|  |
| --- |
|  |

**1. Краткое описание функций, параметров и используемых событий.**

1) Функции для работы с окнами.

* EnableWindow

BOOL EnableWindow (HWND hWnd, BOOL bEnable);

Функция EnableWindow включает в работу или отключает мышь и ввод с клавиатуры в заданное окно или орган управления. Когда ввод данных заблокирован, окно не принимает ввода щелчков мыши и нажатий клавиши. Когда ввод данных разрешается, окно принимает весь ввод данных.

Параметры: hWnd – дескриптор окна, которое нужно включить / отключить; bEnable – показывает, отключаем или включаем окно.

Если окно было предварительно заблокировано, возвращаемое значение – не нуль, если не было – нуль.

* IsWindowVisible

BOOL IsWindowVisible (HWND hWnd);

Функция IsWindowVisible извлекает данные о состоянии видимости заданного окна.

Параметр: hWnd – дескриптор проверяемого окна.

Если заданное окно, его родительское окно, родители родительского окна и так далее, имеют стиль WS\_VISIBLE, возвращаемое значение не нуль. Иначе, возвращаемое значение нулевое.

* ShowWindow

BOOL EnableWindow (HWND hWnd, int nCmdShow);

Функция ShowWindow устанавливает состояние показа определяемого окна.

Параметры: hWnd – дескриптор окна; nCmdShow – определяет, как окно должно быть показано.

Второй параметр игнорируется при первом вызове приложения ShowWindow, если программа, запустившая приложение, предоставляет структуру STARTUPINFO. В противном случае при первом вызове ShowWindow значение должно быть значением, полученным функцией WinMain в ее параметре nCmdShow. В дальнейшем может принимать значения, указанные в таблице:

|  |  |
| --- | --- |
| **Значение** | **Смысл** |
| **SW\_HIDE** | Скрывает окно и активирует другое окно. |
| **SW\_MAXIMIZE** | Развертывает указанное окно. |
| **SW\_MINIMIZE** | Сворачивает указанное окно и активирует следующее окно верхнего уровня в Z-порядке. |
| **SW\_RESTORE** | Активирует и отображает окно. Если окно свернуто или развернуто, система восстанавливает его исходный размер и положение. Приложение должно указать этот флаг при восстановлении свернутого окна. |
| **SW\_SHOW** | Активирует окно и отображает его в текущем размере и положении. |
| **SW\_SHOWDEFAULT** | Устанавливает состояние показа на основе значения SW\_, указанного в структуре STARTUPINFO, переданной в функцию CreateProcess программой, запустившей приложение. |
| **SW\_SHOWMAXIMIZED** | Активирует окно и отображает его как развернутое окно. |
| **SW\_SHOWMINIMIZED** | Активирует окно и отображает его как свернутое окно. |
| **SW\_SHOWMINNOACTIVE** | Отображает окно в виде свернутого окна, но окно не активируется. |
| **SW\_SHOWNA** | Отображает окно в его текущем размере и положении, но окно не активируется. |
| **SW\_SHOWNOACTIVATE** | Отображает окно в его самом последнем размере и положении, но окно не активируется. |
| **SW\_SHOWNORMAL** | Активирует и отображает окно. Если окно свернуто или развернуто, система восстанавливает его исходный размер и положение. Приложение должно указать этот флаг при первом отображении окна. |

Если окно перед этим было видимо, возвращается значение - не нуль. Если окно перед этим было не видимо, возвращаемое значение - нуль.

2) Функции для посыла сообщения окну.

* SendMessage

LRESULT SendMessage (HWND hWnd, UINT Msg, WPARAM wParam,

LPARAM lParam);

Функция SendMessage отправляет заданное сообщение окну или окнам. Функция вызывает оконную процедуру для заданного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообщение.

Параметры: hWnd - дескриптор окна, оконная процедура которого примет сообщение; Msg - определяет сообщение, которое будет отправлено; wParam, lParam - определяют дополнительную конкретизирующую сообщение информацию.

Величина возвращаемого значения определяет результат обработки сообщения; он зависит от отправленного сообщения.

* PostMessage

LRESULT PostMessage (HWND hWnd, UINT Msg, WPARAM wParam,

LPARAM lParam);

Функция PostMessage помещает (вставляет в очередь) сообщение в очередь сообщений, связанную с потоком, который создал заданное окно и возвращает значение без ожидания потока, который обрабатывает сообщение.

