Projet Langage d'assemblage et microprocesseur

Baptiste COSTAMAGNA e2101629

Pierre WADRA e2303942

Rapport sur la Création d'un Explorateur de Répertoire en Assembly MASM32

Dans le cadre de notre projet d'informatique, nous avons conçu un explorateur de répertoire simulant la fonctionnalité de la commande `dir` de Windows permettant de lister récursivement les fichiers présents sur le disque à partir d'un point d'entrée fourni par l'utilisateur. On parlera des éléments importants de la version ligne de commande. Puis on parlera brièvement de la partie interface graphique qui n'a pas été faite entièrement en raison du temps imparti.

Fonctionnement des Composants Clés

1. FindFirstFile et FindNextFile

Ces fonctions de l'API Windows sont essentielles pour parcourir les répertoires. `FindFirstFile` initialise la recherche dans un répertoire, renvoyant une structure `WIN32_FIND_DATA` qui contient des informations sur le premier fichier ou dossier trouvé. `FindNextFile` continue cette recherche, permettant de traiter chaque élément jusqu'à ce que tous les fichiers soient explorés.

2. WIN32_FIND_DATA

Cette structure est utilisée pour stocker des informations sur un fichier ou un dossier trouvé dans un répertoire. Elle contient, entre autres, les attributs du fichier, les horodatages de création, de dernier accès et de modification, ainsi que

les noms de fichiers. Dans notre code, nous utilisons cette structure pour vérifier si l'élément est un dossier et pour récupérer son nom afin de l'afficher.

3. Fonctions crt_strncpy et crt_sprintf

- `crt_strncpy` : Copie un nombre spécifié de caractères d'une chaîne source vers une destination, utilisée ici pour copier le chemin spécifié par l'utilisateur vers une variable locale.
- `crt_sprintf` : Formate une chaîne et l'écrit dans une variable; dans notre cas, utilisée pour concaténer des chemins pour les sous-répertoires.

Voici une analyse de quelques détails du code :

1. Déclarations de Données

- `path` : contient la chaîne "*", utilisée pour `FindFirstFile` afin de récupérer tous les fichiers et dossiers du répertoire spécifié.
- `element` , `folder` : utilisées pour implanter les entrées lors de leur affichage (pour dinstinguer les dossiers des fichiers)
- `niveauRecursion` : un compteur pour suivre la profondeur de récursivité dans la navigation.
- `pointSimple` **et** `pointDouble` : représentent "." et ".." dans les systèmes de fichiers, utilisés pour ignorer le répertoire courant et le répertoire parent dans la navigation.
- `formatAffichage` , `pathFormat` , et d'autres chaînes formatées sont utilisées pour structurer l'affichage et le traitement des chaînes.

2. Fonction `concatene`

Cette procédure est une fonction d'aide pour concaténer deux chaînes. Elle manipule directement les pointeurs de chaînes pour fusionner les contenus en place.

3. Fonction `naviguerRepertoire`

C'est une fonction récursive essentielle qui parcourt les répertoires :

- `sub esp, 1024` : alloue 1024 octets sur la pile pour stocker localement des données telles que les chemins des fichiers. Ce choix de taille assure assez d'espace pour la plupart des chemins sans dépasser la pile.
- `WIN32_FIND_DATA` : (voir ci-dessus).
- `FindFirstFile` **et** `FindNextFile` : (voir ci-dessus)

4. Fonction `visu`

Cette fonction affiche les informations sur chaque fichier ou dossier. Elle gère également l'indentation pour montrer visuellement la structure des dossiers :

- `niveauRecursion` ajuste l'indentation pour chaque niveau de dossier, améliorant la lisibilité de la sortie.
- `crt_printf` est utilisée pour formater et afficher les noms de fichiers et de dossiers avec l'indentation appropriée.

5. Offsets

1. **WIN32_FIND_DATA (318o)**:

la structure est comme suit :

```
typedef struct _WIN32_FIND_DATA {
  DWORD    dwFileAttributes;       // 4 octets
  FILETIME ftCreationTime;         // 8 octets (2 DWORDs)
  FILETIME ftLastAccessTime;       // 8 octets (2 DWORDs)
  FILETIME ftLastWriteTime;        // 8 octets (2 DWORDs)
  DWORD    nFileSizeHigh;          // 4 octets
  DWORD    nFileSizeLow;           // 4 octets
  DWORD    dwReserved0;            // 4 octets
  DWORD    dwReserved1;            // 4 octets
  TCHAR    cFileName[MAX_PATH];    // 260 octets (MAX_PATH est 2
```

```
   TCHAR    cAlternateFileName[14]; // 28 octets (14 TCHARs)
 } WIN32_FIND_DATA, *PWIN32_FIND_DATA;
```

La taille totale est de 16 + 24 + 260 + 28 = 328 octet

1. **PATH (260o)**:
    - Cet espace est réservé pour stocker le chemin d'accès initial ou en cours de traitement. L'espace alloué est de 260 octets, ce qui correspond à la longueur maximale d'un chemin dans Windows (`MAX_PATH`).

    (voir **Limitation de longueur maximale du chemin :** https://learn.microsoft.com/fr-fr/windows/win32/fileio/naming-a-file)
2. **HANDLE (4o)**:
    - Ce petit espace de 4 octets est suffisant pour stocker un handle de fichier, qui est typiquement un pointeur utilisé par le système pour accéder aux ressources du système de fichiers.

    (voir Q2 : https://stackoverflow.com/questions/76059934/masm-x86-in-assembly-what-is-the-difference-between-heap-handle-and-pointer-t)
3. **new PATH (260o)**:
    - Un second espace de 260 octets est réservé pour construire de nouveaux chemins lors de la navigation dans les sous-dossiers. Ce nouvel espace est utilisé pour combiner le chemin actuel avec des noms de sous-dossiers ou fichiers obtenus lors de la traversée.

    La somme de tous ces offsets donne 852.

Conclusion

Nous n’avons pas eu beaucoup de temps pour nous attardé sur la partie interface graphique, ce qui implique qu’elle n’est pas fonctionnelle.

Cependant, ce projet nous a permis comprendre en profondeur le fonctionnement des appels système Windows et la programmation en assembleur, offrant une

perspective pratique sur la programmation de bas niveau.