COMPTE-RENDU DE PROJET

Sujet : Reproduction de la fonction « DIR » de Windows en langage MASM x86

Rapport rédigé & produit par : **Kévin MOREAU**

Professeur responsable : **Vianney LAPOTRE**

École Nationale Supérieure d'Ingénieurs de Bretagne Sud

Apprenti - Ingénieur Expert Cyberdéfense

1ère année

Promotion AIRBUS

06 JUIN 2019

ENSIBS VANNES –CYBERDEFENSE – PROMOTION 2021

Mail : e1604724@etud.univ-ubs.fr

Table des matières

Table des illustrations

Contexte

Ce projet s'inscrit dans le module « Architecture des ordinateurs » du second semestre de la formation Cyberdéfense de l'ENSIBS. Son but est double :

1) Appréhender les mécanismes de bas niveau d'une architecture de processeur complexe.

2) Fournir les prérequis nécessaires au module « investigation numérique » enseigné au quatrième semestre du cycle d'ingénieur en Cyberdéfense.

Le projet

L'application a réalisé dans ce projet est un clone de la fonction Windows 'DIR \s' permettant de lister récursivement les fichiers présents sur le disque à partir d'un point d'entrée fourni par l'utilisateur.

L'application pourra, dans un premier temps être développé dans une version ligne de commande uniquement. De plus, le listing de fichier sera affiché dans une fenêtre permettant de naviguer (scrolling) dans le listing.

/!

Par manque de temps dans la réalisation du projet, la partie suivante ne sera pas traitée dans le projet :

Cependant, une seconde version devra disposer d'une interface graphique permettant à l'utilisateur d'entrée le point d'entrée dans un champ spécifique (DialogBoxParam).

/!

Le développeur devra penser à l'ergonomie de l'affichage afin de faciliter la lecture du listing de fichier produit.

Le rapport

Ce court rapport permettra de rajouter certains détails essentiels à la compréhension du code présent dans le fichier *ProjetDIR.asm.*

Les détails des fonctions et des choix de logique de programmation seront exposés et expliqués dans les pages suivantes.

1. Structure du code – Pseudocode guide

Avant de commencer à partir sur l'explication des fonctions utilisées dans le cadre du projet. Je m'attarde un instant sur l'importance de la structure du code dans la compréhension du programme.
Au départ, j'ai gardé une logique par bloc. Chaque fonction était définie dans une procédure à part qui était appelée lorsque j'en avais besoin. Au fur et à mesure de la montée en puissance de la complexité du code, j'ai décider de renoncer à ce type de structure et à privilégier les alias à l'intérieur d'une seule et unique procédure : *treeFiles*

A l'intérieur de cette procédure se décompose les différentes fonctionnalités du programme.
Pour structurer cette procédure, je suis passé par du pseudo-code pour mieux comprendre le déroulement du programme en prenant en compte la récursivité.
Voici le pseudo-code défini au début du projet afin de m'orienter dans la programmation. Il n'est pas parfait mais permet de me repérer dans le processus. Il est affiné dans le code bien entendu.

```
CurrentDir = CurrentDir (défini par l'utilisateur en dur)

1) FindFirstFile in CurrentDir
   Si un premier fichier est présent
                    Continue
        Sinon
                    Erreur

2) FindFirstFile is a "." Directory or ".."
        Si c'est un dossier "."
                    Goto 4)
        Sinon
                    Continue

        Si c'est un dossier ".."
                    Goto 4)
        Sinon
                    Continue

3) Si c'est un dossier
                    On rentre à l'intérieur
                    On change de dossier courant
                    On liste les fichiers
                    On vérifie leurs correspondances avec "." ou ".."

  Sinon
                    Goto 4

4) FindNextFile in CurrentDir
        S'il y a un autre fichier :
                    On vérifie leurs correspondances avec "." ou ".."
                    Si c'est un dossier :
                                On rentre à l'intérieur
                                On change de dossier courant
                                On liste les fichiers
                                On vérifie leurs correspondances avec "." ou ".."
        S'il n'y a pas de fichier
                    On remonte l'arborescence si besoin
                    On quitte
```

