Модуль №2

Занятие №2

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Синхронные и асинхронные сообщения.
3. Посылка сообщений.
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) Что такое диалоговое окно?
2) Что такое элемент управления?
3) Какие бывают типы диалоговых окон?
4) Какая функция позволяет создать модальный диалог и в чём состоит особенность работы этой функции?
5) Какая функция позволяет создать немодальный диалог?
6) Какие существуют способы отображения немодального диалогового окна?
7) В каком случае при создании диалогового приложения функция **WinMain** должна содержать цикл обработки сообщений?

8) Какие различия между диалоговой процедурой и оконной процедурой?
9) Какое сообщение необходимо обработать в диалоговой процедуре, чтобы обеспечить возможность завершения диалога?
10) Какая функция обеспечивает закрытие модального диалога?
11) Какие действия необходимо выполнить при обработке сообщения **WM_CLOSE** для завершения приложения, созданного на основе немодального диалога?
12) Каким образом в диалоговой процедуре можно запретить вызов стандартного обработчика сообщения?
13) Какие элементы управления относятся к базовым элементам управления?
14) Какие элементы управления относятся к общим элементам управления?
15) Какая функция используется для создания элемента управления?
16) Какая функция позволяет получить дескриптор элемента управления по его идентификатору?
17) Какая функция позволяет получить идентификатор элемента управления по его дескриптору?
18) Каким образом можно сделать элемент управления разрешённым или запрещённым?
19) Что такое статический элемент управления? Какие бывают типы статических элементов управления?
20) Какие действия обычно выполняют при обработке сообщения **WM_INITDIALOG**?

2. Синхронные и асинхронные сообщения

Перед освещением данного вопроса необходимо напомнить слушателям особенности архитектуры Windows-приложения, управляемого сообщениями.

Отметить, что сообщения, посылаемые окну, могут быть **синхронными** и **асинхронными**.

Синхронные сообщения - это сообщения, которые Windows помещает в очередь сообщений приложения. В цикле очередное синхронное сообщение выбирается из очереди функцией **GetMessage**, затем передается Windows посредством функции **DispatchMessage**, после чего Windows отправляет синхронное сообщение оконной процедуре для последующей обработки.

В отличие от синхронных сообщений **асинхронные сообщения** передаются непосредственно оконной процедуре для немедленной обработки, минуя очередь сообщений.

К синхронным сообщениям относятся сообщения о событиях пользовательского ввода, например, клавиатурные сообщения и сообщения мыши. Кроме этого синхронными являются сообщения от таймера (**WM_TIMER**), сообщение о необходимости перерисовки клиентской области окна (**WM_PAINT**) и сообщение о выходе из программы (**WM_QUIT**). Остальные сообщения, как правило, являются асинхронными. Во многих случаях асинхронные сообщения являются результатом обработки синхронных сообщений. Например, когда **WinMain** вызывает функцию **CreateWindow**, Windows создает окно и для этого отправляет оконной процедуре асинхронное сообщение **WM_CREATE**. Когда **WinMain** вызывает **ShowWindow**, Windows отправляет оконной процедуре асинхронные сообщения **WM_SIZE** и **WM_SHOWWINDOW**. Когда **WinMain** вызывает **UpdateWindow**, Windows отправляет оконной процедуре асинхронное сообщение **WM_PAINT**.

3. Посылка сообщений

Ознакомив слушателей с понятиями синхронного и асинхронного сообщений, акцентировать внимание на том, что существует множество задач, для решения которых необходимо уметь «вручную» посылать сообщения окнам, не

дожидаясь их от операционной системы. Для этой цели используются функции API **SendMessage** и **PostMessage**.

