Partie 1

En-tête

Lionel d'Hauenens -=¤ L.A.B.O. ¤=-

Partie 2

Description du projet

Projet asm_windev

Execute_code_bin

Déclaration de Execute_code_bin

```
CONSTANTE
    MEM_COMMIT = 0x1000
    MEM_RELEASE = 0x8000
    PAGE_EXECUTE_READWRITE = 0x40
FIN
```

Procédure globale Affiche_info_syntaxe

```
PROCEDURE PRIVÉE Affiche_info_syntaxe()

Info (  "La chaine qui représente le code machine n'est pas valide ! "+CRLF,...
        "les 3 syntaxes autorisées sont :"+CRLF,...
        "0xEF,0x14,...",...
        "0EFh,14h,...",...
        "EF,14,..."+CRLF,...
        "De plus, le séparateur d'octet doit obligatoirement être une virgule ','")
```

Procédure globale CodeString2CodeBin

```
////////////////////////////////////////////////////////////////////////////////
// Procédure qui converti du code machine représenté sous forme d'une chaîne de caractère
// en code binaire. Ce code binaire est ensuite copié dans le buffer mémoire passé en argument.
//
// Paramètres :
// -----------
//  Type : (Chaîne) CodeString  Chaine de caractères constituée d'une suite d'octets en hexadécimal.
//                              Chaque octet doit obligatoirement être séparé par une virgule ",".
//                              Il y a 3 formes autorisées pour représenter un octet : 0x5E, 5Eh, 5E.
//                              Les 0 de gauche sont ignorés.
// Type : (entier)  CodeBin     Pointeur vers le buffer mémoire
// Type : (boolean) flag_alert  Active (Vrai) ou Désactive (Faux) les messages d'erreurs
//
// Retour :
```

```
// --------
// Renvoie Vrai si tout s'est bien passé et Faux dans le cas contraire.
////////////////////////////////////////////////////////////////////////////////
PROCEDURE PRIVÉE CodeString2CodeBin(CodeString, CodeBin, flag_alert)

QUAND EXCEPTION DANS
    char est un entier sans signe sur 1 octet
    octet_clean est une chaîne
    nb_byte_write_retour est un entier sans signe sur 4 octet

    size_code est un entier sans signe sur 4 octets = ChaîneOccurrence(CodeString,",")+1
    buffer_table est un tableau dynamique de size_code entiers sans signe sur 1 octet

    compteur est un entier sans signe sur 4 octet = 1

    texte_octet est une chaîne  = ExtraitChaîne(CodeString, rangPremier, ",")
    TANTQUE texte_octet <> EOT
        octet_clean = Nettoyer_texte_octet(texte_octet)
        char = Val (octet_clean,"x")
        SI ((char=0 ET octet_clean<>"00") OU (char<>0 ET char=Val(Gauche (octet_clean,1)))) ALORS
            SI (flag_alert) ALORS
                Erreur ("L'octet 0x" + texte_octet + " n "+ compteur +" n'est pas une valeur hexadécimale valide !")
                Affiche_info_syntaxe()
            FIN
            RENVOYER Faux
        FIN
        buffer_table[Compteur] = char
        compteur++
        texte_octet = ExtraitChaîne(CodeString, rangSuivant, ",")
    FIN
    compteur--
    SI (compteur=0) ALORS
        SI (flag_alert) ALORS
            Erreur ("Aucun code machine n'a pu être trouvé  dans la chaine !")
            Affiche_info_syntaxe()
        FIN
        RENVOYER Faux
    FIN

    // Copier le buffer dans la zone mémoire allouée par VirtualAlloc
    AppelDLL32("kernel32", "WriteProcessMemory" , -1, CodeBin, &buffer_table, compteur, &nb_byte_write_retour)

    SI (compteur<>nb_byte_write_retour) ALORS
        SI (flag_alert) ALORS
            Erreur ("Il y a eu un problème lors de la copie du code machine en mémoire !")
        FIN
```

```
        RENVOYER Faux
    FIN
    RENVOYER Vrai

FAIRE
    SI (flag_alert) ALORS
        Erreur ("Il y a eu une erreur inconnue dans la procédure  CodeString2CodeBin !")
    FIN
    RENVOYER Faux
FIN
```

