Programowanie w Windows API

Programowanie w Windows API

Windows API
Windows Application Programming Interface (API) – to zestaw funkcji systemu operacyjnego Windows, które umożliwiają aplikacjom korzystanie z wszystkich usług systemu.

W zbiorze Windows API można wydzielić następujące kategorie funkcji:
- **zarządzenie i administracja**: funkcje umożliwiające serwisowanie i konfigurowanie systemu Windows,
- **diagnostyka**: zestaw funkcji monitorujących wydajność systemu,
- **grafika i multimedia**: funkcje wspomagające tworzenie grafiki 2D i 3D, ponadto usługi multimedialne do zarządzania plikami audio i wideo,
- **serwisy sieciowe**: funkcje realizujące usługi sieciowe,
- **bezpieczeństwo**: funkcje dla kryptografii, autoryzacji i autentykacji,
- **serwisy systemowe**: podstawowe usługi związane z obsługą pamięci, plików, urządzeń peryferyjnych, a także procesów i wątków.
- **Windows User Interface**: zestaw funkcji umożliwiających tworzenie i zarządzanie oknami.

W systemach takich jak: Windows Vista, czy Windows 7 nie tylko istnieje kompatybilność wstecz funkcji API, ale biblioteki te są wciąż rozwijane i uzupełniane o nowe usługi.

Korzystanie z funkcji Windows API jest możliwe z poziomu każdego języka programowania, który pozwala wykorzystywać kod zamknięty w bibliotekach łączonych dynamicznie (DLL). Zatem tworzenie aplikacji z wykorzystaniem niskopoziomowego interfejsu API jest również możliwe w asemblerze.

Prosta aplikacja

Najprostsza aplikacja pod Windows wymaga jedynie funkcji wejścia do programu.

W chwili, gdy taka aplikacja zostanie uruchomiona na ekranie nie widać żadnych efektów jej działania.

Zatem w systemie Windows mogą istnieć aplikacje, które nie posiadają interfejsu graficznego.

```
.386
.model flat

ExitProcess     equ _ExitProcess@4

public  _start

.code

_start:

extrn   ExitProcess          : near

        push    large 0
        call    ExitProcess

end _start
```

Kod prostej aplikacji w asemblerze

```
#include <windows.h>

int __stdcall WinMain (HINSTANCE hThisInstance,
                       HINSTANCE hPrevInstance,
                       LPSTR lpszArgument,
                       int nCmdShow)
{
    return 0;
}
```

Kod prostej aplikacji w C

W programie pisanym w języku asemblera dla zakończenia aplikacji należy wywołać funkcję systemową **ExitProcess**. Funkcja ta przyjmuje jeden parametr – kod powrotu z programu, zwyczajowo 0 oznacza brak błędu. Wartość ta jest przekazywana przez stos.

Funkcja ExitProcess w bibliotekach systemowych nosi nazwę _ExitProcess@4. Stąd dla wygody wprowadzono definicję nazwy ExitProcess.

Aplikacja “Hello world!”

Poniżej zaprezentowano kod (w języku C i w asemblerze) aplikacji wyświetlającej w oknie dialogowym napis powitalny „Hello world!”.

```c
#include <windows.h>

int __stdcall WinMain (HINSTANCE hThisInstance,
                       HINSTANCE hPrevInstance,
                       LPSTR lpszArgument,
                       int nCmdShow)
{
    MessageBox(NULL, "Hello world!", "", MB_OK);
    return 0;
}
```

Aplikacja “Hello world!” w C

MessageBox – funkcja wyświetlająca okno dialogowe komunikatu. Parametry:

1. uchwyt do okna rodzica,
2. adres tekstu do wyświetlenia wewnątrz okna,
3. adres tekstu do wyświetlenia na pasku,
4. styl okna dialogowego: MB_OK – tylko przycisk OK.

```
.386
.model flat

ExitProcess     equ _ExitProcess@4
MessageBox      equ _MessageBoxA@16

MB_OK           =   00000000h

public  _start

.data

empty   db 0
message db "Hello world!"

.code

_start:

extrn   MessageBox          : near

        push    large MB_OK
        push    large offset empty
        push    large offset message
        push    large 0
        call    MessageBox

extrn   ExitProcess         : near

        push    large 0
        call    ExitProcess

end _start
```

