Урок 4. Обработка сообщений клавиатуры, мыши, таймера

Обработка клавиатурных сообщений

При создании приложений рекомендуется реализовать все функциональные возможности программы не только с помощью мыши, но и с помощью клавиатуры.

Синхронизация событий клавиатуры

Приложение «узнает» о нажатиях клавиш посредством сообщений, которые посылаются оконной процедуре (WndProc). Когда пользователь ***нажимает*** и ***отпускает*** клавиши:

1. Драйвер клавиатуры передает информацию о нажатии клавиш в Windows.
2. Windows сохраняет эту информацию (в виде сообщений) в системной очереди сообщений.
3. Затем она передает сообщения клавиатуры, по одному за раз, в очередь сообщений программы (приложения), содержащей окно, имеющее **фокус ввода** (input focus).
4. Затем программа отправляет сообщения соответствующей оконной процедуре (с помощью функции DispatchMessage).

Смысл этого двухступенчатого процесса – сохранение сообщений в системной очереди сообщений и дальнейшая их передача в очередь сообщений приложения – в синхронизации.

Если пользователь печатает на клавиатуре быстрее, чем программа может обрабатывать поступающую информацию, Windows сохраняет информацию о дополнительных нажатиях клавиш в системной очереди сообщений, поскольку одно из дополнительных нажатий может быть переключением фокуса ввода на другую программу. Информацию о последующих нажатиях следует затем направлять в другую программу.

Таким образом, Windows корректно синхронизирует такие события клавиатуры.

Для отражения различных событий клавиатуры Windows посылает программам ***восемь*** различных сообщений. Программы могут игнорировать многие из них, так как в большинстве случаев, в этих сообщениях содержится значительно больше закодированной информации, чем нужно приложению.

Игнорирование событий клавиатуры

Хотя клавиатура является основным источником пользовательского ввода программ для Windows, приложению не обязательно реагировать на каждое получаемое от клавиатуры сообщение - Windows сама обрабатывает многие сообщения клавиатуры.

Приложению не нужно отслеживать нажатия клавиш, поскольку Windows извещает программу об эффекте, вызванном их нажатием. Например, когда пользователь выбирает пункт меню, Windows посылает программе сообщение, что выбран пункт меню, независимо от того, был ли он выбран с помощью мыши или клавиатуры.

В программах для Windows часто используются быстрые ("горячие") клавиши (keyboard accelerators) для быстрого доступа к часто употребляемым пунктам меню (в качестве быстрых клавиш обычно используют функциональные клавиши или различные комбинации клавиш, такие быстрые клавиши определяются в описании ресурсов про-

граммы). Windows сама преобразует быстрые клавиши в командные сообщения (WM_COMMAND) меню, приложению не нужно этого делать явно.

Окна диалога также имеют интерфейс клавиатуры, но обычно программам не нужно отслеживать клавиатурные события, когда активно окно диалога. Интерфейс клавиатуры обслуживается самой Windows, и Windows посылает сообщения программе о действиях, соответствующих нажимаемым клавишам.

В окнах диалога и внутри окон верхнего уровня могут содержаться окна редактирования (edit) для ввода текста (это окна, в которых пользователь набирает строки символов). Windows управляет всей логикой окон редактирования и дает программе окончательное содержимое этих окон, после того, как пользователь завершит ввод текста.

Понятие фокуса ввода

Клавиатура должна разделяться между всеми приложениями, работающими под Windows. Некоторые приложения могут иметь больше одного окна, и клавиатура должна разделяться между этими окнами в рамках одного и того же приложения.

Когда на клавиатуре нажата клавиша, только одна оконная процедура может получить сообщение об этом. Окно, которое получает это сообщение клавиатуры, является окном, имеющим фокус ввода.

Концепция фокуса ввода тесно связана с концепцией ***активного*** окна. Окно, имеющее фокус ввода – это либо активное окно, либо дочернее окно активного окна. Определение активного окна:

1) если у активного окна имеется панель заголовка, то Windows выделяет ее;
2) если у активного окна вместо панели заголовка имеется рамка диалога, то Windows выделяет ее;
3) если активное окно минимизировано, то Windows выделяет текст заголовка в панели задач.

Наиболее часто дочерними окнами являются кнопки, переключатели и другие элементы управления, которые обычно присутствуют в окне диалога. Сами по себе дочерние окна никогда не могут быть активными. Если фокус ввода находится в дочернем окне, то активным является родительское окно этого дочернего окна. То, что фокус ввода находится в дочернем окне, обычно показывается посредством мигающего курсора (для полей редактирования), рамки вокруг надписи на кнопке (для кнопок) или другими привлекающими внимание пользователя средствами.

Если активное окно минимизировано, то окна с фокусом ввода нет. Windows продолжает слать программе сообщения клавиатуры, но эти сообщения выглядят иначе, чем сообщения, направленные активным и еще не минимизированным окнам.

