Reading Contents

Topic 1: Windows Operating System

Software can be categorized into two main types: **System Software** and **Application Software**. System Software is concerned to the system or its resources while Application Software is related to some application. Operating System is one of the good examples of System software that acts as a manager for the System's resources. It arbitrates and schedules the resources among the processes to avoid any kind of conflicts.
There are varieties of Operating Systems available in the market but Windows is one of the important Operating Systems developed by Microsoft. It is widely used inside the PCs, Laptops, enterprise servers, handheld devices and cell phones etc.

Core Features of Windows Operating System

1. **Memory Management**. Memory Management is the vital or key feature of Windows Operating System. The OS is responsible for managing primary as well as secondary memory. Virtual address space on secondary memory is also managed by OS to transfer data between primary and secondary storage and accommodate a large process in small memory space. All kinds of services and related data structures are supported by Operating Systems to efficiently manage the Virtual memory space.
2. **File systems**. Windows OS manages all kinds of files and folders on disk in hierarchical form. Large file naming space of 255 characters is supported by the OS. Number of APIs are provided by the OS to access the files in both Sequential and Random mode.
3. **Processors**. Windows Operating System provides support to multiprocessors and multi cores systems. It also arbitrates among the cores or processors to efficiently divide and allocate computational tasks to them.
4. **Resource naming and location**. In Windows OS, the resources like processes or devices are treated as objects and each object is assigned a unique name to identify, locate and access it.
5. **Multitasking**. Windows OS supports multitasking. It manages processes, threads, and other independent units, and their asynchronous execution. Tasks can be preempted and scheduled according to dynamically calculated priorities.
6. **Communication and Synchronization**. The Windows OS provides constructs to manage inter-process communication and synchronization within a computer or networked computers.
7. **Security and Protection**. The Windows OS has a strong security mechanism to protect resources from illegal and accidental access. A user cannot access other user data without assigning privileges.

Topic 2: Windows Evolution

Windows exist in several versions. New versions of Windows are introduced from time to time. Actually certain new APIs are included in the new version to improve or extend its functionalities. The following major themes or features are considered while developing a new version.

- **Scalability**. A new version of Windows OS runs on different platforms including PCs, enterprise servers, multiprocessing systems, mobiles and systems having large memory space.
- **Performance**. A newer version of Windows certainly improves performance compared to previous versions.
- **Integration**. A newer version must integrate with new technologies like web services, .NET technologies, multimedia etc.
- **Ease of use**. Certain new APIs and improved GI in the new version can ensure ease of use.
- **Enhanced API**. Introducing new APIs or enhancements in existing APIs should be the main theme of the new version.

Disk Operating System (DOS)

In the 1980s, Microsoft Disk Operating System was used inside the IBM PCs incorporating Intel Processor. It was a text based and command line operating system. It was a single user OS. Its filing system was based on FAT and was able to access up to the maximum of 4GB files.

Windows History

Keeping the demand of graphical user interface, Microsoft developed its first version Windows 3.1. In this version, DOS Kernel and FAT based file systems were used.

After that in the 1990s, certain new versions of Windows named Windows 95, 97 and 98 were introduced supporting the 32-bit architecture of Intel's processors.

Later on **Windows NT** versions were introduced supporting a file system based on new technology called NTFS. Its security and file system was better than the previous versions.

Windows Server 2008 OS was developed for professional use to manage enterprise and server applications. Support for multi-core technology and 64-bit applications was provided in this OS. Other Windows versions supporting 32-bit, 64-bit architecture, multi-core and multiprocessing were also introduced including Windows XP, Windows Vista, Windows 7, 8 and Windows 10.

Topic 3: Windows Market Role

Several competitors of Windows OS like UNIX, Linux etc. exist in the market; however, Windows has its own unique status in the market. It has several significant advantages over other operating systems.