Aplikacja “Hello world!” w asemblerze

__stdcall – konwencja wywoławcza

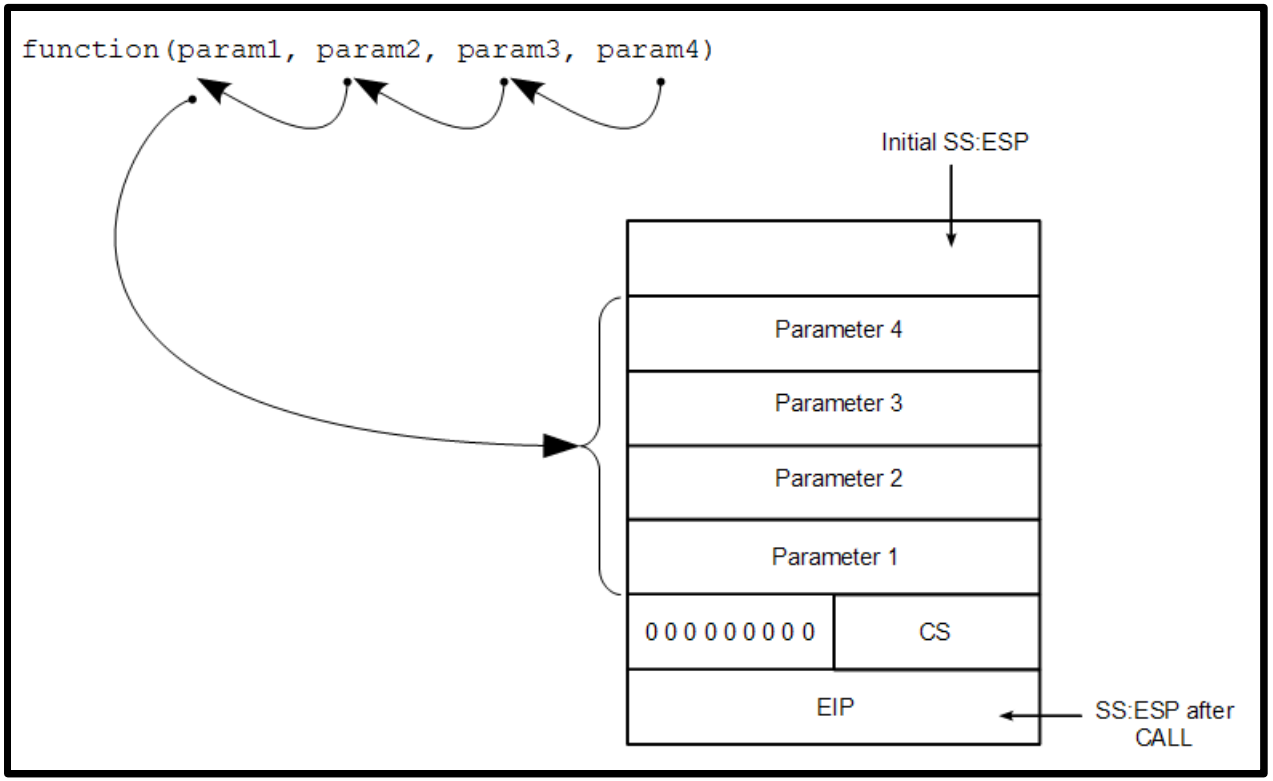
Konwencje wywoławcze określają kolejność z jaką parametry odkładane są na stosie oraz wskazują, która funkcja (wywoływana czy wywołująca) zdejmuje je później ze stosu.

Według konwencji **stdcall** działają funkcje Windows API. Tutaj parametry odkładane są w kolejności od prawej do lewej. Parametry usuwa ze stosu funkcja wywoływana.

Poniższa tabela prezentuje cechy najczęściej używanych konwencji wywoławczych.