Параметры: hWnd - дескриптор окна, оконная процедура которого примет сообщение; Msg - определяет сообщение, которое будет поставлено в очередь; wParam, lParam - определяют дополнительную конкретизирующую сообщение информацию.

Если функция завершается успешно, величина возвращаемого значения - не нуль. Если функция завершается ошибкой, величина возвращаемого значения нулевая.

3) Поиск окон.

* FindWindow

HWND FindWindow(LPCTSTR lpClassName, LPCTSTR lpWndName);

Функция FindWindow извлекает дескриптор окна верхнего уровня, имя класса и имя окна которого соответствуют заданным строкам. Эта функция не ищет дочерние окна. Эта функция не выполняет регистрозависимый поиск.

Параметры: lpClassName - указатель на строку с нулевым символом в конце, которая определяет имя класса или класс атома; lpWndName – указатель на строку с нулевым символом в конце, которая определяет имя окна (заголовок окна). Если этот параметр - ПУСТО (NULL), соответствуют имена всех окон.

Если функция завершилась успешно, возвращаемое значение - дескриптор окна, которое имеет заданное имя класса и имя окна. Если функция завершается ошибкой, возвращаемое значение - ПУСТО (NULL).

* GetWindow

HWND GetWindow (HWND hWnd, UINT uCmd);

Функция GetWindow извлекает дескриптор окна, который имеет определенное отношение к заданному окну.

Параметры: hWnd – дескриптор окна; uCmd – определяет отношение между заданным окном и окном, дескриптор которого должен быть извлечен. Этот параметр может быть одним из следующих значений:

|  |  |
| --- | --- |
| **Значение** | **Предназначение** |
| GW\_CHILD | Извлеченный дескриптор идентифицирует дочернее окно наверху Z - последовательности, если заданное окно - родительское окно; иначе, найденный дескриптор получит значение ПУСТО (NULL). Функция проверяет только дочерние окна заданного окна. |
| GW\_HWNDFIRST | Найденный дескриптор идентифицирует окно того же самого типа, которое является самым высоким в Z - последовательности. |
| GW\_HWNDLAST | Найденный дескриптор идентифицирует окно того же самого типа, который является самым низким в Z - последовательности. |
| GW\_HWNDNEXT | Найденный дескриптор идентифицирует окно ниже определяемого окна в Z - последовательности. |
| GW\_HWNDPREV | Найденный дескриптор идентифицирует окно выше определяемого окна в Z - последовательности. |
| GW\_OWNER | Найденный дескриптор идентифицирует окно владельца заданного окна, если таковой имеется. |

Если функция завершается успешно, возвращаемое значение - дескриптор окна. Если окно с заданным отношением к определяемому окну не существует, возвращаемое значение - ПУСТО (NULL).

* EnumChildWindows

BOOL EnumChildWindows (HWND hWndParent,

WNDENUMPROC lpEnumFunc, LPARAM lParam);

Функция EnumChildWindows перечисляет дочерние окна, которые принадлежат определенному родительскому окну, в свою очередь, передавая дескриптор каждого дочернего окна в функцию повторного вызова, определяемую программой. Функция работает до тех пор, пока не будет перечислено последнее дочернее окно или функция повторного вызова не возвратит значение ЛОЖЬ (FALSE).

Параметры: hWndParent - дескриптор родительского окна, дочерние окна которого должны быть перечислены; lpEnumFunc - указатель на определяемую программой функцию повторного вызова; lParam - устанавливает определяемое программой значение, которое будет передано в функцию повторного вызова.

Если функция завершилась успешно, возвращается значение не нуль. Если функция завершилась ошибкой, возвращаемое значение - ноль.

* EnumWindows

BOOL EnumWindows (WNDENUMPROC lpEnumFunc, LPARAM lParam);

Функция EnumWindows перечисляет все окна верхнего уровня на экране, передавая дескриптор каждого окна, в свою очередь, в определяемую программой функцию повторного вызова. EnumWindows действует до тех пор, пока последнее окно верхнего уровня не будет перечислено, или пока функция повторного вызова не возвратит значение ЛОЖЬ (FALSE).

Параметры: lpEnumFunc - указатель на определяемую программой функцию повторного вызова; lParam - устанавливает определяемое программой значение, которое будет передано в функцию повторного вызова.

