Модуль №1

Занятие №3

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Принципы обработки сообщений мыши.
3. Принципы обработки нажатия клавиш.
4. Практическая часть.
5. Подведение итогов.
6. Домашнее задание.

1. Повторение пройденного материала

Данное занятие необходимо начать с краткого повторения материала предыдущего занятия. При общении со слушателями можно использовать следующие контрольные вопросы:

1) Для какой цели служит утилита Spy++?
2) Почему рекомендуется использование венгерской нотации при именовании переменных?
3) Чем отличается спецификация **__cdecl** от **__stdcall (WINAPI, CALLBACK)**?
4) Какова структура минимального WinAPI-приложения?
5) Какие основные действия выполняются в главной функции программы?
6) Какие действия выполняются в цикле обработки сообщений, и при каком условии этот цикл прерывается?
7) Какую роль в приложении играет оконная процедура?
8) Что такое функция обратного вызова (CALLBACK–функция)?

9) Какое значение должна вернуть оконная процедура, если она выполняет обработку сообщения?
10) Какое значение должна вернуть оконная процедура, если для сообщения обработка не выполняется?
11) Что такое окно сообщения?

2. Принципы обработки сообщений мыши

Переходя к рассмотрению данного вопроса, обратить внимание слушателей, что оконная процедура получает сообщения мыши в случае, если мышь проходит через окно, а также при щелчке внутри окна, даже если окно не активно или не имеет фокуса ввода. Если мышь перемещается по клиентской области окна, то оконная процедура получает сообщение **WM_MOUSEMOVE**. Если кнопка мыши нажимается или отпускается внутри клиентской области, то оконная процедура получает следующие сообщения:

- **WM_LBUTTONDOWN** – нажата левая кнопка мыши;
- **WM_MBUTTONDOWN** - нажата средняя кнопка мыши;
- **WM_RBUTTONDOWN** – нажата правая кнопка мыши;
- **WM_LBUTTONUP** – отпущена левая кнопка мыши;
- **WM_MBUTTONUP** – отпущена средняя кнопка мыши;
- **WM_RBUTTONUP** – отпущена правая кнопка мыши.
- **WM_LBUTTONDBLCLK** – двойной щелчок левой кнопкой мыши;
- **WM_MBUTTONDBLCLK** – двойной щелчок средней кнопкой мыши;
- **WM_RBUTTONDBLCLK** - двойной щелчок правой кнопкой мыши.

Прокрутка колесика вызывает сообщение **WM_MOUSEWHEEL**.

Для всех этих сообщений значение параметра **lParam** содержит положение мыши. При этом в младшем слове (младшие 2 байта) находится значение координаты **x**, а в старшем слове (старшие 2 байта) — значение координаты **y**. **Отсчет координат ведется от левого верхнего угла клиентской области окна**. Эти значения можно извлечь из **lParam** при помощи макросов **LOWORD** и **HIWORD**.

```
    WORD LOWORD(DWORD dwValue); // Возвращает младшее слово из указанного
    // 32-битного значения
    WORD HIWORD(DWORD dwValue); // Возвращает старшее слово из указанного
    // 32-битного значения
```

Следует особо подчеркнуть, что окно будет получать сообщения о двойном щелчке (**DBLCLK**) только в том случае, если стиль соответствующего класса окна содержит флаг **CS_DBLCLKS**. Поэтому перед регистрацией класса окна нужно присвоить полю **style** структуры **WNDCLASSEX** значение, включающее этот флаг. Если класс окна определен без флага **CS_DBLCLKS** и пользователь делает двойной щелчок левой кнопкой мыши, то оконная процедура последовательно получает сообщения **WM_LBUTTONDOWN**, **WM_LBUTTONUP**, **WM_LBUTTONDOWN** и **WM_LBUTTONUP**. Если класс окна определен с флагом **CS_DBLCLKS**, то после двойного щелчка оконная процедура получит сообщения **WM_LBUTTONDOWN**, **WM_LBUTTONUP**, **WM_LBUTTONDBLCLK** и **WM_LBUTTONUP**.

