

## **Découverte de JWAsm**

Je vais tenter d'écrire un programme 64 bits sous Windows 7 et utilisant JWAsm. Chaque amélioration de la version précédente entraînera la création d'une nouvelle étape. La première étape est l'étape 00, c'est-à-dire en partant du source original trouvé sur le site de japeth.

## Etape 00

```
;--- a simple Windows GUI program which creates an  
;--- "empty" window, then enters a "message loop".  
;--- Does not contain directives for Win64 SEH.  
;---  
;--- to create the binary enter:  
;--- JWasm -win64 Win64_2.asm  
;--- Link Win64_2.obj kernel32.lib user32.lib
```

```
option casemap:none
```

```
HINSTANCE typedef QWORD  
HWND typedef QWORD  
HMENU typedef QWORD  
HICON typedef QWORD  
HBRUSH typedef QWORD  
HCURSOR typedef QWORD  
WPARAM typedef QWORD  
LPARAM typedef QWORD  
LPSTR typedef QWORD  
LPVOID typedef QWORD  
UINT typedef DWORD
```

```
NULL equ 0  
WS_OVERLAPPEDWINDOW equ 0CF0000h  
CW_USEDEFAULT equ 80000000h  
SW_SHOWDEFAULT equ 10  
SW_SHOWNORMAL equ 1  
IDC_ARROW equ 32512  
IDI_APPLICATION equ 32512  
WM_DESTROY equ 2  
CS_VREDRAW equ 1  
CS_HREDRAW equ 2  
COLOR_WINDOW equ 5
```

```
proto_WNDPROC typedef proto :HWND, :QWORD, :WPARAM, :LPARAM  
WNDPROC typedef ptr proto_WNDPROC
```

```
WNDCLASSEX struct 8  
cbSize DWORD ?  
style DWORD ?  
lpfnWndProc WNDPROC ?  
cbClsExtra DWORD ?  
cbWndExtra DWORD ?  
hInstance HINSTANCE ?  
hIcon HICON ?  
hCursor HCURSOR ?  
hbrBackground HBRUSH ?  
lpszMenuName LPSTR ?  
lpszClassName LPSTR ?  
hIconSm HICON ?  
WNDCLASSEX ends
```

```
POINT struct  
x SDWORD ?  
y SDWORD ?  
POINT ends
```

```
MSG struct 8  
hwnd HWND ?  
message DWORD ?  
wParam WPARAM ?  
lParam LPARAM ?  
time DWORD ?  
pt POINT <>  
MSG ends
```

```

GetModuleHandleA proto :LPSTR
GetCommandLineA proto
ExitProcess proto :UINT
LoadIconA proto :HINSTANCE, :LPSTR
LoadCursorA proto :HINSTANCE, :LPSTR
RegisterClassExA proto :ptr WNDCLASSEXA
CreateWindowExA proto :DWORD, :LPSTR, :LPSTR, :DWORD, :SDWORD, :SDWORD, :SDWORD, :SDWORD, :HWND, :HMEN
ShowWindow proto :HWND, :SDWORD
UpdateWindow proto :HWND
GetMessageA proto :ptr MSG, :HWND, :SDWORD, :SDWORD
TranslateMessage proto :ptr MSG
DispatchMessageA proto :ptr MSG
PostQuitMessage proto :SDWORD
DefWindowProcA proto :HWND, :UINT, :WPARAM, :LPARAM

```

```
WinMain proto :HINSTANCE, :HINSTANCE, :LPSTR, :UINT
```

**.data**

```

ClassName db "SimpleWinClass",0
AppName db "Our First Window",0

```

**.data?**

```

hInstance HINSTANCE ?
CommandLine LPSTR ?

```

**.code**

```
start:
```

```

and    rsp, -16
invoke GetModuleHandleA, NULL
mov    hInstance, rax
invoke GetCommandLineA
mov    CommandLine, rax
invoke WinMain, hInstance, NULL, CommandLine, SW_SHOWDEFAULT
invoke ExitProcess, eax

```

```
WinMain proc hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:UINT
```

```

local wc:WNDCLASSEXA
local msg:MSG
local hwnd:HWND

```

```
and    rsp, -16 ;to make the stack 16byte aligned
```

```
mov    hInst, rcx ;hInst is the name of the shadow space variable!
```

```

mov    wc.cbSize, SIZEOF WNDCLASSEXA
mov    wc.style, CS_HREDRAW or CS_VREDRAW
; mov    rax, OFFSET WndProc ;using LEA is preferable
lea    rax, [WndProc]
mov    wc.lpfWndProc, rax
mov    wc.cbClsExtra, NULL
mov    wc.cbWndExtra, NULL
mov    wc.hInstance, rcx
mov    wc.hbrBackground, COLOR_WINDOW+1
mov    wc.lpszMenuName, NULL
; mov    rax, OFFSET ClassName ;using LEA is preferable
lea    rax, [ClassName]
mov    wc.lpszClassName, rax
invoke LoadIconA, NULL, IDI_APPLICATION
mov    wc.hIcon, rax
mov    wc.hIconSm, rax
invoke LoadCursorA, NULL, IDC_ARROW
mov    wc.hCursor, rax
invoke RegisterClassExA, addr wc
invoke CreateWindowExA, NULL, ADDR ClassName, ADDR AppName, \