Обрабатывая сообщения WM_SETFOCUS и WM_KILLFOCUS, оконная процедура может определить, когда окно имеет фокус ввода. Сообщение WM_SETFOCUS показывает, что окно получило фокус ввода, а WM_KILLFOCUS – что окно потеряло его. В ответ на эти сообщения функция окна не может запретить получение или потерю фокуса ввода, так как эти сообщения носят чисто информирующий характер.

Программный интерфейс Windows содержит две функции, позволяющие узнать или изменить окно, владеющее фокусом ввода, – GetFocus и SetFocus.

Замечание. Когда пользователь в программе набирает текст, обычно маленький символ подчеркивания, маленький прямоугольник или вертикальная черточка (или

какой-либо другой признак) показывает ему место, где следующий набираемый символ появится на экране. При программировании для Windows этот признак называется ***каретка*** (caret). Слово же курсор относится к битовому образу, отражающему положение мыши на экране.

Категории клавиатурных сообщений

Сообщения, которые приложение получает от Windows о событиях, относящихся к клавиатуре, различаются на ***аппаратные*** (keystrokes) и ***символьные*** (characters). Такое положение соответствует двум представлениям о клавиатуре.

1. Клавиатуру можно считать набором клавиш. Нажатие на клавишу является аппаратным событием; отпускание этой клавиши также является аппаратным событием.
2. Клавиатура также является устройством ввода, генерирующим отображаемые символы. Клавиша, в зависимости от состояния клавиш <Ctrl>, <Shift> и <CapsLock>, может стать источником нескольких символов. Для сочетаний двух аппаратных событий, которые генерируют отображаемые символы, Windows посылает программе и оба аппаратных события и символьное сообщение.

Некоторые клавиши не генерируют символов. Это такие клавиши, как клавиши переключения, функциональные клавиши, клавиши управления курсором и специальные клавиши, такие как <Insert> и <Delete>. Для таких клавиш Windows вырабатывает только ***аппаратные*** сообщения.

Аппаратные сообщения

Когда пользователь нажимает клавишу, Windows помещает либо сообщение WM_KEYDOWN, либо сообщение WM_SYSKEYDOWN в очередь сообщений окна, имеющего фокус ввода. Когда клавиша отпускается, Windows помещает в очередь сообщений либо сообщение WM_KEYUP, либо сообщение WM_SYSKEYUP.

Вывод: несистемные аппаратные сообщения – это сообщения WM_KEYDOWN и WM_KEYUP, а системные - WM_SYSKEYDOWN и WM_SYSKEYUP.

Системные аппаратные сообщения WM_SYSKEYDOWN и WM_SYSKEYUP более важны для Windows, чем для приложений. Эти сообщения обычно вырабатываются при нажатии клавиш в сочетании с клавишей <Alt>. Эти сообщения вызывают опции меню программы или системного меню, или используются для системных функций, таких как смена активного приложения (<Alt+Tab>).

Программы обычно игнорируют сообщения WM_SYSKEYDOWN и WM_SYSKEYUP и передают их в функцию DefWindowProc. Оконная процедура, в конце концов, получает другие сообщения, являющиеся результатом этих аппаратных сообщений клавиатуры (например, выбор меню).

При необходимости включить в код программы инструкции для обработки системных аппаратных сообщений клавиатуры следует после обработки этих сообщений передать их в DefWindowProc:

```
case WM_SYSKEYDOWN:
        MessageBox(hWnd, "Message WM_SYSKEYDOWN", "Key", MB_OK);
        return(DefWindowProc(hWnd, message, wParam, lParam));
case WM_SYSKEYUP:
```

```
MessageBox(hWnd, "Message WM_SYSKEYUP", "Key", MB_OK);
return(DefWindowProc(hWnd, message, wParam, lParam));
```

Что произойдет, если не передать системные аппаратные сообщения в DefWindowProc? Почти все, что влияет на окно программы, в первую проходит через оконную процедуру. Windows самостоятельно обрабатывает сообщения только в том случае, если они передаются в DefWindowProc.

Например, пусть при обработке пришедшего в оконную процедуру сообщения добавлены строки для следующей обработки системных сообщений:

```
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
case WM_SYSCHAR:
        return 0;
```

При этом происходит запрет всех операций с клавишей <Alt> (команды меню, <Alt+Tab> и др.), когда окно программы имеет фокус ввода.

Несистемные сообщения WM_KEYDOWN и WM_KEYUP обычно вырабатываются для клавиш, которые нажимаются и отпускаются без участия клавиши <Alt>. Приложение может использовать или не использовать эти сообщения клавиатуры. Сама Windows их проигнорирует.

Обычно сообщения о нажатии и отпускании появляются парами. Однако если пользователь оставит клавишу нажатой так, чтобы включился автоповтор, то Windows посылает оконной процедуре серию сообщений WM_KEYDOWN (или WM_SYSKEYDOWN) и одно сообщение WM_KEYUP (или WM_SYSKEYUP), когда, в конце концов, клавиша будет отпущена.

