Unit 2: Operating System Principles

2.4. The Windows API - Naming, Conventions, Types

Roadmap for Section 2.4.

- Windows API principles
- Portable programming - the standard C library
- Comparing UNIX and Windows programming styles: an example program

Windows API - Overview

- APIs to Windows systems evolved over time:
  - Win16 - introduced with Windows 2.0
  - Win32 - introduced with Windows NT, Windows 95
  - Win64 – introduced with Windows 64-bit edition
- “Windows API” summarizes all of the above
  - In these slides, Windows API refers to Win32 and Win64

Windows API - major functionality

- File System and Character I/O
- Direct File Access and File Attributes
- Structured Exception Handling
- Memory Management and Memory-Mapped Files
- Security
- Process Management
- Inter-process Communication
- Threads and Scheduling, Windows Synchronization

Windows API Principles

- System resources are *kernel objects* referenced by a *handle* (handle vs. UNIX file descriptors & PIDs)
- *Kernel objects* must be manipulated via Windows API
- Objects – files, processes, threads, IPC pipes, memory mappings, events – have security attributes
- Windows API is rich & flexible:
  - convenience functions often combine common sequences of function calls
- Windows API offers numerous synchronization and communication mechanisms

Windows API principles (contd.)

- Thread is unit of executions (instead of UNIX process)
  - A process can contain one or more threads
- Function names are long and descriptive (as in VMS)
  - *WaitForSingleObject()*
  - *WaitForMultipleObjects()*

Windows API Naming Conventions

- Predefined data types are in uppercase
  - BOOL (32 bit object to store single logical value)
  - HANDLE
  - DWORD (32 bit unsigned integer)
  - LPTSTR
  - LPSECURITY_ATTRIBUTE
- Prefix to identify pointer & const pointer
  - LPTSTR (defined as TCHAR *)
  - LPCTSTR (defined as const TCHAR *)

    (Unicode: *TCHAR* may be 1-byte *char* or 2-byte *wchar_t*)
  - See \$MSDEV\INCLUDE\WINDOWS.H, WINNT.H, WINBASE.H

    (MSDEV=C:\Program Files\Microsoft Visual Studio\...)

64-bit vs. 32-bit Windows APIs

- Pointers and types derived from pointer, e.g. handles, are 64-bit long
  - A few others go 64, e.g. WPARAM, LPARAM, LRESULT, SIZE_T
  - Rest are the same, e.g., 32-bit INT, DWORD, LONG

Win32 and Win64 are referred to as the Windows API

API	Data Model	int	long	pointer
Win32	ILP32	32	32	32
Win64	LLP64	32	32	64
UNIXes	LP64	32	64	64

Differences from UNIX

- HANDLEs are opaque (no short integers)
  - No analogy to file descriptors 0,1,2 in Windows
- No distinctions between HANDLE and process ID
  - Most functions treat file, process, event, pipe identically
- Windows API processes have no parent-child relationship
  - Although the Windows kernel keeps this information
- Windows text files have CR-LF instead of LF (UNIX)
- Anachronisms: “long pointer“ (32 bit)
  - LPSTR, LPVOID

Using Windows API

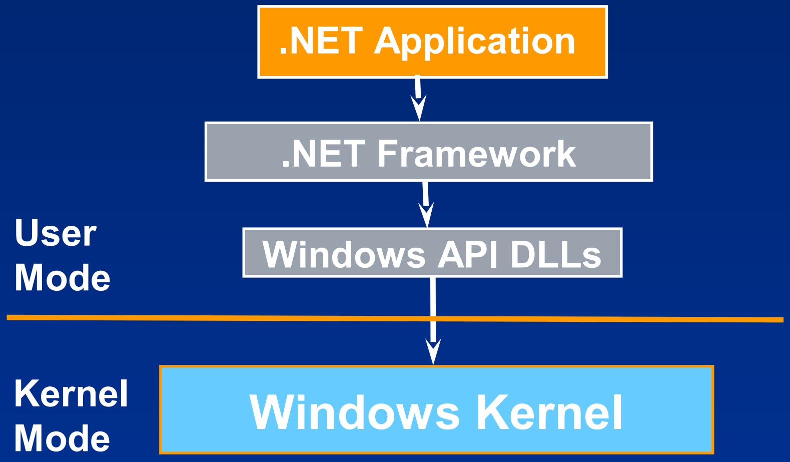
- By calling the Windows API’s functions directly from your code
  - Why (when)?
    - Performance (i.e. speed at execution time)
- Using them indirectly by calling functions from libraries/frameworks built over Windows OS
  - Why?
    - Convenience (i.e. ease of programming)
  - Examples: MFC, .NET Framework, UWP (see next slides)

MFC Library

- Microsoft Foundation Classes (MFC)
  - a library that wraps portions of the Windows API in C++ classes
  - classes are defined for many of the handle-managed Windows objects and also for predefined windows and common controls of GUI
  - Version 1.0 was introduced in 1992, last version in Dec. 2014; usually a new version of MFC shipped with every new version of Visual Studio
  - Its use was reduced after Microsoft introduced .NET Framework
- Alternatives to MFC:
  - Microsoft Windows Template Library (WTL)
  - Borland Object Windows Library (OWL) and, later on, Visual Component Library (VCL), used in the ’90