```

```

        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,\
        CW_USEDEFAULT, CW_USEDEFAULT,CW_USEDEFAULT, NULL, NULL,\
        hInst, NULL
mov     hwnd, rax
invoke ShowWindow, hwnd, SW_SHOWNORMAL
invoke UpdateWindow, hwnd
.while (1)
    invoke GetMessageA, ADDR msg, NULL, 0, 0
    .break .if (!rax)
    invoke TranslateMessage, ADDR msg
    invoke DispatchMessageA, ADDR msg
.endw
mov     rax, msg.wParam
ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM

and     rsp, -16
.if (rcx == WM_DESTROY)
    invoke PostQuitMessage, NULL
.else
    invoke DefWindowProcA, rcx, edx, r8, r9
    ret
.endif
xor     rax, rax
ret
WndProc endp

end start

```

Avant de le compiler, vous devez définir les options de compilation et d'édition des liens. Voici mes options sous RadAsm :

```
4,O,C:\JWasm64\PORC.EXE /v,1
```

```
3,O,C:\JWasm64\JWasm.EXE -Fl -q -Sa -Sg -Sn -W4 -c -win64 -Zp8  
-I\WinInc\Include C:\MyProjects\JWasm\JWasm.Asm
```

```
5,O,C:\JWasm64\jwlink format windows pe runtime windows res JWasm.res file  
JWasm.obj
```

Voici le contenu du fichier MANIFEST pour Windows 7.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">  
<assemblyIdentity  
  version="1.0.0.0"  
  processorArchitecture="*"  
  name="IDCat.IView64.SimpleViewer"  
  type="win32"  
</>  
<description>Visionneuse d'images en 64 bits</description>  
<dependency>  
  <dependentAssembly>  
    <assemblyIdentity  
      type="win32"  
      name="Microsoft.Windows.Common-Controls"  
      version="6.0.0.0"  
      processorArchitecture="*"  
      publicKeyToken="6595b64144ccf1df"  
      language="*"  
    </>  
  </dependentAssembly>  
</dependency>  
</assembly>
```

Si vous compilez ce source, il fonctionne mais comporte un bug au niveau de la deuxième ligne de la **WndProc**. La valeur du message **WM\_DESTROY** est comparée au registre **RCX** alors que le registre dédié au message est le registre **EDX**.

Il est à noter que dans chaque fonction, on commence par aligner la pile afin d'être certain qu'elle se trouve à une adresse multiple de 16.

```
and    rsp,-16
```

# ETAPE 01

Dans cette étape, j'ai reformaté le code source, inclus de nouveaux prototypes et corrigé le bug cité plus haut.

J'ai également ajouté le code permettant de connaître le dossier depuis lequel le programme a été lancé.

```
includelib Lib\kernel32.lib
includelib Lib\user32.lib
includelib Lib\gdi32.lib
includelib Lib\comctl32.lib
includelib Lib\ws2_32.lib
includelib Lib\shell32.lib
includelib Lib\SHLWAPI.lib

; -----
-----

option casemap:none

; -----
-----

HINSTANCE typedef QWORD
HWND      typedef QWORD
HOBJECT   typedef QWORD
HMENU     typedef QWORD
HICON     typedef QWORD
HBRUSH    typedef QWORD
HCURSOR   typedef QWORD
WPARAM    typedef QWORD
LPARAM    typedef QWORD
LPSTR     typedef QWORD
LPVOID    typedef QWORD
UINT      typedef DWORD

; -----
-----

NULL          EQU 0
MAX_PATH      EQU 260

; -----
-----

WS_OVERLAPPEDWINDOW EQU 0CF0000h
CW_USEDEFAULT        EQU 80000000h
SW_SHOWDEFAULT      EQU 10
SW_SHOWNORMAL       EQU 1
IDC_ARROW           EQU 32512
IDI_APPLICATION     EQU 32512
WM_DESTROY          EQU 2
CS_VREDRAW          EQU 1
CS_HREDRAW          EQU 2
COLOR_WINDOW        EQU 5

proto_WNDPROC typedef PROTO :HWND, :QWORD, :WPARAM, :LPARAM
WNDPROC typedef ptr proto_WNDPROC
```

```

; -----
-----
WNDCLASSEXA          STRUCT      8
    cbSize            DWORD       ?
    style             DWORD       ?
    lpfnWndProc       WNDPROC     ?
    cbClsExtra        DWORD       ?
    cbWndExtra        DWORD       ?
    hInstance         HINSTANCE   ?
    hIcon             HICON       ?
    hCursor           HCURSOR     ?
    hbrBackground     HBRUSH      ?
    lpszMenuName      LPSTR       ?
    lpszClassName     LPSTR       ?
    hIconSm           HICON       ?
WNDCLASSEXA          ENDS