Также как и все синхронные сообщения, аппаратные сообщения клавиатуры также становятся в очередь сообщений. Приложение с помощью функции GetMessageTime может получить время нажатия и отпускания клавиши относительно старта системы.

Для всех аппаратных сообщений клавиатуры 32-разрядная переменная lParam, передаваемая в оконную процедуру, состоит из шести полей:

1) счетчика повторений (число нажатий клавиши),
2) скан-кода OEM (Original Equipment Manufacturer) (код клавиши, генерируемый аппаратурой компьютера),
3) флага расширенной клавиатуры (1, если сообщение клавиатуры появилось в результате работы с дополнительными клавишами расширенной клавиатуры IBM),
4) кода контекста (1, если нажата клавиша <Alt>),
5) флага предыдущего состояния клавиши (0, если в предыдущем состоянии клавиша была отпущена, и 1, если в предыдущем состоянии она была нажата),
6) флага состояния клавиши (0, если клавиша нажимается, и 1, если клавиша отпускается).

Важным параметром аппаратных сообщений клавиатуры является параметр wParam. В этом параметре содержится виртуальный код клавиши (virtual key code), идентифицирующий нажатую или отпущенную клавишу.

Для всех аппаратных сообщений клавиатуры параметр wParam содержит код виртуальной клавиши, соответствующей нажатой клавише. Этот параметр используется приложением для идентификации клавиши. Код виртуальной клавиши не зависит от аппаратной

реализации клавиатуры. Коды виртуальных клавиш имеют символьные обозначения, определенные в заголовочных файлах Windows, и имеют префикс VK_.

Примечание. Большинство программ для Windows игнорируют все, кроме нескольких сообщений о нажатии и отпускании клавиш.

Сообщения WM_SYSKEYDOWN и WM_SYSKEYUP адресованы Windows, и приложению не всегда нужно их отслеживать. Если приложение обрабатывает сообщение WM_KEYDOWN, то сообщение WM_KEYUP оно обычно тоже игнорирует.

Сообщения WM_KEYDOWN удобны для обработки сообщений о нажатии клавиш управления курсором, функциональных клавиш и специальных клавиш (таких как <Insert> или <Delete>), не генерирующих символьные сообщения.

Определение состояния клавиш сдвига и клавиш-переключателей

Параметры wParam и lParam аппаратных сообщений клавиатуры ничего не сообщают о состоянии клавиш сдвига (<Shift>, <Ctrl>, <Alt>) и клавиш-переключателей (<CapsLock>, <NumLock>, <ScrollLock>).

Приложение может получить текущее состояние любой виртуальной клавиши с помощью функции GetKeyState.

Функцию GetKeyState следует использовать осторожно, т. к. она не отражает состояние клавиатуры в реальном времени.

Функция GetKeyState отражает состояние клавиатуры на момент, когда последнее сообщение от клавиатуры было выбрано из очереди.

Функция GetKeyState не позволяет получать информацию о клавиатуре независимо от обычных сообщений от клавиатуры. Программе необходимо получить сообщение от клавиатуры до того, как GetKeyState сможет определить состояние клавиш.

Эта синхронизация дает преимущество, потому что, если нужно узнать положение переключателя для конкретного сообщения клавиатуры, GetKeyState обеспечивает возможность точной информации, даже если сообщение обрабатывается уже после того, как состояние переключателя было изменено.

Если нужна информация о текущем положении клавиши, то можно использовать функцию GetAsyncKeyState.

Пример обработки клавиатурных сообщений, определяя при этом положения клавиш сдвига на момент события и их текущее состояние

```
case WM_KEYDOWN:
{       int nVirtKey = (int) wParam; // виртуальный код клавиши
        switch(nVirtKey)
        {
                // нажатие клавиши Space (пробел)
                case VK_SPACE:
                // состояние на момент события
                if(GetKeyState(VK_SHIFT) < 0)
MessageBox(hWnd, "Одновременно <Space>+<Shift>", "<Space>", MB_OK);
                break;
// нажатие клавиши F1 для определения тек. состояния др. клавиши
                case VK_F1:
                // текущее состояние
                if(GetAsyncKeyState(VK_CONTROL))
MessageBox(hWnd, "Текущее состояние <Ctrl> - нажата", "<F1>", MB_OK);
```

```
break;
        }
}; return 0;
```

Символьные сообщения

Преобразование аппаратных сообщений клавиатуры в символьные сообщения делает Windows:

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

Функция GetMessage заполняет поля структуры msg данными следующего сообщения из очереди. Вызов DispatchMessage вызывает соответствующую оконную процедуру. Между двумя этими функциями находится функция TranslateMessage.

Функция TranslateMessage преобразует аппаратные сообщения клавиатуры в символьные сообщения. Если этим сообщением является WM_KEYDOWN (WM_SYSKEYDOWN) и, если нажатие клавиши в сочетании с положением клавиши сдвига генерирует символ, тогда TranslateMessage помещает символьное сообщение в очередь сообщений.

