

Appeler des pointeurs de fonction

Un petit rappel sur les registres, le mode protégé et la pile d'exécution

Un CPU x86, c'est à dire la plus part des processeurs (Intel et AMD) des PCs, comporte des registres qui sont les suivants :

- Registres généraux
 - EAX, le registre à tout faire et utilisé pour le retour de la valeur lors d'appel de fonction (instruction CALL). Il n'est donc pas sauvegardé lors d'appels de fonctions
 - EBX, registre qui doit être sauvegardé avant utilisation dans une fonction
 - ECX, registre compteur, utilisé pour les boucles LOOP et REP. Il n'est pas sauvegardé
 - EDX, registre utilisé pour le retour de valeur de 64 bits (avec EAX). Il n'est pas sauvegarder
- Registres d'index : ils sont sauvegardés pendant les appels de fonctions
 - ESI est le registre qui sert à contenir l'adresse source pour la copie de mémoire avec les instructions MOVS (dites de chaines)
 - EDI est le registre qui sert à contenir l'adresse destination pour la copie de mémoire avec les instructions MOVS
- Registres de piles
 - ESP est le registre qui sert à pointer le sommet de la pile, c'est à dire la dernière valeur empilée
 - EBP est le registre qui sert lors d'appels de fonctions, à conserver l'adresse de pile du début de l'appel de façon à ce que les paramètres gardent le même offset (relativement à EBP) si l'on doit faire des PUSH et des POP (qui modifient ESP). Si l'on utilise ESP, on doit tenir compte de ses variations. Ce registre est conservés lors des appels.
- Registre d'instruction
 - EIP est registre qui pointe la prochaine instruction à exécuter. Il n'est pas accessible ni modifiable directement.

Si vous avez déjà entendu parlé de registres de segments en mode réel, tels que CS, DS, ES, FS, GS, SS...et bien oubliez les, car en mode protégé, il n'y a pas de segments...ou plutôt, un seul de 4Go (en théorie)...

La pile sert à stocker temporairement les informations nécessaires au fonctionnement du programme : adresse de retour de fonction, variables locales...

La pile est une zone mémoire qui croit vers le bas. Au départ, la pile est au plafond et à chaque fois que l'on empile, le sommet descend de 4 octets du plafond. Lorsque l'on dépile, le sommet remonte de 4 octets. Il s'en suit que la mémoire libre est en $ESP - 4$, $ESP - 8$...

Et la mémoire utilisée en ESP , $ESP + 4$, $ESP + 8$...

Lorsque l'on fait un appel de fonction, l'adresse de l'instruction suivant le CALL (adresse de retour) est poussée sur la pile, puis suivent les paramètres suivant la convention d'appel. La pile ressemble donc à ceci :

ESP + 4 * n + 4 : Paramètre-n

...

ESP + 8 : Paramètre 2

ESP + 4 : Paramètre 1

ESP + 0 : Adresse de retour

Si l'on utilise EBP, on écrira le code suivant au début de la procédure :

```
PUSH EBP //on doit sauvegarder EBP avant changement
MOV EBP,ESP //on garde l'emplacement de pile actuel
SUB ESP, taille_des_variables_locales
```

Et l'on écrit à la fin de la procédure :

```
MOV ESP,EBP //on remet ESP à sa place de départ
POP EBP
RET
```

Il s'en suit la disposition de pile suivante (sans variables locales) :

```
ESP + 8 * n + 4 : Paramètre-n
...
ESP + 12 : Paramètre 2
ESP + 8 : Paramètre 1
ESP + 4 : Adresse de retour
ESP + 0 : valeur ancienne de EBP
```

Et relativement à EBP (avec ou sans variables locales):

```
EBP + 8 * n + 4 : Paramètre-n
...
EBP + 12 : Paramètre 2
EBP + 8 : Paramètre 1
EBP + 4 : Adresse de retour
EBP + 0 : valeur ancienne de EBP
EBP - 4 : variable locale 1
EBP - 8 : variable locale 2
...
EBP - 4 * m : variable locale m
```

Un appel de fonction se déroule comme suit, indépendamment de la convention d'appel :