POINT                STRUCT
    x                 SDWORD      ?
    y                 SDWORD      ?
POINT                ENDS

MSG                  STRUCT      8
    hwnd             HWND        ?
    message          DWORD        ?
    wParam           WPARAM      ?
    lParam           LPARAM      ?
    time            DWORD        ?
    pt              POINT        <>
MSG                  ENDS

; -----
-----

GetModuleHandleA    PROTO   :LPSTR
GetCommandLineA     PROTO
ExitProcess         PROTO   :UINT
LoadIconA           PROTO   :HINSTANCE, :LPSTR
LoadCursorA         PROTO   :HINSTANCE, :LPSTR
RegisterClassExA    PROTO   :Ptr WNDCLASSEXA
CreateWindowExA     PROTO
    :DWORD, :LPSTR, :LPSTR, :DWORD, :SDWORD, :SDWORD, :SDWORD, :SDWORD, :HWND, :HMENU, :HINSTANC
E, :LPVOID
ShowWindow          PROTO   :HWND, :SDWORD
UpdateWindow        PROTO   :HWND
GetMessageA         PROTO   :Ptr MSG, :HWND, :SDWORD, :SDWORD
TranslateMessage    PROTO   :Ptr MSG
DispatchMessageA    PROTO   :Ptr MSG
PostQuitMessage     PROTO   :SDWORD
DefWindowProcA      PROTO   :HWND, :UINT, :WPARAM, :LPARAM
SendMessageA        PROTO   :HWND, :DWORD, :WPARAM, :LPARAM
CreateFontA         PROTO
    :DWord, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD, :DWOR
D, :DWORD, :LPSTR
DeleteObject        PROTO   :HWND
DestroyWindow       PROTO   :HWND
GetModuleFileNameA  PROTO   :HINSTANCE, :LPSTR, :DWord
lstrcpyA           PROTO   :LPSTR, :LPSTR
PathFindFileNameA   PROTO   :LPSTR
MessageBoxA         PROTO   :HWND, :LPSTR, :LPSTR, :QWord
PathFileExistsA     PROTO   :LPSTR
MoveWindow          PROTO   :HWND, :DWORD, :DWORD, :DWORD, :DWORD, :DWORD

WinMain             PROTO   :HINSTANCE, :HINSTANCE, :LPSTR, :UINT

; -----
-----
.Data
; -----

```



```

mov     wc.lpszClassName, rax
INVOKE LoadIconA, NULL, IDI_APPLICATION
mov     wc.hIcon, rax
mov     wc.hIconSm, rax
INVOKE LoadCursorA, NULL, IDC_ARROW
mov     wc.hCursor, rax
INVOKE RegisterClassExA, ADDR wc
INVOKE CreateWindowExA, NULL, ADDR ClassName, ADDR AppName, \
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, \
NULL, \
        hInst, NULL
mov     hwnd, rax

INVOKE ShowWindow, hwnd, SW_SHOWNORMAL
INVOKE UpdateWindow, hwnd

.WHILE (1)
    INVOKE GetMessageA, ADDR msg, NULL, 0, 0
    .BREAK .IF (!rax)
    INVOKE TranslateMessage, ADDR msg
    INVOKE DispatchMessageA, ADDR msg
.ENDW

mov     rax, msg.wParam
ret

WinMain     ENDP

; -----
-----

WndProc     PROC     hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM

and     rsp, -16

.if     (edx == WM_DESTROY)
    INVOKE PostQuitMessage, NULL
.else
    INVOKE DefWindowProcA, rcx, edx, r8, r9
    ret
.endif

xor     rax, rax
ret

WndProc     ENDP

; -----
-----

END     Start

```

## ETAPE 02

Ici, je vais normaliser les noms des variables et remplacer les « .IF » et toutes les instructions de ce type par de vraies instructions assembleur.

Dans cette partie, on va se consacrer à l'étude du passage des paramètres pour appeler une fonction. Je commence par le plus simple, une fonction sans paramètre.

```
0000006B          INVOKE  GetCommandLineA
0000006B  4883EC20      *   sub    rsp, 32
0000006F  E800000000    *   call   GetCommandLineA
00000074  4883C420      *   add    rsp, 32
```

On peut déjà remarquer qu'avant d'appeler la fonction, la pile est **décrémentée** de **32** octets, soit 4 quadruples mots. On remarquera, dans le source cette fois que chaque fonction est définie par un **FRAME**. Un **FRAME** peut être considéré comme une structure. Ici, donc le cadre de pile est prévu, quelque soit le type de fonction pour 4 paramètres. Il faudra, une fois que la fonction est terminée, **recorriger** la pile, ce que fait **ADD RSP,32**.

Cette façon de faire change beaucoup de choses par rapport à la programmation classique en 32 bits ou l'on n'utilisait pas cet artifice.

Pour la programmation en 64 bits, je vous conseille ce site <http://x86asm.net/articles/x86-64-tour-of-intel-manuals/>

Autre cas, cette fois avec une fonction prenant 3 paramètres. Le dernier paramètre est sur 32 bits.

