listbox”, “edit”, “scrollbar” и “static”.

Параметр lpWindowName определяет строку, которая выводится в заголовке создаваемого окна.

Параметр dwStyle определяет стиль окна. Не следует путать стиль окна со стилем класса, передаваемым в RegisterClass через структуру WNDCLASS.

Хотя стиль класса определяет некоторые постоянные свойства окон, принадлежащих классу, стиль окна, передаваемый ф. CreateWindow, используется для инициализации локальных свойств окна.

Как и в случае стиля класса, стиль окна также обычно является комбинацией значений (объединенных операцией поразрядного ИЛИ).

Замечание. Для определения стиля главного окна чаще всего используют стиль перекрывающегося окна, для чего через параметр dwStyle передают символическую константу WS_OVERLAPPEDWINDOW, определенную во включаемых файлах следующим образом:

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED|
        WS_CAPTION | WS_SYSMENU | WS_THICKFRAME|
        WS_MINIMIZEBOX | WS_MAXIMIZEBOX)
```

После указания типа окна нужно задать начальные геометрические размеры окна. Если при задании параметров x, y, nWidth и nHeight использовать константу CW_USEDEFAULT, то Windows установит расположение и размеры окна самостоятельно.

При создании окна указываются также дескрипторы его окна-родителя и меню. Если окно является ***главным*** окном приложения, то параметру hWndParent присваивается значение NULL. Значение NULL на месте дескриптора меню hMenu говорит о том, что у окна будет только меню класса, общее для всех окон этого класса.

При создании окна необходимо указать, какой ***экземпляр*** приложения, которое создает окно, что и делается при помощи параметра hInstance.

Последний параметр lpParam используется для передачи окну дополнительных данных (если их нет, то он должен быть равен NULL). При необходимости этот параметр используется в качестве указателя на какие-нибудь данные, на которые программа в дальнейшем могла бы ссылаться.

Отображение окна

Хотя функция **CreateWindow** и создает окно, это не значит, что оно будет автоматически отображаться на экране дисплея. Для отображения окна следует воспользоваться функцией **ShowWindow.** Первым параметром в эту функцию передается дескриптор окна, вторым параметром обычно (для главного окна приложения) является величина, передаваемая в качестве параметра функции WinMain, она задает начальный вид окна на экране.

```
BOOL ShowWindow(HWND hwnd, int nCmdShow);
```

Функция ShowWindow выводит окно на экран. Если второй параметр этой функции имеет значение SW_SHOWNORMAL, то фон рабочей области окна закрашивается той кистью, которая задана в классе окна.

Для перерисовки рабочей области затем необходимо сделать вызов функции:

```
void UpdateWindow(HWND hwnd);
```

Функция UpdateWindow передает функции (процедуре) окна сообщение WM_PAINT. Получив это сообщение, функция обновляет содержимое экрана.

Пример фрагмента кода функции WinMain, в котором осуществляется отображение созданного окна и отрисовка содержимого его внутренней части:

```
        . . .
ShowWindow(hWnd, nCmdShow);    // Отображаем окно
UpdateWindow( hWnd );  // Обновляем содержимое клиентской области окна
        . . .
```

Функции отображения и обновления вызываются, как правило, после создания окна, но порядок и место вызова функций ShowWindow и UpdateWindow не является обязательным. Окно может быть отображено не в момент создания, а позже, по желанию программиста.

Вывод: для отображения окна в типичном приложении необходимо:

1. Зарегистрировать ***класс окна*** функцией **RegisterClass**.
2. Создать окно функцией **CreateWindow**.
3. Отобразить окно функцией **ShowWindow**.
4. Обновить рабочую (клиентскую) область окна функцией **UpdateWindow**.

Цикл обработки очереди сообщений

После создания и отображения окна функция WinMain должна подготовить приложение к получению информации от пользователя через клавиатуру и мышь.

Windows поддерживает ***очередь сообщений (message queue)*** для каждой программы, работающей в данный момент в системе. Когда происходит ввод информации, Windows преобразовывает ее в сообщение, которое помещается в очередь сообщений приложения.

Программа извлекает сообщения из очереди сообщений, выполняя блок команд, известный как ***цикл обработки сообщений (message loop)***. Простейший цикл обработки сообщений имеет следующий вид:

