

Fournir les fonctions CALLBACK autres que stdcall

Le problème est différent des appels de pointeurs de fonction. En effet, on n'a pas besoin d'objet mais simplement d'une structure qui contiendra du code ASM et un pointeur lpfn vers ce code. Il faut transformer un appel cdecl, fastcall, thiscall, (voire pascal) en stdcall de fonction de module BAS.

Nous ne traiterons ici que le cas de fonction renvoyant des valeurs tenant dans les registre EAX et EDX. Sinon, il faudra tenir compte du fait que le premier paramètre est un pointeur vers une zone mémoire.

Transformer en appel stdcall un appel...

Dans tous les cas, on passera le pointeur lpfn comme pointeur de fonction callback à la fonction demandeuse et on passera aux fonctions de constructions de fonction callback, le pointeur AddressOf de la fonction VB.

Le code aura la structuration suivante :

```
Public Type Callback
    lpfn As Long
```

```
    Code(0 To <taille nécessaire>) As Long
End Type
```

```
Public Sub InitCallback(ByRef lpCB As Callback, ByVal lpfnCallback As Long, <autres
paramètres dépendants de la convention d'appel>)
```

```
    With lpCDeclCB
```

```
        'on remplit avec le code fixe et les paramètres dépendant de la fonction
```

```
        .Code(0) = &HXXXXXXXX
```

```
        '.....
```

```
        .Code(n) = &HXXXXXXXX
```

```
        .lpfn = VarPtr(.Code(0))
```

```
    End With
```

```
End Sub
```

Cdecl

Le problème est que la fonction appelée ne doit pas supprimer les paramètres de la pile. Hors VB supprime toujours les paramètres de la pile à la fin des fonctions (stdcall). Il faudra donc dupliquer les paramètres pour laisser la pile dans son état normal.

Il faut donc :

- On alloue de l'espace sur la pile pour un double des paramètres
- On copie les paramètres et l'adresse de retour dans cet emplacement
- On saute dans la fonction

On aura donc la pile suivante avant l'appel à la fonction callback VB :

Paramètre-n

```

...
Paramètre-2
Paramètre-1
Paramètre-n
...
Paramètre-2
Paramètre-1
Return Address

```

En prenant en compte le fait qu'il faut utiliser les registres EDI et ESI et donc les sauvegarder, cela donne le code ASM suivant :

```

mov ecx, 0x12345678 //on copie la taille des donnée dans ECX :
                    //0x12345678 sera remplacé par la taille des paramètres
add ecx, 4          //on ajoute 4 pour l'adresse de retour

lea eax, [esp - 4] //on charge l'adresse de la pile - 4 pour EDI - 4
                    //pour ESI
sub eax, ecx        //on soustrait la taille des paramètres à
                    //l'adresse nouvelle

test [eax], ecx     //on vérifie que la page est chargée avant de
                    //copier au cas où on aurait changé de page

mov [eax], edi      //on sauvegarde les registres EDI et ESI
mov [eax + 4], esi  //

mov esi, esp        //on calcule l'adresse de la source pour la copie
                    //des paramètres

lea edi, [eax + 8] //on calcule l'adresse de la zone des paramètres
                    //copiés

cld                //copie vers le haut

shr ecx, 2         //copie 4 octets par 4 octets
rep movsd         //on copie les ecx DWORDs

mov esp, eax       //on place la pile à l'adresse des (données
                    //copiées + EDI + ESI)

pop edi           //on restaure les registres EDI et ESI
pop esi

mov eax, 0x12345678 //on copie l'adresse de la fonction stdcall à
                    //appeler : 0x12345678 sera remplacé par l'adresse
jmp eax           //on appelle la fonction

```

Fastcall

Le problème est l'ordre des paramètres :

- **S'il y a un ou plusieurs paramètres entiers ou pointeurs (<= 4 octets), on les met en premier dans le prototype VB**
- **On met donc tout autres paramètres en suivant dans leur ordre de déclaration**

Le code dépend du nombre de paramètres pouvant être mis dans les registres :

- S'il n'y en a pas : l'ordre des paramètres ne change pas

- S'il y en a un, le code met celui ci (ECX) en premier paramètre
- S'il y en a deux, le code met ECX en premier et EDX en deuxième

Le code sera le suivant :

```

pop eax //on récupère l'adresse de retour

//ce code sera soit modifié suivant le nombre de « registrables »
//s'il y a un deuxième paramètre pouvant tenir dans un registre
PUSH EDX
//s'il y a un premier paramètre pouvant tenir dans un registre
PUSH ECX

push eax //on remet l'adresse de retour

mov eax,0x12345678 //on copie l'adresse de la fonction stdcall à
//appeler : 0x12345678 sera remplacé par l'adresse
jmp eax //on appelle la fonction

```

Thiscall

Le plus dure est de connaître la structure de l'objet C++, à savoir les variables privées. Il faut pour cela, disposer du fichier .h et traduire les membres de type variable de l'objet en structure VB. **Il ne faut pas confondre le pointeur This des objets COM (et VB) avec le pointeur This des objets C++. Ce dernier a un pointeur vers une vtable, uniquement s'il contient des fonctions virtuelles.** Un objet C++ n'est pas une entité autonome au sens de COM. La seule différence entre une fonction stdcall et une méthode thiscall est la décoration de son nom par le compilateur et le pointeur this dans ECX.

Le code devra donc simplement mettre le pointeur This de ECX comme premier paramètre. Le code sera le suivant :

```

pop eax //on récupère l'adresse de retour
push ecx //on met le pointeur This C++ sur la pile
push eax //on remet l'adresse de retour

mov eax,0x12345678 //on copie l'adresse de la fonction stdcall à
//appeler : 0x12345678 sera remplacé par l'adresse
jmp eax //on appelle la fonction

```

Pascal

Là c'est simple, il suffit que les paramètres soient mis en ordre inverse. Si on a a,b,c dans le prototype pascal, on met c,b,a dans la fonction BAS.