Convention	Passing order	Cleaning
stdcall	Right to left	called
C	Right to left	calling
PASCAL	Left to right	called

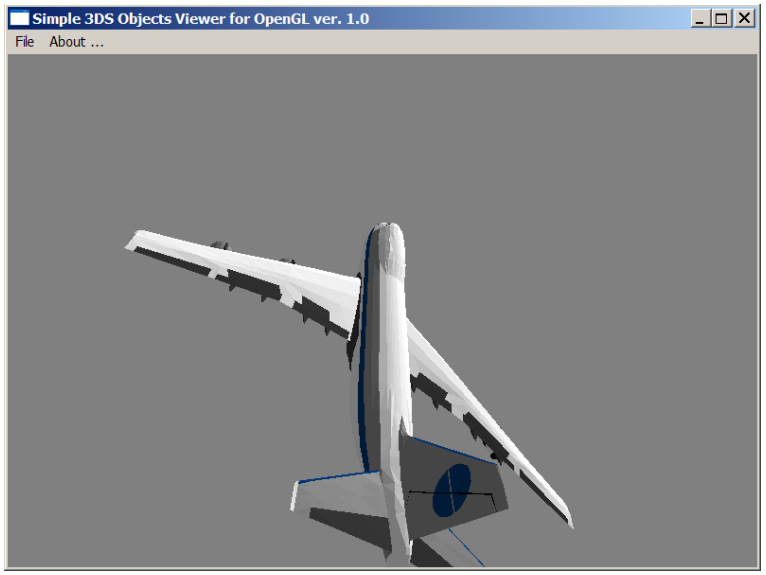
Parametry wywołania funkcji mogą zostać usunięte ze stosu przy pomocy instrukcji RET *n,* gdzie *n* oznacza dodatkową (poza adresem powrotu) liczbę bajtów zdejmowanych ze stosu



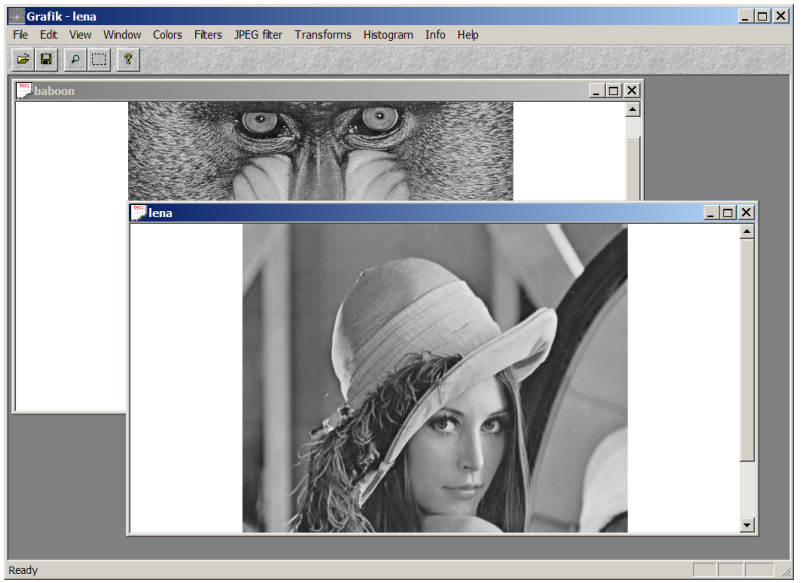
Przekazywanie parametrów w przypadku konwencji stdcall

Single Document Application (SDI)

Aplikacja SDI jest aplikacją posiadającą graficzny interfejs użytkownika (GUI), który zawiera tylko jedno okno dokumentu. Z kolei aplikacja Multiple Data Interface (MDI) może posiadać wiele otwartych okien dokumentu.



Przykład aplikacji SDI



Przykład aplikacji MDI

Zanim otrzymamy działającą aplikację SDI w kodzie programu należy wykonać następujące czynności:

- przygotować pętlę komunikatów,
- przygotować procedurę obsługi komunikatów,
- zarejestrować klasę okna,
- utworzyć okno,
- wyświetlić okno.