Если функция завершилась успешно, возвращается значение не нуль. Если функция завершилась ошибкой, возвращаемое значение - ноль.

* FindWindowEx

HWND FindWindowEx (HWND hwndParent, HWND hwndChildAfter,

LPCTSTR lpszClass, LPCTSTR lpszWindow);

Функция FindWindowEx извлекает дескриптор окна, имя класса и имя окна которого соответствуют заданным строкам. Эта функция ищет дочерние окна, начиная с первого из последующих заданных дочерних окон. Эта функция не исполняет поиск, зависимый от регистра.

Параметры: hwndParent - дескриптор родительского окна, дочерние окна которого должны быть найдены; hwndChildAfter - дескриптор дочернего окна; lpszClass - указатель на строку с нулевым символом в конце, которая определяет имя класса или класс атома; lpszWindow - указывает на строку с нулевым символом в конце, которая определяет имя окна (заголовок окна). Если этот параметр - ПУСТО (NULL), соответствуют имена всех окон.

Если функция завершилась успешно, возвращаемое значение - дескриптор окна, которое имеет заданное имя класса и имя окна. Если функция завершается ошибкой, возвращаемое значение - ПУСТО (NULL).

**2. Выполнение работы.**

1. Для программных экспериментов необходимы три вспомогательных приложения:

* + Программа из ЛР №2
  + Простейшая программа на Borland C++
  + Простейшая программа на C#

2. Создал основное приложение Win32 Project (оно будет управлять окнами других приложений).

При создании окна ЛР №3 программа проверяет, созданы ли вспомогательные окна, и если нет, выводит соответствующее сообщение.

3. Добавил в основное приложение следующие возможности:

* + Спрятать окно вспомогательного приложения, если оно видимо;

HWND ButHide2;

const int ButHide2ID = 1;

ButHide2 = CreateWindow(L"BUTTON", L"Спрятать/показать 2 вспомогательное приложение ", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 50, 10, 500, 30, hWnd, NULL, hInstance, NULL);

if (ButHide2 != NULL) SetWindowLong(ButHide2, GWL\_ID, ButHide2ID);

case ButHide2ID:

hLab02 = FindWindow(NULL, L"SPLab002");

if (IsWindowVisible(hLab02))

ShowWindow(hLab02, SW\_HIDE);

else

ShowWindow(hLab02, SW\_SHOWNOACTIVATE);

break;

* + Сделать недоступным окно вспомогательного приложения;

HWND ButActive3;

const int ButActive3ID = 2;

ButActive3 = CreateWindow(L"BUTTON", L"Сделать (не)доступными окнo вспомогательного приложения 3", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 50, 60, 500, 30, hWnd, NULL, hInstance, NULL);

if (ButActive3 != NULL) SetWindowLong(ButActive3, GWL\_ID, ButActive3ID);

case ButActive3ID :

hLab03 = FindWindow(NULL, L"Dli SP");

if (IsWindowEnabled(hLab03))

EnableWindow(hLab03, FALSE);

else

EnableWindow(hLab03, TRUE);

break;

4. Управление приложением из ЛР №2.

* + Нажать (программно) кнопку и выполнить команду;

HWND ButPress1;

const int ButPress1ID = 3;

ButPress1 = CreateWindow(L"BUTTON", L"ЛР №2: нажать кнопку и выполнить команду", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 50, 110, 500, 30, hWnd, NULL, hInstance, NULL);

if (ButPress1 != NULL) SetWindowLong(ButPress1, GWL\_ID, ButPress1ID);

case ButPress1ID:

hLab01 = FindWindow(NULL, L"Project4");

if (hLab01)

hBtn = FindWindowEx(hLab01, NULL, L"BUTTON", L"Кнопка 1");

if (hBtn)

SendMessage(hBtn, BM\_CLICK, 0, 0);

break;

* + Выполнить команду пункта меню;

case ButMenu1ID:

hLab01 = FindWindow(NULL, L"Project4");

if (hLab01)

SendMessage(hLab01, WM\_COMMAND, IDM\_ABOUT, NULL);

break;

* + Заставить программу выполнить действия, соответствующие нажатию пользователем правой кнопки мыши;

case WM\_RBUTTONDOWN:

{

hLab01 = FindWindow(NULL, L"Project4");

if (hLab01)

SendMessage(hLab01, WM\_RBUTTONDOWN, MK\_RBUTTON, lParam);

