4. Fichiers

4.1 Nom de fichiers sous Win32

- Le chemin complet d'un fichier commence par un lecteur
- Un double backslash indique la racine globale suivie d'un nom de machine et d'un nom de partage pour indiquer un chemin sur un serveur de fichiers de réseau : \\servername\sharename
- Le séparateur dans le chemin est backslash (\), bien que le slash (/) puisse être utilisé dans les paramètres de l'API.
- Les noms des répertoires et des fichiers ne peuvent contenir des caractères ASCII entre 1 et 31 et des caractères du type : < > : « |
- Les noms peuvent contenir des blancs, mais leur utilisation n'est pas recommandée.
- Les noms de répertoires et de fichiers sont « case-insensitive », mais sont « case-retaining »
- Une période . sépare un nom de fichier de son extension. L'extension indique souvent le type de fichier. (ex. .exe pour les exécutables,..).
- . et .., en noms de répertoire, indique le répertoire courant et le répertoire père.

4.2 Gestion des fichiers

- Créer et ouvrir un fichier : CreateFile(...)
- Fermer un fichier : BOOL CloseHandle(HANDLE hObject)
- Lire un fichier
- Ecrire dans un fichier

4.3 Création et Ouverture de fichier

- *HANDLE CreateFile*(*LPCTSTR* lpszName, *DWORD* fwdAccess, *DWORD* fwdShareMode, *LPSECURITY_ATTRIBUTES* lpsa, *DWORD* fdwCreate, *DWORD* fdwAttrsAndFlags, *HANDLE* hTemplateFile)
- Retourne un Handle sur un objet fichier ouvert ou INVALID_HANDLE_VALUE en cas d'échec.
- *DWORD* fdwAccess : est un combinaison de bits gérant l'accès en lecture ou en écriture selon la combinaison GENERIC_READ et/ou GENERIC_WRITE.
- *DWORD* fdwShareMode : est une combinaison de bits telle que
  - 0 : le fichier ne peut être partagé (aucun processus à commencer par le courant ne peut ouvrir un second HANDLE sur ce fichier)
  - FILE_SHARE_READ : d'autres processus, y compris le courant peuvent ce fichier pour des lectures concurrentes.
  - FILE_SHARE_WRITE : d'autres processus, y compris le courant peuvent ce fichier pour des écritures concurrentes.
- *LPSECURITY_ATTRIBUTES* lpsa : pointe sur une structure SECURITY_ATTRIBUTES. Nous utiliserons pour l'instant NULL (structure SECURITY_ATTRIBUTES par défaut).

- *DWORD* fdwCreate : est un combinaison de bits gérant le comportement de la fonction telle que
  - CREATE_NEW : échoue si le fichier existe déjà, sinon crée un nouveau fichier.
  - CREATE_ALWAYS : un fichier existant sera écrasé
  - OPEN_EXISTING : échoue si le fichier n'existe pas
  - OPEN_ALWAYS : Ouvre le fichier et le crée si celui n'existe pas
  - TRUNCATE_EXISTING : La longueur du fichier est mise à zéro. (fdwShareMode GENERIC_WRITE, accès en écriture)
- *DWORD* fdwAttrsAndFlags : indique des attributs du fichier et des flags (il y a 16 flags et attributs), les plus importants sont :
  - FILE_ATTRIBUTE_NORMAL : cet attribut est utilisé seul (les flags peuvent être modifiés).
  - FILE_ATTRIBUTE_READONLY : l'application ne peut ni écrire, ni modifier le fichier.
  - FILE_FLAG_DELETE_ON_CLOSE : très utile pour les fichiers temporaires. Le fichier est effacé quand le dernier HANDLE ouvert est fermé.
  - FILE_FLAG_OVERLAPPED : Ce flag est important pour les entrées/sorties asynchrones (non supporté sous Windows 95).
- *HANDLE* hTemplateFile : est le HANDLE d'un fichier ouvert en GENERIC_READ mode qui indique les attributs à appliquer au nouveau fichier créé, ignorant ainsi fdwAttrsAndFlags. Il permet de faire correspondre les attributs d'un nouveau fichier avec ceux d'un fichier existant. Généralement, ce paramètre est NULL.

