# Cvičení ITU - Událostmi řízená aplikace

## Obsah



## Pokyny k překladu:

**Microsoft Visual Studio 2010**
- Nabootujte do Windows
- Stáhněte si šablonu ze stránek, rozbalte a otevřete **WinApi.vcxproj** ve Visual Studiu 2010.
- V souboru **main.c** je připravena šablona. Po překladu a spuštění se otevře prázdné okno.
- **Pokud se Vám špatně zobrazuje čeština nastavte kódování znaků:** *Project->Properties->Configuration properties->General->Character set* na možnost *Not set*

**Prezentace**
- Stáhněte si zde.

## Princip aplikace ve WinAPI

### Registrace a vytvoření nového okna

Systém Windows je založen na principu zasílání zpráv. Zprávy slouží např. k informaci aplikace o stisku klávesy, posuny myši, změně velikosti okna atd. Každá zpráva se skládá z identifikátoru a dvou parametrů. Klíčovou částí aplikace používající pouze WinAPI je hlavní smyčka aplikace, která zabezpečuje příjem a zpracování jednotlivých zpráv zaslaných aplikaci. Tato smyčka bývá zpravidla umístěna ve funkce `WinMain`, která se volá při spuštění okenní aplikace. Zde s také provádí inicializace aplikace a tvorba hlavního okna.

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

// Global variable

HINSTANCE hInst;

// Function prototypes.
int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int);
LRESULT CALLBACK MainWndProc(HWND, UINT, WPARAM, LPARAM);

// Application entry point.

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
{
    MSG msg;
    BOOL bRet;
```

```
WNDCLASS wcx;          // register class
HWND hWnd;

hInst = hInstance;     // Save the application-instance handle.
```

Nejdříve musíme nadefinovat a zaregistrovat třídu okna. Podle této třídy pak vytvoříme nové okno naší aplikce. Všimněte si parametru `lpfnWndProc` (viz. dále), kde definujeme název funkce, která bude obsluhovat události týkajících se okna. Dále pak jméno třídy. To bude použito jako identifikátor při vytváření vlastního okna.

```
// Fill in the window class structure with parameters that describe the main wind

wcx.style = CS_HREDRAW | CS_VREDRAW;              // redraw if size changes
wcx.lpfnWndProc = (WNDPROC) MainWndProc;                  // points to window p
wcx.cbClsExtra = 0;                               // no extra class memory
wcx.cbWndExtra = 0;                               // no extra window memory
wcx.hInstance = hInstance;                        // handle to instance
wcx.hIcon = LoadIcon(NULL, IDI_APPLICATION);      // predefined app. icon
wcx.hCursor = LoadCursor(NULL, IDC_ARROW);        // predefined arrow
wcx.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);  // white background br
wcx.lpszMenuName =  (LPCSTR) "MainMenu";                   // name of menu resour
wcx.lpszClassName = (LPCSTR) "MainWClass";                 // name of window clas

// Register the window class.

if (!RegisterClass(&wcx)) return FALSE;

// create window of registered class

hWnd = CreateWindow(
    "MainWClass",        // name of window class
    "ITU",       // title-bar string
    WS_OVERLAPPEDWINDOW, // top-level window
    50,                  // default horizontal position
    100,                 // default vertical position
    750,                 // default width
    150,                 // default height
    (HWND) NULL,         // no owner window
    (HMENU) NULL,        // use class menu
    hInstance,           // handle to application instance
    (LPVOID) NULL);      // no window-creation data
if (!hWnd) return FALSE;

// Show the window and send a WM_PAINT message to the window procedure.
// Record the current cursor position.

ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

// loop of message processing

while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)
{
    if (bRet == -1)
    {
        // handle the error and possibly exit
    }
    else
    {
        TranslateMessage(&msg);
```

```
            DispatchMessage(&msg);
        }
    }
    return (int) msg.wParam;
}
```

Zpět na obsah

### Zpracování jednotlivých zpráv

Další důležitou částí aplikace je funkce zpracovávající jednotlivé zprávy doručené danému oknu aplikace. V ukázkové aplikaci je to funkce `MainWndProc`. Můžete zde vidět větvení do příslušných funkcí programu, podle doručené události a volání funkce DefWindowProc zabezpečující standardní zpracování ostatních událostí, které nejsou zpracovávány ve vlastní režii aplikace. Nastavení této funkce pro hlavní okno se provádí při registraci třídy okna ve funkci `WinMain`. Základní typy událostí jsou uvedené v ukázkové aplikaci (zatím se na ně nijak nereaguje).

