Unit 8: File System

8.5. Windows File and Directory Management

Roadmap for Section 8.5

- Windows File I/O API – General Principles
- Moving and Copying Files
- Directory Management
- Directory Searching
- File and Directory Attributes

Windows API I/O: File and Directory Management

- Windows API provides a number of straightforward functions to manage files
  - Delete, copy, rename files
  - Create temporary file names

```
BOOL DeleteFile (LPCTSTR lpszFileName)
```

- Absolute pathnames start with drive letter or server name!
- It is not possible to delete an open file in Windows
  (but it was possible in Windows 9x)

UNIX unlink() decrements link count
(but does not necessarily delete file)

Moving files

```
BOOL CopyFile( LPCTSTR lpszExistingFile,
               LPCTSTR lpszNewFile,
               BOOL fFailIfExists );
```

- Copies the named existing file and assigns new name
- An existing file will be replaced only if
  fFailIfExists == FALSE
- DeleteFile() and CopyFile() do not work for directories
- Win32 does not support any file linking
  (but NTFS and POSIX subsystem do)

Moving Files (contd.)

```
BOOL MoveFile (LPCTSTR lpszExisting,
          LPCTSTR lpszNew);

BOOL MoveFileEx( LPCTSTR lpszExisting,
          LPCTSTR lpszNew, DWORD fdwFlags);
```

- MoveFile() fails if the new file already exists (use MoveFileEx() for existing files)
  - Windows 9x does not implement MoveFileEx()
  - New files can be on different drives / directories (see flags below)
  - New directories must be on the same drive
  - lpszNew == NULL : existing file is deleted
- fdwFlags:
  - MOVEFILE_REPLACE_EXISTING – replace existing destination file
  - MOVEFILE_COPY_ALLOWED – destination may be on different volume

Directory Management

```
BOOL CreateDirectory( LPCTSTR lpszPath,
            LPSECURITY_ATTRIBUTES lpsa );

BOOL RemoveDirectory( LPCTSTR lpszPath );
```

- lpszPath points to null-terminated string with the name of the target directory
  - Only an empty directory can be removed
  - lpsa == NULL will create a null-ACL for the new directory

Directory Management (contd.)

```
BOOL SetCurrentDirectory( LPCTSTR lpszCurDir );

DWORD GetCurrentDirectory( DWORD chCurDir,
          LPTSTR lpszCurDir  );
```

- Each process has a current working directory
  - For each individual drive it keeps a working directory
- GetCurrentDirectory:
  - chCurDir is size of buffer in characters (!)
  - If buffer is too small: GetCurrentDirectory() returns required size (!) or zero on failure
  - Call GetCurrentDirectory twice: first to obtain size of buffer, next to obtain the desired value
    (or use MAX_PATH constant)

Directory Searching

```
HANDLE FindFirstFile( LPCTSTR lpszSearchFile,
        LPWIN32_FIND_DATA lpffd );
```

- Search a directory for files that satisfy a specified name pattern
  - Search handles must be obtained via FindFirstFile() and closed via FindClose()
  - FindFirstFile() examines subdirectories and file names
  - Return of INVALID_HANDLE_VALUE indicates failure
- Parameters:
  - lpszSearchFile points to directory/pathname that can contain wildcard characters (? and *; no regular expressions)
  - lpffd points to data structure with access information

WIN32_FIND_DATA structure

FILE_ATTRIBUTE_ARCHIVE, FILE_ATTRIBUTE_COMPRESSED, FILE_ATTRIBUTE_DIRECTORY, FILE_ATTRIBUTE_ENCRYPTED, FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_NORMAL, FILE_ATTRIBUTE_OFFLINE, FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_REPARSE_POINT, FILE_ATTRIBUTE_SPARSE_FILE, FILE_ATTRIBUTE_SYSTEM, FILE_ATTRIBUTE_TEMPORARY

```
typedef struct _WIN32_FIND_DATA { // wfd

    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;
```

Does not contain path-portion of name

DOS 8.3 name

Directory Searching (contd.)

```
BOOL FindNextFile( HANDLE hFindFile,
           LPWIN32_FIND_DATA lpffd );

BOOL FindClose( HANDLE hFindFile );
```

- FindNextFile() returns FALSE in case of invalid arguments or if no more matching files are found
  - GetLastError() returns ERROR_NO_MORE_FILES
- Use FindClose() to close search handle
  - CloseHandle() will raise an exception
- GetFileInformationByHandle() obtains same info...
- Programs must do wildcard expansion on their own
  - MS-DOS shell (cmd.exe) does not expand wildcards (sh.exe does)

More File and Directory Attributes

```
BOOL GetFileTime( HANDLE hFiles,
            LPFILETIME lpftCreation,
            LPFILETIME lpftLastAccess,
            LPFILETIME lpftLastWrite );
```

- File times are 64-bit unsigned integers
  (time, expressed in 100 nanoseconds units, since January 1, 1601)
- FileTimeToSystemTime() / SystemTimeToFileTime()

  convert into years down to milliseconds (and vice versa)
- CompareFileTime(), SetFileTime()
- NTFS supports all three file times (creation time, modification time and last access time)
- FAT is accurate only for last access time

File Attributes (contd.)

```
DWORD GetFileAttributes( LPCTSTR lpszFileName )
```

- Returns file attribute or 0xFFFFFFFF in case of failure
- Attributes include:
  - FILE_ATTRIBUTE_DIRECTORY
  - FILE_ATTRIBUTE_NORMAL
  - FILE_ATTRIBUTE_READONLY
  - FILE_ATTRIBUTE_TEMPORARY
- SetFileAttribute() changes those attributes for a file

File Flags - controlling read-ahead

- Cache Manager (CM) performs read-ahead and write-back
  - Reading the next block during sequential access
  - Asynchonous read-ahead with history for strided access
    - history of the last two read requests
    - If a pattern can be determined, cache manager extrapolates it
  - Cache Manager un-maps cached data according to access scheme
- File flags can be specified when opening a file with CreateFile()
  - FILE_FLAG_SEQUENTIAL_SCAN
    - Instructs Cache Manager to perform sequential read-ahead
  - FILE_FLAG_ RANDOM_ACCESS
    - Instructs Cache Manager not to perform read-ahead
  - FILE_FLAG_NO_BUFFERING
    - Cache Manager shall not be involved with I/O on this file

Further Reading

- Mark E. Russinovich, David A. Solomon, and Alex Ionescu, “*Windows Internals*”, 6th Edition, Microsoft Press, 2012.
  - Chapter 12 – File Systems (from pp. 391)
    - NTFS On-Disk Structure (from pp. 442)
    - File System Operation (from pp. 407)
  - Chapter 11 – Cache Manager (from pp. 355)
    - File System Interfaces (from pp. 373)

  Remark: these chapters will be in part 2 of 7th edition!
- Jeffrey Richter, “*Advanced Windows*”, 3rd Edition, Microsoft Press, September 1997.
  - Chapter 14 – File Systems
    - Directory Operations (from pp. 637)
- Johnson M. Hart, “*Windows System Programming*”, 3rd Edition, Addison-Wesley, 2004.