

МИНЕСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Отчет к лабораторной работе № 2

Дисциплина: «Системное программирование»

Михаил Чуворкин
28.9.2021

Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо ознакомиться (по литературе или в сети) и сделать краткое описание следующих функций и параметров, а также используемых событий (также для работы будут необходимы функции и события, рассмотренные в работе № 1):

- 1). Тип данных TCHAR, переключение режима кодировки символов, функции для работы со строками (strlen, strcpy, strcat, itoa, atoi).

В настройках проекта на вкладке GENERAL есть параметр CHARACTER SET который указывает в какой кодировке будет компилироваться программа. Если указан параметр «Use Unicode Character set», тип **TCHAR** будет транслироваться в тип **wchar_t**. Если указан параметр «Use Multi-byte character set» то тогда **TCHAR** будет транслироваться в тип **char**.

```
#ifdef _UNICODE
typedef wchar_t TCHAR; // используется «широкий символ»
#else
typedef char TCHAR; // используется обычный символьный тип
#endif
```

size_t strlen(const char* string) - возвращает длину строки.

char* strcpy (
 char* destptr, // указатель на буфер назначения
 const char* srcptr // указатель на исходную строку
) - копирует Си-строку **srcptr**, включая завершающий нулевой символ в строку назначения, на которую ссылается указатель **destptr**. Функция возвращает значение **destptr**.

char* strcat(
 char* destptr, // Указатель на строку назначения, к которой добавятся символы строки **srcptr**
 const char* srcptr // Си-строка, которая добавляется в конец строки **destptr**
) - Объединение строк. Функция добавляет копию строки **srcptr** в конец строки **destptr**. Нулевой символ конца строки **destptr** заменяется первым символом строки **srcptr**, и новый нуль-символ добавляется в конец уже новой строки, сформированной объединением символов двух строк в строке **destptr**. Функция возвращает значение **destptr**.

char* itoa(int value, **char*** string, int radix) - преобразует целое число **value** в строку **string** в формате **radix**. К цифрам числа **value** подбираются ANSI символы типа **char** и записываются в строку **string**. Функция не имеет возвращаемого значения, соответствующего ошибке. Необходимо заботиться о том, чтобы строка для вывода данных была до-статочно длинной, чтобы вместить в себя результат. Максимальная необходимая длина составляет 17 байт.(**itoa** is deprecated, use **_itoa_s** instead for **char***, **_itot_s** for **TCHAR**)

`int atoi(const char* string)` - преобразует строку `string` в целое значение типа `int`. Анализируя строку `string`, `atoi` интерпретирует её содержание, как целое число, которое возвращается как `int`. Строка должна быть нуль-терминированной, то есть оканчиваться символом «\0». Строка должна содержать корректную запись целого числа. В противном случае возвращается 0. Число может завершаться любым символом, который не может входить в состав строкового представления целого числа. Сюда относятся пробелы, знаки пунктуации и другие знаки, не являющиеся цифрами.

2). LOWORD, HIWORD.

`WORD LOWORD(DWORD dwValue);` - извлекает младшее слово (первые 2 байта) из заданной величины.

`WORD HIWORD(DWORD dwValue);` - извлекает старшее слово (2 байта) из заданной величины.

Используется для получения координат мыши

3). Функция SendMessage

```
LRESULT SendMessage(  
    HWND hWnd, // дескриптор окна, принимающего сообщение  
    UINT Msg,  // определяет сообщение, которое будет  
               // отправлено.  
    WPARAM wParam, // доп. информация  
    LPARAM lParam // доп. информация  
);
```

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.

Чтобы отправить сообщение и вернуть немедленно значение, используйте функцию `SendMessageCallback` или `SendNotifyMessage`. Чтобы поместить сообщение в очередь сообщений потока и вернуть немедленно значение, используйте функцию `PostMessage` или `PostThreadMessage`.

Если параметр `hWnd` имеет значение `HWND_BROADCAST`, сообщение отправляется всем окнам верхнего уровня в системе, включая заблокированные или невидимые, не имеющие владельца, перекрывающие и выскакивающие окна; но сообщение не отправляется дочерним окнам.

Например, передать сообщение главному окну

```
SendMessage(hWnd,WM_LBUTTONDOWN,MK_LBUTTON,MAKELONG(100,100));
```

или сообщение для кнопки `SendMessage(hBut1,BM_SETSTATE,TRUE,0);`

4). Стилъ окна: CS_DBLCLKS, сообщение WM_LBUTTONDOWNBLCLK.

Стилъ **CS_DBLCLKS** (0x0008) Функция окна будет получать сообщения при двойном щелчке клавишей мыши.

Сообщение **WM_LBUTTONDOWNBLCLK** посылается, когда пользователь дважды нажимает на левую кнопку мыши, пока курсор находится в клиентской области окна. Если мышь не захвачена, сообщение посылается окну под курсором. Иначе, сообщение посылается окну, которое имеет захват мыши.

5). Сообщение WM_NCHITTEST, процедура DefWindowProc.