4.4 Exemple d'ouverture de fichier

```
int main(
{
HANDLE hIn ;
   /* ouverture du fichier C:\ */
   hIn = CreateFile ("C:\TOTO", GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);

   if (hIn == INVALID_HANDLE_VALUE) {
        printf ("impossible d'ouvrir le fichier TOTO \n");
        return 2;
   }
```

4.5 Exemple de création de fichier

```
int main(
{
HANDLE hourt ;
   hOut = CreateFile (argv [2], GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
   if (hOut == INVALID_HANDLE_VALUE) {
   printf ("Impossible d'ouvrir le fichier de sortie, erreur : \n");
   return 3;
   }
```

4.6 Fermeture de Fichier

- *BOOL* CloseHandle(*HANDLE* hObject)
- retourne TRUE si fermeture réussi, FALSE sinon
- exemple : CloseHandle (hOut);

4.7 Lecture dans un fichier

- *BOOL* ReadFile(*HANDLE* hFile, *LPVOID* lpBuffer, *DWORD* nNumberOfBytesToRead, *LPDWORD* lpNumberOfBytesRead, *LPOVERLAPPED* lpOverlapped);
- Retourne TRUE si la lecture a réussi (même si aucun octet n'a été lu à cause d'une tentative de lecture au-delà de la fin de fichier)
- *HANDLE* hFile : est le Handle du fichier ouvert avec l'accès GENERIC_READ
- *LPVOID* lpBuffer : pointe sur le buffer mémoire qui recevra les données d'entrée.
- *DWORD* nNumberOfBytesToRead : est le nombre d'octets lus dans le fichier.
- *LPDWORD* lpNumberOfBytesRead : pointe sur le nombre d'octets réellement lus à l'appel de ReadFile. (Cette valeur peut être zéro si le HANDLE est positionné en fin de fichier)
- *LPOVERLAPPED* lpOverlapped : pointe sur une structure OVERLAPPED (lecture asynchrone), utiliser NULL.

4.8 Exemple de lecture dans un fichier

```
bResult = ReadFile(hFile, &inBuffer, nBytesToRead, &nBytesRead,
NULL) ;
// Test de fin de fichier
if (bResult &&  nBytesRead == 0, )
{
    // nous sommes en fin de fichier
}
```

4.9 Ecriture dans un fichier

- BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)
- Retourne TRUE si l'écriture a réussi, FALSE sinon.

- HANDLE hFile : : est le Handle du fichier ouvert avec l'accès GENERIC_WRITE
- LPCVOID lpBuffer : pointe sur le buffer mémoire qui recevra les données.
- DWORD nNumberOfBytesToWrite : est le nombre d'octets écrits dans le fichier.
- LPDWORD lpNumberOfBytesWritten : pointe sur le nombre d'octets réellement écrits à l'appel de WriteFile.
- LPOVERLAPPED lpOverlapped : pointe sur une structure OVERLAPPED (lecture asynchrone), utiliser NULL.

4.10 Exemple d'écriture dans un fichier

```
WriteFile (hOut, CBuffer, nIn, &nOut, NULL);
if (nIn != nOut) {
        printf ("erreur d'écriture \n");
        return 4;
        }
```

5. Fichiers d’entrées/sorties standard

5.1 Ouverture d’un fichier d’entrées/sorties standard

5.2 Redirection d’un fichier d’entrées/sorties standard

5. Fichiers d'entrées/sorties standard

- Comme pour Unix, Win32 gère trois fichiers d'entrées/sorties standards : fichier d'entrée standard, fichier de sortie standard, fichier de messages d'erreur. Contrairement à UNIX, Win32 nécessite l'utilisation de HANDLEs sur ces fichiers standards.