```
LRESULT CALLBACK MainWndProc(
    HWND hWnd,        // handle to window
    UINT uMsg,        // message identifier
    WPARAM wParam,    // first message parameter
    LPARAM lParam)    // second message parameter
{

    switch (uMsg)
    {
        case WM_CREATE:
            // Initialize the window.
            return 0;

        case WM_SIZE:
            // Set the size and position of the window.
            return 0;

        case WM_DESTROY:
            // Clean up window-specific data objects.
            PostQuitMessage(0);
            return 0;

        //
        // Process other messages.
        //

        default:
            return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
}
```

Zpět na obsah

## Vykreslování do okna

K užitečným událostem patří mimo jiné událost `WM_PAINT`, která je poslána oknu pokud se má toto okno překreslit. Veškeré zobrazování do okna by mělo být prováděno v reakci na tuto událost. Pokud požadujeme překreslit dané okno, lze použít funkci `InvalidateRect` pro zneplatnění obsahu okna, což je také demonstrováno. V ukázkové aplikaci se volá funkce `onPaint`, ve které se provádí vlastní zobrazování jednotlivých objektů. Vykreslování do okna stejně jako např. na tiskárnu se provádí ve Windows prostřednictvím tzv. **device contextu**. Funkce `onPaint` dostane však jako parametr handle okna, které se má překreslit. Proto je nutné získat device context pomocí funkce `GetDC` případně

BeginPaint jak se provádí v ukázkové aplikaci. Po provedení kreslení se musí přidělený device context uvolnit funkcí DeleteDC a v případě použití funkce BeginPaint zavolat také funkci EndPaint. K samotnému vykreslování slouží funkce např. MoveToEx, LineTo, TextOut (více na http://msdn.microsoft.com/library).

Doplníme globální proměnné, definujeme a deklarujeme funkci `onPaint`, přidáme počítadlo zpráv, ošetříme zprávu `WM_PAINT`.

```
// Global variable

UINT  MessageCount = 0;
UINT  Count = 0;

// Function prototypes.

void onPaint(HWND hWnd);

...

LRESULT CALLBACK MainWndProc( ...

    ++MessageCount;
    InvalidateRect(hWnd, NULL, 0);
    switch (uMsg)
    {
                ...
        case WM_PAINT:
            // Paint the window's client area.
            onPaint(hWnd);
            return 0;
        ...
    }
        ...
}

void onPaint(HWND hWnd)
{
    PAINTSTRUCT ps;                 // information can be used to paint the client ar
    HDC         hDC;                // device context
    char        text[256];          // buffer to store an output text
    HFONT       font;               // new large font
    HFONT       oldFont;            // saves the previous font

    hDC = BeginPaint(hWnd, &ps);    // prepares the specified window for painting

    font = CreateFont(100,0,0,0,0,FALSE,FALSE,FALSE,ANSI_CHARSET,OUT_DEFAULT_PRECIS,C

    oldFont = (HFONT)SelectObject(hDC,font);

    sprintf(text, "%05u, Msg:%05u", Count, MessageCount);  // output text
    TextOut(                        // writes a character string at the specified loc
        hDC,                        // handle to device context
        1,1,
        text,                       // character string
        (int) strlen(text));              // number of characters

    SelectObject(hDC,oldFont);

    DeleteObject(font);

    DeleteDC(hDC);                  // deletes the specified device context
    EndPaint(hWnd, &ps);            // marks the end of painting in the specified win
}
```

Zpět na obsah

# Práce s klávesnicí

Vstup z klávesnice probíhá zejména prostřednictvím zpráv generovaných na základě událostí na klávesnici. Tyto zprávy jsou doručeny vždy oknu, na které "je zaostřeno". Základní druhy zpráv:

- `WM_CHAR` - tento typ zprávy přichází po zadání znaku na klávesnici
- `WM_KEYDOWN` - tento typ zprávy přichází po stisku klávesy na klávesnici (nebo při "autorepeat")
- `WM_KEYUP` - tento typ zprávy přichází po uvolnění klávesy na klávesnici

Pozn.: Zprávy jsou doručovány oknu, na které je "zaostřeno" (nikoli, jak tomu bývá v některých verzích nadstaveb XWindow, nad kterým je myší kursor).

Pro čtení tisknutelných znaků z klávesnice je vhodné ošetřit událost `WM_CHAR`. Pro ostatní klávesy (šipky, funkční klávesy, apod.) je nutné ošetřit zprávy `WM_KEYDOWN`, popř. `WM_KEYUP`.

Překlad zpráv `WM_KEYDOWN` a `WM_KEYUP` na zprávy `WM_CHAR` je realizován v hlavní smyčce aplikace pomocí funkce `TranslateMessage`.