2. CurrentDir et appel de la fonction FindFirstFile

```
start:
    ;Un petit peu de décoration
    push  offset strnull
    push  offset strIntro
    ;Indication du dossier courant
    push  offset CurrentDir
    push  offset strIndicationDossierCourant
    call    crt_printf

    ;Appel de la fonction treeFiles avec le chemin CurrentDir initial en paramètre
    push    offset CurrentDir
    call    treeFiles
```

Figure 1 - Aperçu du début du start

La fonction *FindFirstFile*, comme décrit dans la documentation, a besoin d'un paramètre qui est le chemin courant dans lequel la fonction va venir chercher le premier fichier.

```
;Définition de la fonction FindFirstFile (documentation)
;FindFirstFileA(LPCSTR lpFileName,LPWIN32_FIND_DATAA lpFindFileData);
push    offset datafiles    ; Paramètre structure pour la fonction FindFirstFile
push    [ebp+8]             ; A partir de x32dbg, l'accès à la structure de la pile permet de récupérer le paramètre du dossier courant pour la fonction FindFirstFile (path)
call    FindFirstFile       ; Appel de la fonction FindFirstFile ; resultat dans eax
```

Figure 2 - FindFirstFile : Paramètres et lancement de la fonction

Premièrement, comme décrit dans la documentation, la fonction a besoin d'une structure (ici *datafiles*) de fichier contenant toutes les informations du fichier. La fonction va venir remplir cette structure avec les informations qu'elle trouve sur le premier fichier du dossier courant.

Deuxièmement, c'est le chemin qui a été push dans le *start* qui va être pris en paramètre (on a pris soin, juste avant d'appeler la fonction, d'enregistrer l'état de la pile, c'est pour ça qu'on peut se permettre de récupérer le contenu de [ebp+8] en paramètre, car en réalité cela correspond bien à *push offset CurrentDir* réalisé dans le *start.*

La fonction renvoie deux choses. La première, de façon indirecte, est le remplissage de la structure passée en paramètre, ainsi *datafiles* va contenir l'ensemble des informations décrits dans sa structure de données.

L'autre résultat est lui contenu dans le registre *EAX*. Il s'agit d'une valeur correspondant à la bonne exécution de la fonction. C'est pour cela que, juste après, on compare la valeur de *EAX* avec la valeur *INVALID_HANDLE_VALUE*, pour vérifier que la fonction s'est bien déroulée, et dans le cas contraire, nous renvoyer l'erreur de la fonction grâce au *call GetLastError*.

```
; Vérification au cas où présence d'erreur
; On compare la valeur du résultat. Rappel : Le résultat est stocké dans EAX
cmp     eax, INVALID_HANDLE_VALUE ; if eax = INVALID_HANDLE_VALUE, call GetLastError
mov     [esp], eax ; Insertion de la valeur de eax en haut de la pile
je      GetLastError ; renvoi vers la fonction check_error si INVALID_HANDLE_VALUE.
```

Figure 3 - Réflexion en cas d'erreur

3. Correspondance avec les dossiers ‘.’ et ‘..’

Une fois le fichier trouvé et la structure remplie, il est obligatoire de tester une condition. En effet, sous Windows et dans chaque dossier, se trouve deux dossiers obligatoirement présents : ‘.’ et ‘..’.

Ainsi, il est nécessaire d’appliquer au cas par cas chaque fichier pour savoir s’il correspond avec l’un de ces deux dossiers. Car en effet, on verra plus tard lors de la récursivité que si on déclare ‘.’ et ‘..’ comme des dossiers à traiter, on risque de boucler à l’infini car dans ces mêmes dossiers se trouve également un ‘.’ et un ‘..’.