5.1 Ouverture d'un fichier d'entrées/sorties standard

- HANDLE GetStdHandle(DWORD nStdHandle);
- Retourne un HANDLE valide, sinon INVALID_HANDLE_VALUE.
- DWORD nStdHandle : peut prendre trois valeurs, STD_INPUT_HANDLE, STD_OUTPUT_HANDLE, STD_ERROR_HANDLE.
- Les entrées/sorties standards correspondent normalement à la console (en sortie) et au clavier (en entrée), mais peuvent être redirigées.
- GetStdHandle ne crée pas de nouveau handle sur le device standard et ne duplique pas le handle. Plusieurs appels avec le même argument retourneront le même handle.

- Fermer le handle d'un fichier d'entrées/sorties standard, le rende indisponible par la suite (souvent le handle correspondant n'est donc pas fermé).

5.2 Redirection d'un fichier d'entrées/sorties standard

- HANDLE SetStdHandle(DWORD nStdHandle, HANDLE hHandle);
- Retourne TRUE, sinon FALSE en cas d'échec
- DWORD nStdHandle : peut prendre trois valeurs, STD_INPUT_HANDLE, STD_OUTPUT_HANDLE, STD_ERROR_HANDLE.
- HANDLE hHandle : le HANDLE du fichier de redirection
- La méthode générale pour rediriger un fichier d'entrées/sorties standard dans un processus utilise SetStdHandle suivie de GetStdHandle.
- Deux noms de chemin réservés existe pour les entrées/sorties de la console : "CONIN$" et "CONOUT$". Il est donc possible d'utiliser la console en ouvrant des handles sur "CONIN$" et "CONOUT$".

6. Entrées/sorties Console

- 6.1 Lecture Console
- 6.2 Ecriture Console
- 6.3 Processus et console

6. Entrées/sorties Console

- Nous pouvons utiliser ReadFile et WriteFile pour des entrées/sorties sur la console.
- Il existe néanmoins des fonctions d'entrées/sorties spécifiques : ReadConsole et WriteConsole. Elles ont pour avantage de traiter des caractères génériques (TCHAR) plutôt que des octets et selon un mode de console établit par la fonction SetConsoleMode.
- BOOL SetConsoleMode(HANDLE hConsoleHandle, DWORD fdevMode)
- Retourne TRUE, sinon FALSE en cas d'échec
- HANDLE hConsoleHandle : indique un HANDLE sur l'entrée ou la sortie de la console ("CONIN$" et "CONOUT$").
- DWORD fdevMode : gère le traitement des caractères
  - ENABLE_LINE_INPUT : la fonction de lecture console (ReadConsole) renvoie la ligne quand un retour chariot est rencontré
  - ENABLE_ ECHO_INPUT : les caractères sont affichés à l'écran en même temps qu'ils sont lus
  - ENABLE_PROCESSED_INPUT : ce flag configure le système pour traiter les caractères, d'effacement, de retour chariot, de retour à la ligne.
  - ENABLE_PROCESSED_OUTPUT : ce flag configure le système pour traiter les caractères, de tabulation, de sonnerie, d'effacement, de retour chariot, de retour à la ligne.

6.1 Lecture Console

- BOOL ReadConsole(HANDLE hConsoleInput, LPVOID lpBuffer, DWORD nNumberOfCharsToRead, LPDWORD lpNumberOfCharsRead, LPVOID lpReserved)
- Fonction similaire à ReadFile
- Les deux paramètres nNumberOfCharsToRead et lpNumberOfCharsRead exprime des quantités de caractères génériques et non d'octets
- LpReserved doit être NULL (dans la documentation Microsoft ! !), inutilisé ou non documenté ?

6.2 Ecriture Console

- BOOL WriteConsole(HANDLE hConsoleOutput, CONST VOID *lpBuffer, DWORD nNumberOfCharsToWrite, LPDWORD lpNumberOfCharsWritten, LPVOID lpReserved)
- Fonction similaire à WriteFile
- Les deux paramètres nNumberOfCharsToWrite et lpNumberOfCharsWritten exprime des quantités de caractères génériques et non d'octets
- LpReserved doit être NULL ...

6.3 Processus et console

- Un processus n'a qu'une console à la fois
- Certaines applications n'ont pas de console par défaut comme les applications GUI Win32
- BOOL AllocConsole(VOID), crée alors une nouvelle console associée au processus courant (échoue si le processus a déjà une console)
- BOOL FreeConsole(VOID), détache un processus de sa console.