Procédure globale Executer_code_machine

```
////////////////////////////////////////////////////////////////////////////////
// Cette procédure exécute du code machine relogeable dans un Thread
//
// Paramètres :
// ------------
//  Type : (Chaine)     code_binaire_texte  Chaine de caractères constituée d'une suite d'octets en hexadécimal.
//                                          Chaque octet doit obligatoirement être séparé par une virgule ",".
//                                          Il y a 3 formes autorisées pour représenter un octet : 0x5E, 5Eh, 5E.
//                                          Les 0 de gauche sont ignorés
//  Type : (Indéfini)   parametre_user      Objet quelconque dont windev peut obtenir un pointeur.
//                                          Info : Un tableau de pointeurs permet de gérer plusieurs paramètres user ;)
//                                          C'est une valeur optionnelle et peut être NULL.
//  Type : (boolean)    flag_alert          Active (Vrai) ou Désactive (Faux) les messages d'erreurs
//
// Retour :
// --------
// Renvoie Vrai si tout s'est bien passé et Faux dans le cas contraire.
////////////////////////////////////////////////////////////////////////////////
PROCEDURE Executer_code_machine(code_binaire_texte, parametre_user, flag_alert)

    ThreadMode (threadSectionCritique)

QUAND EXCEPTION DANS
    code_binaire est un entier sans signe sur 4 octets
    parametres est un tableau fixe de 3 entiers sans signe sur 4 octets
    id_thread est un entier sans signe sur 4 octets
    handle_thread est un entier sans signe sur 4 octets
    h_kernel32 est un entier sans signe sur 4 octets
    h_user32 est un entier sans signe sur 4 octets
    size_code_bin est un entier sans signe sur 4 octets
    lpExitCode est un entier sans signe sur 4 octets
    b est un boolean
```

```
 // Préparer les paramètres à passer au code machine
////////////////////////////////////////////////////
h_kernel32 = ChargeDLL("Kernel32.dll")

// Paramètres de base
/////////////////////
parametres[1] = AppelDLL32("kernel32", "GetProcAddress" , h_kernel32 , "LoadLibraryA")
parametres[2] = AppelDLL32("kernel32", "GetProcAddress" , h_kernel32, "GetProcAddress")

// Paramètre Optionnel (Pointeur mémoire ou NULL)
// Pointeur sur les paramètres user
////////////////////////////////////////////////////
SI (TypeVar (parametre_user)= wlVariant) ALORS
    SI (parametre_user..Type=wlEntier) ALORS
        SI (parametre_user=Null) ALORS parametres[3] = 0
    FIN
SINON
    parametres[3] = &parametre_user
FIN

// Allouer la mémoire pour le code binaire
size_code_bin = ChaîneOccurrence(code_binaire_texte,",")+1
code_binaire = AppelDLL32("kernel32", "VirtualAlloc" , 0, size_code_bin, MEM_COMMIT, PAGE_EXECUTE_READWRITE)

// Convertir le  code (format texte) en code binaire
b= CodeString2CodeBin (code_binaire_texte, code_binaire, flag_alert)
SI (PAS b) ALORS
    // Libérer la mémoire contenant le code machine
    AppelDLL32("kernel32", "VirtualFree", code_binaire, 0, MEM_RELEASE)
    RENVOYER Faux
FIN

// Exécuter le code machine dans un thread
handle_thread = AppelDLL32("Kernel32","CreateThread",0,0,code_binaire,&parametres,0,&id_thread)
// Attendre la fin du Thread
AppelDLL32("kernel32", "WaitForSingleObject", handle_thread, 0xFFFFFFFF)
// Libérer la mémoire contenant le code machine
AppelDLL32("kernel32", "VirtualFree", code_binaire, 0, MEM_RELEASE)
// Vérifier le code de retour du thread
AppelDLL32("kernel32", "GetExitCodeThread" , handle_thread, &lpExitCode)
SI (lpExitCode=0) ALORS
    SI (flag_alert) ALORS
        Erreur ("Le code relogeable a eu un problème lors de son exécution ! :(")
    FIN
    RENVOYER Faux
```

```
    FIN

    // Tout va bien :)
    RENVOYER Vrai

FAIRE
    // ERREUR !
    // essayer de fermer tout ce que l'on peut
    AppelDLL32("kernel32", "TerminateThread" , handle_thread, 0)
    AppelDLL32("kernel32", "VirtualFree", code_binaire, 0, MEM_RELEASE)
    SI (flag_alert) ALORS
        // et signaler qu'il y a eu un problème
        Erreur ("Le code relogeable n'a pas pu être exécuté ! :(")
    FIN
    RENVOYER Faux
FIN
```