Pętla komunikatów

System Windows przekazuje aplikacji użytkownika informacje o wszelkich zdarzeniach (np. związanych z klawiaturą i myszą) za pomocą komunikatów. Komunikaty trafiają do kolejki FIFO danej aplikacji. Aby nasz program uczynić w pełni funkcjonalnym musi on odczytywać z kolejki napływające komunikaty i następnie obsługiwać je w sposób zależny od rodzaju zdarzenia. Obsługa komunikatów jest realizowana poprzez specjalnie przygotowaną procedurę, zwaną procedurą obsługi komunikatów. Aby komunikaty napływały do procedury obsługi muszą być odczytywane z kolejki (funkcja **GetMessage**) oraz dalej przesyłane do procedury obsługi za pomocą funkcji **DispatchMessage**.

```c
#include <windows.h>

int __stdcall WinMain (HINSTANCE hThisInstance,
                       HINSTANCE hPrevInstance,
                       LPSTR lpszArgument,
                       int nCmdShow)
{
    MSG msg;
    while(GetMessage(&msg, 0, 0, 0))
    {
        DispatchMessage(&msg);
    }
    return 0;
}
```

Pętla komunikatów w C

```
GetMessage      equ _GetMessageA@16
DispatchMessage equ _DispatchMessageA@4

MSG     STRUC
        hwnd        dd  ?
        message     dd  ?
        wParam      dd  ?
        lParam      dd  ?
        time        dd  ?
        pt      dd  ?
MSG     ENDS

.data
        (....)
Msg             MSG <>
        (....)
.code
        (....)
extrn   GetMessage          : near
@strt0: push    large 0
        push    large 0
        push    large 0
        lea     eax, Msg
        push    eax
        call    GetMessage
        cmp     eax,0
        je      @strt1
extrn   DispatchMessage     : near
        lea     eax, Msg
        push    eax
        call    DispatchMessage
        jmp     @strt0
@strt1: nop
        (....)
end
```

Pętla komunikatów w asemblerze

Programowanie w Windows API

Procedura obsługi komunikatów

Procedura obsługi komunikatów przyjmuje cztery parametry: uchwyt do okna (hWnd), id komunikatu (uMsg) oraz dodatkowe parametry (wParam, lParam), których znaczenie zależy od rodzaju komunikatu. Te komunikaty, które nie wymagają specyficznej obsługi należy przesłać do domyślnej procedury obsługi **DefWindowProc**.

```
DWORD CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam,LPARAM lParam)
{
    HDC         hDC;
    PAINTSTRUCT ps;
    switch(uMsg)
    {
        case WM_CLOSE:

            DestroyWindow(hWnd);

        break;

        case WM_DESTROY:

            PostQuitMessage(0);

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

Procedura obsługi komunikatów w C (podstawowa obsługa)

```
PostQuitMessage     equ _PostQuitMessage@4
DefWindowProc       equ _DefWindowProcA@16
DestroyWindow       equ _DestroyWindow@4
WM_CLOSE            = 0010h
WM_DESTROY          = 0002h
.code
        (...)
WndProc proc    near
        push    ebp
        push    esp
        pop     ebp
        mov     eax,[ebp + 12]
        cmp     eax, WM_CLOSE
        je      @wndp0
        cmp     eax, WM_DESTROY
        je      @wndp1
        jmp     @wndp2
extrn   DestroyWindow       : near
@wndp0: mov     eax, [ebp + 8]
        push    eax
        call    DestroyWindow
        jmp     @wndp2
extrn   PostQuitMessage     : near
@wndp1: push    large 0
        call    PostQuitMessage
extrn   DefWindowProc       : near
@wndp2: mov eax,[ebp + 20]
        push    eax
        mov eax,[ebp + 16]
        push    eax
        mov eax,[ebp + 12]
        push    eax
        mov eax,[ebp + 8]
        push    eax
        call    DefWindowProc
        pop ebp
        ret 16
WndProc endp
        (...)
end
```

Procedura obsługi komunikatów w asemblerze (podstawowa obsługa)

Klasa okna

Klasa okna jest strukturą opisującą podstawowe cechy okna: kolor tła, ikony, postać kursora myszy itp., a także wskazuje procedurę obsługi komunikatów. Klasa okna musi zostać zarejestrowana (**RegisterClassEx**) jeszcze przed utworzeniem okna.