7. Gestion des erreurs

- DWORD GetLastError( ) : retourne la valeur du code d'erreur du dernier appel à une fonction de l'API Win32.
- L'appel à une fonction plutôt qu'un code erreur global comme errno d'Unix, permet la gestion des erreurs entre plusieurs threads.
- La fonction FormatMessage convertit le numéro d'erreur en un message explicite.
- DWORD FormatMessage(DWORD dwFlags, LPCVOID lpSource, DWORD dwMessageId, DWORD dwLanguageId, LPTSTR lpBuffer, DWORD nSize, va_list *Arguments)
- Retourne la longueur du message (en octets ou en caractère UNICODE, selon le mode utilisé)
- En ce qui concerne les paramètres (voir le manuel en ligne de Microsoft)
- Exemple : FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM, NULL, GetLastError(),LANG_NEUTRAL, (LPTSTR) &LpMsgErr,0, NULL);

8. Gestion des répertoires

- 8.1 Créer un répertoire
- 8.2 Effacer un répertoire
- 8.3 Récupérer le répertoire Courant
- 8.4 Positionner le répertoire Courant
- 8.5 Exemple

8. Gestion des répertoires

- Chaque processus a un répertoire courant (hérité du processus père)
- Les fonctions de manipulation des répertoires sont GetCurrentDirectory, SetCurrentDirectory, CreateDirectory, RemoveDirectory.

8.1 Créer un répertoire

- BOOL CreateDirectory(LPCTSTR lpPathName, LPSECURITY_ATTRIBUTES lpSecurityAttributes)
- Retourne TRUE, sinon FALSE en cas d'échec (utiliser alors GetLastError())
- LPCTSTR lpPathName : pointe sur une chaîne de caractères indiquant le chemin du répertoire à créer (une longueur limite par défaut est de MAX_PATH caractères)
- LPSECURITY_ATTRIBUTES lpSecurityAttributes : pointe sur une structure SECURITY_ATTRIBUTES, NULL pour le moment.

8.2 Effacer un répertoire

- BOOL RemoveDiretory(LPCTSTR lpPathName)
- Retourne TRUE, sinon FALSE en cas d'échec (utiliser alors GetLastError())
- LPCTSTR lpPathName : pointe sur une chaîne de caractères indiquant le chemin du répertoire à effacer

8.3 Récupérer le répertoire Courant

- DWORD GetCurrentDirectory(DWORD nBufferLength, LPTSTR lpBuffer)
- Retourne le nombre de caractères écrits dans le buffer (sans le caractère NULL de fin ce chaîne), sinon 0 en cas d'échec (utiliser alors GetLastError()) (ex. cas ou le buffer n'est pas assez grand).
- DWORD nBufferLength : indique la longueur du buffer en caractères pour recueillir la chaîne du répertoire courant (incluant le caractère NULL de terminaisons de chaîne)
- LPTSTR lpBuffer : pointe sur le buffer de la chaîne du répertoire courant. Cette chaîne est un chemin absolu.

8.4 Positionner le répertoire Courant

- BOOL SetCurrentDirectory(LPCTSTR lpPathName);
- Retourne TRUE, sinon FALSE en cas d'échec (utiliser alors GetLastError())
- LPCTSTR lpPathName : pointe sur le buffer de la chaîne du nouveau répertoire courant. Ce chemin peut être absolu ou relatif. Dans tous les cas le chemin absolu est déduit et indique le nouveau répertoire courant.

8.5 Exemple

```
int _tmain (int argc, LPTSTR argv [])
{

   TCHAR pwdBuffer [DIRNAME_LEN];
   DWORD LenCurDir;
   DWORD MsgLen;

   /* recupere le répertoire courant */
   LenCurDir = GetCurrentDirectory (DIRNAME_LEN, pwdBuffer);
...
}
```

9. Gestion des lecteurs

9.1 Lecteurs logiques disponibles

9.2 Noms des lecteurs logiques disponibles

9.3 Espace disque disponible

9. Gestion des lecteurs

- Ces fonctions de l'API Win32 permettent un certain nombre d'opération sur les disques : les lecteurs disponibles, leur nom, et l'espace disponible.
- Les fonctions de manipulation des lecteurs sont GetLogicalDrives, GetLogicalDrivesStrings, GetDiskFreeSpace.