Ainsi on va venir tester, pour chaque fichier que l’on va traiter récursivement, s’il correspond à l’un de ces deux dossiers, voici comment l’on procède :

```
push    1 ; third argument : size
push    offset datafiles.cFileName ; second argument.
push    offset strDot ; first argument (".")
call    crt_strncmp
;crt_strncmp renvoie 0 dans eax si les deux fichiers sont égaux.
```

Figure 4 - Comparaison entre deux strings

On déclare une variable qui va contenir la chaîne de caractère ‘.’ que l’on va venir comparer avec le nom du fichier (*datafiles.cFileName*) grâce à la fonction *crt_strncmp* qui permet de comparer deux chaînes de caractères.
Le résultat est contenu dans le registre *EAX*, on compare ainsi la valeur de *EAX* avec *0* pour déterminer si les deux fichiers sont égaux. Si c’est le cas, on passe directement au fichier suivant sans avoir à rentrer dans ce dossier car on ne le souhaite pas.

```
add     esp, 12 ;Restructuration de la pile (chaque argument (les 3 présents ci-dessus) prend 4 octets soit 4x3 = 12 octets pour bien replacer la pile)
cmp     eax, 0 ; On compare le résultat de la fonction crt_strncmp avec 0 (ce qui signifie que les deux sont égaux)
je      treeFiles_nextFile ; Si égal, tout s'est bien passé, on passe au fichier suivant via la fonction treeFiles_nextFile
```

Figure 5 - Go next file si le fichier correspond à la string

On applique exactement la même procédure pour la chaîne de caractère ‘..’, elle ne sera pas détaillée ci-dessous.
Par contre si ce n’est pas égal, on va tester vérifier que ce fichier possède les attributs d’un dossier. Cette fonctionnalité est programmée dans l’alias *treeFiles_corpsFunction*. Pour savoir si le fichier est un dossier, on compare son attribut avec une variable déjà implémentée qui est *FILE_ATTRIBUTE_DIRECTORY*

```
; 2) On vérifie si le fichier traité est un dossier
mov     eax, offset datafiles.dwFileAttributes
cmp     byte ptr[eax], FILE_ATTRIBUTE_DIRECTORY ;rappel issu de x32dbg : [cmp byte ptr eax,10]
;Si c'est un dossier : eax == FILE_ATTRIBUTE_DIRECTORY
```

Figure 6 - Test de détermination si le fichier est un dossier

Si ce n’est pas le cas, alors on passe au fichier suivant. Par contre si la fonction renvoie une valeur indiquant que le fichier est un dossier, il faut procéder à certaines opérations décrites juste après.

4. Le fichier est un dossier... que faire ?

La fin de la partie précédente comparait la valeur d'attribut d'un fichier avec la valeur associée à l'attribut d'un dossier. Ainsi, si la comparaison s'avère être bonne (les deux valeurs sont égales), cela signifie que notre fichier traité et un dossier qui ne correspond ni à '.' ni à '..', il faut donc le traiter de façon bien précise, et c'est là que la récursivité et la notion de niveau apparaisse.

Premièrement, une fois qu'on a défini que le fichier est un dossier, on va définir ce dossier comme nouveau dossier courant grâce à la fonction *SetCurrentDirectory*.

```
;Sinon, si c'est un dossier on push son nom en haut de la pile et on appelle la fonction SetCurrentDirectory qui permet de définir le nouveau dossier courant
;Documentation de la fonction SetCurrentDirectory : BOOL SetCurrentDirectory(LPCTSTR lpPathName);
push    offset datafiles.cFileName ; first arg : lpPathName
call    SetCurrentDirectory ; appel de la fonction


; 3) On compare la valeur retournée par la fonction SetCurrentDirectory avec 0 pour déterminer une erreur
cmp     eax, 0
;Si c'est OK, la fonction renvoie 1
;Ainsi, si la fonction n'a pas marché que le datafiles.cFileName ne peut pas être défini comme dossier courent
;Alors, on passe au fichier suivant
je      treeFiles_nextFile
;   Sinon, on incrémente le niveauDossier en passant par le registre EDX. On rentre alors dans un sous-dossier, le niveau change
mov     edx, offset niveauDossier
inc     dword ptr [edx] ; ici on incrémente le pointeur, c'est donc la valeur de niveauDossier qui va changer

;     Puis on applique la récursivité en rapellant la fonction avec le dossier défini en paramètre de la fonction
;     Et on boucle tant qu'il y a des fichiers. La sortie de la boucle se situe à la comparaison avec ERROR_NO_MORE_FILES de la fonction treeFiles_nextFile (ligne 190)
push    offset CurrentDir
call    treeFiles
```

