Модуль №4

Занятие №2

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Динамическое создание меню. Модификация меню.
3. Практическая часть.
4. Подведение итогов.
5. 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) Зачем в приложении необходимо меню? Что именно следует помещать в меню?
2) Какие различают виды меню?
3) Чем отличается пункт меню «команда» от пункта «подменю»?
4) Что такое сепаратор? Для чего он используется?
5) Что такое «горячие» клавиши в меню?
6) Что означает троеточие в конце названия пункта меню? Обязательно ли его указывать?
7) В чем отличие между разрешенными, запрещенными и недоступными пунктами меню?

8) Каким образом можно присоединить меню к главному окну приложения?
9) Какие функции API позволяют получить дескриптор главного меню и дескриптор подменю?
10) Какое сообщение приходит в оконную процедуру главного окна при выборе пункта меню, определяющего команду?
11) Какое сообщение приходит в оконную процедуру главного окна непосредственно перед активизацией выпадающего меню?
12) Какое сообщение операционная система отправляет главному окну приложения при навигации по меню?
13) Какая функция API применяется для изменения статуса пунктов меню?
14) Какая функция API позволяет установить или снять отметку на пункте меню?
15) В чем заключается удобство использования ресурса таблицы строк?

2. Динамическое создание меню. Модификация меню

Обратить внимание слушателей на то, что способ создания меню с помощью средств интегрированной среды разработки приложений не является единственным. Альтернативным подходом является программное (динамическое) создание меню.

Следует ознакомить слушателей с набором функций API, предназначенных для создания и модификации меню.

Функция API **CreateMenu** предназначена для создания главного меню приложения:

```
HMENU CreateMenu(VOID);
```

Необходимо отметить, что меню, созданное этой функцией, изначально будет пустым, т.е. не будет содержать ни одного пункта. Заполнить меню пунктами можно с помощью функций API **AppendMenu** или **InsertMenu**.

Функция API **CreatePopupMenu** предназначена для создания всплывающего меню приложения:

```
HMENU CreatePopupMenu(VOID);
```

Всплывающее меню, созданное этой функцией, также изначально будет пустым. Заполнить меню пунктами можно с помощью уже упомянутых функций API **AppendMenu** или **InsertMenu**.

Функция API **AppendMenu** применяется для добавления пунктов к концу меню.

```
BOOL AppendMenu(
    HMENU hMenu, // дескриптор меню, к которому добавляется новый пункт
    UINT uFlags, // внешний вид и правило поведения добавляемого пункта меню
    UINT_PTR uIDNewItem, // идентификатор для нового пункта меню или
      // дескриптор выпадающего меню
    LPCTSTR lpNewItem // содержимое нового пункта меню – зависит от второго
      // параметра
);
```

Второй параметр может иметь одно или несколько значений (флагов), приведенных ниже. В последнем случае флаги объединяются с помощью побитовой операции «ИЛИ».

- **MF_POPUP** – создает всплывающее меню. В этом случае третий параметр функции содержит дескриптор всплывающего меню.
- **MF_CHECKED** – помещает отметку рядом с пунктом меню.
- **MF_DEFAULT** – пункт меню установлен в качестве применяемого по умолчанию. Имя этого пункта выделяется жирным шрифтом. В этом случае при открытии подменю двойным щелчком

мыши, Windows автоматически выполнит команду по умолчанию, закрыв при этом подменю.

- **MF_ENABLED –** делает пункт меню доступным для выбора.
- **MF_DISABLED** – делает пункт меню недоступным для выбора.
- **MF_GRAYED** – выделяет серым цветом пункт меню и запрещает его выбор.
- **MF_UNCHECKED** – пункт меню не имеет отметки.
- **MF_SEPARATOR –** указывает, что пункт меню является разделителем (сепаратором).
- **MF_STRING** – отображает пункт меню с использованием текстовой строки, которая задается в четвертом параметре.

Функция API **InsertMenu** позволяет вставить новый пункт в меню.

```
BOOL InsertMenu(
    HMENU hMenu, // дескриптор меню, которое модифицируется
    UINT uPosition, // идентификатор или позиция пункта меню, перед которым
      // происходит вставка нового пункта
    UINT uFlags, // интерпретация второго параметра, а также внешний вид и
      //правило поведения нового пункта меню
    PTR uIDNewItem, // идентификатор для нового пункта меню или
      // дескриптор выпадающего меню
    LPCTSTR lpNewItem // содержимое нового пункта меню – зависит от третьего
      // параметра
);
```

Второму параметру функции **uPosition** передается либо идентификатор пункта меню, либо позиция пункта меню. Выбор варианта интерпретации задается в третьем параметре:

- **MF_BYCOMMAND** – в этом случае второй параметр должен содержать идентификатор пункта меню, перед которым происходит вставка нового пункта;
- **MF_BYPOSITION** – в этом случае второй параметр должен содержать относительную позицию пункта меню с отсчетом от нуля.