9.1 Lecteurs logiques disponibles

- DWORD GetLogicalDrives(VOID)
- Retourne un masque de bits représentant les lecteurs de disques courants. Le bit de poids le plus faible (0) correspond au lecteur A, le bit 1 au lecteur B ...etc.

9.2 Noms des lecteurs logiques disponibles

- DWORD GetLogicalDriveStrings(DWORD nBufferLength, LPTSTR lpBuffer);

- Retourne la longueur de la chaîne copiée dans le buffer (caractère NULL de terminaisons de chaîne non compris), sinon 0 en cas d'échec.
- DWORD nBufferLength : indique la taille maximale,,en caractères, du buffer pointé par lpBuffer (sans le caractère NULL de terminaison de chaîne).
- LPTSTR lpBuffer : pointe sur le buffer qui reçoit la liste des chaînes de caractères, une chaîne pour chaque lecteur valide (terminée par un caractère NULL). Un double caractère NULL termine la liste.
- Exemple : c:\<null>d:\<null><null>

9.3 Espace disque disponible

- BOOL GetDiskFreeSpace(LPCTSTR lpRootPathName, LPDWORD lpSectorsPerCluster, LPDWORD lpBytesPerSector, LPDWORD lpNumberOfFreeClusters, LPDWORD lpTotalNumberOfClusters)
- Retourne TRUE, sinon FALSE en cas d'échec (utiliser alors GetLastError())
- LPCTSTR lpRootPathName : pointe sur la chaîne qui indique le répertoire racine du disque concerné. Si NULL, alors il s'agit de la racine du disque courant.
- LPDWORD lpSectorsPerCluster : pointe sur une variable indiquant le nombre de secteurs par cluster.
- LPDWORD lpBytesPerSector : pointe sur une variable indiquant le nombre d'octets par secteur.
- LPDWORD lpNumberOfFreeClusters : pointe sur une variable indiquant le nombre total de clusters libres sur le disque.
- LPDWORD lpTotalNumberOfClusters : pointe sur une variable indiquant le nombre total de clusters sur le disque.

10. Accès direct aux fichiers et attributs de fichiers

10. Accès direct aux fichiers et attributs de fichiers

10.1 Pointeurs de Fichier

- Tout comme UNIX, les bibliothèques C et presque tous les systèmes d'exploitation, Win32 gère un pointeur de fichier sur chaque HANDLE ouvert, indiquant la position de l'octet courant sur le fichier. WriteFile et ReadFile transfèrent alors des données séquentiellement à partir de cette position.
- Ce pointeur est initialisé à zéro (début de fichier) et avance avec les lectures et écritures successives.
- L'opération cruciale dans l'accès direct au fichier est le positionnement de ce pointeur : SetFilePointer sous Win32.
- Cette opération va nous permettre de voir comment Win32 gère l'adressage 64 bis de NTFS.

10.2 Positionnement du pointeur de fichier

- DWORD SetFilePointer(HANDLE hFile, LONG lDistanceToMove, PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod)
- Retourne les 32-bits de poids faible du nouveau pointeur de fichier (les 32-bits de poids fort se trouvent dans le contenu du pointeur lpDistanceToMoveHigh, si non NULL). En cas d'erreur la valeur retournée est 0xFFFFFFFF.
- HANDLE hFile : c'est le HANDLE d'un fichier en lecture et/ou écriture
- LONG lDistanceToMove : en fonction de la valeur de dwMoveMethod, c'est un déplacement de type LONG signé ou une position dans le fichier de type LONG non signé.
- PLONG lpDistanceToMoveHigh : pointe sur les 32-bits de poids fort du déplacement du pointeur de fichier. Si cette valeur est NULL, la fonction ne peut être utilisée que sur des fichiers de longueur limitée à 2^{32}-2 octets.
- DWORD dwMoveMethod : indique un des modes de déplacement suivant :
  - FILE_BEGIN : Position à partir du début du fichier, lDistanceToMove pointe sur une valeur non signée

- FILE_CURRENT : Déplace le pointeur à partir de la position courante, , lDistanceToMove pointe sur une valeur signée
- FILE_END : Déplace le pointeur à partir de la fin de fichier.