```
0000001F          INVOKE  GetModuleFileNameA,rax,ADDR  szModuleFileName,SIZEOF  szModule-
FileName
0000001F  4883EC20      *   sub    rsp, 32
00000023  488BC8        *   mov    rcx, rax
00000026  488D1500000000 *   lea   rdx, szModuleFileName
0000002D  41B804010000  *   mov    r8d,  SIZEOF  szModuleFileName
00000033  E800000000    *   call   GetModuleFileNameA
00000038  4883C420      *   add    rsp, 32
```

Comme on peut le voir dans le code ci-dessus, aucun paramètre n'est passé par la pile. On se content d'initialiser les registres **RCX** pour le premier paramètre, **RDX** pour le second paramètre, **R8** pour le troisième paramètre et **R9** s'il y avait un quatrième paramètre.

Le cadre de pile permet de traiter les paramètres comme des variables locales. L'initialisation du troisième paramètre se fait par l'intermédiaire du registre **R8D**, qui est la partie **32 bits** du registre **R8**.

De D8 à R15, on peut accéder au double mot (**R8D**) ou au mot (**R8W**), à l'octet (**R8L**).

Si avant d'appeler une fonction vous initialisez **RCX**, **RDX**, **R8** ou encore **R9**, faites attention. Si **INVOKE** est utilisée, vous risquez que la valeur d'un de ces registres soit écrasée. Vous risquez aussi d'avoir un message d'erreur de la part de JWasm vous indiquant un écrasement du contenu du registre (register override).

En programmation 32 bits je n'aimais déjà pas beaucoup **INVOKE** à cause de l'override avec **EAX**, en 64 bits, je pense le bannir une fois que le mécanisme d'appel des fonctions sera bien compris.

Sur le même type, on trouve l'appel de la fonction WinMain.

```
0000007F                                INVOKE
    WinMain,hInstance,NULL,lpszCommandLine,SW_SHOWDEFAULT
0000007F 4883EC20 *    sub rsp, 32
00000083 488B0D00000000 *    mov rcx, hInstance
0000008A 48C7C200000000 *    mov rdx, NULL
00000091 4C8B0500000000 *    mov r8, lpszCommandLine
00000098 41B90A00000000 *    mov r9d, SW_SHOWDEFAULT
0000009E E813000000 *    call WinMain
000000A3 4883C420 *    add rsp, 32
```

Comment se fait l'appel d'une fonction comportant plus de 4 paramètres ?

Un exemple au hasard avec la fonction CreateWindowEx.

```
00000165                                INVOKE
    CreateWindowExA,NULL,ADDR szClassName,ADDR szAppName,
WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
NULL,NULL, __hInst,NULL
00000165 4883EC60 *    sub rsp, 96
00000169 B900000000 *    mov ecx, NULL
0000016E 488D1500000000 *    lea rdx, szClassName
00000175 4C8D0500000000 *    lea r8, szAppName
0000017C 41B90000CF00 *    mov r9d, WS_OVERLAPPEDWINDOW
00000182 C744242000000080 *    mov dword ptr [rsp+32], CW_USEDEFAULT
0000018A C744242800000080 *    mov dword ptr [rsp+40], CW_USEDEFAULT
00000192 C744243000000080 *    mov dword ptr [rsp+48], CW_USEDEFAULT
0000019A C744243800000080 *    mov dword ptr [rsp+56], CW_USEDEFAULT
000001A2 48C744244000000000 *    mov qword ptr [rsp+64], NULL ; RCX
000001AB 48C744244800000000 *    mov qword ptr [rsp+72], NULL ; RCX
000001B4 488B4510 *    mov rax, __hInst ; A déplacer
000001B8 4889442450 *    mov [rsp+80], rax
000001BD 48C744245800000000 *    mov qword ptr [rsp+88], NULL ; RCX
000001C6 E800000000 *    call CreateWindowExA
000001CB 4883C460 *    add rsp, 96
```

La fonction **CreateWindowEx** prend **12 paramètres**, il faut commencer par soustraire 12 x 8, soit 96 octets. Les quatre premiers paramètres sont passés via **RCX, RDX, R8** et **R9**, les suivants utiliseront le cadre de pile. On commencera l'initialisation de ce cadre de pile à partir du cinquième paramètre, ce qui correspond à la position **RSP + 32**.

Si l'on avait fait un passage manuel des paramètres, j'aurais remplacé le **MOV ECX,NULL** par **XOR ECX,ECX**. J'en aurais profité pour initialisé le dernier paramètre avec **ECX**. J'aurais également fait **MOV RAX,\_\_hInst** juste après l'initialisation du registre **R9D**.

J'ai mis en **ROUGE**, le code qui peut à mon avis être amélioré.

A noter enfin que JWasm prèere utiliser l'instruction **LEA Reg,Adresse** plutôt que **MOV Reg,OFFSET Adresse**. Pourquoi pas.

Enfin ,la variable \_hWnd me semble inutile.

En 32 bits, je procédais comme suit :