Кроме того, в третьем параметре можно дополнительно указать один или несколько флагов, приведенных выше при рассмотрении функции **AppendMenu**.

Функция API **ModifyMenu** применяется для изменения существующего пункта меню.

```
BOOL ModifyMenu(
    HMENU hMenu, // дескриптор меню, которое модифицируется
    UINT uPosition, // идентификатор или позиция пункта меню, который будет
      // изменен
    UINT uFlags, // интерпретация второго параметра, а также внешний вид и
      //правило поведения изменяемого пункта меню
    PTR uIDNewItem, // идентификатор для изменяемого пункта меню или
      // дескриптор выпадающего меню
    LPCTSTR lpNewItem // новое содержимое изменяемого пункта меню – зависит
      // от третьего параметра
);
```

Назначение параметров функции **ModifyMenu** аналогично функции **InsertMenu**.

Функция API **DeleteMenu** применяется для удаления отдельного пункта из указанного меню.

```
BOOL DeleteMenu(
    HMENU hMenu, // дескриптор меню, которое модифицируется
    UINT uPosition, // идентификатор или позиция пункта меню, который будет
      // удален
    UINT uFlags // интерпретация второго параметра
);
```

Следует отметить, что если удаляемым пунктом является выпадающее меню, то оно уничтожается и высвобождается память.

Акцентировать внимание слушателей на том, что после вызова функций **AppendMenu**, **InsertMenu** или **DeleteMenu** с целью модификации главного меню следует обязательно вызвать рассмотренную ранее

функцию API **DrawMenuBar** для повторного отображения изменившейся полосы меню.

Функция API **DestroyMenu** предназначена для уничтожения меню и высвобождения памяти, занимаемой меню.

```
BOOL DestroyMenu(
      HMENU hMenu // дескриптор уничтожаемого меню
);
```

Функция API **GetMenuString** получает строку текста указанного пункта меню.

```
int GetMenuString(
    HMENU hMenu, // дескриптор меню
    UINT uIDItem, // идентификатор или позиция пункта меню, текст которого
      // необходимо получить
    LPTSTR lpString, // указатель на строковый буфер, в который будет
      // записана строка текста
    int nMaxCount, // максимальное число символов, которое должно быть
      // записано в буфер
    UINT uFlag // интерпретация второго параметра
);
```