```
#include <windows.h>

HINSTANCE hInstance;

int __stdcall WinMain (HINSTANCE hThisInstance,
                       HINSTANCE hPrevInstance,
                       LPSTR lpszArgument,
                       int nCmdShow)
{
    WNDCLASSEX              wnc;
    hInstance               = hThisInstance;
    wnc.cbSize              = sizeof(WNDCLASSEX);
    wnc.cbClsExtra          = 0;
    wnc.cbWndExtra          = 0;
    wnc.hbrBackground       = (HBRUSH)COLOR_WINDOW + 1;
    wnc.hCursor             = LoadCursor(NULL, IDC_ARROW);
    wnc.hIcon               = LoadIcon(NULL, IDI_APPLICATION);
    wnc.hIconSm             = LoadIcon(NULL, IDI_APPLICATION);
    wnc.hInstance           = hInstance;
    wnc.lpfnWndProc         = (WNDPROC)WndProc;
    wnc.lpszClassName       = "MAINCLASS";
    wnc.lpszMenuName        = NULL;
    wnc.style               = 0;
    RegisterClassEx(&wnc);
    (...)
    return 0;
}
```

Rejestracja klasy okna w C

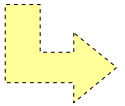
```
LoadIcon            equ _LoadIconA@8
LoadCursor          equ _LoadCursorA@8
RegisterClassEx     equ _RegisterClassExA@4
GetModuleHandle     equ _GetModuleHandleA@4
GetStockObject      equ _GetStockObject@4

CS_HREDRAW          = 0002h
CS_VREDRAW          = 0001h

WNDCLASSEX  STRUC
            cbSize      dd  ?
            style       dd  ?
            lpfnWndProc dd  ?
            cbClsExtra  dd  ?
            cbWndExtra  dd  ?
            hInstance   dd  ?
            hIcon       dd  ?
            hCursor     dd  ?
            hbrBackground   dd  ?
            lpszMenuName    dd  ?
            lpszClassName   dd  ?
            hIconSm     dd  ?
WNDCLASSEX  ENDS
```

Rejestracja klasy okna w asemblerze (cdn.)

Klasa okna (kontynuacja)

```
.data
        (...)
        hInstance           dd 0
        className           db 'MYCLASS', 0
        wndClassEx          WNDCLASSEX <SIZE WNDCLASSEX,CS_HREDRAW or CS_VREDRAW,,0,0,>
        (...)
.code
        (...)
extrn   GetModuleHandle     : near
        push    large 0
        call    GetModuleHandle
        mov     hInstance,eax
        (...)
extrn   LoadIcon            : near
        push    large IDI_MAINICON
        push    hInstance
        call    LoadIcon
extrn   LoadCursor          : near
        mov     wndClassEx.hIcon,eax
        mov     wndClassEx.hIconSm,eax
        push    large IDC_CROSS
        push    large 0
        call    LoadCursor
extrn   GetStockObject      : near
        mov     wndClassEx.hCursor,eax
        push    large DKGRAY_BRUSH
        call    GetStockObject
```

Rejestracja klasy okna w asemblerze (kont.)



```
extrn   RegisterClassEx     : near
        mov     wndClassEx.hbrBackground,eax
        mov     eax,hInstance
        mov     wndClassEx.hInstance,eax
        lea     eax,className
        mov     wndClassEx.lpszClassName,eax
        xor     eax,eax
        mov     wndClassEx.lpszMenuName,eax
        lea     eax,WndProc
        mov     wndClassEx.lpfnWndProc,eax
        lea     eax,wndClassEx
        push    eax
        call    RegisterClassEx
        cmp     eax,0
        jne     @init0
        (...)
end
```

Tworzenie i wyświetlanie okna

Kiedy klasa okna zostanie już zarejestrowana można utworzyć obiekt okna za pomocą funkcji **CreateWindowEx**. Takie okno nie jest jeszcze widoczne. Należy jeszcze je wyświetlić korzystając z funkcji **ShowWindow**.