}

break;

case WM\_RBUTTONUP:

{

hLab01 = FindWindow(NULL, L"Project4");

if (hLab01)

SendMessage(hLab01, WM\_RBUTTONUP, MK\_RBUTTON, lParam);

}break;

* + Заставить окно переместиться по экрану (послав, соответствующие сообщения, так как это окно умеет перемещаться при движении мыши в области клиента);

case WM\_LBUTTONDOWN:

{

SendMessage(hLab01, WM\_LBUTTONDOWN, MK\_LBUTTON, lParam);

}

break;

case WM\_LBUTTONUP:

{

SendMessage(hLab01, WM\_LBUTTONUP, MK\_LBUTTON, lParam);

}

break;

case WM\_MOUSEMOVE:

{

hLab01 = FindWindow(NULL, L"Project4");

if (hLab01) {

POINT coords;

coords.x = LOWORD(lParam);

coords.y = HIWORD(lParam);

ClientToScreen(hWnd, &coords);

ScreenToClient(hLab01, &coords);

SendMessage(hLab01, WM\_MOUSEMOVE, wParam, MAKELONG(coords.x, coords.y));

}

}

break;

5. В основное приложение добавить кнопку, при нажатии на которую программа найдет и пронумерует все запущенные в системе окна (а также и их дочерние). Окнам следует дать имена с номерами.

Инициализируем необходимые переменные:

HWND hStatic;

HWND hStatic1;

int windowCounter=0;

int ChildCounter = 0;

int ChildCounter1 = 0;

int num = 74;

char AllMesseger[400];

HWND Child;

char NameChild[10];

Объявим функцию обратного вызова для переименования и нумерации окон:

BOOL CALLBACK RenameChild(HWND hWnd, LPARAM lParam);

BOOL CALLBACK RenameWindow(HWND hWnd, LPARAM lParam);

И определим ее:

BOOL CALLBACK RenameChild(HWND hWnd, LPARAM lParam) {

ChildCounter++;

ChildCounter1++;

if (ChildCounter == num) {

GetWindowTextA(hWnd, NameChild, MAX\_LOADSTRING);

Child = hWnd;

}

TCHAR newName[MAX\_LOADSTRING] = \_T("Дочернее");

TCHAR windowNumber[10];

\_itot\_s(ChildCounter, windowNumber, 10);

\_tcscat\_s(newName, windowNumber);

SendMessage(hWnd, WM\_SETTEXT, NULL, LPARAM(newName));

return TRUE;

}

BOOL CALLBACK RenameWindow(HWND hWnd, LPARAM lParam) {

windowCounter++;

TCHAR newName[MAX\_LOADSTRING] = \_T("Окно");

TCHAR windowNumber[10];

\_itot\_s(windowCounter, windowNumber, 10);

\_tcscat\_s(newName, windowNumber);

SendMessage(hWnd, WM\_SETTEXT, NULL, LPARAM(newName));

EnumChildWindows(hWnd, &RenameChild, lParam);

return TRUE;

} GetWindowText(tW, lpWinTitle, 256 - 1);

string str = "Дочернее окно №" + to\_string(i);

wstring s1 = wstring(str.begin(), str.end());

const wchar\_t\* s = s1.c\_str();

SendMessage(tW, WM\_SETTEXT, 0, (LPARAM)s);

TCHAR lpWinTitle1[256];

GetWindowText(tW, lpWinTitle1, 256 - 1);

return true;

}

}

Теперь кнопки:

case ButNumber1ID:

GetWindowTextA(hWnd, AllMesseger, MAX\_LOADSTRING);

EnumWindows(&RenameWindow, NULL);

Helper(hWnd);

strcat\_s(AllMesseger, NameChild);

strcat\_s(AllMesseger, " ");

Helper(Child);

SetWindowTextA(hStatic, AllMesseger);

char str[MAX\_LOADSTRING];

char str1[MAX\_LOADSTRING];

\_itoa\_s(ChildCounter, str, 10);

strcat\_s(str, " - количество окон ,");

\_itoa\_s(windowCounter, str1, 10);

strcat\_s(str1, " - количество дочерних окон ");

strcat\_s(str, str1);

SetWindowTextA(hStatic1, str);

break;

6. Запустить все приложения (четыре + плюс стандартные Windows и т.д. и пронумеровать окна (визуально определить примерное их количество, а также определить какие элементы являются окнами, а какие нет).