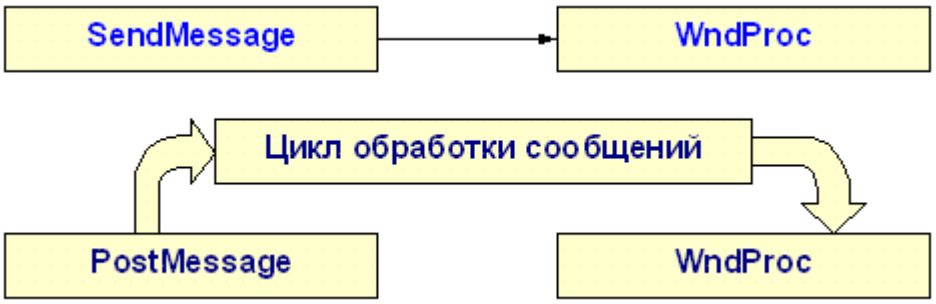
```
LRESULT SendMessage(
    HWND hWnd, // дескриптор окна, которому отправляется сообщение
    UINT Msg, // идентификатор сообщения
    WPARAM wParam, // дополнительная информация о сообщении
    LPARAM lParam  // дополнительная информация о сообщении
);
```

```
BOOL PostMessage(
    HWND hWnd, // дескриптор окна, которому отправляется сообщение
    UINT Msg, // идентификатор сообщения
    WPARAM wParam, // дополнительная информация о сообщении
    LPARAM lParam  // дополнительная информация о сообщении
);
```

Как видно, обе функции имеют одинаковую сигнатуру, а их сходство состоит в том, что они предназначены для посылки сообщения в окно некоторого приложения. Однако вышеприведенные функции имеют определённые отличия.

Функция **SendMessage** вызывает оконную процедуру определенного окна и не завершает свою работу до тех пор, пока оконная процедура не обработает сообщение. Иначе говоря, выполнение программы не будет продолжено, пока сообщение не будет обработано. Оконная процедура, которой отправляется сообщение, может быть той же самой оконной процедурой, другой оконной процедурой той же программы или даже оконной процедурой другого приложения. Таким образом, функция **SendMessage** посылает асинхронное сообщение указанному окну.

Функция **PostMessage** посылает синхронное сообщение указанному окну. В отличие от **SendMessage**, функция **PostMessage** не вызывает явно оконную процедуру, а всего лишь помещает сообщение в очередь сообщений. Выполнение самой функции на этом завершается, и работа приложения может быть продолжена.



Следует отметить, если первый параметр вышерассмотренных функций имеет значение **HWND_BROADCAST**, то сообщение посылается всем окнам верхнего уровня, существующим в настоящий момент в системе.

Для сравнительного анализа работы функций **SendMessage** и **PostMessage** предлагается рассмотреть следующий код:

```
#include <windows.h>
#include <tchar.h>

LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);
TCHAR szClassWindow[] = TEXT("Каркасное приложение");

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                     LPTSTR lpszCmdLine, int nCmdShow)
{
      HWND hWnd;
      MSG Msg;
      WNDCLASSEX wcl;
      wcl.cbSize = sizeof(wcl);
      wcl.style = CS_HREDRAW | CS_VREDRAW;
      wcl.lpfnWndProc = WindowProc;
      wcl.cbClsExtra = 0;
      wcl.cbWndExtra = 0;
      wcl.hInstance = hInst;
      wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
      wcl.hCursor = LoadCursor(NULL, IDC_ARROW);
      wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
      wcl.lpszMenuName = NULL;
      wcl.lpszClassName = szClassWindow;
      wcl.hIconSm = NULL;
      if (!RegisterClassEx(&wcl))
            return 0;
      hWnd = CreateWindowEx(0, szClassWindow,
            TEXT("Синхронные и асинхронные сообщения"),
            WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 500, 200,
            NULL, NULL, hInst, NULL);
      ShowWindow(hWnd, nCmdShow);
      UpdateWindow(hWnd);
      while(GetMessage(&Msg, NULL, 0, 0))
      {
            TranslateMessage(&Msg);
            DispatchMessage(&Msg);
      }
```

```
	return Msg.wParam;
}

LRESULT CALLBACK WindowProc (HWND hWnd, UINT message, WPARAM wParam,
                             LPARAM lParam)
{
	switch(message)
	{
		case WM_DESTROY:
			PostQuitMessage(0);
			break;
		case WM_KEYDOWN:
			if(wParam == VK_UP)
			{
				HWND h = FindWindow(TEXT("SciCalc"),
					TEXT("Калькулятор"));
				if(h)
					SendMessage(h, WM_CLOSE, 0, 0);
			}
			else if(wParam == VK_DOWN)
			{
				HWND h = FindWindow(TEXT("SciCalc"),
					TEXT("Калькулятор"));
				if(h)
					PostMessage(h, WM_CLOSE, 0, 0);
			}
			else if(wParam == VK_LEFT)
			{
				HWND h = FindWindow(TEXT("SciCalc"),
					TEXT("Калькулятор"));
				if(h)
					SendMessage(h, WM_QUIT, 0, 0);
			}
			else if(wParam == VK_RIGHT)
			{
				HWND h = FindWindow(TEXT("SciCalc"),
					TEXT("Калькулятор"));
				if(h)
					PostMessage(h, WM_QUIT, 0, 0);
			}
			break;
		default:
			return DefWindowProc(hWnd, message, wParam, lParam);
	}
	return 0;
}
```