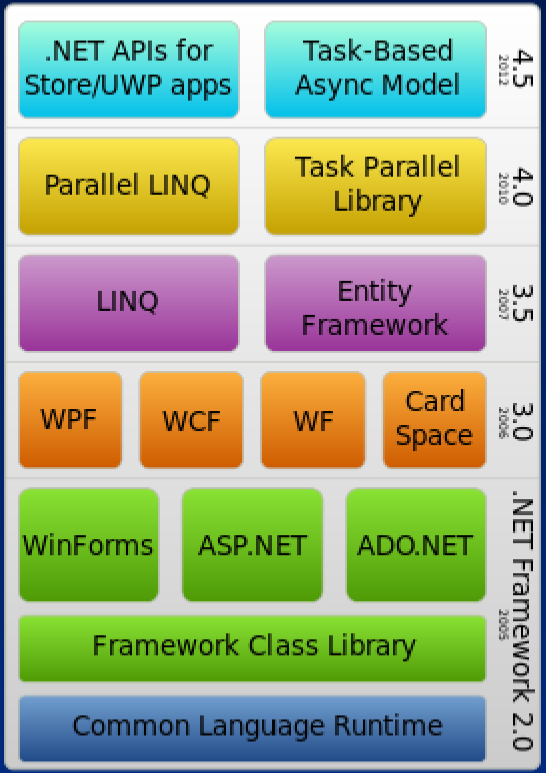
.NET Framework (and CLR)

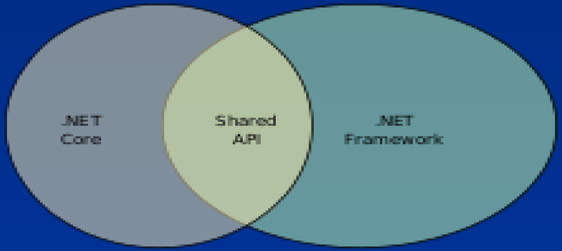
- .NET is a software framework for writing apps on Windows, which provides:
  - A large class library known as Framework Class Library (FCL)
  - An **application virtual machine**, called Common Language Runtime (CLR)
- .NET Framework is built on standard Windows APIs
  - It is not a subsystem
  - It does not call undocumented Windows system calls



.NET Framework

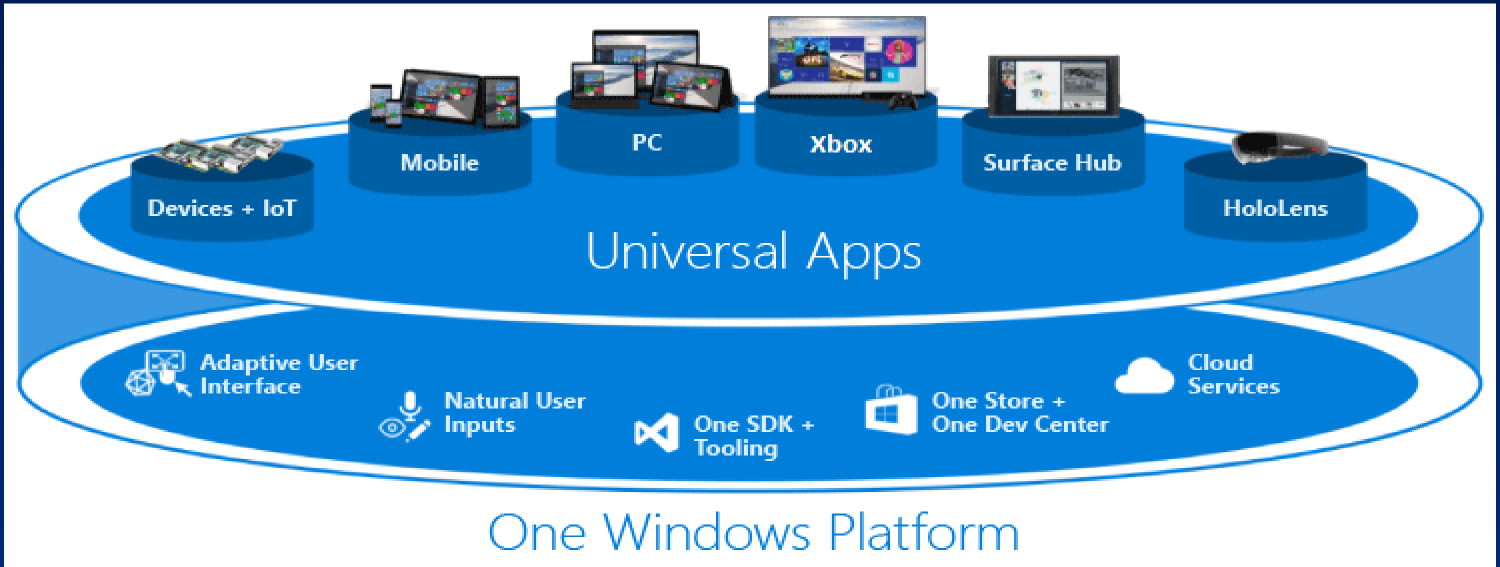
- .NET Framework components and history –>
- Alternative implementations:
  - Mono
  - .NET Micro Framework
  - .NET Core
- .NET Core:
  - It is a cross-platform free and open-source managed software framework similar to .NET Framework
  - Version 1.0 was released on 27 June 2016
  - It consists of CoreCLR, a complete cross-platform runtime implementation of CLR, the virtual machine that manages the execution of .NET programs, and CoreFX, which is a partial fork of FCL
  - It shares a subset of .NET Framework APIs, but it comes also with its own API that is not part of .NET Framework