```
#include <windows.h>

HWND        hMainWnd;
HINSTANCE   hInstance;

int __stdcall WinMain (HINSTANCE hThisInstance,HINSTANCE hPrevInstance,LPSTR lpszArgument,int nCmdShow)
{
    (...)
    hMainWnd = CreateWindowEx(NULL, "MAINCLASS", "SDI application", WS_OVERLAPPED | WS_SYSMENU,
                              CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL,
                              NULL, hInstance, NULL);
    ShowWindow(hMainWindow, SW_SHOWNORMAL);
    (...)
    return 0;
}
```

Tworzenie i pokazywanie okna w C

```
WS_OVERLAPPEDWINDOW = (00cfh shl 16)
CW_USEDEFAULT       = (8000h shl 16)
SW_SHOWNORMAL       = 0001h
.data
        (...)
        hMainWnd            dd 0
        appName             db 'SDI application',0
        className           db 'MYCLASS',0
        (...)
```

Tworzenie i pokazywanie okna w asemblerze (cdn.)

Tworzenie i wyświetlanie okna (kontynuacja)

```
.code
        (...)
extrn   CreateWindowEx      : near
        push    large 0
        push    hMyInstance
        push    hMyMenu
        push    large 0
        push    large CW_USEDEFAULT
        push    large CW_USEDEFAULT
        push    large CW_USEDEFAULT
        push    large CW_USEDEFAULT
        push    large WS_OVERLAPPEDWINDOW
        lea     eax,appName
        push    eax
        lea     eax,className
        push    eax
        push    large 0
        call    CreateWindowEx
extrn   ShowWindow          : near
        mov     hMainWnd,eax
        push    large SW_SHOWNORMAL
        push    eax
        call    ShowWindow
        (...)
end
```

Tworzenie i pokazywanie okna w asemblerze (kontynuacja)

Rysowanie w obszarze klienckim okna

System operacyjny Windows przesyła komunikat WM_PAINT w chwili, kiedy zachodzi potrzeba odświeżenia zawartości klienckiej części okna aplikacji – zmiana rozmiaru okna, przesunięcie innego okna nad oknem aplikacji, etc. Stąd kod odświeżający zawartość okna powinien być uruchamiany komunikatem WM_PAINT.

Rysowanie wymaga uchwytu do tzw. **kontekstu urządzenia** dla klienckiej części okna, który zwraca funkcja **BeginPaint**. Należy później pamiętać, aby kontekst ten zwolnić funkcją **EndPaint**.

Rysowanie samo w sobie wymaga **pióra** oraz **pędzla**, które tworzymy za pomocą funkcji **CreatePen** oraz **CreateSolidBrush**. Obiekty te usuwamy w chwili, kiedy nie będą już nam potrzebne (**DeleteObject**). (Ważne!)

Obiekty pióra i pędzla wybieramy do kontekstu funkcją **SelectObject.** Uchwyty do domyślnego pióra oraz pędzla należy zapamiętać i później przywrócić. (Ważne !)

```
#include <windows.h>
(...)
DWORD CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC         hDC;
    HPEN        hPen, hOldPen;
    HBRUSH      hBrush, hOldBrush;
    PAINTSTRUCT ps;
    switch(uMsg)
    {
        (...)
        case WM_PAINT:
            hDC         = BeginPaint(hWnd, &ps);
            hPen        = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
            hBrush      = CreateSolidBrush(RGB(0, 255, 0));
            hOldPen     = (HPEN)SelectObject(hDC, hPen);
            hOldBrush   = (HBRUSH)SelectObject(hDC, hBrush);
            Ellipse(hDC, 0, 0, 300, 200);
            SelectObject(hDC, hOldPen);
            SelectObject(hDC, hOldBrush);
            DeleteObject(hPen);
            DeleteObject(hBrush);
            EndPaint(hWnd, &ps);
        break;
        (...)
    }
    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
(...)
```