Сообщение **WM_NCHITTEST** отправляется в окно, чтобы определить, какая часть окна соответствует определенной координате экрана. Это может произойти, например, при перемещении курсора, при нажатии или отпуске кнопки мыши или в ответ на вызов функции, такой как `WindowFromPoint`. Если мышь не захвачена, сообщение отправляется в окно под курсором. В противном случае сообщение отправляется в окно, которое захватывает мышь.

Параметр `lParam` сообщения **WM_NCHITTEST** содержит экранные координаты острия курсора. Функция `DefWindowProc` проверяет эти координаты и возвращает значение местоположения курсора, которое указывает место острия.

```
LRESULT DefWindowProc(  
    HWND    hWnd, // дескриптор окна, принимающего сообщение  
    UINT    Msg,  // определяет сообщение, которое будет  
отправлено.  
    WPARAM wParam, // доп. информация (зависит от Msg)  
    LPARAM lParam // доп. информация (зависит от Msg)  
);
```

Функция `DefWindowProc` вызывается оконной процедурой по умолчанию, чтобы обеспечить обработку по умолчанию любого сообщения окна, которые приложение не обрабатывает. Эта функция гарантирует то, что обрабатывается каждое сообщение. Функция `DefWindowProc` вызывается с теми же самыми параметрами, принятыми оконной процедурой.

6). `SetCapture(hWnd); ReleaseCapture();`

```
HWND SetCapture(  
    HWND hWnd // Дескриптор окна в текущем потоке, который  
должен захватить мышь.  
);
```

Функция `SetCapture` устанавливает захват мыши в заданном окне, принадлежащем текущему потоку. `SetCapture` захватывает ввод данных от мыши или когда мышь находится над захватывающим окном, или когда нажималась кнопка мыши, в то время, когда мышь была над захватывающим окном, а кнопка все еще находилась в нажатом состоянии. Только одно окно одновременно может захватить мышь. Возвращаемое значение - дескриптор окна, которое перед этим захватило мышь. Если такого окна нет, возвращаемое значение - ПУСТО (NULL).

`BOOL ReleaseCapture(VOID);`

Функция `ReleaseCapture` освобождает захват мыши окном в текущем потоке и восстанавливает обычную обработку ввода данных от мыши. Окно, которое захватило мышь, получает весь ввод данных от мыши, независимо от позиции курсора, кроме тех случаев, когда кнопкой мыши щелкают в то время, когда острие курсора находится в окне другого потока. Если функция завершается успешно, возвращаемое значение не ноль.

Задание

1. Создать приложение Win32 Project (в Microsoft Visual Studio). Запустив приложение, убедиться в том, что оно работает, а окно с помощью мыши перемещается по экрану (курсор мыши находится в заголовке окна).
2. В окно приложения добавить две кнопки, поля ввода и вывода. Первая кнопка должна возводить в квадрат целое число, введенное пользователем, вторая кнопка должна заставить первую кнопку “нажаться” и выполнить код (предусмотреть два возможных варианта воздействия на первую кнопку).

Создадим кнопки и поля ввода и вывода:

```
HWND btnMaster; // управляющая кнопка
const int idBtnMaster = 0;

HWND btnSlave; // управляемая кнопка
const int idBtnSlave = 1;

HWND editText; // для введения числа для возведения в квадрат
const int idEditText = 2;

HWND staticText; // для вывода квадрата
const int idStaticText = 3;

btnMaster = CreateWindow(L"BUTTON", L"Square it!", WS_VISIBLE | WS_CHILD |
    BS_PUSHBUTTON, 125, 75, 100, 25, hWnd, (HMENU)idBtnMaster, hInstance, NULL);

btnSlave = CreateWindow(L"BUTTON", L"Push first", WS_VISIBLE | WS_CHILD |
    BS_PUSHBUTTON, 275, 75, 100, 25, hWnd, (HMENU)idBtnSlave, hInstance, NULL);

editText = CreateWindow(L"EDIT", L"I am EDIT", WS_VISIBLE | WS_CHILD | NULL |
    WS_BORDER, 125, 15, 250, 25, hWnd, (HMENU)idEditText, hInstance, NULL);

staticText = CreateWindow(L"STATIC", L"I am STATIC", WS_VISIBLE | WS_CHILD | NULL,
    125, 125, 250, 125, hWnd, (HMENU)idStaticText, hInstance, NULL);
```

И Обработчики нажатия кнопок:

```
case idBtnMaster:
{
    // первый вариант
    //SendMessage(btnSlave, BM_CLICK, 0, 0);
    // второй вариант
    SendMessage(btnSlave, WM_LBUTTONDOWN, 0, 0);
}
```

```

        SendMessage(btnSlave, WM_LBUTTONUP, 0, 0);
    }
    break;
case idBtnSlave:
{
    TCHAR editTextContent[MAX_LOADSTRING];
    GetWindowText(editText, editTextContent, MAX_LOADSTRING);
    // _ttoi универсальна для различных вариантов TCHAR
    // https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/atoi-
    atoi-l-wtoi-wtoi-l?redirectedfrom=MSDN&view=msvc-160
    int number = _ttoi(editTextContent);
    // _itot_s - макрос для вызова нужной функции, соотв. типу текущего
    варианта TCHAR
    // https://docs.microsoft.com/en-us/cpp/c-runtime-library/routine-
    mappings?view=msvc-160
    _itot_s(number * number, editTextContent, MAX_LOADSTRING, 10);
    SetWindowText(staticText, editTextContent);
}
    break;

```

3. По щелчку на правой кнопке мыши определить и вывести координаты курсора.