Рассмотреть со слушателями следующее приложение, в котором демонстрируется программный способ создания меню.

```
                                    // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"

                                  // MenuBarDlg.h
#pragma once
#include "header.h"

class CMenuBarDlg
{
public:
      CMenuBarDlg(void);
```

```
public:
	static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
	static CMenuBarDlg* ptr;
	BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
	void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
	void Cls_OnClose(HWND hwnd);
};


                                // MenuBarDlg.cpp

#include "MenuBarDlg.h"

#define ID_FILE_EXIT WM_USER
#define ID_EDIT_DELETE WM_USER+1
#define ID_EDIT_FINDNEXT WM_USER+2
#define ID EDIT TIMEDATE WM USER+3
#define ID_EDIT_GOTO WM_USER+4
#define ID_VIEW_STATUSBAR WM_USER+5
#define ID_FILE_NEW WM_USER+6
#define ID_FILE_OPEN WM_USER+7
#define ID_FILE_SAVE WM_USER+8
#define ID_FILE_SAVE_AS WM_USER+9
#define ID_FILE_PAGE_SETUP WM_USER+10
#define ID_FILE_PRINT WM_USER+11
#define ID_EDIT_UNDO WM_USER+12
#define ID_EDIT_CUT WM_USER+13
#define ID_EDIT_COPY WM_USER+14
#define ID_EDIT_PASTE WM_USER+15
#define ID_EDIT_FIND WM_USER+16
#define ID_EDIT_REPLACE WM_USER+17
#define ID_EDIT_SELECT_ALL WM_USER+18

CMenuBarDlg* CMenuBarDlg::ptr = NULL;

CMenuBarDlg::CMenuBarDlg(void)
{
	ptr = this;
}

void CMenuBarDlg::Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, 0);
}

BOOL CMenuBarDlg::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
{
	HMENU subFileMenu = CreatePopupMenu();
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_NEW, TEXT("New"));
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_OPEN, TEXT("Open..."));
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_SAVE, TEXT("Save"));
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_SAVE_AS, TEXT("Save As..."));
	AppendMenu(subFileMenu, MF_SEPARATOR, 0, 0);
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_PAGE_SETUP,
		TEXT("Page Setup..."));
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_PRINT, TEXT("Print..."));
	AppendMenu(subFileMenu, MF_SEPARATOR, 0, 0);
	AppendMenu(subFileMenu, MF_STRING, ID_FILE_EXIT, TEXT("Exit"));
```

```
      HMENU subEditMenu = CreatePopupMenu();
      AppendMenu(subEditMenu, MF_STRING | MF_GRAYED, ID_EDIT_UNDO,
                  TEXT("Undo"));
      AppendMenu(subEditMenu, MF_SEPARATOR, 0, 0);
      AppendMenu(subEditMenu, MF_STRING | MF_GRAYED, ID_EDIT_CUT,
                  TEXT("Cut"));
      AppendMenu(subEditMenu, MF_STRING | MF_GRAYED, ID_EDIT_COPY,
                  TEXT("Copy"));
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_PASTE, TEXT("Paste"));
      AppendMenu(subEditMenu, MF_STRING | MF_GRAYED, ID_EDIT_DELETE,
                  TEXT("Delete"));
      AppendMenu(subEditMenu, MF_SEPARATOR, 0, 0);
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_FIND, TEXT("Find... "));
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_FINDNEXT, TEXT("Find Next"));
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_REPLACE, TEXT("Replace..."));
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_GOTO, TEXT("Go To..."));
      AppendMenu(subEditMenu, MF_SEPARATOR, 0, 0);
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_SELECT_ALL,
                  TEXT("Select All"));
      AppendMenu(subEditMenu, MF_STRING, ID_EDIT_TIMEDATE, TEXT("Time/Date"));
      HMENU subViewMenu = CreatePopupMenu();
      AppendMenu(subViewMenu, MF STRING | MF CHECKED, ID VIEW STATUSBAR,
                  TEXT("Status Bar"));
      HMENU mainMenu = CreateMenu();
      AppendMenu(mainMenu, MF_POPUP, (UINT_PTR)subFileMenu, TEXT("File"));
      AppendMenu(mainMenu, MF_POPUP, (UINT_PTR)subEditMenu, TEXT("Edit"));
      AppendMenu(mainMenu, MF_POPUP, (UINT_PTR)subViewMenu, TEXT("View"));
      SetMenu(hwnd, mainMenu);
      return TRUE;
}

// Обработчик сообщения WM_COMMAND будет вызван при выборе пункта меню
void CMenuBarDlg::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                UINT codeNotify)
{
      HMENU hMenu = GetMenu(hwnd);
      TCHAR buf[30] = {0};
      GetMenuString(hMenu, id, buf, 30, MF_BYCOMMAND);
      MessageBox(hwnd, buf, TEXT("Динамическое создание меню"),
                  MB_ICONINFORMATION | MB_OK);
}


BOOL CALLBACK CMenuBarDlg::DlgProc(HWND hwnd, UINT message, WPARAM wParam,
                                   LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
      }
      return FALSE;
}


                                // MenuBar.cpp
#include "MenuBarDlg.h"
```

```
int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                     int nCmdShow)
{
      CMenuBarDlg dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                           CMenuBarDlg::DlgProc);
}
```

3. Практическая часть

1. Написать приложение, которое отображает увеличивающийся каждую секунду счетчик. Запуск счетчика осуществляется при помощи пункта меню «Счетчик | Старт», а остановка при помощи пункта меню «Счетчик | Стоп». При этом меню создается динамически. Необходимо предусмотреть, чтобы нельзя было выбрать пункт меню «Счетчик | Старт», если счетчик уже запущен, а пункт меню «Счетчик | Стоп», если счетчик еще не запущен.
2. Реализовать второй вариант счетчика. Вместо двух пунктов меню «Счетчик | Старт» и «Счетчик | Стоп» присутствует только один. Если счетчик еще не запущен, то он называется «Счетчик | Старт», а если уже запущен, то «Счетчик | Стоп».

4. Подведение итогов

Подвести общие итоги занятия. Ещё раз отметить способы динамического создания главного меню и подменю. Напомнить слушателям способы добавления и вставки в произвольную позицию пункта меню, а также способы добавления и вставки подменю. Подчеркнуть, какие существуют варианты идентификации пункта меню: по идентификатору и

по порядковому номеру. Отметить способы модификации и удаления пунктов меню, а также способ уничтожения меню целиком. Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

5. Домашнее задание

Разработать приложение «Перевод меню», аналогичное приложению из домашнего задания прошлого занятия (см. Модуль 4 Занятие 1). Основное отличие состоит в том, что единственное меню создается программным способом, а перевод меню осуществляется путем модификации каждого пункта меню.