```
PUSH    EAX
INVOKE  ShowWindow,EAX,SW_SHOWNORMAL
CALL    UpdateWindow
```

En 64 bits cela n'est plus possible, cela se fait comme indiqué ci-dessous :

```
000001CF 48898578FFFFFF      mov     _hWnd,rax

000001D6                                INVOKE  ShowWindow,_hWnd,SW_SHOWNORMAL
000001D6 4883EC20            *      sub  rsp, 32
000001DA 488B8D78FFFFFF      *      mov  rcx, _hWnd
000001E1 BA01000000          *      mov  edx, SW_SHOWNORMAL
000001E6 E800000000          *      call ShowWindow
000001EB 4883C420            *      add  rsp, 32
000001EF                                INVOKE  UpdateWindow,_hWnd
000001EF 4883EC20            *      sub  rsp, 32
000001F3 488B8D78FFFFFF      *      mov  rcx, _hWnd
000001FA E800000000          *      call UpdateWindow
000001FF 4883C420            *      add  rsp, 32
```

Par contre, cela ne me plaît toujours pas. Pourquoi ne pas soustraire directement 64 octets de la pile et faire un **PUSH** du registre **ECX** ?

En fait, je suis d'accord pour la soustraction des 64 octets, mais faire un MOV suivi d'un PUSH puis un peu plus loin d'un **POP**, cela fait perdre plus de temps qu'il n'en fait gagner. A mon avis, les deux instructions **ADD RSP, 32** et **SUB RSP,32** sont inutiles. Avant d'appeler la fonction il faut faire **SUB RSP,64** et juste après l'appel de **UpdateWindow**, faire **ADD RSP,64**.

Il en va de même dans la boucle de traitement des messages dont voici le code :

```
00000203                                     .WHILE      (1)
00000203      *      @C0002:
00000203      INVOKE  GetMessageA,ADDR  _Msg,NULL,0,0
00000203  4883EC20      *      sub  rsp, 32
00000207  488D4D80      *      lea  rcx, _Msg
0000020B  48C7C200000000      *      mov  rdx, NULL
00000212  41B800000000      *      mov  r8d, 0
00000218  41B900000000      *      mov  r9d, 0
0000021E  E800000000      *      call GetMessageA
00000223  4883C420      *      add  rsp, 32
00000227                                     .BREAK     .IF (!rax)
00000227  4823C0      *      and  rax, rax
0000022A  7424      *      jz   @C0003
0000022C      INVOKE  TranslateMessage,ADDR  _Msg
0000022C  4883EC20      *      sub  rsp, 32
00000230  488D4D80      *      lea  rcx, _Msg
00000234  E800000000      *      call TranslateMessage
00000239  4883C420      *      add  rsp, 32
0000023D      INVOKE  DispatchMessageA,ADDR  _Msg
0000023D  4883EC20      *      sub  rsp, 32
00000241  488D4D80      *      lea  rcx, _Msg
00000245  E800000000      *      call DispatchMessageA
0000024A  4883C420      *      add  rsp, 32
0000024E                                     .ENDW
0000024E  EBB3      *      jmp  @C0002
00000250      *      @C0003:
```

Autre cas :

```
00000277      INVOKE  DefWindowProcA,rcx,edx,r8,r9
00000277  4883EC20      *      sub  rsp, 32
0000027B  E800000000      *      call DefWindowProcA
00000280  4883C420      *      add  rsp, 32
00000284                                     ret
00000284  C9      *      leave
00000285  C3      *      retn
```

Je voudrais revenir sur la modification du pointeur de pile.

```
SUB   RSP,32
CALL  Fonction_1
ADD   RSP,32
SUB   RSP,32
CALL  Fontion_2
ADD   RSP,32
```

Pourrait être remplacé par :

```
SUB   RSP,64
CALL  Fonction_1
ADD   RSP,32
CALL  Fonction_2
ADD   RSP,32
```

Mais pourquoi ne pas faire plus simple :

```
SUB   RSP,32
CALL  Fonction_1
CALL  Fonction_2
ADD   RSP,32
```

Je vais donc mettre en application toutes ces modifications dans cette étape.

# Modification de l'appel de la fonction CreateWindowEx

Code Avant :