- Tous les arguments sont ajustés à la taille du bus de donnée soit 32bits pour les PC
- Tous les arguments sont placés soit sur la pile, soit dans des registres
- L'adresse de retour est empilée sur la pile
- Un saut est fait vers la fonction
- La fonction crée son cadre de pile avec EBP si nécessaire (si oui, elle sauvegarde EBP)
- Si la fonction utilise ESI, EDI et EBX, elle les sauvegarde sur la pile
- La fonction s'exécute
- La valeur de retour est mise :
 - Dans le registre EAX pour des entiers ou pointeurs ≤ 4 octets
 - Dans les registres EDI :EAX pour des entiers ou pointeurs de 8 octets
 - Dans les registres du coprocesseur arithmétique pour les réels
 - Dans une zone mémoire dont l'adresse est passée en tout premier paramètre à la fonction pour des valeurs de retour de taille > 8 octets. Cette adresse est renvoyée dans EAX.
- Les registres sauvegardés sont restaurés
- Le cadre de pile est restauré
- Suivant la convention d'appel, la fonction retire les paramètres de la pile et rend la main à l'appelant, ou rend simplement la main.

Les objets légers et les pointeurs de fonctions

Je ne sais pas si vous avez remarqué le nombre de AddressOf que l'on peut faire pour les objets légers...mais cela veut dire que l'on a des pointeurs de fonctions dans la vtable...Nous pouvons donc penser à utiliser une vtable pour appeler des pointeurs de fonctions...le seul problème restant est que l'état de la pile et des registres change suivant la convention d'appel...

La méthode sera donc la suivante : on fait créer un objet léger avec une interface possédant une seule fonction qui a la signature (ou à peu près) de la fonction à appeler. On aura donc une vtable à 4 entrées (les méthodes de IUnknown et notre fonction). La quatrième entrée de la vtable sera donc un pointeur vers un morceau de code asm (compilé bien sûr) pour transformer un appel méthode COM en un appel de pointeur de fonction avec la bonne convention d'appel...

Mais vous avez dit « convention d'appel »...sans être indiscret, qu'est-ce que cela peut être ?

Je part du principe que la valeur de retour tient dans les registres et qu'un pointeur vers la mémoire n'est pas nécessaire...Sinon, ça se complique encore un peu plus...on verra ça en fin de tutoriel...

Les conventions d'appels

Une convention d'appel est la façon dont les paramètres sont passés à la fonction appelée. Il en existe un certain nombre.

Nom de la convention	Passage des paramètres	Nettoyage de la pile	Décoration des noms	Notes
Stdcall	Sur la pile de droite à gauche	Appelé	Un _ devant le nom Un @ et la taille des paramètres après le nom	Utilisé par VB et par les APIs Windows
Cdecl	Sur la pile de droite à gauche	Appelant	Un _ devant sauf si extern « C »	Utilisé par défaut par les compilateurs C
Fastcall	Dans les registres ECX et EDI et sur la pile de droite à gauche	Appelé	Un @ devant Un @ et la taille des paramètres après le nom	Utilisé par les compilateurs Borland
Pascal	Sur la pile de gauche à droite	Appelé	????	Obsolète
Thiscall	Sur la pile de droite à gauche et un pointeur This dans ECX	Appelé	Un ? avant Un @ suivi d'un bazar indiquant la signature de la fonction	Utilisé pour les objets C++ (class)
Register	Dans trois registres et sur la pile	Appelé	????	Utilisé par Delphi

La convention stdcall

C'est la seule convention d'appel que VB supporte de façon native. Tous les paramètres sont passés sur la pile de droite à gauche (par rapport à l'ordre de déclaration). Le premier argument sera donc à ESP+4. Le deuxième argument sera à esp+8...et ainsi de suite...

C'est la fonction qui retire les paramètres de la pile avant le RET.

La pile ressemble à ceci :

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

La convention d'appel des méthodes COM

C'est une convention stdcall. La seule différence est qu'il y a en plus, le paramètre this de type pointeur (32 bits).

La pile ressemble à ceci :

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

La convention d'appel cdecl

La convention cdecl est identique à stdcall à l'exception que ce n'est pas la fonction qui retire les paramètres de la pile mais la fonction qui appelle la fonction.

La pile ressemble à ceci :

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

La convention d'appel fastcall

La seule différence avec la convention stdcall, c'est que les deux premiers paramètres entiers ou pointeurs (<= 4 octets) sont passés dans les registres ECX et EDX. Les types réels et supérieurs à 4 octets sont passés sur la pile. Cela signifie que l'ordre des paramètres ne sera pas forcément le même dans : ECX, EDX, la pile...