Это символьное сообщение будет следующим, после сообщения о нажатии клавиши, которое функция GetMessage извлечет из очереди.

Существует четыре вида символьных сообщений: несистемные – WM_CHAR, WM_DEADCHAR, системные – WM_SYSCHAR, WM_SYSDEADCHAR.

- Сообщения WM_SYSCHAR и WM_SYSDEADCHAR являются следствием сообщений WM_SYSKEYDOWN.

В большинстве случаев программы для Windows могут игнорировать все сообщения за исключением WM_CHAR.

- Параметр lParam, передаваемый в оконную процедуру как часть символьного сообщения, является таким же, как и параметр lParam аппаратного сообщения клавиатуры, из которого сгенерировано символьное сообщение. Параметр wParam – это код символа ASCII.

Если Windows-приложению необходимо обрабатывать символы клавиатуры, то ему придется обрабатывать сообщение WM_CHAR. Наиболее типичным кодом обработки сообщения WM_CHAR является следующий код:

```
case WM_CHAR:
{       char chCharCode = (char)wParam; // ASCII-код символа
        switch( chCharCode )
        {
                case '\b': . . .; // введен символ Backspace
                        break;
                case '\t': . . .; // введен символ Tab
                        break;
                case '\r': . . .; // введен символ Enter
                        break;
                case 'A': . . .; // введен прописной символ A
                        break;
                case 'a': . . .; // введен строчный символ a
```

```
                        break;
. . .
                default: . . .; // введены другие символы
                break;
        }
}; return 0;
```

Для обеспечения возможности работы с символами кириллицы фирма Microsoft разработала расширенный набор символов с кириллицей. В терминологии Windows такие таблицы кодов называются наборами символов OEM (Original Equipment Manufacturer).

По умолчанию в контекст отображения выбирается системный шрифт, для которого используется набор символов ANSI.

Для одинаковых символов наборы ANSI и OEM используют разные коды, это приводит к необходимости перекодировки символов, например, при переносе текстов, подготовленных в среде MS-DOS, в среду Windows. В составе программного интерфейса Windows имеются функции, которые берут на себя работу по преобразованию и перекодировке символов.

Обработка сообщений "мыши"

Определить присутствие мыши можно с помощью функции GetSystemMetrics, передав ей в качестве параметра значение SM_MOUSEPRESENT. Если мышь есть, эта функция возвращает ненулевое значение.

Для определения количества кнопок можно использовать вызов GetSystemMetrics с параметром SM_CMOUSEBUTTONS.

Когда пользователь перемещает мышь, Windows перемещает по экрану растровую картинку, которая называется "курсор мыши" (mouse cursor). Курсор мыши имеет вершину (hot spot) размером в один пиксель, точно указывающий положение мыши на экране.

В драйвере дисплея содержатся несколько предопределенных курсоров мыши, которые могут использоваться в программах. Наиболее типичным курсором является наклонная ***стрелка***, которая называется IDC_ARROW (вершина курсора – острие стрелки). Курсор IDC_WAIT в виде ***песочных часов*** обычно используется для индикации того, что программа чем-то занята.

Замечание. Программисты могут сами проектировать свои собственные курсоры.

Курсор, устанавливаемый по умолчанию для конкретного окна, задается при определении класса окна:

```
wndclass.hCursor = LoadCursor( NULL, IDC_ARROW );
```

Сообщения мыши, связанные с рабочей областью окна

Windows посылает сообщения клавиатуры тому окну, которое имеет фокус ввода. Сообщения мыши отличаются: оконная процедура получает сообщения мыши и когда мышь проходит через ее окно, и при щелчке внутри окна, даже если окно не активно или не имеет фокуса ввода.

В Windows для мыши определен набор из 21 сообщения. Однако 11 из этих сообщений не относятся к рабочей области, и программы для Windows обычно игнорируют их.

Если мышь перемещается по рабочей области окна, оконная процедура получает сообщение WM_MOUSEMOVE.

Если кнопка мыши нажимается или отпускается внутри рабочей области окна, оконная процедура получает следующие сообщения:

- о нажатии – WM_LBUTTONDOWN, WM_RBUTTONDOWN, WM_MBUTTONDOWN;
- об отпускании – WM_LBUTTONUP, WM_RBUTTONUP, WM_MBUTTONUP;
- о двойном щелчке – WM_LBUTTONDBLCLK, WM_RBUTTONDBLCLK, WM_MBUTTONDBLCLK.

Для всех сообщений, связанных с рабочей областью, значение параметра lParam содержит положение мыши. Младшее слово – это координата X, а старшее – координата Y относительно верхнего левого угла рабочей области окна.

Координаты X и Y можно извлечь из параметра lParam с помощью макросов LOWORD и HIWORD.