```
#define BUFSIZE 65535

LRESULT APIENTRY MainWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    static int nchLast;          // amount of chars in the input buffer
    static RECT TextArea;        // client area
    static PTCHAR pchInputBuf;   // input buffer
    int chWidth;                 // width of the actual character
    int i;                       // counter
    int nVirtKey;               // virtual-key code
    HDC hdc;                    // handle to device context
    PAINTSTRUCT ps;             // required by BeginPaint

    switch (uMsg)
    {
        case WM_CREATE:
            // Allocate a buffer to store keyboard input.
            pchInputBuf = (LPTSTR) GlobalAlloc(GPTR, BUFSIZE * sizeof(TCHAR));
            nchLast = 0;
            pchInputBuf[nchLast] = TEXT('\0');
            break;

        case WM_CHAR:
          switch (wParam) {
                case 0x08:  // backspace
                case 0x0A:  // linefeed
                case 0x1B:  // escape
                    MessageBeep((UINT) -1);
                    break;

                case 0x09:  // tab
                    // Convert tabs to four consecutive spaces.
                    for (i = 0; i < 4; i++) SendMessage(hWnd, WM_CHAR, 0x20, 0);
                    break;

                case 0x0D:  // carriage return
                    // Record the carriage return and position the caret at the begin
                    pchInputBuf[nchLast++] = 0x0D;
                    break;
```

```
                default:    // displayable character
                    // Store the character in the buffer.
                    pchInputBuf[nchLast++] = (TCHAR) wParam;
                    break;
            }
            InvalidateRect(hWnd, NULL, TRUE);
            break;

        case WM_KEYDOWN:
            switch (wParam) {
                case VK_LEFT:   // LEFT ARROW
                    break;

                case VK_RIGHT:  // RIGHT ARROW
                    nVirtKey = GetKeyState(VK_SHIFT);
                    if (nVirtKey & 0x8000) {
                    }
                    break;
            }
            break;

        case WM_SIZE:
            GetClientRect(hWnd, &TextArea);
            break;

        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);

            // Set the clipping rectangle, and then draw the text into it.
            DrawText(hdc, pchInputBuf, nchLast, &TextArea, DT_LEFT);

            EndPaint(hWnd, &ps);
            break;

        case WM_DESTROY:
            PostQuitMessage(0);

            // Free the input buffer.
            GlobalFree((HGLOBAL) pchInputBuf);
            break;

