File Management

Mario Tayah and Jim Fawcett

CSE775 – Distributed Objects

Spring 2007, Revised Spring 2012

Introduction

- A file is a set of data which has been organized, stored and named.
- Files are used for:
  - Long term storage of data and programs.
  - Form of program to program communication.
- A file system is a system for organizing directories and files, generally in terms of how it is implemented in the disk operating system.
- A file system usually holds an interface to allow the user to:
  - Add
  - Edit
  - Remove

  Any file or directory.
- Some of the common file systems are:
  - FAT and NTFS on Windows Systems
  - UFS and JFS on Unix Systems.

Window File System

- Windows has four main file systems:
  - NTFS file system, most recent window's file system includes security, long names…

    note that this system is not supported on diskettes as well as all windows 9X OS.
  - FAT and FAT32 no support for windows security supported on diskettes as well as windows 9X OS.
  - CDFS used to access information stored on CD.
  - UDF support for DVD reading.
  - Other file systems include: NFS, SAN, CIFS
  - Throughout this presentation we will be focusing on the NTFS file system.

File Management Operations

- The file system supports functions that allow you to perform functionality in four major categories:
  - Creating, Deleting, and Maintaining Files
  - Reading From and Writing to Files
  - Obtaining and Setting File Information
  - Reading and setting Security and Access Rights
  - File and Directory Linking
  - File Compression and Decompression
  - File Encryption
  - Sparse Files

Creating, Deleting, Maintaining Files

- Win32 Functionality:
  - Naming a File
  - Creating and Opening Files
  - Creating and Using a Temporary File
  - Moving and Replacing Files
  - Closing Files
  - Deleting Files
  - Defragmenting Files

File naming

- Filespecification in windows:
  - A file path starts either with a disc drive name(c:, d:…) or with (\\) indicating the global root on the machine.
  - The path seperator character is the "\"although in some APIs it is the /.
  - Directory and file names cannot contain any of the ASCII characters that fall in the range 1-31, including mainly: < > : " | ? * \ /
  - Directory as well as file names can have spaces but, in accessing them you should inclose the space seperated name with "".
  - Can be as long as 255 characters
  - "." Usually separates the extension from the filename but, a filename can hold many "."
  - Moreover, in the path, a "." indicates the current directory while a ".." the upper/parent directory.

Creating & Opening Files

- You use the "CreateFile" function to open an already existing file or creating a new one.
- To the right are two examples:
  - Upper one, open a file for writing.
  - Lower one, create a new file for reading.

```
#include <windows.h>
#include <stdio.h>

HANDLE hFile;

hFile = CreateFile(TEXT("myfile.txt"),     // file to create
                   GENERIC_WRITE,          // open for writing
                   0,                      // do not share
                   NULL,                   // default security
                   CREATE_ALWAYS,          // overwrite existing
                   FILE_ATTRIBUTE_NORMAL | // normal file
                   FILE_FLAG_OVERLAPPED,   // asynchronous I/O
                   NULL);                  // no attr. template

if (hFile == INVALID_HANDLE_VALUE)
{
    printf("Could not open file (error %d)\n", GetLastError());
    return 0;
}
```

```
#include <windows.h>
#include <stdio.h>

HANDLE hFile;

hFile = CreateFile(TEXT("myfile.txt"),    // file to open
                   GENERIC_READ,          // open for reading
                   FILE_SHARE_READ,       // share for reading
                   NULL,                  // default security
                   OPEN_EXISTING,         // existing file only
                   FILE_ATTRIBUTE_NORMAL, // normal file
                   NULL);                 // no attr. template

if (hFile == INVALID_HANDLE_VALUE)
{
    printf("Could not open file (error %d)\n", GetLastError());
    return 0;
}
```

Creating and Using Temporary Files

- The windows file system interface provides functions to allow Applications to use temporary files by the following functions:
  - GetTempFileName : Creates a name for a temporary file. If a unique file name is generated, an empty file is created and the handle to it is released; otherwise, only a file name is generated.
  - GetTempPath: retrieves the path to the directory where temporary files should be created.
- Find to the right an illustrative code fragment.

```
// Create a temporary file.
uRetVal = GetTempFileName(lpPathBuffer, // directory for tmp files
                          "NEW",        // temp file name prefix
                          0,            // create unique name
                          szTempName);  // buffer for name
if (uRetVal == 0)
{
    printf ("GetTempFileName failed with error %d.\n",
            GetLastError());
    return (3);
}

// Create the new file to write the upper-case version to.
hTempFile = CreateFile((LPTSTR) szTempName, // file name
                       GENERIC_READ | GENERIC_WRITE, // open r-w
                       0,                    // do not share
                       NULL,                 // default security
                       CREATE_ALWAYS,        // overwrite existing
                       FILE_ATTRIBUTE_NORMAL,// normal file
                       NULL);                // no template
if (hTempFile == INVALID_HANDLE_VALUE)
{
    printf ("CreateFile failed with error %d.\n",
            GetLastError());
    return (4);
}
```