```
        . . .
MSG  msg;
while( GetMessage(&msg, NULL, 0, 0))
        {
TranslateMessage( &msg ); // преобразование скэн-кода клавиши в сообщение
DispatchMessage( &msg );  // отправка сообщения окну-адресату приложения
        }
        . . .
```

Цикл обработки начинается с извлечения сообщений из очереди при помощи функции:

```
BOOL GetMessage(MSG FAR *lpmsg, HWND hwnd, UINT uMsgFilterMin, UINT
                uMsgFilterMax);
```

Если при вызове этой функции указать вторым аргументом NULL, то программа будет получать сообщения от всех окон, созданных программой. При помощи параметров uMsgFilterMin и uMsgFilterMax можно отфильтровать сообщения, получаемые программой. Если на их месте передать 0, то программа будет получать **все** сообщения.

После вызова функции GetMessage приложение получает структуру msg типа MSG с информацией о сообщении. Структура MSG содержит информацию об окне, которому предназначено сообщение, численное значение самого сообщения, время и координаты (для сообщений от мыши) возникновения сообщения, а также два дополнительных параметра wParam и lParam, смысл и значение которых зависят от особенностей сообщения.

Каждое получаемое приложением сообщение (за исключением WM_QUIT) направлено одному из окон приложения. Поскольку приложение не должно прямо вызывать функцию обработки окна, для передачи сообщения нужному окну используется функция:

```
LONG DispatchMessage(const MSG FAR *lpmsg);
```

Эта функция передает msg обратно в Windows. Windows отправляет сообщение для его обработки соответствующей оконной процедуре – таким образом, Windows вызывает оконную процедуру.

После того, как оконная функция (WndProc( )) обработает сообщение, Windows возвращает управление в приложение к следующему за вызовом DispatchMessage коду (см. пример), и цикл обработки сообщений снова возобновляет работу, вызывая GetMessage. Если поле message сообщения msg, извлеченного из очереди сообщений, равно любому значению, кроме WM_QUIT, то функция GetMessage возвращает ненулевое значение. Сообщение WM_QUIT, извлеченное из очереди приложения заставляет ***прервать*** цикл обработки сообщений.

Перед вызовом функции DispatchMessage могут быть помещены специальные функции, производящие над помещенными в очередь сообщениями какие-то действия. Например, преобразование некоторых сообщений, полученных с помощью клавиатуры, в более понятные сообщения можно при помощи функции

```
BOOL TranslateMessage(const MSG FAR* lpmsg);
```

Завершение работы приложения

Приложение завершает свою работу тогда, когда функция WinMain передает управление Windows. Передать управление можно в любом месте WinMain, в том числе и до входа в цикл обработки сообщений.

Однако после входа в цикл обработки сообщений единственным способом завершить приложение является посылка в очередь приложения сообщения WM_QUIT. Сообщение WM_QUIT, таким образом, является средством завершения работы приложения. Оно обычно помещается в очередь сообщений только функцией главного окна данного приложения. Делается это вызовом функции PostQuitMessage.

Когда функция GetMessage извлекает сообщение WM_QUIT из очереди, она возвращает нуль, что приводит к завершению работы приложения.

Пример фрагмента кода функции WinMain, в котором осуществляется завершение работы приложения:

```
        . . .
// цикл обработки сообщений
while( GetMessage(&msg, NULL, 0, 0))
 {
 ...
 }                              // завершается после получения WM_QUIT
. . .
return TRUE;                    // завершение работы приложения
```

Оконная процедура

После того, как класс окна зарегистрирован, окно создано и отображено и приложение вошло в цикл обработки сообщений начинается обработка поступающих сообщений, при этом главная роль отводится ***функции (процедуре) окна***.

Реальная работа приложения происходит в оконных функциях. Именно оконная процедура определяет то, что выводится в рабочую область окна и то, как окно реагирует на пользовательский ввод.

Оконной процедуре можно назначать любое имя, в программе Windows может содержаться более одной оконной процедуры.

Оконная процедура всегда связана с определенным классом окна, который регистрируется при помощи функции RegisterClass. Функция CreateWindowEx (устар. CreateWindow) создает окно на основе ***определенного*** класса (CS_) окна. На основе одного и того же класса можно создать ***несколько*** окон.

Оконная процедура всегда определяется следующим образом:

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

Все четыре параметра оконной процедуры (WndProc) идентичны первым четырем полям структуры MSG. Первым параметром (hWnd) является дескриптор окна, получающего сообщение. Если в программе создается ***несколько*** окон на основе одного и того же класса окна (и, следовательно, одной и той же оконной процедуры), тогда параметр hWnd идентифицирует конкретное окно, которое получает сообщение.

Функция окна имеет одно очень важное свойство: она вызывается непосредственно ОС Windows и не может вызываться приложением напрямую.

Простейшая функция окна выглядит следующим образом:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
        {
        return( DefWindowProc(hWnd, message, wParam, lParam));
        }
```

Функция DefWindowProc играет ***ключевую*** роль в формировании информационных потоков сообщений Windows, и ее указание в функции окна **обязательно**. Перед тем, как передать управление функции DefWindowProc, можно перехватить и обработать практически любое сообщение.

Обработка сообщений

Каждое получаемое окном сообщение идентифицируется ***номером***, который содержится в параметре msg оконной процедуры. Если оконная процедура обрабатывает сообщение, то ее возвращаемым значением должен быть 0.

Все сообщения, не обрабатываемые оконной процедурой, должны передаваться функции DefWindowProc (такой механизм позволяет Windows обрабатывать окно совместно с приложением). При этом значение, возвращаемое функцией DefWindowProc, должно быть ***возвращаемым значением*** оконной процедуры.

Типичный вид функции окна:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
switch (msg)
{
case WM_PAINT:    // Обновить содержимое клиентской области окна
{
                              . . .
}; break;
```

```
case WM_DESTROY: // Пользователь удалил окно
{
                              . . .
                              // В функции главного окна следует в очередь
                              // сообщений послать сообщение WM_QUIT
PostQuitMessage(0);
}; break;

default: return DefWindowProc(hWnd, msg, wParam, lParam);
}
return 0;
}
```

Функция окна получает сообщение из двух источников: из цикла обработки сообщений и от Windows:

- Из цикла обработки сообщений поступают сообщения ввода: перемещение и нажатие клавиш мыши, нажатие и отпускание клавиш клавиатуры и, если установлен генератор событий таймера, сообщения от таймера.
- Windows посылает функции окна сообщения поддержки окна напрямую, минуя очередь сообщений приложения и цикл обработки сообщений. Эти сообщения обычно вызваны событиями, требующими немедленной реакции по изменению вида окна.

Отображение содержимого окна, сообщение WM_PAINT

Сообщение WM_PAINT сообщает программе, что часть или вся рабочая область окна ***недействительна (invalid)***, и ее следует ***перерисовать***.

Ситуации, когда рабочая область может становиться недействительной:

- При первом создании окна недействительна вся рабочая зона (другое название – клиентская часть) окна, поскольку программа еще ничего не нарисовала. Сообщение WM_PAINT, посылаемое, когда приложение вызывает функцию UpdateWindow, заставляет оконную процедуру что-то(?) нарисовать в рабочей области.
- Когда пользователь изменяет размер окна, в стиле которого заданы флаги CS_HREDRAW и CS_VREDRAW, рабочая область также становится недействительной. Операционная система вслед за этим посылает в оконную процедуру сообщение WM_PAINT.
- Когда пользователь минимизирует окно программы, а затем вновь восстанавливает до прежнего размера, то в Windows содержимое рабочей области не сохраняется (в графической среде это бы привело к тому, что пришлось бы хранить слишком много данных). Вместо этого Windows делает недействительным ***все*** окно. Затем оконная процедура получает сообщение WM_PAINT и сама восстанавливает содержимое окна.
- Когда пользователь перемещает окна так, что они перекрываются, Windows не сохраняет ту часть окна, которая закрывается другим окном. Когда эта часть позже открывается, Windows отмечает эту область как недействительную. Оконная процедура получает сообщение WM_PAINT для восстановления содержимого окна.
- Если по логике работы приложения при обработке того или другого сообщения требуется изменить содержимое окна, то приложение может само при помощи функции InvalidateRect объявить любую область окна как недействительную (т.е. требующую

обновления), а затем сообщить Windows, что необходимо перерисовать часть или все окно при помощи функции UpdateWindow:

```
InvalidateRect(hWnd, NULL, TRUE);
UpdateWindow( hWnd );
```

Замечание. Первый параметр функции InvalidateRect является идентификатором (дескриптором) окна, для которого выполняется операция. Второй параметр - указатель на структуру типа **RECT**, определяющую прямоугольную область, подлежащую обновлению (если он равен NULL, то недействительной объявляется вся внутренняя часть окна). Третий параметр определяет необходимость стирания фона окна (если параметр задан как TRUE, фон окна подлежит стиранию).

Вывод. Сообщение WM_PAINT посылается ОС окну, когда его часть требует перерисовки, и в очереди сообщений потока, владеющего этим окном, не находится других необработанных сообщений.

Если окно содержит одну или несколько областей, подлежащих обновлению, то приложение получает одно сообщение WM_PAINT, в котором определена область, охватывающая все указанные области.

Обработка сообщения WM_PAINT почти всегда начинается с вызова функции BeginPaint:

```
. . .
case WM_PAINT: // Обновить содержимое клиентской области окна
{
        PAINTSTRUCT  ps;
        HDC hDC = BeginPaint(hWnd, &ps);
        . . . // вызов функций GDI для контекста HDC
```

а заканчивается вызовом функции EndPaint:

```
        . . . // вызов функций GDI для контекста HDC
        EndPaint(hWnd, &ps);
}; break;
```

В обеих функциях первый параметр – это дескриптор окна приложения (hWnd), а второй (&ps) – это указатель на структуру типа PAINTSTRUCT. В этой структуре содержится некоторая информация, которую оконная процедура может использовать для рисования в рабочей области окна.

В частности, одно из полей этой структуры представляет собой самый маленький прямоугольник, содержащий все области окна, требующие перерисовки. Ограничив свои действия по рисованию только этой прямоугольной областью, приложение может ускорить процесс перерисовки.

При обработке вызова BeginPaint, Windows производит следующие действия:

- обновляет фон рабочей области с помощью кисти (brush), которая указывалась при регистрации класса окна;
- делает всю рабочую область ***действительной*** (не требующей перерисовки);
- возвращает дескриптор контекста устройства (например, pDC, этот дескриптор необходим для вывода в рабочую область текста и графики).

При использовании дескриптора контекста устройства, возвращаемого функцией BeginPaint, приложение не может рисовать вне рабочей области. Функция EndPaint освобождает дескриптор устройства, после чего приложению его нельзя использовать.

Замечание 1. Функцию BeginPaint следует вызывать только в ответ на сообщение WM_PAINT. Каждый вызов этой функции должен сочетаться с последующим вызовом функции EndPaint.

Замечание 2. Так как приложение практически никогда при обработке сообщения WM_PAINT «не знает» размеров всей своей клиентской области, то перед рисованием оно может получить эту информацию при помощи функции:

```
BOOL GetClientRect(HWND hWnd, LPRECT rect);
```

Первый параметр – дескриптор окна, второй – указатель на переменную типа RECT, в эту переменную функция GetClientRect помещает информацию о размере рабочей области в пикселях.

Удаление окна, сообщение WM_DESTROY

Сообщение WM_DESTROY показывает, что Windows находится в процессе ликвидации окна в ответ на полученную от пользователя команду “Закрыть”, выберет пункт “Закрыть” из системного меню или нажмет комбинацию клавиш Alt+F4.

Главное окно стандартно реагирует на это сообщение, вызывая функцию:

```
PostQuitMessage( 0 );
```

Эта функция ставит сообщение WM_QUIT в очередь сообщений приложения. Это заставляет функцию WinMain прервать цикл обработки сообщений и выйти в систему, завершив работу приложения.

Список литературы

Основная

1. Федорова Г.Н. Разработка программных модулей программного обеспечения для компьютерных систем: учебник для студ. учреждений сред. проф. образования/ Г.Н. Федорова.- М.: Издательский центр "Академия", 2016.- 336 с.

Дополнительная

1. Архипова Е.Н. Программирование для Windows: введение в интерфейс Win API.
2. Безруков В.А. Win32 API. Программирование/ учебное пособие. – СПб: СПбГУ ИТМО, 2009. – 90 с.
3. Гунько А.В. Программирование (в среде Windows): учебное пособие/ А.В. Гунько – Новосибирск: Изд-во НГТУ, 2019.– 155 с.
4. Литвиненко Н. А. Технология программирования на C++. Win32 API-приложения. — СПб.: БХВ-Петербург, 2010. — 288 с.
5. Побегайло А. П. Системное программирование в Windows. — СПб.: БХВ-Петербург, 2006. - 1056 с.

6. Саймон Ричард. Microsoft Windows API. Справочник системного программиста. Второе издание, дополненное: Пер. с англ./Ричард Саймон — К.: ООО «ТИД «ДС», 2004. — 1216 с.
7. Щупак Ю.А. Win32 API. Разработка приложений для Windows. - СПб.: Питер, 2008. - 592 с.
8. Харт, Джонсон, М. Системное программирование в среде Windows, 3-е издание.: Пер. с англ.- М.: Издательский дом "Вильямс", 2005.- 592 с.

Контрольные вопросы:

1. Какая функция является точкой входа в Windows-программу?
2. Что такое дескриптор копии приложения? Что он идентифицирует?
3. Какие действия обычно выполняет функция WinMain?
4. Для чего необходимо регистрировать класс окна?
5. Какие характеристики класса окон можно задать при его регистрации? Какие поля структуры, описывающей класс окна, являются наиболее важными?
6. Что такое оконная процедура? Для чего она предназначена? Кто ее вызывает в процессе работы приложения?
7. Может ли приложение создавать окна, принадлежащие классам, которые приложение самостоятельно не регистрировало?
8. Что такое дескриптор окна? Для чего он необходим?
9. После создания окна оно обязательно должно появиться на экране? Что необходимо сделать для отображения окна?
10. Для чего служит цикл обработки сообщений? Откуда приложение извлекает очередное сообщение? Каким образом и кем сообщения создаются?
11. Сколько очередей сообщений существует? Для кого предназначены сообщения?
12. Каким образом цикл обработки сообщений прерывается, давая возможность приложению завершить свою работу? Кто обычно посылает сообщение (и какое), которое прерывает цикл обработки сообщений?
13. Из каких источников оконная процедура получает сообщения?
14. Какое сообщение приходит окну, если часть его рабочей области (или вся она) требует перерисовки? Что такое недействительная область?
15. В каких ситуациях рабочая область может становиться недействительной?
16. С вызова какой функции должна начинаться обработка сообщения WM_PAINT? Вызовом какой функции она должна заканчиваться? Для чего вызываются эти функции?
17. Как обычно реагирует стандартное главное окно приложения на получение сообщения о его удалении? Что при этом происходит с приложением?
18. Когда приходит сообщение об удалении окна? Что произойдет, если остальные окна приложения будут реагировать на это сообщение так же, как и главное окно?

Задание: ответить на контрольные вопросы и прислать их на проверку (файл должен быть в формате pdf)