Анализируя вышеприведенный код, следует отметить, что при отправке сообщения **WM_CLOSE** главному окну приложения «Калькулятор» обе функции (**SendMessage** и **PostMessage**) дают одинаковый результат – приложение закрывается. Следует напомнить слушателям, что при стандартной обработке сообщения **WM_CLOSE** вызывается функция **DestroyWindow** для разрушения главного окна приложения и его дочерних окон. Вызов этой функ-

ции, в свою очередь, приводит к посылке сообщения **WM_DESTROY**, при обработке которого оконная процедура обычно вызывает функцию **PostQuitMessage**. Как известно, вызов функции **PostQuitMessage** ставит сообщение **WM_QUIT** в очередь сообщений приложения. Впоследствии, функция **GetMessage** выбирает это сообщение из очереди, возвращает нулевое значение и завершает тем самым цикл обработки сообщений и, следовательно, приложение.

При отправке сообщения **WM_QUIT** главному окну приложения «Калькулятор» функции **PostMessage** и **SendMessage** дают различный результат. В первом случае (**PostMessage**) сообщение будет помещено в очередь сообщений приложения, что впоследствии приведёт к завершению приложения по вышеописанной причине. Во втором случае (**SendMessage**) сообщение будет направлено непосредственно оконной процедуре в обход очереди сообщений. В результате это не приведёт к закрытию приложения, так как сообщение **WM_QUIT** не ассоциируется с каким-либо окном и не предназначено для обработки оконной процедурой.

4. Практическая часть

Написать приложение «Слайд-шоу»: на форме диалога расположен статический элемент управления **Picture Control**, на котором через каждую секунду (или другой временной интервал) меняется изображение. Изображения (**Bitmaps** – растровые битовые образы) должны храниться в ресурсах приложения. Для использования в программе растрового битового образа, его предварительно следует загрузить функцией API **LoadBitmap**:

```
HBITMAP LoadBitmap(
  HINSTANCE hInstance,   // дескриптор приложения
  LPCTSTR lpBitmapName   // имя растрового битового образа
);
```

Например, в ресурсах приложения имеется растровый битовый образ с идентификатором **IDB_BITMAP1**, тогда его дескриптор можно получить следующим образом:

```
HBITMAP hBmp = LoadBitmap(GetModuleHandle(NULL),
                          MAKEINTRESOURCE(IDB_BITMAP1));
```

Для установки изображения на «статик» ему следует отправить сообщение **STM_SETIMAGE**, передав в **WPARAM** значение **IMAGE_BITMAP**, а в **LPARAM –** дескриптор растрового битового образа.

```
SendMessage(hStatic, STM_SETIMAGE, WPARAM(IMAGE_BITMAP), LPARAM(hBmp));
```

5. Подведение итогов

Подвести общие итоги занятия. Отметить различие между синхронным и асинхронным сообщениями. Подчеркнуть, что посылка сообщений является основным средством программного взаимодействия между разными окнами приложения и даже между разными приложениями. Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

6. Домашнее задание

1. Разработать приложение «убегающий статик». Суть приложения: на форме диалогового окна расположен статический элемент управления. Пользователь пытается подвести курсор мыши к «статику», и, если кур-

сор находится близко со «статиком», элемент управления «убегает». Предусмотреть перемещение «статика» только в пределах диалогового окна.

2. Разработать диалоговое приложение, позволяющее перетаскивать мышью «статик», расположенный на форме.