```
00000165 4883EC60 * sub rsp, 96
00000169 B900000000 * mov ecx, NULL
0000016E 488D1500000000 * lea rdx, szClassName
00000175 4C8D0500000000 * lea r8, szAppName
0000017C 41B90000CF00 * mov r9d, WS_OVERLAPPEDWINDOW
00000182 C744242000000080 * mov dword ptr [rsp+32], CW_USEDEFAULT
0000018A C744242800000080 * mov dword ptr [rsp+40], CW_USEDEFAULT
00000192 C744243000000080 * mov dword ptr [rsp+48], CW_USEDEFAULT
0000019A C744243800000080 * mov dword ptr [rsp+56], CW_USEDEFAULT
000001A2 48C744244000000000 * mov qword ptr [rsp+64], NULL
000001AB 48C744244800000000 * mov qword ptr [rsp+72], NULL
000001B4 488B4510 * mov rax, __hInst
000001B8 4889442450 * mov [rsp+80], rax
000001BD 48C744245800000000 * mov qword ptr [rsp+88], NULL
000001C6 E800000000 * call CreateWindowExA
000001CB 4883C460 * add rsp, 96
```

Taille du code : **105 octets**

Code Après :

```
00000165 4883EC60 sub rsp, 96
00000169 4833C9 xor rcx, rcx
0000016C 488D1500000000 lea rdx, szClassName
00000173 4C8D0500000000 lea r8, szAppName
0000017A 488B4510 mov rax, __hInst
0000017E 41B90000CF00 mov r9d, WS_OVERLAPPEDWINDOW
00000184 C744242000000080 mov dword ptr [rsp+32], CW_USEDEFAULT
0000018C C744242800000080 mov dword ptr [rsp+40], CW_USEDEFAULT
00000194 C744243000000080 mov dword ptr [rsp+48], CW_USEDEFAULT
0000019C C744243800000080 mov dword ptr [rsp+56], CW_USEDEFAULT
000001A4 48894C2440 mov qword ptr [rsp+64], rcx
000001A9 48894C2448 mov qword ptr [rsp+72], rcx
000001AE 4889442450 mov [rsp+80], rax
000001B3 48894C2458 mov qword ptr [rsp+88], rcx
000001B8 E800000000 call CreateWindowExA
000001BD 4883C460 add rsp, 96
```

Taille du code : **91 octets**

Autre Variante :

```
00000165 4883EC60 sub rsp, 96
00000169 B800000080 mov eax, CW_USEDEFAULT
0000016E 4833C9 xor rcx, rcx
00000171 89442420 mov dword ptr [rsp+32], eax
00000175 89442428 mov dword ptr [rsp+40], eax
00000179 89442430 mov dword ptr [rsp+48], eax
0000017D 89442438 mov dword ptr [rsp+56], eax
00000181 488D1500000000 lea rdx, szClassName
00000188 4C8D0500000000 lea r8, szAppName
0000018F 488B4510 mov rax, __hInst
00000193 41B90000CF00 mov r9d, WS_OVERLAPPEDWINDOW
00000199 48894C2440 mov qword ptr [rsp+64], rcx
0000019E 48894C2448 mov qword ptr [rsp+72], rcx
000001A3 4889442450 mov [rsp+80], rax
000001A8 48894C2458 mov qword ptr [rsp+88], rcx
000001AD E800000000 call CreateWindowExA
000001B2 4883C460 add rsp, 96
```

Taille du code : **50 octets**

Soit un gain de plus de 50% par rapport à la version proposée par JWAsm !

## Etape N°3

Dans cette étape, j'ai ajouté le traitement des messages :

- WM\_CREATE
- WM\_SIZE
- WM\_NOTIFY

Je souhaite que le programme ait deux boîtes à onglet en haut et que chacune de ces deux boîtes à onglets gère 3 fenêtres.

Boîte à onglets de gauche

- Fenêtre contenant la liste des dossiers
- Fenêtre de pré-visualisation d'une image
- Fenêtre d'information sur une image

Boîte à onglet de droite

- Fenêtre des miniatures
- Fenêtre de zoom
- Fenêtre des effets spéciaux

Actuellement, le résultat est bon même si l'affichage n'est pas génial.

Cette nouvelle méthode de passage des paramètres est très intéressante et permet d'économiser pas mal d'octets. La fonction `CreateWindowEx` possède 12 paramètres. Assez souvent, lors du traitement du message `WM_CREATE` de la fenêtre principale, l'on crée les fenêtres enfants.

Si comme moi vous avez 8 fenêtres filles, cela fait un grand nombre d'initialisation à faire dont une grande partie est complètement inutile. Par inutile, je veux dire qu'une fois faite pour une fenêtre, c'est pareil pour les autres.