Moving & Replacing Files

- **To copy:**
  - Before a file can be copied, it must be closed or opened only for reading. No thread can have the file opened for writing. To copy an existing file to a new one, use the CopyFile or CopyFileEx function. Applications can specify whether CopyFile and CopyFileEx fail if the destination file already exists.
- **To replace:**
  - The ReplaceFile function replaces one file with another file, with the option of creating a backup copy of the original file.
- **To move:**
  - A file must also be closed before an application can move it. The MoveFile and MoveFileEx functions copy an existing file to a new location and deletes the original.
  - The MoveFileEx function also allows an application to specify how to move the file. The function can replace an existing file, move a file across volumes, and delay moving the file until the operating system is restarted.

Closing & deleting Files

- **To close:**
  - To use operating system resources efficiently, an application should close files when they are no longer needed by using the CloseHandle function. If a file is open when an application terminates, the system closes it automatically.
  - The following codes closes the file named Myfile.txt, whose handle is stored in the hFile variable:

    CloseHandle(hFile);
  - Note that closing a file does not delete the file from disk.

- **To delete:**
  - The DeleteFile function can be used to delete a file on close. A file cannot be deleted until all handles to it are closed. If a file cannot be deleted, its name cannot be reused. To reuse a file name immediately, rename the existing file.
  - The following codes closes and deletes the file named Myfile.txt, whose handle is stored in the hFile variable:

    CloseHandle(hFile);
    DeleteFile(TEXT("myfile.txt"));

Defragmenting Files

- When a file is written to a disk, the file cannot be written in contiguous clusters. Noncontiguous clusters slow down the process of reading and writing a file.
- To optimize files for fast access, a volume can be defragmented.
- The process of defragmentation simply moves fragments of the file to try to make them as close to each other as possible if not contigous to each other → allowing faster access
- The following are the steps to perform defragmentation of a file:
- Use the FSCTL_GET_VOLUME_BITMAP control code to find a place on the volume that is large enough to accept an entire file.
    - Note If necessary, move other files to make a place that is large enough. Ideally, there is enough unallocated clusters after the first extent of the file that you can move subsequent extents into the space after the first extent.
  - Use the FSCTL_GET_RETRIEVAL_POINTERS control code to get a map of the current layout of the file on the disk.
  - Walk the RETRIEVAL_POINTERS_BUFFER structure returned by FSCTL_GET_RETRIEVAL_POINTERS.
  - Use the FSCTL_MOVE_FILE control code to move each cluster as you walk the structure.

- So, as you can see from the steps indicated above, the process of defragmenting constitutes finding the bits and pieces of the file and trying to put them next to each other

Obtaining and Setting File Information

- Win32 Functionality:
  - Retrieving and Changing File Attributes
  - Retrieving File Type Information
  - Determining the Size of a File
  - Testing for the End of a File
  - Searching for One or More Files
  - Setting and Getting the Timestamp of a File
  - Determining the Current Character Set Code Page

Retrieving and Changing File Attributes

- To get the file attributes you can use:
  - **GetFileAttributes**
  - **GetFileAttributesEx**
- To set file attributes use:
  - **CreateFile**
  - **SetFileAttributes**

Note that applications cannot set all the file attributes.

Retrieving File Type & size Information

- In order to get the file type information use:
  - **GetFileType :** which retrieves the type of a file: disk, character (such as a console), pipe, or unknown.
  - **GetBinaryType:** which determines whether a file is executable, and if so, the type of executable file it is.
- In order to determine the size of a file use:
  - **GetFileSize:** which retrieves the size of a file in bytes.

Testing for "End of a File"

- The **ReadFile** function checks for the end-of-file condition (eof) differently for synchronous and asynchronous read operations as follows:
  - Synchronous: the synchronous read operation gets to the end of a file, ReadFile returns TRUE, and sets the variable pointed to by lpNumberOfBytesRead to 0 (zero).
  - Asynchronous: An asynchronous read operation can encounter the end of a file during the initiating call to ReadFile, or during subsequent asynchronous operation.
- The code fragment on the right shows how to check for the end of file

```
// Attempt a synchronous read operation
bResult = ReadFile(hFile, &inBuffer, nBytesToRead, &nBytesRead, NULL);

// Check for eof
if (bResult &&  nBytesRead == 0, )
{
    // At the end of the file
}
```

Searching for One or More Files

- An application can search the current directory for all file names that match a given pattern by using the following:
  - FindFirstFile
  - FindFirstFileEx
  - FindNextFile
  - FindClose

  Note that the pattern must be a valid file name and can include wildcard characters.

Setting/Getting Timestamp & Character Set Code

- **Timestamp:**
  - Applications can retrieve and set the date and time a file is created, last modified, or last accessed by using:
    - GetFileTime
    - SetFileTime
- **Character Set Code**:
  - in order to access/set the character set code use:
    - AreFileApisANSI which determines whether the file I/O functions are using the ANSI or OEM character set code page.
    - SetFileApisToANSI which causes the functions to use the ANSI code page.
    - SetFileApisToOEM which causes the functions to use the OEM code page.