Модифицировать стиль класса окна можно также вызовом функции API **SetClassLong**, которая позволяет, в общем случае, изменить для класса указанного окна 32-битное значение, связанное со структурой **WNDCLASSEX**:

```
DWORD SetClassLong(
    HWND hWnd, // дескриптор окна
    int nIndex, // значение, определяющее, что нужно изменить, например:
      // GCL_STYLE – изменить стиль окна,
      // GCL_HICON – изменить дескриптор курсора,
      // GCL_HCURSOR – изменить дескриптор иконки
    LONG dwNewLong // новое 32-битное значение
);
```

Функция API **GetClassLong** позволяет получить 32-битное значение из структуры **WNDCLASSEX**, связанной с классом указанного окна:

```
DWORD GetClassLong(
    HWND hWnd, // дескриптор окна
    int nIndex // значение, определяющее, что нужно получить из WNDCLASSEX
);
```

Пример изменения стиля класса окна:

```
UINT style = GetClassLong(hWnd, GCL_STYLE);
SetClassLong(hWnd, GCL_STYLE, style | CS_DBLCLKS);
```

3. Принципы обработки нажатия клавиш

Ознакомив слушателей с принципами обработки сообщений мыши, следует перейти к обсуждению принципов обработки нажатия клавиш. Отметить, что одно из широко используемых сообщений порождается при нажатии клавиши. Это сообщение называется **WM_CHAR**. Для сообщений **WM_CHAR** параметр **wParam** содержит ASCII-код нажатой клавиши. **LOWORD** (**lParam**) содержит количество повторов, генерируемых при удерживании клавиши в нажатом положении. **HIWORD** (**lParam**) представляет собой битовую карту со следующими значениями битов:

- 15: равен 1, если клавиша отпущена, и 0, если она нажата.
- 14: устанавливается, если клавиша уже была нажата перед посылкой сообщения.
- 13: устанавливается в 1, если дополнительно нажата клавиша <**Alt**>.
- 12-9: используется системой.
- 8: устанавливается в 1, если нажата клавиша функциональной или дополнительной части клавиатуры.
- 7-0: код клавиши (scan-код).

Символьные сообщения **WM_CHAR** передаются в оконную процедуру в промежутке между аппаратными сообщениями клавиатуры. Например, если пользователь, удерживая клавишу <**Shift**>, нажимает клавишу <**A**>, отпускает клавишу <**A**> и затем отпускает клавишу <**Shift**>, то оконная процедура получит пять сообщений:

Сообщение	**Виртуальная клавиша или ANSI-код**
WM_KEYDOWN	Виртуальная клавиша VK_SHIFT
WM_KEYDOWN	Виртуальная клавиша A
WM_CHAR	ANSI-код символа A
WM_KEYUP	Виртуальная клавиша A
WM_KEYUP	Виртуальная клавиша VK_SHIFT

Следует подчеркнуть, что имеет смысл обрабатывать только те аппаратные сообщения **WM_KEYDOWN** и **WM_KEYUP**, которые содержат (в **wParam**) виртуальные коды для клавиш управления курсором, клавиш <**Shift**>,<**Ctrl**>,<**Alt**> функциональных клавиш (**VK_LEFT, VK_DOWN, VK_SHIFT, VK_CTRL, VK_MENU, VK_RETURN, VK_TAB** и т.д.). В то же время аппаратные сообщения для символьных клавиш могут игнорироваться. Ввод информации от символьных клавиш гораздо удобнее обрабатывать, используя символьное сообщение **WM_CHAR**.

Получить состояние указанной виртуальной клавиши можно с помощью функции API **GetKeyState**. Это состояние показывает, нажата ли клавиша, отпущена или переключена в то или иное состояние.

```
SHORT GetKeyState (int nVirtKey);
```

Параметр **nVirtKey** задаёт код виртуальной клавиши. Возвращаемое значение – состояние клавиши, которое закодировано в двух битах. Если старший бит равен 1, то клавиша нажата, в ином случае, она отпущена. Если младший бит равен 1, то клавиша переключена, т.е. переведена во включенное состояние.

Отметить, что полный перечень всех макросов виртуальных клавиш представлен в файле **winuser.h.**