♦ Il est possible d'utiliser cette fonction pour obtenir la longueur du fichier (en indiquant un déplacement 0 à partir de la fin de fichier)

10.3 Attributs de fichier et parcours de répertoire

♦ Il est possible de parcourir les fichiers d'un répertoire et d'obtenir leurs attributs

♦ Cette recherche utilise FindFirstFile, FindNextFile et FindClose

♦ Des informations sur un fichier ouvert peuvent aussi être obtenues par la fonction BOOL GetFileInformationByHandle(HANDLE hFile, LPBY_HANDLE_FILE_INFORMATION lpFileInformation) dans la structure BY_HANDLE_FILE_INFORMATION (Cf. Manuel).

10.3.1 Parcours d'un répertoire

- HANDLE FindFirstFile(LPCTSTR lpFileName, LPWIN32_FIND_DATA lpFindFileData)
- Examine les sous répertoires et fichiers, à la recherche du premier nom correspondant à l'expression régulière passée en paramètre.
- Retourne le HANDLE correspondant, sinon INVALID_HANDLE_VALUE en cas d'échec
- LPCTSTR lpFileName : pointe sur un nom de chemin ou répertoire qui contient les caractères ? et * des expressions régulières.
- LPWIN32_FIND_DATA lpFindFileData : pointe sur la structure WIN32_FIND_DATA qui contient des informations sur le fichier ou sous répertoire.

10.3.2 Structure WIN32_FIND_DATA

- typedef struct _WIN32_FIND_DATA {

```
DWORD dwFileAttributes;
FILETIME ftCreationTime;
FILETIME ftLastAccessTime;
FILETIME ftLastWriteTime;
DWORD    nFileSizeHigh;
DWORD    nFileSizeLow;
DWORD    dwReserved0;
DWORD    dwReserved1;
TCHAR    cFileName[ MAX_PATH ];
TCHAR    cAlternateFileName[ 14 ];
} WIN32_FIND_DATA;
```

- DWORD dwFileAttributes : ce sont les attributs du fichier et les flags (il y a 16 flags et attributs) dont nous avons parlé pour CreateFile (ex. la

condition (dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) !=0 permet de détecter si on a trouvé un sous répertoire ou un fichier)

- FILETIME ftCreationTime : date de création
- FILETIME ftLastAccessTime : date du dernier accés
- FILETIME ftLastWriteTime : date de la dernière modification
- DWORD nFileSizeHigh : 32-bits de poids fort de la taille du fichier
- DWORD nFileSizeLow : 32-bits de poids faible de la taille du fichier
- DWORD dwReserved0 : réservé (non documenté)
- DWORD dwReserved1 : réservé (non documenté)
- TCHAR cFileName[MAX_PATH] : nom du fichier
- TCHAR cAlternateFileName[14] : autre nom du fichier (version DOS : 8 caractères et 3 d'extension)

10.3.3 Parcours d'un répertoire (suite)

- BOOL FindNextFile(HANDLE hFindFile, LPWIN32_FIND_DATA lpFindFileData)
- S'utilise à la suite de FindFirstFile (conserve l'expression régulière de recherche FindFirstFile)
- Retourne FALSE en cas d'arguments non valides ou de fichier non trouvé (GetLastError() est alors égal à ERROR_NO_MORE_FILES). Une fois la recherche finie, la fonction ferme le HANDLE de recherche ouvert par FindFirstFile (pas besoin de CloseHandle).
- HANDLE hFindFile : indique le HANDLE de recherche retourné par l'appel précédent a FindFirstFile.
- LPWIN32_FIND_DATA lpFindFileData : pointe sur la structure WIN32_FIND_DATA qui contient des informations sur le fichier ou sous répertoire trouvé.
- Le Handle de recherche peut être fermé (fermeture anticipée) par la fonction BOOL FindClose (HANDLE hFindFile)
- FindClose retourne TRUE, sinon FALSE en cas d'échec