Above 90% PCs are based on Intel's processors and Windows is the most appropriate OS for Intel PCs. In the world of desktop, the most dominant OS is the Microsoft Windows which enjoys a market share of above 80%. Windows is not confined to the desktop, it also has support for diverse platforms including multi-core, multiprocessing, servers and mobiles etc.

Due to its dominance role, certain applications and software development tools are available in the market that can easily integrate with Windows OS and can develop windows applications ranging from small scale to enterprise level.

One of the key features of Windows OS is its rich GUI that makes its use very convenient. This interface can be easily customized according to the local setup. The size, color and visibility of graphical interface objects can also be changed by the user.

Compared to other operating systems, certain modern features exist in Windows due to which most of the developers develop their applications for Windows targeting the huge market of Windows.

Topic 4: Windows, Standards and Open Systems

Open Source Software is a software that is publically available with its source code to use, modify and distribute with original rights. It is developed by the community rather than a single company or vendor. In contrast, proprietary software is copyrighted and only available to use under a license.

Windows Operating system is a proprietary and copyrighted software of Microsoft corporation. It is provided for use only under a License agreement. Without purchasing a license, its use is illegal and is an act of copyright infringement.

Being a closed system, Windows has the following strengths.

· As Windows components are provided and updated only by a single vendor, its implementation remains uniform throughout the world. Further, extensions in Window components or APIs are only vendor-specific and so no non-standard extension is possible except for platform differences.

Windows also support various types of hardware platforms like open systems.

Interoperability of Windows: Windows provide interoperability with non-window components.

- Windows OS provides support to the Standard C and C++ libraries. We can install and use any C compilers on Windows systems.
- Socket is a resource that is required for interface when two computers are interconnected to each other. Windows also supports sockets to communicate among devices having different computer architectures and access to TCP/IP and other networking protocols.
- It also supports the Remote Procedure Calls (RPCs) architecture to call the remote functions in distributed client-server based applications.
- Windows also supports the X Windows system which is open source, cross platform software providing GUI in a distributive network environment.

Topic 5: Windows Principles

- In Windows OS, all the system resources including processes, threads, memory, pipes, DLL etc. are represented by objects which are identified and referenced by a handle. These objects cannot be directly accessed. In case, if any application approaches to access these objects directly, Windows throws an appropriate exception. The only way to access and operate on these objects is a set of APIs provided by Windows. Several APIs can be related to a single object to manipulate it differently.
- A long list of parameters is associated with each API where each parameter has its own significance but only few parameters are specified for a specific operation.
- To perform the task of multitasking and multi-threading efficiently, Windows provides a number of synchronization constructs to arbitrate among the resources.
- The names of Windows APIs are long and descriptive for its proper and convenient use.
- Some pre-defined data types required for Windows APIs are:
    - BOOL (for storing a single logical value)
    - HANDLE (a handle for object)
    - LPTSTR (a string pointer)
    - DWORD (32-bit unsigned integer)
- Windows Data types avoid the pointer operator (*).
- Some lowercase prefix letters with variable names are used to identify the type of variable. This notation is called Hungarian notation. For example, in the variable name **lpszFilename**, 'lpsz' is Hungarian notation representing a long pointer to zero terminated string.
- **windows.h** is a header file including all the APIs prototypes and data types

Topic 6: 32-Bit and 64-Bit Source Code Portability

Windows keeps two versions of each API, one for 32-bit and other for 64-bit. A 32-bit code can be run on 64-bit hardware but will be unable to exploit some features of 64-bit like accessing large disk space or using large pointer or 64-bit operation.

Latest versions of Windows support both 32 and 64-bit architectures by keeping two versions of each API, one for 32-bit and other for 64-bit.

Interoperability of 32 and 64-bit: A single source code can be built for 32-bit as well as 64-bit versions. To decide whether executable code of 32 or 64-bit is generated by the compiler at runtime, it depends on its settings or configuration. Further, to decide which version of API is used, it is also based on the compiler's configuration.

A 32-bit code can run on 64-bit hardware successfully but will be unable to use some features of 64-bit like large disk space, large pointer etc.