        default:
            return DefWindowProc(hWnd, uMsg, wParam, lParam);
    }
}
```

V příkladě vykreslujeme při každé změně textu všechny znaky, což není nutné. Při vkládání nového znaku můžeme vykreslovat pouze nový znak. K tomu je ale třeba mít pozici posledně vloženého znaku (viz. další kapitola).

Zpět na obsah

## Práce s myší

S myší se (podobně jako s klávesnicí) pracuje převážně pomocí zpráv generovaných na základě událostí na myši a posílaných oknu, které se nachází "pod myší". Základními zprávami jsou:

- WM_MOUSEMOVE
- WM_xBUTTONDOWN

- `WM_xBUTTONUP`
- `WM_xBUTTONDBLCLK`

kde, x je L, M, R.

Jelikož zprávy jsou zasílány oknu "pod myší" bez ohledu na "zaostření" okna a to pouze pro klientskou oblast okna (ne pro rámeček) je nezbytné ošetřit stav tlačítek myši "bezpečně".

Jednou z možností je použití funkcí `SetCapture` a `ReleaseCapture`. Fukce SetCapture "přesměruje" zprávy myši oknu nastaveného parametrem funkce. Od této chvíle jsou všechny zprávy myši zasílány tomuto oknu. Uvolnění myši se provede funkcí ReleaseCapture. Toto zrušení přesměrování je třeba provést explicitně ihned po ukončení relevantní akce, například po vyjetí myši z okna, puštění tlačítka myši, apod. Jinak hrozí nesprávná funkce myši v celé aplikaci či systému.

Uvažujme příklad, kdy chceme přetáhnout data z jedné aplikace do jiné a zaměřme se prozatím pouze na funkčnost myši. Při stisku tlačítka "přichytíme" myš k "zaostřenému" oknu. Při uvolnění tlačítka známe pozici kursoru, ovšem v souřadnicích klientské oblasti (pozor, souřadnice kursoru jsou relativní k počátku, tzn. při opuštění okna vlevo nahoru získáme záporná čísla). Funkce `ClientToScreen` převede souřadnice z klientské oblasti do souřadnic celé obrazovky.
Nakonec potřebujeme zjistit, nad kterým oknem bylo tlačítko myši uvolněno. Další "kouzelná" funkce `WindowFromPoint` najde okno, které obsahuje zadaný bod.

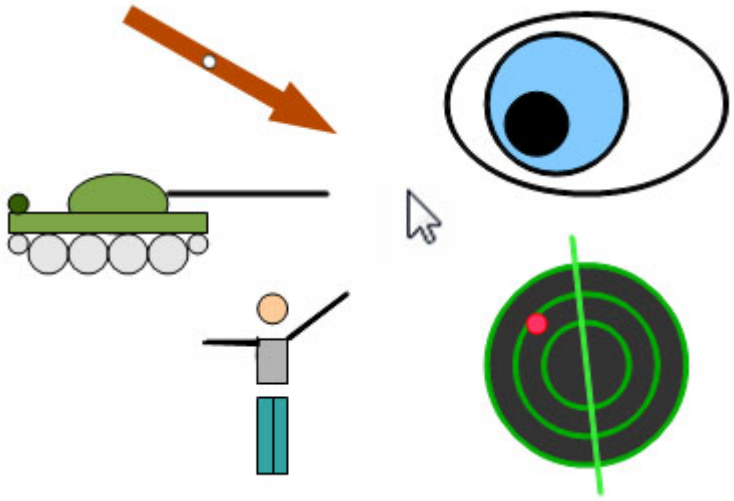
```
POINT ReleasePoint;         // point where mouse button was released
HWND hwndTmp;               // window handle where mouse button was released

        ...
    case WM_LBUTTONDOWN:
        SetCapture(hWnd);
        break;

    case WM_LBUTTONUP:
        ReleasePoint.x = (int)LOWORD(lParam);
        ReleasePoint.y = (int)HIWORD(lParam);
        if (ReleasePoint.x > 0x7FFF) ReleasePoint.x -= (long)0xFFFF;
        if (ReleasePoint.y > 0x7FFF) ReleasePoint.y -= (long)0xFFFF;

        ClientToScreen(hWnd, &ReleasePoint);

        hwndTmp = WindowFromPoint(ReleasePoint);

        SetWindowText( hwndTmp, ... );