Figure 7 - Assignation d'un fichier comme dossier courant

La fonction renvoie une valeur dans *EAX* qu'on va exploiter pour déterminer une erreur d'assignation.

Si l'assignation a bien fonctionnée, la fonction renvoie *1* dans EAX.
Par la suite nous allons utiliser une nouvelle variable intitulée niveauDossier qui permet de se repérer dans l'arborescence des chemins courants. Ainsi, étant donné que le dossier courant a changé (on est entré en profondeur dans l'arborescence), il faut donc changer la valeur de *niveauDossier* pour pouvoir indiquer au programme qu'on est bien dans un niveau différent du dossier courant initial.

Cette variable va être extrêmement pratique dans l'utilisation d'une partie du programme qui sera détaillée juste après, qui est la tabulation.

Une fois le niveau incrémenté, il ne nous reste plus qu'à rappeler la procédure pour effectuer de manière récursive toutes les opérations vues précédemment mais cette fois-ci, en ayant changé le dossier courant (car nous sommes cette fois-ci plus dans le même dossier mais bien en profondeur).

5. L’appel à un autre fichier dans le répertoire courant

Je m’attarde rapidement là-dessus pour expliquer le fonctionnement de la fonction *FindNextFile*. Le fonctionnement est assez simple et nous aurons besoin de certaines explications pour la fin du rapport.

Cette fonction est appelée à chaque fois après un *FindFirstFile*. Elle nécessite plusieurs paramètres :

```
;Recherche des fichiers suivants :
treeFiles_nextFile:
  push    offset datafiles ; Second argument ; C'est ici qu'on rappelle la structure initialement remplie par la fonction FindFirstFile, et ce, récursivement.
  push    [ebp-4] ; First argument : Définition du handle nécessaire à la fonction
  call    FindNextFile ; Fonction FindNextFileA ; resultat dans eax

  ; Verification si présence d'erreur dans le handle
  cmp     eax, 0 ; FindNextFile renvoie un "binaire" 0 ou 1 dans eax selon le bon fonctionnement
          ; de la fonction. 1 signifie que l'opération s'est bien passé
          ; Les informations de la fonction sont stockés dans la structure
  jne     treeFiles_checkNamesNext ; Si il n'y a pas d'erreur, on passe au fichier suivant
  call    GetLastError ; Sinon on renvoie l'erreur correspondante

  ; Ensuite, si il n'y a plus de fichier dans le dossier on continue, sinon on saute à l'étape finale
  cmp     eax, ERROR_NO_MORE_FILES
  je      treeFiles_filesEnd ; Si plus de fichier dans le dossier, on passe à l'étape finale
```

Figure 8 - Récupération des infos concernant le fichier suivant

Elle utilise la même structure de données que la fonction *FindFirstFile*. En fait la fonction *FindNextFile* va remplacer les données de l’ancien fichier contenues dans la structure datafiles par le prochain fichier. Par ailleurs, la fonction a besoin d’un handle pour fonctionner.

Si un prochain fichier est présent, la fonction renvoie une valeur dans le registre *EAX*. Si la valeur est égale à *0*, cela signifie que la fonction renvoie une erreur.