Read/Write To files

- The windows file system provides the following functions to allow application to read and write to files:
  - **ReadFile**
  - **ReadFileEx**
  - **WriteFile**
  - **WriteFileEx**
- In order to read/write to a file, you need to hold a handle to that file, a handle can be defined to provide reading capability or/and writing capability.
- These functions read and write a specified number of bytes at the location indicated by the file pointer.
- When the file pointer reaches the end of a file and the application attempts to read from the file, no error occurs, but no bytes are read.

Note that these functions do not provide any formatting.

File Security & Access Rights

- Windows has the notion of securable objects which are objects that are secured by the operating system→ the operating system through a module named "Access Control " says weather a certain application/user is eligible for accessing a certain resource.
- Files are one of the securable objects in NTFS→ access to them is also secured through the "Access Control" module.
- The access control reads security descriptors defined on certain resources, these security descriptors specify who is eligible to access/use this resource and in what way.
- You can specify the security descriptor for a file through:
  - CreateFile, CreateDirectory, or CreateDirectoryEx at creation time.
- If you specify NULL for the *lpSecurityAttributes* parameter, the file or directory gets a default security descriptor → inherits its parent security descriptor.
- To retrieve the security descriptor of a file or directory object, use:
  - GetNamedSecurityInfo or GetSecurityInfo
- To change the security descriptor of a file or directory object use:
  - SetNamedSecurityInfo or SetSecurityInfofunction.

File Encryption

- NTFS provides an additional layer of file protection which is the Encrypted File System, or EFS.
- EFS provides cryptographic protection of individual files on NTFS file system volumes using a public-key system.

File Compression and Decompression

- The NTFS file system volumes support file compression on an individual file basis.
- NTFS file system uses the "Lempel-Ziv compression" which is lossless→ no data is loss in the compression process.
- On the NTFS file system, compression is performed transparently. This means it can be used without requiring changes to existing applications. The compressed bytes of the file are not accessible to applications→ the application does not deal with compressed files, it only deals with the uncompressed files.
- The NTFS file system provides functions to provide the following main compression/decompression processes:

  - Decompressing multiple files
  - Decompressing multiple files
  - Reading from compressed files

Decompressing a Single File

- An application can decompress a single compressed file by performing the following tasks :
  - Open the source file by calling the LZOpenFile function.
  - Open the destination file by calling LZOpenFile.
  - Copy the source file to the destination file by calling the LZCopy function and passing the handles returned by LZOpenFile.
  - Close the files by calling the LZClose function.

Decompressing multiple files

- An application can decompress multiple files by performing the following tasks :
  - Open the source files by calling the LZOpenFile function.
  - Open the destination files by calling LZOpenFile.
  - Copy the source files to the destination files by calling the LZCopy function.
  - Close the files by calling the LZClose function.

Reading from Compressed Files

- More complex operation for the compressed file manipulation is available :
  - an application can decompress a compressed file a portion at a time by using the LZSeek and LZRead functions.

    this is useful when it is necessary to extract parts of large files. For example, a font manufacturer may have compressed files containing font metrics in addition to character data.
  - To use the information in these files, an application would need to decompress the file; however, most applications would use only part of the file at any particular time.
  - To get information about font metrics, the application would extract data from the header.
  - To get information from the text, the application would reposition the file pointer by calling LZSeek and extract character data by calling LZRead.

Sparse Files

- Definition:
  - A file in which much of the data is zeros is said to contain a sparse data set
- How does it work:
  - Support for sparse files is introduced in the NTFS file system as a way to make the disk space usage more efficient.
  - When the sparse file functionality is enabled, the system does not allocate hard drive space to a file except in regions where it contains nonzero (useful) data.
  - When a write operation is attempted where a large amount of the data in the buffer is zeros, the zeros are not written to the file. Instead, the file system creates an internal list containing the locations of the zeros in the file, and this list is consulted during all read operations.
  - When a read operation is performed in areas of the file where zeros were located, the file system returns the appropriate number of zeros in the buffer allocated for the read operation. In this way, maintenance of the sparse file is transparent to all processes that access it.
  - You can see, that this process, increases the save and read operations.
  - Note that the default data value of a sparse file is zero; however, it can be set to other values.

File and Directory Linking

- Definition : create a system representation of a file or directory in a location in the directory structure that is different from the file or directory object that is being linked to (similar to virtual directory).
- There are two types of links supported in the NTFS file system:
  - hard links
  - junctions.
- The NTFS file system also provides the distributed link tracking service, which automatically tracks links as they are moved so the link won’t get broken.

File Management References

- File Management Functions
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_management_functions.asp
- File Management Control Codes
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_management_control_codes.asp
- File Management Structures
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_management_structures.asp
- File Management Enumerations
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_management_enumerations.asp

End of Presentation