        ReleaseCapture();
        break;
```

Zpět na obsah

# Samostatný úkol

## Zadání

S využitím přiložených kódů vytvořte aplikaci, která bude prostřednictvím grafického výstupu informovat o pohybu myši ve svém okně. Aplikace musí vhodně zvoleným způsobem reagovat na stisk tlačítka myši či stisk klávesy (pohyb grafického objektu po obrazovce, změna barevnosti, výpis textu,...). Konkrétní zadání bude specifikováno na každém cvičení!
Demonstrační aplikace je k dispozici ZDE.
Příklady možných grafických výstupů:



**V rámci každého cvičení bude kladen specifický požadavek na funkčnost aplikace - zatímco jedna skupina bude upravovat barevnost výstupu na základě stisku tlačítka myši, jiná bude stiskem klávesy pohybovat objekty, zobrazovat titulky, či vytvářet konkrétní výstup.**

## Odevzdání a bodové ohodnocení

Úkol se bude hodnotit na příslušném cvičení, poté odevzdávat do IS.
**Úkol musí být předveden cvičícímu - pouhé odevzdání do IS nestačí!!!!**
**Body se udělují pouze za práci na cvičení!**
**Do IS odevzdávejte pouze soubor main.c**
Můžete získat max **5 bodů** následovně:

- 1.5 bodu - vykreslení obsahu okna s informací o pohybu myši po obrazovce.
- 1 bod - reakce na stisk klávesy a myši.
- 1.5 bod - splnění specifického požadavku na funkčnost.
- 1 bod - připravenost při předvedení aplikace (schopnost orientace v kódu, pohotová a správná odpověď na otázku)

Opravdu zdařené řešení si lze prohlédnout ZDE (Autorem je Jan Ohnheiser).

## Doporučený postup

1. Vytvořte hlavní okno aplikace(můžete využít nachystané kostry ZDE).

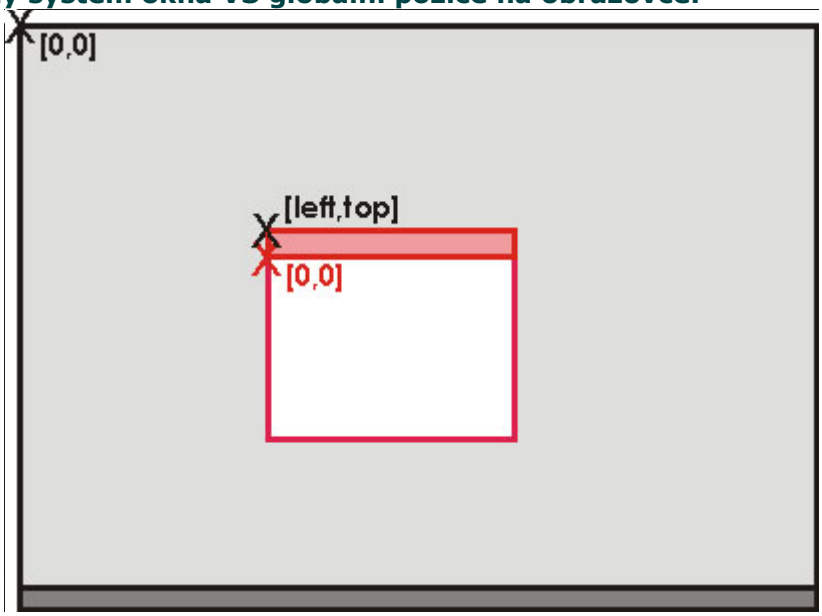
2. Vytvořte vlastní kód pro vykreslení obsahu okna (ideálně parametrizovatelný)
   Využitelné jsou mj. následující funkce:
   - *MoveToEx(), LineTo(), TextOut()* pro vykreslování čar a textu.
   - *Ellipse(), Rectangle(), Polygon()* pro kreslení jednoduchých tvarů.
   - *GetWindowRect()*, *GetClientRect()* zajistí pozici a rozměry okna, resp. jeho klientské části.
   - *CreatePen(), CreateSolidBrush()* vytvoří pero, resp. štětec, se kterým se budou dané tvary vykreslovat-

   Dokumentaci k uvedeným i dalším funkcím lze nalézt na MSDN.
   Ukázky konstrukcí, které dělají občas problémy:

```
//nakreslení čáry z bodu [ax, ay] do bodu [bx, by]
MoveToEx(mhDC, ax, ay, NULL);
LineTo(mhDC, bx, by);

//zjistění pozice okna na obrazovce
RECT pozice_okna;
GetWindowRect(hWnd, &pozice_okna);
//k dispozici jsou nyni udaje: pozice_okna.left, pozice_okna.right, p

//nastavení barvy štětce do DC
```

```
HBRUSH brush, oldbrush
brush = CreateSolidBrush(RGB(140, 40, 20));
oldbrush = SelectObject(hDC, brush);
```

**Lokální souřadný systém okna VS globální pozice na obrazovce:**




3. Zjištění lokální pozice kurzoru myši lze provést např. kombinací funkcí *GetCursorPos()* a *ScreenToClient()*.

```
POINT p
GetCursorPos(&p);
ScreenToClient(hWnd, &p);
```

4. Doplňte funkcionalitu reakcí na stisk kláves a pohyb myši (přidat nové *case* větve ve funkci *MainWndProc()*). Inspirujte se v kapitolkách Práce s klávesnicí a Práce s myší
5. VOLITELNÉ - Dodejte vlastní rozšíření (grafika, animace, práce s textem, lokalizace kurzoru i mimo oblast okna,...)

---


*Tel.: 54114-1145, Fax: 54114 1270*
*Stránku upravil: Jan Kohút, 11.10.2019*