Значение параметра wParam показывает состояние кнопок мыши и клавиш <Shift> и <Ctrl>. Можно проверить параметр wParam с помощью битовых масок, определенных в заголовочных файлах:

- MK_LBUTTON – левая кнопка нажата.
- MK_RBUTTON – правая кнопка нажата.
- MK_MBUTTON – средняя кнопка нажата.
- MK_SHIFT – клавиша <Shift> нажата.
- MK_CONTROL – клавиша <Ctrl> нажата.

Пример, проверяющий, была ли нажата левая кнопка мыши при ее движении по рабочей области окна, и рисующий на этом месте пиксель:

```
case WM_MOUSEMOVE:
{
        // состояние кнопок мыши
        UINT fwKeys = wParam;
        // горизонтальная позиция курсора
        int xPos = LOWORD( lParam );
        // вертикальная позиция курсора
        int yPos = HIWORD( lParam );
        if(fwKeys & MK_LBUTTON)
        {
                HDC hDC = GetDC( hWnd );
                SetPixel(hDC, xPos, yPos, 0);
                ReleaseDC( hWnd, hDC );
        }
};  return 0;
```

При движении мыши по рабочей области окна, Windows не вырабатывает сообщение WM_MOUSEMOVE для всех возможных положений мыши. Количество сообщений WM_MOUSEMOVE зависит от устройства мыши и скорости, с которой оконная процедура может обрабатывать сообщения о движении мыши.

Если щелкнуть кнопкой мыши в рабочей области неактивного окна, Windows сделает активным окно, в котором был произведен щелчок, и затем передаст оконной процедуре сообщение WM_LBUTTONDOWN.

Если приложение получает сообщение WM_LBUTTONDOWN, то оно может уверенно считать, что в данный момент его окно активно.

Однако оконная процедура может получить сообщение WM_LBUTTONUP, не получив вначале сообщения WM_LBUTTONDOWN. Это может случиться, если кнопка мыши нажимается в одном окне, мышь перемещается в другое окно, и кнопка отпускается.

Аналогично, оконная процедура может получить сообщение WM_LBUTTONDOWN без соответствующего ему сообщения WM_LBUTTONUP, если кнопка мыши отпускается во время нахождения мыши в другом окне.

В этих правилах есть исключения:

- Оконная процедура может захватить мышь (capture the mouse) и продолжать получать сообщения мыши, даже если она находится вне рабочей области окна.
- Если системное модальное окно сообщений или системное модальное окно диалога находится на экране, никакая другая программа не может получать сообщения от мыши.

Обработка нажатия клавиш <Shift> и <Ctrl> и кнопок мыши при получении сообщений мыши, связанных с рабочей областью окна, через параметр wParam передается значение, позволяющее определить, были ли одновременно с этим нажаты кнопки мыши или клавиши <Shift> и <Ctrl> клавиатуры.

Например, если обработка должна зависеть от состояния клавиш <Shift> и <Ctrl>, то приложение могло бы воспользоваться следующей логикой:

```
UINT fwKeys = wParam; // состояние кнопок мыши
if(MK_SHIFT & fwKeys)
{
        if(MK_CONTROL & fwKeys)
        { /* нажаты клавиши <Shift> и <Ctrl> */ }
        else
        { /* нажата клавиша <Shift> */ }
}
else
{
        if( MK_CONTROL & fwKeys )
        { /* нажата клавиша <Ctrl> */ }
        else
        { /* клавиши <Shift> и <Ctrl> не нажаты */ }
}
```

Функция GetKeyState также может возвращать состояние кнопок мыши или клавиш <Shift> и <Ctrl>, используя виртуальные коды клавиш VK_LBUTTON, VK_RBUTTON, VK_MBUTTON, VK_SHIFT и VK_CONTROL. При нажатой кнопке или клавише возвращаемое значение функции GetKeyState отрицательно.

Функция GetKeyState возвращает состояние кнопки мыши и клавиши в связи с обрабатываемым в данный момент сообщением, т.е. информация о состоянии должным образом синхронизируется с сообщением.

Двойные щелчки мыши

Двойным щелчком мыши называются два, следующих одно за другим в быстром темпе, щелчка мыши. Для того, чтобы два последовательных щелчка считались двойным

щелчком, они должны произойти в течение очень короткого промежутка времени, который зовется временем двойного щелчка (double-click time).

Если необходимо, чтобы оконная процедура получала сообщения двойного щелчка мыши, то следует включить идентификатор CS_DBLCLKS при задании стиля класса окна перед вызовом функции RegisterClass, например:

```
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
```

- Если CS_DBLCLKS не включить в стиль класса окна, и пользователь в быстром темпе щелкнет левой кнопкой мыши, то оконная процедура получит сообщения о двух одинарных щелчках мыши в следующей последовательности:
  WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDOWN, WM_LBUTTONUP.
- Если же CS_DBLCLKS включить в стиль окна, то при двойном щелчке оконная процедура получает следующие сообщения:
  WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLK, WM_LBUTTONUP.
  Сообщение WM_LBUTTONDBLCLK просто заменяет второе сообщение WM_LBUTTONDOWN.