On appelle par la suite *GetLastError* pour plus d’informations. Au-delà de ça, il faut également comparer *EAX* avec la valeur indiquant qu’il n’y a plus de fichier à traiter dans le dossier courant. Si c’est le cas, on arrive donc à la fin du dossier courant.

On appelle l’alias de fin du programme pour remonter de niveau si nécessaire.

6. Fin du dossier courant, on remonte !

Ainsi, lorsqu'il n'y a plus de fichier à traiter, on remonte de dossier en effectuant cette liste d'opérations :

```
; On arrive dans cette fonction quand il n'y a plus de fichier dans le dossier courant :
; à la fin de cette fonction, la récursivité s'applique avec tous les résultats précédents
treeFiles_filesEnd:
  ;On remonte le dossier courant en poussant ".." et en l'appellant dans la fonction SetCurrentDirectory
  push    offset strDoubleDot
  call    SetCurrentDirectory

  ;Une fois remonté, il ne faut pas oublier de décrémenter le niveau de notre dossier. On effectue la même opération
  ; que les lignes (166 & 167) en décrementant le niveau à l'inverse de l'incrémenter
  mov     edx, offset niveauDossier
  dec     dword ptr [edx]
  ;On restructure la pile
  add     esp, 4
  ;On quitte proprement la fonction
  leave
    ret
```

Figure 9 - Procédure de fin de dossier courant

On pousse la chaîne de caractère *'..'* et on appelle la fonction *SetCurrentDirectory*. Cette opération permet de remonter le dossier courant pour revenir au précédent. Cela permet ainsi de lister le reste des fichiers du *dossier n-1* si jamais il y en a après le dossier rencontré.

On profite, par la même occasion, de décrémenter la variable *niveauDossier* pour indiquer au programme qu'on est bien remonté. Ensuite on quitte et on retourne lorsque la récursivité est terminée et qu'il n'y a plus aucun fichier à traiter dans le dossier initial ainsi que dans ses sous-dossiers.

7. L’affichage et la tabulation

Etape clé du programme, l’affichage des fichiers / dossiers en fonction de leur dossier dépendant. C’est ici que la variable *niveauDossier* va permettre de structurer l’affichage de nos fichiers :

```
treeFiles_tabulationDossier:
  mov    edx, offset niveauDossier ; Par défaut, le niveau est à 0 (correspondant au 'root' du chemin choisi, ici ./*)
  mov    eax, [esp] ; On déplace l'adresse du haut de la pile dans eax
  cmp    [edx], eax ; On compare l'adresse du haut de la pile avec l'adresse edx, précédemment défini. Plus concrétement, on compare niveauDossier avec le niveau que l'on atteint avec la récursivité.
  je     treeFiles_corpsFunction ; Si c'est égal, on passe à la prochaine étape qui est l'affichage du fichier, au début du programme on ne tabule donc plus car nous sommes au même niveau
  ;treeFiles_corpsFunction permet également de sortir de la boucle créer par le jmp treeFiles_tabulationDossier défini en fin d'alias

  ; Si c'est différent, on print un tab, cela signifie qu'on est à l'intérieur d'un dossier, on affiche donc une tabulation juste avant d'afficher le nom du fichier.
  ; Cela permet donc une structuration récursive efficace sans retour à la ligne entre chaque print
  push   offset tabulation ; On pousse le contenu de tabulation (soit 2 espaces), correspondant à un TAB en haut de la pile
  call   crt_printf ; On l'affiche dans le terminal.

  add    esp, 4 ; On restructure la stack
  inc    dword ptr [esp] ; On incrémente la valeur de l'adresse d'esp puis on recommence la fonction de tabulation tant que
  jmp    treeFiles_tabulationDossier
```

Figure 10 - Boucle permettant l'insertion de tabulation avant le nom du fichier

Pour commencer, on initialise une variable que l’on nomme ici *i*.

Cette variable va permettre de de déterminer le nombre de tabulation à afficher avant d’afficher le nom du fichier.