A source code developed for 64-bit architecture cannot easily run on a 32-bit machine. For this purpose, re-compilation of the program is required and suitable configuration is made in the compiler to generate a 32-bit executable code.

Topic 7: When to use Standard C Library for File Operations

Windows provides a set of built-in APIs to perform I/O operations. A related API with specific parameters is invoked for the concerned resource and I/O operation is performed.

Similarly, certain C/C++ standard functions are available to perform I/O operations. For example, fopen(), fclose(), fread(), fwrite() etc. are C functions that can be used to perform I/O operations related to files.

Differences between Windows APIs and Standard Functions

Standard C functions can be used inside the source code to run on Windows platform because Windows has system calls at low level to support C/C++ functions for I/O operations.

If it is required to run a program on cross-platform, then it is preferred to use the standard C/C++ library functions inside the source code. However, in this case, the advanced Windows

features like locking, synchronization, asynchronous I/O, inter process communication etc. cannot be achieved.

In case, if portability is not focused and required to avail the advanced Windows features, then it is preferred to use the Windows APIs.

Topic 8: A Simple File Copy Program Using Standard C Library

```
// cpC.  Basic File Copy Program:  C Library Implementation
// Copy File1 to File2
#include<stdio.h>
#include<errno.h>
#define BUF_SIZE 256

int main(int argc, char *argv[])     {

        FILE *inFile, *outFile;
        char rec[BUF_SIZE];
        size_t  bytesIn, bytesOut;

        if(argc!=3)    {
          printf("Usage: cp file1 file2\n");
          return 1;     }

        inFile=fopen(argv[1], "rb");
        if(inFile==NULL)  {
          perror(argv[1]);
          return 2;
                                }

        outFile=fopen(argv[2], "wb");
        if(outFile==NULL)   {
          perror(argv[2]);
          return 3;                }

        /* Process the input file a record at a time */
        while((bytesIn = fread(rec, 1, BUF_SIZE, inFile)) > 0)
        {
          bytesOut=fwrite(rec, 1, bytesIn, outFile);
```

```
        if(bytesOut != bytesIn)   {
                perror("Fatal write error");
                return 4;               }
        }
        fclose(inFile);
        fclose(outFile);
        return 0;
}
```

This program is used to copy one file to another using C standard functions. In this program, a buffer of size 256 bytes is used in which the chunks of file are copied one by one.
The source file is opened in read binary mode and the destination file in write binary mode using the C **fopen()** function.
If both are successfully opened, then a file is read inside a loop chunk by chunk using **fread()** function and written onto the destination file using **fwrite()** function. After a few iterations, the file will be written to the destination file and both files are closed.

Topic 9: A Simple File Copy Program Using Windows APIs

Program 1 - 2

```
// cpC.  Basic File Copy Program:  Windows Implementation
// Copy File1 to File2
#include<stdio.h>
#include<windows.h>
#include<stringapiset.h>
#define BUF_SIZE 16384
int main(int argc, char *argv[])     {
HANDLE  hIn, hOut;
DWORD  nIn, nOut;
        CHAR buffer[BUF_SIZE];
        LPWSTR lpwszFile1, lpwszFile2;
        INT iLen1, iLen2;

if(argc !=3)   {
          fprintf(stderr, "Usage: cp file1 file2\n");
          return 1;   }
        lpwszFile1 = (LPTSTR)malloc(510);
```

```
        lpwszFile2 = (LPTSTR)malloc(510);
        iLen1 = MultiByteToWideChar(CP_ACP, 0, argv[1], -1, lpwszFile1, 510);
        iLen2 = MultiByteToWideChar(CP_ACP, 0, argv[2], -1, lpwszFile2, 510);
        hIn=CreateFile(lpwszFile1, GENERIC_READ, FILE_SHARE_READ, NULL,
                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
        if (hIn==INVALID_HANDLE_VALUE)  {
            fprintf(stderr, "Cannot open input file. Error: %x\n", GetLastError());
            return 2;       }

        hOut=CreateFile(lpwszFile2, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
                FILE_ATTRIBUTE_NORMAL, NULL);
        if (hOut== INVALID_HANDLE_VALUE)   {
                fprintf(stderr, "cannot open output file. Error: %x\n", GetLastError());
                CloseHandle(hIn);
            return 3;
        }

        while((ReadFile(hIn, buffer, BUF_SIZE, &nIn, NULL) && nIn > 0)  {
          WriteFile(hOut, buffer, nIn, &nOut, NULL);
          if (nIn != nOut)   {
                fprintf("Fatal write error: %x\n", GetLastError());
                CloseHandle(hIn);
                CloseHandle(hOut);
                return 4;
                }
        }
        CloseHandle(hIn);
        CloseHandle(hOut);
        return 0;
}
```