Сообщения мыши, связанные с нерабочей областью окна

Если мышь оказывается вне рабочей области окна, но все еще внутри окна, то Windows посылает оконной процедуре сообщения мыши, связанные с нерабочей областью (нерабочая область включает в себя панель заголовка, меню, рамку окна и полосы прокрутки).

Обычно нет необходимости обрабатывать сообщения мыши нерабочей области. Вместо этого их передают в DefWindowProc, чтобы Windows могла выполнить системные функции. В этом смысле сообщения мыши нерабочей области похожи на системные сообщения клавиатуры.

Сообщения мыши нерабочей области почти полностью такие же, как и сообщения мыши рабочей области. В названия сообщений входят буквы NC, что означает нерабочая (nonclient).

Например, если мышь перемещается внутри нерабочей области окна, то оконная процедура получает сообщение WM_NCMOUSEMOVE.

Однако параметры wParam и lParam для сообщений нерабочей области отличаются от соответствующих параметров для сообщений мыши рабочей области:

1. Параметр wParam показывает зону нерабочей области, в которой произошло перемещение или щелчок. Его значение приравнивается одному из идентификаторов, начинающихся с HT, что означает тест попадания (hit-test).
2. Переменная lParam содержит в младшем слове значение координаты X, а в старшем – Y. Однако эти координаты являются координатами экрана, а не координатами рабочей области. Значения координат X и Y верхнего левого угла экрана равны 0.

Приложение может преобразовать экранные координаты в координаты рабочей области окна и, наоборот, с помощью функций Windows ScreenToClient и ClientToScreen.

Сообщение теста попадания

Сообщение WM_NCHITTEST (тест попадания в нерабочую область – nonclient hit-test) предшествует всем остальным сообщениям мыши рабочей и нерабочей области. Параметр lParam содержит значения X и Y экранных координат положения мыши. Параметр wParam не используется.

В приложениях Windows это сообщение обычно передается в DefWindowProc. В этом случае Windows использует сообщение WM_NCHITTEST для выработки всех остальных сообщений на основе положения мыши.

Для сообщений мыши нерабочей области возвращаемое значение функции DefWindowProc при обработке сообщения WM_NCHITTEST передается как параметр wParam в сообщении мыши.

Если функция DefWindowProc после обработки сообщения WM_NCHITTEST возвращает значение HTCLIENT, то Windows преобразует экранные координаты в координаты рабочей области и вырабатывает сообщение мыши рабочей области.

Замечание. Используя сообщение WM_NCHITTEST, приложение может полностью запретить работу с мышью в том или ином окне (чего делать, конечно, не рекомендуется), включив в оконную процедуру при обработке сообщений следующий код:

```
case  WM_NCHITTEST:
return ( LRESULT ) HTNOWHERE;
```

Этой обработкой полностью запрещаются все сообщения мыши рабочей и нерабочей области окна.

Кнопки мыши просто не будут работать до тех пор, пока мышь будет находится где-либо внутри окна, включая значок системного меню, кнопки минимизации, максимизации и закрытия окна.

Захват мыши

Оконная процедура обычно получает сообщения мыши только тогда, когда курсор мыши находится в рабочей или в нерабочей области окна.

Но иногда программе может понадобиться получать сообщения от мыши и тогда, когда курсор мыши находится вне окна. Если это необходимо сделать, то приложение может произвести захват (capture) мыши.

Захватить мышь можно вызвав функцию SetCapture. После вызова этой функции, Windows посылает все сообщения мыши в оконную процедуру того окна, чей дескриптор окна был передан в функцию SetCapture.

Пока мышь захвачена, системные функции клавиатуры тоже не действуют.

Сообщения мыши в захваченном состоянии всегда остаются сообщениями рабочей области, даже если мышь оказывается не в рабочей области окна. Параметр lParam по-прежнему показывает положение мыши в координатах рабочей области. Эти координаты, однако, могут стать отрицательными, если мышь окажется левее или выше рабочей области.

Освободить мышь, т.е. возвратить обработку мыши в нормальный режим, можно при помощи функции ReleaseCapture.

Использование таймера

Таймер в Windows является устройством ввода информации, которое периодически извещает приложение о том, что истек заданный интервал времени.

Приложение сообщает системе Windows интервал времени, а затем Windows периодически посылает приложению сообщения WM_TIMER, сигнализируя об истечении интервала времени.