Procédure globale Nettoyer_texte_octet

```
////////////////////////////////////////////////////////////////////////////////
// Procédurent qui formate une chaîne censée représenter un octet en hexadécimal.
//
// Le préfixe "0x" et le suffixe "h" sont autorisés mais pas obligatoires.
//
// Paramètres :
// ------------
//  texte_octet : (Chaîne)  texte_octet : chaîne de caractère à nettoyer.
//
// Retour :
//---------
// Renvoie une chaîne de caractère sur 2 caractères représentant un octet en hexadécimal.
// Si la chaïne d'origine n'a pas pu être formatée la chaîne "INVALIDE" est renvoyée.
////////////////////////////////////////////////////////////////////////////////
PROCEDURE PRIVÉE Nettoyer_texte_octet(texte_octet)

QUAND EXCEPTION DANS

    s est une chaîne = texte_octet

    // nettoyer les espaces
    s = SansEspace(s)

    SI (Taille(s)=0) ALORS RENVOYER "INVALIDE"       // octet invalide

    // passe tout en majuscule
    s = Majuscule(s)
```

```
    // enlever les infos sur la base 16
    SI (Gauche (s,2)="0X") ALORS
        s = Droite (s,Taille(s)-2)
    SINON
        SI (Droite(s,1)="H") ALORS
            s = Gauche (s,Taille(s)-1)
        FIN
    FIN

    // Nettoyer les zéros de gauche
    TANTQUE (Taille(s)>0 ET Gauche (s,1)="0")
        s = Droite (s,Taille(s)-1)
    FIN

    // vérifier et fromater l'octet texte nettoyé
    SI (Taille(s)>=0 ET Taille (s)<=2) ALORS
        RENVOYER Droite ("00"+s,2)              // texte octet sur 2 caractères  :)
    FIN

    RENVOYER "INVALIDE"                         // octet invalide !!!

FAIRE
    RENVOYER "INVALIDE"                         // octet invalide !!!
FIN
```

Partie 3

Description des fenêtres

Fenêtre Windev Wake Up

Code des champs

Initialisation de BTN_Executer_le_code

Clic sur BTN_Executer_le_code

```
    ////////////////
   // Obligatoire //
  ////////////////
 // Code machine à exécuter au format texte
//////////////////////////////////////////

code_binaire_texte est une chaîne=...
"EB,65,00,00,00,00,00,00,00,00,75,73,65,72,33,32,2E,64,6C,6C,"+...
"00,4D,65,73,73,61,67,65,42,6F,78,41,00,4A,65,20,73,75,69,73,"+...
"20,6C,65,20,74,65,78,74,65,20,70,61,72,20,64,E9,66,61,75,74,"+...
"20,64,75,20,63,6F,64,65,20,72,65,6C,6F,67,65,61,62,6C,65,2E,"+...
"20,3A,29,00,57,69,6E,64,65,76,2E,2E,2E,20,57,61,6B,65,55,70,"+...
"20,21,00,60,E8,00,00,00,00,5D,81,ED,6D,00,00,00,8D,85,0A,01,"+...
"00,00,50,64,FF,35,00,00,00,00,64,89,25,00,00,00,00,8B,7C,24,"+...
"2C,8B,07,89,85,02,00,00,00,8B,47,04,89,85,06,00,00,00,8D,85,"+...
"0A,00,00,00,50,FF,95,02,00,00,00,89,85,0A,00,00,00,8D,85,15,"+...
"00,00,00,50,FF,B5,0A,00,00,00,FF,95,06,00,00,00,89,85,15,00,"+...
"00,00,8D,9D,21,00,00,00,8D,85,54,00,00,00,8B,4F,08,09,C9,74,"+...
"05,8B,01,8B,59,04,6A,10,50,53,6A,00,FF,95,15,00,00,00,31,C0,"+...
"40,64,8B,25,00,00,00,00,64,8F,05,00,00,00,00,83,C4,04,89,44,"+...
"24,1C,61,C2,04,00,60,E8,00,00,00,00,5D,81,ED,10,01,00,00,8B,"+...
"7C,24,2C,8D,85,F1,00,00,00,89,87,B8,00,00,00,31,C0,89,87,B0,"+...
"00,00,00,61,31,C0,C3"

     ///////////////
    // Optionnel //
   ///////////////
  // Préparation des 2 paramètres à passer au code machine
 // parametre_user peut être NULL
//////////////////////////////////////////////////
texte_titre est une chaîne ASCIIZ sur 100 = "Je suis le titre passé en paramètre :)"
```

```
texte_message est une chaîne ASCIIZ sur 256 = "Je suis le texte passé en paramètre :)"
parametre_user est un tableau fixe de 2 entier sans signe sur 4 octets
parametre_user[1]= &texte_titre
parametre_user[2]= &texte_message

 // Test d'exécution du code avec passage d'une table de 2 paramètres
////////////////////////////////////////////////////////////////////
SI (Executer_code_machine(code_binaire_texte, parametre_user, Vrai))ALORS
    Info ("Le code machine avec paramètres vient d'être exécuté avec succès :)")
SINON
    Erreur ("Le code machine avec paramètres n'a pas pu être exécuté :(")
FIN

// Test d'exécution du code sans paramètres
///////////////////////////////////////////////////
SI (Executer_code_machine(code_binaire_texte,Null, Vrai) )ALORS
    Info ("Le code machine sans paramètre vient d'être exécuté avec succès :)")
SINON
    Erreur ("Le code machine sans paramètre n'a pas pu être exécuté :(")
FIN
```

Partie 4

Table des matières

Table des matières

Projet asm_windev

2 En-tête Partie 1
2 **En-tête**

4 Projet Partie 2
4 **Execute_code_bin**

11 Fenêtres Partie 3
11 **Windev Wake Up**
12 Code des champs