In this program, Windows APIs are used instead of C standard functions. In the main program, the words in capital letters like HANDLE, DWORD, CHAR, LPWSTR and INT are Windows data types. As the file paths given in command line parameters are in ASCII format, the parameters will be first converted to Unicode using the **MultiByteToWideChar()** function.

After that, a file is opened for reading purposes and its handle is stored in hIn. Similarly, another file is created for writing purposes and its handle is stored in hOut.

If files are successfully opened, then the source file is read in a loop and written to the destination file. At the end both the handles for files are closed.

Topic 10: A Simple File Copy Program Using Windows Convenience Function

Numerous Windows functions are used to perform various tasks at low level. However, Windows has a set of **Convenience functions** that combine several functions to perform a common task. In most cases, these functions improve the performance because several tasks are performed by a single function.

For example, **CopyFile()** is a convenience function that replaces the algorithms used for creating, opening, reading and writing one file to another.

Program 1 - 3

```
/* cpC.  Basic File Copy Program:  Windows Implementation
                        using convenience function CopyFile() */
// Copy File1 to File2

#include<stdio.h>
#include<windows.h>
#define BUF_SIZE 256
LPWSTR lpwszFile1, lpwszFile2;
INT iLen1, iLen2;

int main(int argc, char *argv[])        {
if(argc !=3)  {
          fprintf(stderr, "Usage: cp file1 file2\n");
          return 1;
               }
        lpwszFile1 = (LPTSTR)malloc(510);
        lpwszFile2 = (LPTSTR)malloc(510);
        iLen1 = MultiByteToWideChar(CP_ACP, 0, argv[1], -1, lpwszFile1, 510);
        iLen2 = MultiByteToWideChar(CP_ACP, 0, argv[2], -1, lpwszFile2, 510);

if (!CopyFile(lpwszFile1, lpwszFile2, FALSE)
        {
                        fprintf(stderr, "CopyFile Error: %x\n", GetLastError());
                        return 2;
        }
```

```
    return 0;
}
```

Lecture 11: Windows File System

Windows supports various file systems.

- **NT File System**

NTFS is an important file system supported by Windows, its main features are:

  - **Security**: One user cannot access other user data without privileges.
  - **Fault tolerance** (if a portion of disk corrupts, it works because different copies of information are maintained in this files system).
  - **Encryption** (data encrypted/decrypted, provide security)
  - **Compression** (Space capacity increased due to data compression)
  - Supports very huge file size

- **File Allocation Tables**
  - FAT12, FAT16, FAT32 (Start version of File Systems, also supported by Windows)
- Compact Disk File System (CDFS)
  - Support CD
- Supports Universal Disk Format (UDF) & Live File System (LFS) also

Lecture 12: File Naming Conventions

- Windows OS supports a number of File Systems. Each file system has its own mechanism for naming files.