В качестве примера обработки событий мыши, а также обработки нажатия клавиш привести следующий код:

```
// Файл WINDOWS.H содержит определения, макросы, и структуры
// которые используются при написании приложений под Windows.
#include <windows.h>
#include <tchar.h>


//прототип оконной процедуры
LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

TCHAR szClassWindow[] = TEXT("Каркасное приложение"); /*Имя класса окна*/

INT WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                     LPTSTR lpszCmdLine, int nCmdShow)
{
      HWND hWnd;
      MSG Msg;
      WNDCLASSEX wcl;

      /* 1. Определение класса окна  */

      wcl.cbSize = sizeof (wcl);    // размер структуры WNDCLASSEX
      wcl.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS; // окно сможет
      // получать сообщения о двойном щелчке (DBLCLK)
      wcl.lpfnWndProc = WindowProc; // адрес оконной процедуры
      wcl.cbClsExtra = 0;     // используется Windows
      wcl.cbWndExtra  = 0;    // используется Windows
      wcl.hInstance = hInst;  // дескриптор данного приложения
      // загрузка стандартной иконки
      wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
      // загрузка стандартного курсора
      wcl.hCursor = LoadCursor(NULL, IDC_ARROW);
      // заполнение окна белым цветом
      wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
      wcl.lpszMenuName = NULL;      // приложение не содержит меню
      wcl.lpszClassName = szClassWindow;  //имя класса окна
      wcl.hIconSm = NULL; // отсутствие маленькой иконки


      /*  2. Регистрация класса окна  */

      if (!RegisterClassEx(&wcl))
            return 0;   // при неудачной регистрации - выход

      /*  3. Создание окна  */

      //создается окно и  переменной hWnd присваивается дескриптор окна
      hWnd = CreateWindowEx(
            0,          // расширенный стиль окна
            szClassWindow,    // имя класса окна
            TEXT("Каркас  Windows приложения"), // заголовок окна
            /* Заголовок, рамка, позволяющая менять размеры, системное меню,
                  кнопки развёртывания и свёртывания окна  */
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,    // x-координата левого верхнего угла окна
            CW_USEDEFAULT,    // y-координата левого верхнего угла окна
            CW_USEDEFAULT,    // ширина окна
```

```
            CW_USEDEFAULT,    // высота окна
            NULL,             // дескриптор родительского окна
            NULL,             // дескриптор меню окна
            hInst,            // идентификатор приложения, создавшего окно
            NULL);            // указатель на область данных приложения


      /* 4. Отображение окна */

      ShowWindow(hWnd, nCmdShow);
      UpdateWindow(hWnd);     // перерисовка окна

      /* 5. Запуск цикла обработки сообщений  */

      // получение очередного сообщения из очереди сообщений
      while (GetMessage(&Msg, NULL, 0, 0))
      {
            TranslateMessage(&Msg); // трансляция сообщения
            DispatchMessage(&Msg);  // диспетчеризация сообщений
      }
      return lpMsg.wParam;
}


LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMessage, WPARAM wParam,
                            LPARAM lParam)
{
      TCHAR str[50];
      switch(uMessage)
      {
            case WM_LBUTTONDBLCLK:
                  MessageBox(
                        0,
                        TEXT("Двойной щелчок левой кнопкой мыши"),
                        TEXT("WM_LBUTTONDBLCLK"),
                        MB_OK | MB_ICONINFORMATION);
                  break;
            case WM_LBUTTONDOWN:
                  MessageBox(
                        0,
                        TEXT("Нажата левая кнопка мыши"),
                        TEXT("WM_LBUTTONDOWN"),
                        MB_OK | MB_ICONINFORMATION);
                  break;
            case WM_LBUTTONUP:
                  MessageBox(
                        0,
                        TEXT("Отпущена левая кнопка мыши"),
                        TEXT("WM_LBUTTONUP"),
                        MB_OK | MB_ICONINFORMATION);
                  break;
            case WM_RBUTTONDOWN:
                  MessageBox(
                        0,
                        TEXT("Нажата правая кнопка мыши"),
                        TEXT("WM_RBUTTONDOWN"),
                        MB_OK | MB_ICONINFORMATION);
                  break;
            case WM_MOUSEMOVE:
```

```
                // текущие координаты курсора мыши
                wsprintf(str, TEXT("X=%d  Y=%d"),
                      LOWORD(lParam), HIWORD(lParam));
                SetWindowText(hWnd, str);//строка выводится в заголовок окна
                break;
          case WM_CHAR:
                wsprintf(str, TEXT("Нажата клавиша %c"),
                            (TCHAR) wParam); // ASCII-код нажатой клавиши
                MessageBox(0, str, TEXT("WM_CHAR"),
                            MB_OK | MB_ICONINFORMATION);
                break;
          case WM_DESTROY: // сообщение о завершении программы
                PostQuitMessage(0);     // посылка сообщения WM_QUIT
                break;
          default:
          // все сообщения, которые не обрабатываются в данной оконной
          // функции направляются обратно Windows на обработку по умолчанию
          return DefWindowProc(hWnd, uMessage, wParam, lParam);
    }
    return 0;
}
```

Необходимо дать подробные комментарии к приведенному коду. В частности, следует отметить, что при обработке сообщения **WM_MOUSEMOVE** с помощью функции **wsprintf** форматируется строка, содержащая текущие координаты мыши, для последующего вывода в заголовок окна. Вывод строки в заголовок окна осуществляется функцией API **SetWindowText**:

```
BOOL SetWindowText (
HWND hWnd, // дескриптор окна, в котором должен быть изменен текст
LPCTSTR lpString //указатель на строку, содержащую новый текст
);
```

Далее следует ознакомить слушателей с группой функций, позволяющих получать информацию о размерах и расположении окна, а также изменять размеры, расположение и характеристики отображения окна.

