Programmation Win32 : Créer une fenêtre

I – Préalable :

Avant de créer une fenêtre, on doit récupérer :

- Un handle sur l'exécutable qui va créer le processus où se trouve la fenêtre

GetModuleHandle qui prend NULL comme argument et qui retourne dans eax ce handle (type HINSTANCE)

- La ligne de commande avec laquelle on lance l'exécutable

GetCommandLine qui ne prend pas d'argument et qui retourne dans eax la ligne de commande (type LPSTR)

On appelle ensuite la fonction WinMain en lui passant en arguments :

- l'handle du processus actuel
- l'handle précédente (NULL pour nous)
- la ligne de commande
- le type d'affichage de la fenêtre (cf WinMain dans la doc. de l'API Win32)

Dans la fonction WinMain :

- on crée, au minimum, la fenêtre principale
- on gère les messages envoyés par les différents éléments

II - Création de la fenêtre :

1 – Définition de la classe :

Pour créer une fenêtre, il faut d'abord déclarer la classe de cette fenêtre, c'est-à-dire ses caractéristiques de base.

Pour cela, on utilise une structure de classe de fenêtre : WNDCLASSEX

On déclare wc en WNDCLASSEX.

Ses différents champs sont :

- **cbSize** : taille de la structure

```
mov wc.cbSize, SIZEOF WNDCLASSEX
```

- **style** : style de la classe de fenêtre

```
mov wc.style, CS_HDRAW + CS_VDRAW
```

- **lpfnWndProc** : adresse de la procédure de gestion de la fenêtre

```
mov wc.lpfnWndProc, offset WndProc
```

- **cbClsExtra** : nombre d'octets supplémentaires à allouer après la classe de fenêtre.

```
mov   wc.cbClsExtra, NULL
```

- **cbWndExtra** : nombre d'octets supplémentaires à allouer après l'instance de fenêtre.

```
mov   wc.cbWndExtra, NULL
```

- **hInstance** : handle sur l'exécutable (retourné par GetModuleHandle)

- **hbrBackground** : brosse utilisée pour le fond de la fenêtre

- **lpszMenuName** : nom de la ressource utilisée pour définir un menu
(NULL si pas de menu)

- lpszClassName : nom de la classe de fenêtre (type de fenêtre : EDIT, SimpleWinClass, etc...)

- **hIcon** : icône de la fenêtre.

- **hIconSm** : petite icône de la fenêtre.

- **hCursor** ; curseur utilisé par la fenêtre.

2 – Affichage de la fenêtre :

Une fois une classe définie, on peut créer plusieurs fenêtres avec le même WNDCLASSEX.
Il faut : - enregistrer une fois pour toutes le WNDCLASSEX avec RegisterClassEx
- créer chaque fenêtre en mémoire avec CreateWindowEx qui retourne un handle de fenêtre
- afficher chaque fénêtre avec ShowWindow

RegisterClassEx : enregistre la classe de fenêtre dont on lui passe l'adresse en paramètre.
invoke RegisterClassEx, addr wc

CreateWindowEx : crée une fenêtre :
Paramètres (dans l'ordre) :
-Style étendu de la fenêtre (voir doc. sur CreateWindowsEx)
-Adresse du nom de la classe de fenêtre (type de fenêtre : EDIT, SimpleWinClass, etc...)
-Adresse du nom de la fenêtre (le titre)
-Style simple de la fenêtre (WS_CHILD : fenêtre fille, WS_OVERLAPPED : fenêtre avec bordure et titre...)
-Coordonnée horizontale de la fenêtre (par rapport à l'écran pour une fenêtre parente, par rapport à la fenêtre parente pour une fenêtre fille)
-Coordonnée verticale de la fenêtre (par rapport à l'écran pour une fenêtre parente, par rapport à la fenêtre parente pour une fenêtre fille)
-Largeur de la fenêtre
-Hauteur de la fenêtre
-Handle de la fenêtre parente (NULL si c'est une fenêtre parente)
-Handle de menu ou identifiant du contrôle (EDIT, bouton, etc…)
-Handle sur l'exécutable (retourné par GetModuleHandle)
-Utilisé pour créer un MDI (Multiple Document Interface)
Retourne un handle de fenêtre dans eax.

ShowWindow : affiche une fenêtre :
Paramètres :
-Handle de la fenêtre à afficher
-Type d'affichage de la fenêtre (SW_SHOWNORMAL pour la première fois)

III – Envoi des messages :

Il faut créer une boucle permettant de recevoir et transmettre les différents évènements générés par nos fenêtres.
Cette boucle infinie est bâtie sur le modèle suivant :

messages:
GetMessage : récupère le message
TranslateMessage : le rend lisible (notamment pour les touches du clavier)
DispatchMessage : envoie le message à la fonction de gestion de la fenêtre
jmp messages

GetMessage : paramètres :
- Adresse où stocker le message reçu (variable de type MSG)
- Handle de la fenêtre dont on veut récupérer les messages (NULL = toutes les fenêtres)
- Valeur du plus petit n° de message à récupérer
- Valeur du plus grand n° de message à récupérer

TranslateMessage : paramètre :
- Adresse du message à traduire (variable de type MSG)

DispatchMessage : paramètre :
- Adresse du message à traduire (variable de type MSG)

IV – Fonction de gestion de la fenêtre :

Le nom de cette fonction est définie dans le champ **lpfnWndProc** de la classe de fonction.
Elle est chargée de gérer les messages reçus par les fenêtres de cette classe.

Elle prend en arguments :

- le handle de la fenêtre ayant envoyé le message : hWnd
- la nature du message : uMsg
- le premier paramètre du message : wParam
- le second paramètre du message : lParam

uMsg sert à déterminer quel est le message ayant provoqué l'appel à la fonction.
On doit souvent gérer :

- WM_DESTROY qui est provoqué par la demande de fermeture de la fenêtre
- WM_CREATE qui est envoyé lors de la création de la fenêtre
- WM_COMMAND qui est envoyé par les éléments de la fenêtre (menu ou contrôle)
- WM_PAINT qui est envoyé lors d'une demande de redessiner une portion de la fenêtre.

Il y a une très grande quantité de messages possibles (voir la documentation de l'API Win32) .

wParam et lParam servent à affiner le traitement du message.
Par exemple, pour WM_COMMAND, on peut savoir de qui vient le message (si lParam==0 alors c'est d'un menu, sinon, c'est d'un contrôle et on récupère son identifiant, ainsi que son code de notification dans wParam)

On doit appeler, si le message reçu n'est pas géré explicitement par notre programme, la fonction de message par défaut DefWindowProc, qui prend les mêmes paramètres que la fonction de gestion de la fenêtre.