Возможные случаи применения таймера в Windows:

- Режим автосохранения – таймер может предложить программе сохранять работу пользователя на диске всегда, когда истекает заданный интервал времени.
- Поддержка обновления информации о состоянии – программа может использовать таймер для вывода на экран обновляемой в реальном времени, постоянно меняющейся информации, связанной либо с системными ресурсами, либо с процессом выполнения отдельной задачи.
- Завершение демонстрационных версий программ – некоторые демонстрационные версии программ рассчитаны на свое завершение через какое-либо заданное время после запуска. Таймер может сигнализировать таким приложениям, когда их время истекает.
- Эмуляция многозадачности – хотя Windows является вытесняющей многозадачной средой, иногда самое эффективное решение для программы – как можно быстрее вернуть управление Windows. Если программа должна выполнять большой объем работы, она может разделить задачу на части и обрабатывать каждую часть при получении сообщения от таймера.
- Задание темпа изменения – графические объекты в играх или окна с результатами в обучающих программах могут нуждаться в задании установленного темпа изменения.

Сообщения от таймера

Присоединить таймер к программе можно при помощи вызова функции SetTimer. Функция SetTimer содержит целый параметр, задающий интервал в миллисекундах – это значение определяет темп, с которым Windows посылает программе сообщения WM_TIMER.

Для остановки потока сообщений от таймера приложение должно вызвать функцию KillTimer. Вызов KillTimer очищает очередь сообщений от всех необработанных сообщений WM_TIMER.

Поскольку приложения Windows получают сообщения WM_TIMER из обычной очереди сообщений, приложение не должно беспокоиться о том, что его работа будет прервана внезапным сообщением WM_TIMER.

В этом смысле таймер похож на клавиатуру и мышь: драйвер обрабатывает асинхронные аппаратные прерывания, а Windows преобразует эти прерывания в регулярные, структурированные, последовательные сообщения.

Итак, сообщения таймера ставятся в обычную очередь сообщений и обрабатываются как все остальные сообщения. Поэтому, если приложение задает функции SetTimer интервал 1000 миллисекунд, то ему не гарантируется получение сообщения каждую секунду (интервал будет колебаться). Если приложение занято больше чем секунду, то оно вообще не получит ни одного сообщения WM_TIMER в течение этого времени. Фактически,

- Windows обрабатывает сообщение WM_TIMER во многом так же, как сообщения WM_PAINT. Оба эти сообщения имеют низкий приоритет, и программа получит их, только если в очереди нет других сообщений.
- Сообщения WM_TIMER похожи на сообщения WM_PAINT и в другом смысле: Windows не хранит в очереди сообщений несколько сообщений WM_TIMER. Вместо этого Windows объединяет несколько сообщений WM_TIMER из очереди в одно сообщение. В результате приложение не может определить число “потерянных” сообщений WM_TIMER.
- Таймер можно использовать одним из трех способов, в зависимости от параметров функции SetTimer.

Первый способ использование таймера

Этот простейший способ заставляет Windows посылать сообщения WM_TIMER обычной оконной процедуре приложения. Вызов функции SetTimer в этом случае примет такой вид:

```
SetTimer( hWnd, 1, 1000, NULL );
```

- Первый параметр – дескриптор того окна, чья оконная процедура будет получать сообщения WM_TIMER.
- Вторым параметром является идентификатор таймера, значение которого должно быть отлично от нуля. В этом примере он произвольно установлен в 1.
- Третий параметр – это 32-разрядное беззнаковое целое, которое задает интервал в миллисекундах (значение 1000 задает генерацию сообщений WM_TIMER один раз в секунду).

Поток сообщений WM_TIMER можно в любое время остановить (даже во время обработки сообщения WM_TIMER), вызвав функцию:

```
KillTimer( hWnd, 1 );
```

Вторым параметром здесь является тот же идентификатор таймера, который использовался при вызове функции.

Замечание. Приложение должно перед завершением программы уничтожить все активные таймеры.

Когда оконная процедура получает сообщение WM_TIMER, значение wParam равно значению идентификатора таймера (который равен 1 в приведенном примере), а lParam равно 0.

Если приложению необходимо несколько таймеров, для каждого из них необходимо использовать свой идентификатор, например:

```
#define TIMER_SEC      1
#define TIMER_MIN      2
...
SetTimer( hWnd, TIMER_SEC, 1000, NULL );
```

```
SetTimer( hWnd, TIMER_MIN, 60000, NULL );
```

Значение параметра wParam позволяют различать передаваемые в оконную процедуру сообщения WM_TIMER. Логика обработки сообщения WM_TIMER может выглядеть примерно так:

```
case WM_TIMER:
{       UINT wTimerID = (UINT)wParam;
        switch(wTimerID)
        {
                // действия происходят один раз в секунду
                case TIMER_SEC:
                        . . .
                        break;
                // действия происходят один раз в минуту
                case TIMER_MIN:
                        . . .
                        break;
        }
}; return 0;
```

Замечание. Для того чтобы установить новое время срабатывания для существующего таймера, следует уничтожить таймер функций KillTimer и снова установить его функцией SetTimer.

Второй способ использования таймера

При первом способе установки таймера сообщения WM_TIMER посылаются в обычную оконную процедуру.