On initialise *i* à *0* au début du cycle, car nous sommes dans le dossier courant défini au départ. Par ailleurs, *niveauDossier*, au début du programme est lui aussi défini à *0*.

Ainsi, lorsque *niveauDossier* et *i* sont égaux, on affiche simplement le nom du fichier.
Si par contre, comme on l’a vu dans les sections précédentes, on rentre dans un sous-dossier du dossier courant, la variable *niveauDossier* va venir s’incrémenter. On se retrouve donc dans le cas où *i* vaut toujours *0* et *niveauDossier* vaut *1*.

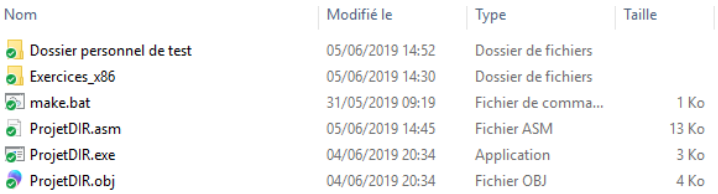
Ainsi, si la comparaison n’est pas bonne, on va afficher une tabulation (représentée dans le programme par deux espaces). Une fois la tabulation affichée, on incrémente *i*. *i* passe donc à 1 puis on rappelle la fonction de tabulation tant que *niveauDossier* et *i* ne sont pas égaux.

Enfin, *niveauDossier* et *i* se retrouve donc au même niveau, on peut alors afficher le fichier juste à coté de la tabulation et sortir de la boucle.

Les tabulations vont donc s’accumuler naturellement jusqu’à l’affichage du fichier.

8. Aperçu et conclusion de projet

Pour illustrer un petit peu le fonctionnement du programme, voici un aperçu final du projet avec le dossier courant du projet en question :



Nom	Modifié le	Type	Taille
Dossier personnel de test	05/06/2019 14:52	Dossier de fichiers	
Exercices_x86	05/06/2019 14:30	Dossier de fichiers	
make.bat	31/05/2019 09:19	Fichier de comma...	1 Ko
ProjetDIR.asm	05/06/2019 14:45	Fichier ASM	13 Ko
ProjetDIR.exe	04/06/2019 20:34	Application	3 Ko
ProjetDIR.obj	04/06/2019 20:34	Fichier OBJ	4 Ko

Figure 11 - Aperçu graphique du dossier sous Windows 10

```
Dossier courant : .\* ,voici le contenu du dossier courant :

Dossier personnel de test
  1002136_black-folder-png.png
  CNI.txt
  monpasseport.txt
  Passeport.txt
  programmer-x86-assembly.jpg
  stackoverflow.png
  tellmeeverything.png
  x86.png
Exercices_x86
  Calculs
    Calculs.asm
    Calculs.exe
    Calculs.obj
    make.bat
  HelloWorld
    HelloWorld.asm
    HelloWorld.exe
    HelloWorld.obj
    make.bat
  Messagebox
    make.bat
    MessageBox.asm
    MessageBox.exe
    MessageBox.obj
  MinusculesMajuscules
    make.bat
    MinenMaj.asm
    MinenMaj.exe
    MinenMaj.obj
  TP1.pdf
  VariablesLocales
    make.bat
    VariablesLocales.asm
    VariablesLocales.exe
    VariablesLocales.obj
make.bat
ProjetDIR.asm
ProjetDIR.exe
ProjetDIR.obj

Fin du programme, tous les fichiers / dossiers ont ete listes
Appuyez sur une touche pour continuer...
```

Figure 12 - Aperçu du dossier avec notre programme en lignes de commandes

Pour conclure, ce projet fut une véritable découverte de l'assembleur pour ma part. Un projet adapté aux bases à avoir en assembleur puis une première approche de MASM et du débuggeur qui fût d'une aide précieuse.

Enfin, merci à certaines personnes de la promotion pour leur aide précieuse (notamment avec la récursivité...).

Merci pour votre lecture,
Kévin Moreau