La pile ressemble à ceci :

Paramètre-n

...
Paramètre-4
Paramètre-3
Return Address

À noter que le parameter 3 n'est pas forcément celui qui est le troisième dans la déclaration de la fonction. Il faut tenir compte des types qui peuvent entrer dans un registre de CPU.

La convention d'appel thiscall

C'est une convention d'appel stdcall et un pointeur This se trouve dans ECX. C'est la convention en C++ pour les objets.

La convention d'appel pascal

C'est une convention stdcall avec les paramètres en ordre inverse, passés de gauche à droite.

Transformer un stdcall méthode en Stdcall

L'adresse de la fonction à appeler sera stocké dans la structure de l'objet léger dans le troisième DWORD, c'est à dire à l'offset 8 par rapport au début de la structure.

La pile ressemble à ceci pour un appel d'une méthode d'objet:

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

Et nous voulons ceci :

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

Il faut donc supprimer le pointeur This de la pile et faire un JUMP à l'adresse de la méthode qui est stockée dans la structure de l'objet (pointée par This) à l'offset 8.

Il faudra donc le code suivant en ASM :

```
POP ECX    //on récupère l'adresse de retour et on la retire de la pile
POP EAX    //on récupère le pointeur This et on la retire de la pile
PUSH ECX   //on remet l'adresse de retour sur la pile
JMP DWORD PTR [EAX + 8] //on fait un saut vers la fonction à appeler
```

Voici donc un moyen d'appeler des fonctions stdcall par un objet. Le code ASM donnera les octets suivants :

Le code suivant n'est pas spécifique à la fonction appelée. On pourra donc le mettre dans une variable globale au module. Il faudra déclarer un type contenant un tableau fixe de Long (pour être aligné) afin d'être sûr que la variable ne soit pas détruite avant l'objet qui pourrait encore en avoir besoin. Et pourquoi pas une constante ? Parce que le code exécutable doit être dans une zone mémoire en lecture écriture...

Nous aurons donc pour tous les code ASM, le type suivant :

```
Type asmCode
    Code(0 To Taille) As Long
End Type
Private m_asmCode as asmCode
```

On ajustera Taille au nombre de DWORDs nécessaires au code compilé. Si le dernier morceau du code fait moins qu'un DWORD, on ajoutera des NOPs (&H90) après le code ASM

Cdecl

Comme pour stdcall, il faut aussi retirer le pointeur This de la pile avant d'appeler la fonction. Le problème restant est qu'au retour de la fonction appelée, les paramètres ne seront pas retirés de la pile. Il faut donc pouvoir exécuter du code juste après l'appel à la fonction avant de rendre la main à VB.

Ajoutons à tout cela, que dans un soucis d'encapsulation, la taille des paramètres à retirer de la pile est propre à une fonction, donc à un objet. Le code de suppression des paramètres de la pile devra se trouver dans la structure de l'objet.

Il faudra donc :

- Retirer le pointeur This de la pile
- Conserver l'adresse de retour finale dans l'objet pour compléter le code ASM inclu
- Appeler la fonction
- Au retour, on se retrouve dans l'objet :
 - On remet sur la pile, l'adresse de retour définitif
 - On fait un RET avec une taille de paramètre contenue dans l'objet

On aura une structure d'objet comme suit :

```
Private Type typFunctionCallerHeap
    'le pointeur vers la vtable
    pVTable As Long
    'le compteur de référence pour savoir quand on doit libérer la mémoire de l'objet
    cCount As Long

    'données attachées
    '-----
    'un pointeur vers la fonction
    lpfn As Long
    'la taille des arguments de la fonction
    cbArgSize As Long

    'le code ASM supplémentaire
    lpPushReturnAddress As Long
    lpRetAddress As Long
    lpRet As Long
```

End Type

Cela donne le code ASM suivant :

```
pop ecx //return address
pop eax //pointeur this
mov [eax + 20],ecx //stocke l'adresse de retour finale
lea ecx,[eax + 16] //charge l'adresse du code de retour
push ecx
jmp dword ptr [eax + 8]

//on trouvera ce code dans les trios derniers membres de la
structure
push 0x12345678 //remplacé pendant l'exécution par l'adresse de
retour
ret 0x1234 //remplacé par la taille des paramètres de la
fonction
```