С помощью второго способа можно заставить Windows пересылать сообщения другой функции этого же приложения. Функция, которая будет получать эти таймерные сообщения, называется функцией обратного вызова (call-back). Эта функция приложения, которую вызывает Windows. Приложение сообщает Windows адрес этой функции, а позже Windows вызывает ее (оконная процедура фактически является такой функцией обратного вызова).

Как и оконная процедура, функция обратного вызова должна определяться как CALLBACK, поскольку Windows вызывает ее вне кодового пространства программы. Параметры функции обратного вызова и ее возвращаемое значение зависят от назначения функции обратного вызова.

В случае использования функции обратного вызова для таймера, входными параметрами являются те же параметры, что и параметры оконной процедуры. Таймерная функция обратного вызова не имеет возвращаемого в Windows значения.

Допустим, что в качестве имени таймерной функции обратного вызова выбрано имя TimerProc, тогда эта функция должна иметь следующее определение (она будет обрабатывать только сообщения WM_TIMER):

```
void CALLBACK TimerProc( HWND hWnd, UINT iMsg, UINT iTimerID, DWORD dwTime )
        {
        // обработка сообщений WM_TIMER
        . . .
        }
```

Входной параметр hWnd – дескриптор окна, задаваемый при вызове функции SetTimer.

Windows будет посылать функции TimerProc только сообщения WM_TIMER, следовательно, параметр iMsg всегда будет равен WM_TIMER.
Значение iTimerID – это идентификатор таймера, а значение dwTime – системное время.
Как уже говорилось, при использовании первого способа установки таймера требуется следующий вызов:

```
SetTimer(hWnd, 1, 1000, NULL);
```

При использовании функции обратного вызова для обработки сообщений WM_TIMER (второй способ использования таймера), четвертый параметр функции SetTimer заменяется адресом функции обратного вызова:

```
SetTimer(hWnd, 1, 1000, (TIMERPROC)TimerProc);
```

Третий способ использование таймера

Третий способ установки таймера напоминает второй, за исключением того, что параметр hWnd функции SetTimer устанавливается NULL, а второй параметр (обычно идентификатор таймера) игнорируется. Функция возвращает ID таймера (возвращаемое функцией значение будет равно NULL, если таймер недоступен):

```
UINT iTimerID = SetTimer(NULL, 0, 1000, (TIMERPROC)TimerProc);
```

Параметр hWnd, который будет передаваться системой Windows в таймерную функцию TimerProc при этом способе установки таймера, также будет равен NULL.

Первый параметр функции KillTimer при удалении установленного таймера также должен быть равен NULL. Идентификатор таймера должен быть равен значению, возвращаемому функцией SetTimer:

```
KillTimer(NULL, iTimerID);
```

Замечание. Такой метод установки таймера используется редко. Он удобен, если в программе в разное время делается много вызовов функции SetTimer, и при этом не запоминаются те таймерные идентификаторы, которые уже использовались.

Контрольная работа №2

по дисциплине "Прикладное программирование"

1. Что происходит, когда пользователь нажимает или отпускает клавишу на клавиатуре?
2. Что такое фокус ввода? Как он связан с понятием активного окна?
3. Может ли приложение, обрабатывающее сообщения WM_SETFOCUS и WM_KILLFOCUS повлиять на приобретение или потерю фокуса ввода окном?
4. Когда в окно поступают аппаратные сообщения клавиатуры, какие типы аппаратных сообщений существуют?
5. Какие сообщения Windows являются аппаратными клавиатурными сообщениями?
6. В чем специфика обработки системных аппаратных клавиатурных сообщений?
7. Что такое виртульный код клавиши? Что он идентифицирует? Зависит ли этот код от аппаратной реализации клавиатуры?
8. Как приложение может получить информацию о нажатии на определенную клавишу на момент, когда последнее сообщение от клавиатуры было выбрано из очереди?
9. Как приложение может получить информацию о нажатии на определенную клавишу в текущий момент времени?
10. Почему чаще всего приложение обрабатывает не аппаратные сообщения, а символьные? Что такое символьные сообщения, как они формируются?
11. Как следует модифицировать цикл обработки сообщений для того, чтобы приложение могло получать символьные сообщения?
12. Какое сообщение является символьным клавиатурным сообщением и какую дополнительную информацию оно несет с собой?
13. Какие наборы символов использует Windows, чем они отличаются и где применяются?
14. Как можно определить наличие мыши и ее характеристики?
15. Чем является курсор мыши, что такое вершина курсора мыши?
16. Где приложение обычно определяет форму и вид, которые приобретает курсор, когда он находится над поверхностью окна?
17. Какие сообщения могут поступать приложению при работе пользователя с мышью?
18. От чего зависит количество сообщений WM_MOUSEMOVE, которые получает окно приложения?
19. Обязательно ли приложение, получившее сообщение WM_LBUTTONDOWN, получит и сообщений WM_LBUTTONUP? В каких случаях это будет обязательно, а в каких нет?
20. Что необходимо сделать, чтобы окно приложения получало сообщения о двойных щелчках мыши? Что будет происходить в противном случае?

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