Напишем обработчик для нажатия на правую кнопку мыши и вывод координат в staticText

```

case WM_RBUTTONDOWN:
{
    POINT mouseP;
    mouseP.x = LOWORD(lParam);
    mouseP.y = HIWORD(lParam);
    TCHAR strX[10], strY[10];
    _itot_s(mouseP.x, strX, 10);
    _itot_s(mouseP.y, strY, 10);
    // _tcscat_s - так же макрос для TCHAR, аналог strcat
    // _T(x) - так же для TCHAR - используется для приведения к соотв. типу
    // не удалось понять, дают ли макросные функции сохранение \0 в конце строки
    _tcscat_s(strX, _T("\n"));
    _tcscat_s(strX, strY);
    SetWindowText(staticText, strX);
}
break;

```

4. Прodelать программные эксперименты и проследить за работой событий (для эксперимента можно записывать информацию о событии в строковую переменную, а затем вывести ее в окне сообщений):

- WM_LBUTTONDBLCLK: какие события и сколько раз срабатывают при двойном щелчке (WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP);
- WM_NCHITTEST: сколько раз возникает данное событие при работе с мышью.

Добавим стиль окна, определяющий возможность обрабатывать двойные нажатия

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

Создадим глобальную строку для записи сообщений мыши:

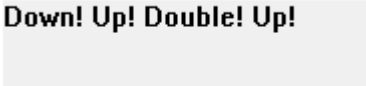
```
TCHAR mouseExpStr[200];
```

Напишем обработчик для нажатий на ЛКМ:

```
case WM_LBUTTONDOWN:
{
    _tcscat_s(mouseExpStr, _T("Down! "));
    SetWindowText(staticText, mouseExpStr);
}
break;
case WM_LBUTTONUP:
{
    _tcscat_s(mouseExpStr, _T("Up! "));
    SetWindowText(staticText, mouseExpStr);
}
break;
case WM_LBUTTONDBLCLK:
{
    _tcscat_s(mouseExpStr, _T("Double! "));
    SetWindowText(staticText, mouseExpStr);
}

break;
```

По результатам эксперимента, вместо второго сообщения `WM_LBUTTONDOWN` возникает сообщение `WM_LBUTTONDBLCLK`



Down! Up! Double! Up!

Создадим счетчик для записи количества сообщений `WM_NCHITTEST` и окно `STATIC` для вывода

```
HWND staticNCHITTEST;
const int idStaticNCHITTEST = 4;
int counterNCHITTEST = 0;

staticNCHITTEST = CreateWindow(L"STATIC", L"I am STATIC", WS_VISIBLE |
WS_CHILD | NULL, 125, 275, 250, 50, hWnd, (HMENU)idStaticNCHITTEST, hInstance,
NULL);

case WM_NCHITTEST:
{
    counterNCHITTEST++;
    TCHAR strNCHITTEST[10];
    _itot_s(counterNCHITTEST, strNCHITTEST, 10);
    SetWindowText(staticNCHITTEST, strNCHITTEST);
    return DefWindowProc(hWnd, message, wParam, lParam);
}
break;
```

По результатам эксперимента, сообщения `WM_NCHITTEST` возникают при нажатии на любую кнопку мыши и перемещении мыши на 1 пиксель в регионе окна `hWnd` (т.е. при нажатии на кнопки и перемещении в регионе кнопок оно не возникает).

5. Разработать программный код, который позволит пользователю с помощью мыши перемещать окно, но мышь при этом нажимается и перемещается в клиентской области окна.

```
int mouseIsDown = 0; // для хранения текущего состояния кнопки мыши
```

```
POINT oldPoint; // для хранения прошлого состояния положения курсора
```

Обновляем `mouseIsDown` в обработчиках сообщений `WM_LBUTTONDOWN` и `WM_LBUTTONUP`

И обрабатываем сообщение перемещения мыши:

```
case WM_MOUSEMOVE:
{
    POINT coords;
    coords.x = LOWORD(lParam);
    coords.y = HIWORD(lParam);
    if (mouseIsDown)
    {
        //SetCapture(hWnd);
        RECT mWindowRect;
        GetWindowRect(hWnd, &mWindowRect);
        MoveWindow(hWnd, mWindowRect.left + coords.x - oldPoint.x,
mWindowRect.top + coords.y - oldPoint.y, mWindowWidth, mWindowHeight, true);
    }
    else {
        //ReleaseCapture();
        oldPoint = coords;
    }
}

break;
```

6. Сохранить разработанное приложение так, чтобы его можно было использовать в программных экспериментах следующей лабораторной работы.

Полный код программы доступен здесь: <https://pastebin.com/zQ21e6ML>