Certain Limitations for File Naming

- Letters like A, B, C etc. are used to represent Drive, Network Drives are represented by higher letters like N, K, L etc.
- Double slash (\\) in the start of path represents remote resource
- Forward slash (\ ) or backslash (/) is used as a path name separator
- The first 31 ASCII characters (control characters) cannot be used in file names
- Special symbols like \, /, colon (:), pipe (|) etc. cannot be used in filenames
- File and directory names are case insensitive
- File name and extensions are separated by (.)
- Max size for file name is 255 and for path is 260 characters.

- File extension takes 2 – 4 bytes
- Single dot (**.**) represents current directory while double dot (**..**) represents one step back (up) directory

Lecture 13: Creating and Opening Files (Using Windows API)

The CreateFile() API

CreateFile() API with a list of parameters is used to open or create a new file. Its return type is HANDLE to an open file object in case of successful opening or creation. The parameters are:

lpFileName: It is a string pointer that points to a filename to be opened or created.

dwDesiredAccess: It is a 32-bit double word which specifies the GENERIC_READ and WRITE access.

dwShareMode: This mode specifies how the file is shared?

- 0 signifies that file will not be shared
- FILE_SHARE_READ allows the file to be shared for concurrent read
- FILE_SHARE_WRITE allows the file to be shared for writing.

lpSecurityAttributes: points to a security attributes structure.

dwCreate: signifies whether to create a new file or overwrite an existing one.

- CREATE_NEW: Creates a new file. If the file already exists, then fail.
- CREATE_ALWAYS: Creates a new file or overwrites an existing one.
- OPEN_EXISTING: open an existing file or fail if the file does not exist
- OPEN_ALWAYS: open an existing file, if it does not exist, then create it.

dwFlagsAndAttributes: It signifies attributes of the newly created file.
hTemplateFile: It is the Handle of an open file that specifies attributes to apply to a newly created file.

Lecture 14: Reading/Writing a File

The **ReadFile()** API is used to read data from file to buffer.

Syntax:

BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);

- If the file is not opened in concurrent mode, then ReadFile() starts reading from the current position.
- If the current location is End of File, then no Errors occur and *lpNumberOfBytesRead is set to zero
- The function returns FALSE if it fails in case any of the parameter is invalid

Parameters

- HANDLE hFile is the file handle
- LPVOID lpBuffer is the address of the array that stores the data read from the file.
- DWORD nNumberOfBytesToRead is the number of bytes to be read from the file
- LPDWORD lpNumberOfBytesRead is the number of bytes actually read
- LPOVERLAPPED lpOverlapped is used for concurrent processing.

The **WriteFile()** API is used to write data from buffer to file.

Syntax:

BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);

To write through the current size of file, the file must be opened with FILE_FLAG_WRITE_THROUGH option

Lecture 15: Closing a File

After opening & using a file, it is required to close and invalidate the file handles in order to release the system resources.

The **CloseFile()** API is used to close a file. A file handle is passed as parameter to this API as a result of which the API will return True or False value. If the operation is successful, then it returns True value. In case if the handle is already invalid, then it will return False value.

Lecture 16: Generic Characters

- Windows supports both ASCII (8-bit standard) and Unicode (16-bit standard) characters
- Windows have different API function variants to deal with both Unicode and ASCII characters.
- Unicode is used to represent characters in other languages like Arabic, Chinese, Urdu etc.
- The structure of a program is different for Unicode and ASCII characters.
- Generic program or code works for both the ASCII and Unicode.
- Windows provides a set of data types for ASCII, a set of data types for Unicode and a set of data types for Generic code like TCHAR, LPTSTR, LPCTST etc.
- For Unicode the following two macros should be used before including <windows.h>
    - #define UNICODE: it includes all the function headers
    - #define _UNICODE: it includes variable types
    - These macros will force the generic data type to be replaced by Unicode data types.
- sizeof() operator should be used in coding instead of hard coding if required.
- Use generic C Library functions like _itot(), _stprintf(), _tcscopy() etc.
- Use the _TEXT or T_macro for representing string constants.
- <tchar.h> file should be included for generic C functions.
- Most of the APIs in the latest version of Windows expect to pass the parameters in Unicode, therefore a generic code should be written instead of ASCII code.