10.3.4 Autres attributs de fichiers et répertoires

- FindFirstFile et FindNextFile permettent d'obtenir certaines informations sur les attributs des fichiers et répertoires.
- D'autres fonctions, incluant celle qui permet de configurer les attributs d'un fichier ou d'un répertoire, traite directement le HANDLE d'un fichier déjà ouvert plutôt que le HANDLE de recherche dans le parcours d'un répertoire.
- Nous pouvons ainsi utiliser GetFileTime, GetFileAttributes, GetTempFileName

10.4 Taille d'un fichier

- DWORD GetFileSize(HANDLE hFile, LPDWORD lpFileSizeHigh)
- Retourne les 32-bits de poids faible de la taille du fichier, sinon 0xFFFFFFFF en cas d'erreur (attention l'erreur ici n'est pas obligatoire si la taille du fichier dépasse 2^{32}-2 octets, notamment si l'adressage 64 bits est utilisé. Nous utiliserons alors GetLastError() pour vérifier NO_ERROR)
- HANDLE hFile : c'est le HANDLE d'un fichier en lecture et/ou écriture
- LPDWORD lpFileSizeHigh : pointe sur les 32-bits de poids fort de la taille du fichier

10.5 Récupérer les dates d'un fichier ou d'un répertoire

- BOOL GetFileTime(HANDLE hFile, LPFILETIME lpCreationTime, LPFILETIME lpLastAccessTime, LPFILETIME lpLastWriteTime)
- HANDLE hFile : HANDLE du fichier ouvert
- LPFILETIME lpCreationTime : date de création
- LPFILETIME lpLastAccessTime : date du dernier accès
- LPFILETIME lpLastWriteTime : date de la dernière modification

10.6 Fonctions de conversion du temps

- Le format des dates de fichier, ici comme dans la structure WIN32_FIND_DATA sont des entiers non signés de 64 bits d'unité 100 nanosecondes exprimant le décalage depuis la date du 1ier janvier 1601 (10^7 unités par seconde)
- Des fonctions telles que FileTimeToSystemTime et SystemTimeToFileTime permettent cette conversion
- BOOL FileTimeToSystemTime (CONST FILETIME *lpFileTime, LPSYSTEMTIME lpSystemTime)
- CONST FILETIME *lpFileTime : pointe sur la date de fichier à convertir
- LPSYSTEMTIME lpSystemTime : pointe sur la structure qui recevra la date système
- Pour information :

```
typedef struct _FILETIME {
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
    } FILETIME;
```