Fastcall

Alors là, ça se complique au niveau prototype de fonction...car il faut mettre les paramètres, non pas dans leur ordre de définition mais dans l'ordre de la convention fastcall :

- S'il y a un ou plusieurs paramètres entiers ou pointeurs (≤ 4 octets), on les met en premier
- On met donc tout autres paramètres en suivant

Par exemple, si l'on a la déclaration suivante :

```
Int Fct(double d, int a, float f, int* b, int c);
```

On mettra la déclaration suivante dans le fichier ODL :

```
Int Fct([in]int a,[in,out]int*b,[in] double d,[in]float f,[in]int c);
```

Une fois les paramètres dans le bon ordre, il ne reste plus qu'à :

- Retirer le pointeur This (comme toujours)
- On met les deux premiers paramètres dans les registres ECX et EDX si besoin
- On remet l'adresse de retour
- On fait un saut dans la fonction

La structure de l'objet sera la suivante :

```
Private Type typFunctionCallerHeap
```

```
'le pointeur vers la vtable
```

```
pVTable As Long
```

```
'le compteur de référence pour savoir quand on doit libérer la mémoire de l'objet
```

```
cCount As Long
```

```
'données attachées
```

```
'-----
```

```
'un pointeur vers la fonction
```

```
lpfn As Long
```

```
'stockage de l'adresse de retour lors de l'appel
```

```
lpRet As Long
```

```
'nombre d'arguments entiers et pointeurs
```

```
cArgCount As Long
```

```
'code ASM pour gérer l'appel
```

```
asmCode(0 To 5) As Long
```

'puisque le code ASM est différent pour chaque objet, la vtable aussi
VTable As VTable
End Type

Cela nous donne le code suivant :

```
pop ecx //conserve la return address
pop eax //conserve le pointeur this
mov [eax + 12], ecx //On conserve l'adresse de retour dans
l'objet

//premier paramètre dans ECX, s'il y en a un et NOP sinon
pop ecx

//deuxième paramètre dans EDX, s'il y en a un et NOP sinon
pop edx

//On remet l'adresse de retour
push [eax + 12]

//on saute vers la fonction à appeler
jmp dword ptr [eax + 8]
```

Comme le code ASM dépend de la fonction à appeler, on doit mettre le code (et donc la vtable qui pointe vers) dans l'objet.

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. Cette variable est à passer au constructeur de l'objet COM d'appel de la méthode de la classe C++. Il ne faut pas confondre le pointeur This des objets COM (et VB) avec le pointeur This des objets C++. Ce dernier n'a pas de pointeur vers une vtable. 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 est pratiquement celui du code pour stdcall puisqu'il faut seulement mettre un pointeur vers une structure de l'objet C++ dans ECX avant l'appel (ce pointeur This est stocké dans la structure de l'objet COM après l'adresse de la fonction):

```
pop ecx //conserve la return address
pop eax //conserve le pointeur this
push ecx //remet d'adresse de retour sur la pile
mov ecx, [eax + 12] //on copie le pointeur This depuis l'objet
jmp DWORD ptr [eax + 8] //on appelle la fonction
```

Pascal

C'est exactement le même code que pour stdcall à l'exception que les paramètres de la fonctions doivent être écrits (dans le fichier ODL) à l'envers de leur déclaration en pascal. SI les paramètres sont a, b, c en pascal, il seront c, b, a dans le fichier ODL.

Si la fonction renvoie un type de taille supérieure à 8 octets

Dans ce cas, un pointeur vers la zone mémoire prévue pour la valeur de retour est passé :

- TOUJOURS sur la pile pour n'importe quel convention d'appel
- TOUJOURS avant tout autre paramètre pour n'importe quel convention d'appel

Il suffit donc de déclarer ce paramètre dans la liste des paramètres comme un long.

Pour stdcall, cdecl, thiscall et pascal, on ajoute un paramètre Long en premier paramètre.

Pour fastcall, on met le paramètre Long après les deux premiers paramètres pouvant être passés dans les registre (s'il y en a). Ce paramètre de valeur de retour n'est jamais passé par registres.