```
sub    rsp,96
xor    rcx,rcx
lea    rdx,szWC_TABCONTROL
lea    r8,szNullString
mov    rax,CW_USEDEFAULT
mov    r9d,WS_CHILD + WS_VISIBLE + TCS_FORCEICONLEFT + TCS_TOOLTIPS
mov    dword ptr [rsp+32],eax
mov    dword ptr [rsp+40],eax
mov    dword ptr [rsp+48],eax
mov    dword ptr [rsp+56],eax
mov    rax, __hWnd
mov    qword ptr [rsp+72],IDC_TAB_01
mov    [rsp+64],rax
mov    qword ptr [rsp+88],rcx
mov    rax,hInstance
mov    [rsp+80],rax
call   CreateWindowExA

mov    hTabLeft,rax
```

Ceci est la façon typique de créer une fenêtre. Dans mon cas, que va-t-il changer pour la fenêtre suivante qui est aussi un contrôle TAB ? Juste l'identifiant du contrôle qui va devenir **IDC\_TAB\_02** au lieu de **IDC\_TAB\_01**. Donc pourquoi répéter tout ce code ?

Voici donc le code de création du deuxième contrôle TAB.

```
xor    rcx,rcx
lea    rdx,szWC_TABCONTROL
lea    r8,szNullString
mov    r9d,WS_CHILD + WS_VISIBLE + TCS_FORCEICONLEFT + TCS_TOOLTIPS
mov    qword ptr [rsp+72],IDC_TAB_02
call   CreateWindowExA

mov    hTabRight,rax
```

Six lignes pour créer une fenêtre, qui dit mieux ? Soit une réduction du code de 50% fatalement. Maintenant, la prochaine fenêtre à créer n'a pas besoin d'identifiant, on va donc remplacer **IDC\_TAB\_02** par **zéro** ou **NULL**. Pour cela, je vais utiliser le registre **RCX** qui est déjà à 0.

```
xor    rcx,rcx
lea    rdx,szClass_Folders
lea    r8,szNullString
mov    r9d,WS_CHILD + WS_VISIBLE
mov    qword ptr [rsp+72],rcx
call   CreateWindowExA

mov    hFolders,rax
```

Il me reste à créer encore 5 fenêtres. On peut maintenant gagner encore une ligne de code et créer une fenêtre en 5 lignes !

```
xor    rcx,rcx
lea    rdx,szClass_Preview
lea    r8,szNullString
mov    r9d,WS_CHILD + WS_VISIBLE
call   CreateWindowExA
```

Dans la mesure où l'on utilise toujours la même zone de la pile pour les paramètres de cette fonction, on peut alors se servir du fait qu'une partie soit déjà initialisée.

Auriez-vous, un jour, imaginé créer une fenêtre avec seulement 5 lignes de code ?

Moi pas.

## Etape N°4

Maintenant il est possible de cliquer sur un onglet et d'activer la fenêtre correspondante. Pour le vérifier, j'affiche le nom de la classe de la fenêtre.

Le processus est simple. Lorsque j'ai créé les onglets j'ai stocké dans le **IParam** le handle de la fenêtre correspondante. Dans le **WM\_NOTIFY**, je demande quel est l'onglet sélectionné et ensuite je récupère le **LPARAM** au moyen de **TCM\_GETITEM**.

Si je traite le message **TCN\_SELCHANGING**, je cache la fenêtre **ShowWindow,\_Tci.IParam,SW\_HIDE**. Si je traite le message **TCN\_SELCHANGE**, je montre la fenêtre **ShowWindow,\_Tci.IParam,SW\_SHOW**.

Dans le futur du programme, d'autres messages seront certainement traités, mais mettre des onglets en place est tellement facile que je veux en faire profiter tout le monde.

Le traitement du message **WM\_NOTIFY** est le principal ajout de cette étape. Je traite aussi le message **WM\_PAINT**, mais il n'a rien de spectaculaire ni de bien utile. En plus, je ne me suis pas cassé la tête pour l'affichage du texte, pas de double buffer, du débutant.

Voici la structure **PAINTSTRUCT** façon 64 bits :

```
PAINTSTRUCT      STRUCT      8
    hdc           HDC         ?
    fErase        DWORD       ?
    rcPaint       RECT        <>
    fRestore      DWORD       ?
    fIncUpdate    DWORD       ?
    rgbReserved   BYTE        32 dup (?)
                 DWORD       ?
```

PAINTSTRUCT ENDS

```
LPPAINTSTRUCT    typedef PTR PAINTSTRUCT
```

Le message **MW\_NOTIFY** fait une utilisation intensive de la structure **NMHDR**, voici comment elle est définie :

```
NMHDR            STRUC
    hwndFrom      HWND        ?
    idFrom        QWORD       ?
    code          DWORD       ?
                 DWORD       ?
NMHDR            ENDS

LPMNHDR          Typedef    PTR NMHDR
```

J'ai complété la structure par un double mot bidon afin que la structure ait une taille multiple de 8. Ce sera à vérifier dans le futur quand on utilisera des structures incorporant **NMHDR**.

## Voici le code de traitement du message WM\_NOTIFY