♦ Structure SYSTEMTIME : date système

```
typedef struct _ SYSTEMTIME {
WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

♦ WORD wYear : année en cours

♦ WORD wMonth : le mois en cours (1 pour janvier ...)

♦ WORD wDayOfWeek : le jour de la semaine (0 pour lundi..)

♦ WORD wDay : le jour du mois

♦ WORD wHour : l'heure courante

- WORD wMinute : la minute courante
- WORD wSecond : la seconde courante
- WORD wMilliseconds : la milliseconde courante
- Pour information : GetSystemTime() et SetSystemTime() permettent une manipulation de la date système.
- BOOL SystemTimeToFileTime (CONST SYSTEMTIME *lpSystemTime, LPFILETIME lpFileTime) est la fonction réciproque de FileTimeToSystemTime.

10.7 Récupération des attributs d'un fichier

- DWORD GetFileAttributes(LPCTSTR lpFileName)
- Retourne les attributs du fichier codés (le fdwAttrsAndFlags de CreateFile)
- Les valeurs d'attributs peuvent être : FILE_ATTRIBUTE_DIRECTORY, FILE_ATTRIBUTE_NORMAL, FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_HIDDEN, etc.
- La fonction BOOL SetFileAttributes (LPCTSTR lpFileName, DWORD dwFileAttributes), change les attributs d'un fichier dont le nom est dans la chaîne pointé par lpFileName)

11. Tableaux des Correspondances Win32/ Unix/ C

- 11.1 Entrées/sorties standards
- 11.2 Gestion des messages d'erreurs
- 11.3 Gestion des répertoires
- 11.4 Gestion des fichiers
- 11.5 Gestion des fichiers
- 11.6 Gestion des fichiers

11.1 Tableaux des Correspondances Win32/ Posix / C Ansi : Entrées/sorties standards :

Win32	Unix	Bibliothèque C	Remarques
AllocConsole	Terminal I/O	N/A	
FreeConsole	terminal I/O	N/A	
ReadConsole	read	getc, scanf, gets	
SetConsoleMode	ioctl	N/A	
WriteConsole	write	putc, printf, puts	

11.2 Gestion des répertoires

Win32	Unix	Bibliothèque C	Remarques
CreateDirectory	mkdir	N/A	Make a new directory
FindClose	closedir	N/A	Close a directory search handle
FindFirstFile	opendir, readdir	N/A	Find first file matching a pattern
FindNextFile	readdir	N/A	Find subsequent files
GetCurrentDirectory	getcwd	N/A	
GetFullPathName	N/A	N/A	
GetSystemDirectory	Well-known path names	N/A	
RemoveDirectory	rmdir, unlink	remove	
SearchPath	Use opendir; readdir	N/A	Search for a file on a specified path
SetCurrentDirectory	chdir, fchdir		Change the working directory

11.3 Gestion des messages d'erreurs

Win32	Unix	Bibliothèque C	Remarques
FormatMessage	strerror	perror	
GetLastError	errorno	errno	Use the UNIX or C library global variable
SetLastError	errorno	errno	Set the UNIX or C library global variable

11.4 Gestion des fichiers

Win32	Unix	Bibliothèque C	Remarques
CloseHandle (file handle)	close	fclose	CloseHandle is not limited to files
CopyFile	open; read; write; close	fopen; fread; fwrite; fclose	Duplicate a file
CreateFile	open, creat	fopen	Open/create a file
DeleteFile	unlink	remove	Delete a file
FlushFileBuffers	fsynch	fflush	Write file buffers
GetFileAttributes	stat, fstat, lstat	N/A	
GetFileInformationbyHandle	stat, fstat, lstat	N/A	Fill structure with file info
GetFileSize	stat, fstat, lstat	ftell, fseek	Get length of file in bytes
GetFileTime	stat, fstat, lstat	N/A	
GetFileType	stat	fstat	Check for character stream device or file

11.5 Gestion des fichiers

Win32	Unix	Bibliothèque C	Remarques
GetStdHandle	use file desc 0, 1, or 2	use stdin, stdout, stderr	
GetTempFileName	use C library	tmpnam, mktemp	Create a unique file name
GetTempFileName; CreateFile	use C library	tmpfile	Creates a temporary file
GetTempPath	/temp path	N/A	Directory for temp files
MoveFile	use C library	rename	Rename a file
MoveFile	use C library	rename	Rename a directory
MoveFileEx	use C library	rename	Rename a file
N/A	link, unlink	N/A	Win32 does not support links
N/A	link	N/A	Win32 does not support links
N/A	symlink	N/A	Create a symbolic link
N/A	readlink	N/A	Read name in a symbolic link

11.6 Gestion des fichiers

Win32	Unix	Bibliothèque C	Remarques
N/A, ReadFile returns 0 bytes	N/A, read returns 0 bytes	feof	Test for end of file
N/A, use multiple ReadFiles	readv	N/A, use multiple freads	Scatter read
N/A, use multiple WriteFiles	writev	N/A, use ultiple fwrites	Gather write
ReadFile	read	fread	Read data from a file
SetEndOfFile	truncate, ftruncate	trunctate to 0 only	UNIX commands specify position
SetFileAttributes	fcntl	N/A	
SetFilePointer	lseek	fseek	Get position of file pointer
SetFilePointer (to 0)	lseek (0)	rewind	
SetFileTime	utime	N/A	
SetStdHandle	close, dup, or dup2, or fnctl	freopen	dup2 or fcntl
WriteFile	write	fwrite	Write data to a file