Lecture 17: Generic Functions

- Besides Generic data types, Windows also supports generic functions.
- Examples of generic functions are:
  _tcscmp() instead of lstrcmp(), _tcscmpi() instead of lstrcmpi()

- Some functions that deal with Unicode characters and strings and work with locale settings transparently are:
    - CharUpper()
    - IsCharAlphaNumeric()
    - CompareString()
- Generic main() function is modified in terms of data types of its parameters and its name as mentioned below:
  int _tmain(int argc, LPTSTR argv[])
- <windows.h> and <tchar.h> header files must be included before the main() function.
- Windows usually have two versions for each API function. For example, TextOut will have two variants i.e. TextOutA(16 bit API) and TextOutW(32 bit API)
- The use of Generic data types and Unicode macro enables the compiler to decide which version to choose

Lecture 18: Unicode Strategies

While writing a new code or enhancing an existing one, a programmer can adapt any of the following strategies based on requirements.

- Ignore Unicode Policy
- Use 8 bit only
- Ignore Unicode
- Use data types like char or CHAR
- Most of the Windows cannot be used ignoring unicode
- Use standard library functions like printf(), scanf(), atoi() etc.

Using Generic Code Policy

- The Unicode macro is used to switch between 8 bit and Unicode
- Generic functions are used
- Generic data types are used

Using Unicode Policy

- Use Unicode only
- Unicode functions are used
- Wide characters data types are used

Unicode and 8-bit Policy

- Handle Unicode and 8-bit

- Write code for both Unicode and 8-bit
- Decision regarding which option to choose is made at runtime
- Use of Unicode only strategy is now increasing popularly
- Use of generic code makes the code more flexible

Lecture 19: Reporting Errors

- Windows Operating System provides the facility to report errors, if occurs.
- It is an important and salient feature of Windows to display error messages for user convenience.
- The important Windows function for reporting errors is **GetLastError()**. It returns the code for the last system error.
- Another function **FormatMessage()** translates the error code into meaningful English or language selected in preferences.
- A message string reference against the error code is stored in another parameter.
- Another function **LocalFree()** is also used with FormatMessage() to deallocate the allocated local memory.

Lecture 20: Example for Reporting Errors

The following two header files must be included for different functions, prototypes, and data types.

<environment.h> and **<everything.h>**

<environment.h> includes a Unicode macro for the environment of the program.

<everything.h> includes all the header files required for typical Windows program

Example:

```
#include<everything.h>
VOID ReportError(LPCTSTR userMessage, DWORD exitCode, BOOL printErrorMessage)
{
        DWORD eMsgLen, errNum,=GetLastError();
        LPTSTR lpvSysMsg;
   _ftprintf(stderr, _T("%s\n"), userMessage)
    if (printErrorMessage)
```

```
    {
        eMsgLen = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER  |
        FORMAT_MESSAGE_FORM_SYSTEM, NULL, errNum, MAKELANGID(LANG_NEUTRAL,
        SUBLANG_DEFAULT), (LPTSTR) &lpvSysMsg, 0, NULL);

        if (eMsgLen > 0)
        {
                _fprintf(stderr, _T("%s\n"), lpvSysMsg);
        }
        else
        {
                _ftprintf (stderr, _T("Last Error Number; %d\n"), errNum);
        }
       if (lpvSysMsg != NULL)
       LocalFree(lpvSysMsg);
    }
    if (exitCode>0)
    {
        ExitProcess(exitCode);
        return;
    }
}
```

Here this function **ReportError()** receives three parameters. Inside this function, the result of **GetLast()** function is stored in the variable, errNum. A generic string variable "lpvSysMsg" is declared for storing the error message.

If the user likes to print the error message, then the error code will first format and convert it into a string using the **FormatMessage()** function.
If message length is greater than 0, then it will print the message, otherwise will print the error code and memory is deallocated.