Функция API **GetWindowRect** позволяет получить размеры прямоугольника окна:

```
BOOL GetWindowRect(
HWND hWnd, //дескриптор окна
LPRECT lpRect //указатель на структуру RECT
);
```

```
typedef struct tagRECT {
LONG left;
LONG top;
LONG right;
LONG bottom;
} RECT;
```

Поля этой структуры задают координаты левого верхнего угла (**left, top**) и правого нижнего угла (**right, bottom**) прямоугольника.

Акцентировать внимание слушателей на том, что в структуре **RECT** будут указаны экранные координаты левого верхнего и правого нижнего углов окна. Отметить, что отсчёт координат ведётся относительно левого верхнего угла экрана (0,0).

Функция API **GetClientRect** позволяет получить размеры прямоугольника, охватывающего клиентскую (рабочую) область окна:

```
BOOL GetClientRect(
HWND hWnd, //дескриптор окна
LPRECT lpRect //указатель на структуру RECT
);
```

Акцентировать внимание слушателей на том, что в структуре **RECT** будут указаны координаты левого верхнего и правого нижнего углов клиентской области окна. Поскольку отсчёт координат в данном случае ведётся относительно левого верхнего угла рабочей области окна, то координаты (**left, top**) будут равны (0,0).

Функция API **MoveWindow** позволяет переместить окно, а также изменить его размеры:

```
BOOL MoveWindow(
    HWND hWnd,//дескриптор окна
    int X, //новая координата X левого верхнего угла окна
    int Y, //новая координата Y левого верхнего угла окна
```

```
            int nWidth, //новая ширина окна
            int nHeight, //новая высота окна
            BOOL bRepaint //необходимость немедленной перерисовки окна
        );
```

Функция API **BringWindowToTop** активизирует окно и переносит его в верхнее положение, если оно находится позади других окон:

```
        BOOL BringWindowToTop(
          HWND hWnd //дескриптор окна
        );
```

Далее следует ознакомить слушателей с двумя функциями поиска окон. Отметить, что для поиска окна верхнего уровня служит функция API **FindWindow**:

```
        HWND FindWindow(
            LPCTSTR lpClassName, //имя класса окна
            LPCTSTR lpWindowName // заголовок окна
        );
```

Напомнить слушателям, что имя класса окна, равно как и заголовок окна можно узнать, используя утилиту Spy++.

Затем привести прототип ещё одной поисковой функции API **FindWindowEx**, которая служит для поиска дочерних окон (например, для поиска элементов управления диалогового окна):

```
HWND FindWindowEx(
HWND hwndParent, //дескриптор родительского окна
HWND  hwndChildAfter,  //дескриптор  дочернего  окна,  после  которого  следует
//начать поиск, либо 0 – для поиска, начиная с первого дочернего окна
LPCTSTR lpszClass, //имя класса окна
LPCTSTR lpszWindow // заголовок окна
);
```

4. Практическая часть

1) Написать приложение, в котором ведётся подсчёт количества «кликов» левой, правой и средней кнопки мыши. Обновляемую статистику необходимо выводить в заголовок окна.
2) Предположим, что существует прямоугольник, границы которого на 10 пикселей отстоят от границ клиентской области окна. Необходимо при нажатии левой кнопки мыши выводить в заголовок окна сообщение о том, где произошел щелчок мышью: внутри прямоугольника, снаружи или на границе прямоугольника. При нажатии правой кнопки мыши необходимо выводить в заголовок окна информацию о размере клиентской области окна (ширина и высота клиентской области окна).

5. Подведение итогов

Подвести общие итоги занятия. Напомнить слушателям о том, какие существуют сообщения мыши и клавиатурные сообщения, проанализировав дополнительную информацию, которая приходит с этими сообщениями. Подчеркнуть, в каком случае следует обрабатывать сообщение WM_CHAR, а в каком случае – WM_KEYDOWN и WM_KEYUP. Напомнить слушателям о том, как получить состояние указанной виртуальной клавиши. Подводя итоги, акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

6. Домашнее задание

1) Написать приложение, позволяющее при нажатии левой кнопки мыши изменить текст в заголовке окна стандартного приложения «Калькулятор», а при нажатии правой кнопки мыши сместить вправо кнопку «пуск», изменив на ней надпись.

2) Написать приложение, обладающее следующей функциональностью:

- при нажатии кнопки <Enter> окно позиционируется в левый верхний угол экрана с размерами (300X300);
- с помощью клавиш управления курсором осуществляется перемещение окна.