```
.Code

Main_WmNotify PROC    __hWnd:HWND, __ControlId:WPARAM, __lpNmHdr:LPNMHDR
LOCAL    __hTab:HWND
LOCAL    _Tci:TCITEM

and     rsp,-16

mov     __hWnd,rcx
mov     eax,DWord PTR (NMHDR PTR [r8]).idFrom
mov     __lpNmHdr,r8

cmp     eax,IDC_TAB_01
je      @TabControls

cmp     eax,IDC_TAB_02
je      @TabControls

xor     rax,rax
ret

@TabControls :

mov     rax,hTabLeft
mov     rcx,hTabRight
cmp     (NMHDR PTR [r8]).idFrom,IDC_TAB_01
cmovne rax,rcx
mov     _hTab,rax

cmp     (NMHDR PTR [r8]).code,TCN_SELCHANGING
jne     @Tab_01_Msg_01

xor     r8,r8
xor     r9,r9

INVOKE SendMessageA,_hTab,TCM_GETCURSEL,r8,r9

cmp     rax,-1
je      @Exit

lea     r9,_Tci
mov     r8,rax
mov     DWord PTR (TCITEM PTR [r9]).mask_,TCIF_PARAM

INVOKE SendMessageA,_hTab,TCM_GETITEM,r8,r9

test    rax,rax
jz      @Exit

xor     edx,edx        ; SW_HIDE

INVOKE ShowWindow,_Tci.lParam,edx

xor     eax,eax
ret

@Tab_01_Msg_01 :

cmp     (NMHDR PTR [r8]).code,TCN_SELCHANGE
jne     @Exit

xor     r8,r8
xor     r9,r9

INVOKE SendMessageA,_hTab,TCM_GETCURSEL,r8,r9
```

```

        cmp     rax,-1
        je     @Exit

        lea   r9,_Tci
        mov   r8,rax

        mov   DWORD PTR (TCITEM PTR [r9]).mask_,TCIF_PARAM

        INVOKE SendMessageA,_hTab,TCM_GETITEM,r8,r9

        test  rax,rax
        jz    @Exit

        INVOKE ShowWindow,_Tci.lParam,SW_SHOW

        ret

@Exit :

        ret
Main_WmNotify   ENDP

```

Cette fonction pourrait être améliorée en revoyant les branchements conditionnels afin qu'ils satisfassent le prefetch. Peut être dans le futur ou un jour ou je serai plus courageux...

## Autre remarque à propos de JWAsm

Dans une procédure vous avez deux appels de fonctions :

- ⤴ CreateFont
- ⤴ CreateWindowEx

CreateFont nécessite de faire SUB RSP,112 alors que CreateWindowEx n'a besoin que de faire SUB RSP,96.

En utilisant la directive INVOKE, JWAsm va générer le code suivant :

```
SUB    RSP,112
CALL   CreateFont
ADD    RSP,112
SUB    RSP,96
CALL   CreateWindowEx
ADD    RSP,96
```

C'est pas terrible.

Quel que soit l'ordre des fonctions appelées, on sait que le plus GRAND SUB RSP,xxxx est celui de CreateFont. Alors pourquoi ne pas écrire de la façon suivante :

```
SUB    RSP,112
CALL   CreateFont
CALL   CreateWindowEx
ADD    RSP,112
```

Ce que savent faire certains assembleur mais que ne fait pas JWAsm. Donc la directive INVOKE est à bannir.

Voici un exemple concret après modification du code source :

```
sub     rsp,112

mov     ecx,14
xor     edx,edx
xor     r8,r8
xor     r9,r9
lea     rax,szTahoma
mov     DWORD_PTR [rsp+32],FW_ULTRALIGHT
mov     DWORD_PTR [rsp+40],edx
mov     DWORD_PTR [rsp+48],edx
mov     DWORD_PTR [rsp+56],edx
mov     DWORD_PTR [rsp+64],edx           ; ANSI_CHARSET
mov     DWORD_PTR [rsp+72],OUT_TT_ONLY_PRECIS
mov     DWORD_PTR [rsp+80],edx         ; CLIP_DEFAULT_PRECIS
mov     DWORD_PTR [rsp+88],CLEARTYPE_QUALITY
mov     DWORD_PTR [rsp+96],VARIABLE_PITCH or FF_DONTCARE
mov     [rsp+104],rax
call    CreateFontA

mov     hFont_Main,rax

xor     rcx,rcx
lea     rdx,szWC_TABCONTROL
lea     r8,szNullString
mov     rax,CW_USEDEFAULT
mov     r9d,WS_CHILD + WS_VISIBLE + TCS_FORCEICONLEFT + TCS_TOOLTIPS
mov     DWORD_PTR [rsp+32],eax
mov     DWORD_PTR [rsp+40],eax
mov     DWORD_PTR [rsp+48],eax
mov     DWORD_PTR [rsp+56],eax
mov     rax,___hWnd
mov     qword ptr [rsp+72],IDC_TAB_01
mov     [rsp+64],rax
mov     qword ptr [rsp+88],rcx
mov     rax,hInstance
mov     [rsp+80],rax
call    CreateWindowExA

mov     hTabLeft,rax
```