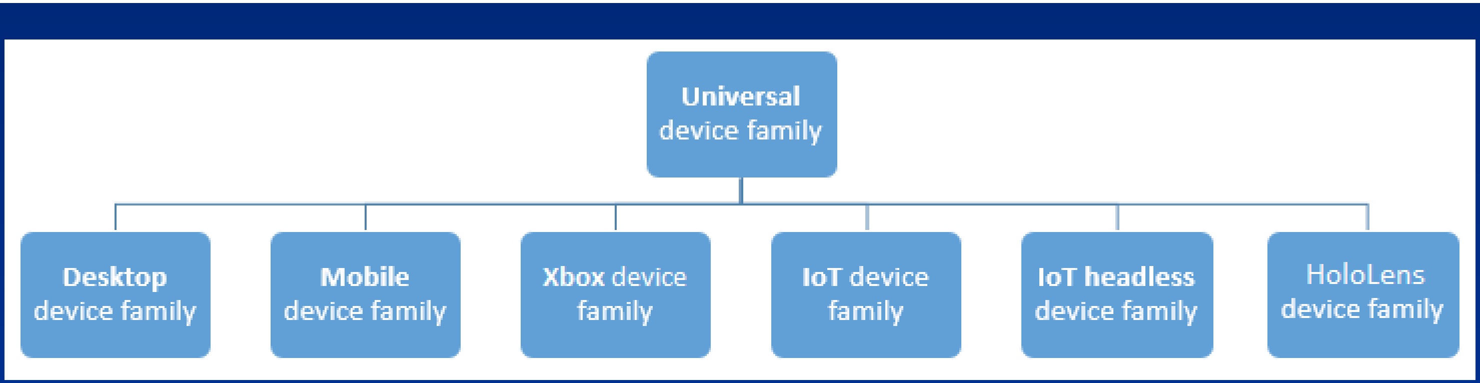




Universal Windows Platform (UWP)

- A new application model, introduced in Windows 10




Universal device family

Desktop device family

Mobile device family

Xbox device family

IoT device family

IoT headless device family

HoloLens device family

See https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

Portability: The Standard C Library

- Included in the Windows API
- C library contains functions with limited capability to manage OS resources (e.g.; files)
- Often adequate for simple programs
- Possible to write portable programs
- Include files:
  - <stdlib.h>, <stdio.h>, <string.h>

Example Application

- Sequential file copy:
  - The simplest, most common, and most essential capability of any file system
  - Common form of sequential processing
- Comparing programs:
  - Quick way to introduce Windows API essentials
  - Contrast different approaches
  - Minimal error processing

Sequential File Copy

UNIX:

- File descriptors are integers; error value: -1
- read()/write() return number of bytes processed,
  - 0 indicates EOF
  - Positive return value indicates success
- close() works only for I/O objects
- I/O is synchronous
- Error processing depends on perror() & errno (global)

Basic cp file copy program. UNIX Implementation

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 512  /* or 4096 */

int main (int argc, char *argv []) {
   int input_fd, output_fd;
   ssize_t bytes_in, bytes_out;
   char rec [BUF_SIZE];
   if (argc != 3) {
      printf ("Usage: cp file1 file2\n");
      return 1;
   }
   input_fd = open (argv [1], O_RDONLY);
   if (input_fd == -1) {
      perror (argv [1]); return 2;
   }
   output_fd = open(argv[2],
         O_WRONLY|O_CREAT|O_TRUNC, 0600);
   if (output_fd == -1) {
      perror (argv [2]); return 3;
}
```

```
/* Process the input file a record
   at atime. */


while ((bytes_in = read
   (input_fd, &rec, BUF_SIZE)) > 0) {
      bytes_out =
         write (output_fd, &rec, bytes_in);
      if (bytes_out != bytes_in) {
         perror ("Fatal write error.");
         return 4;
      }
   }
   close (input_fd);
   close (output_fd);
   return 0;
}
```

File Copy with Standard C Library

- Open files identified by pointers to FILE structures
  - NULL indicates invalid value
  - Pointers are „handles“ to open file objects
- Call to fopen() specifies whether file is text or binary
- Errors are diagnosed with perror() of ferror()

- Portable between UNIX and Windows
- Competitive performance
- Still constrained to synchronous I/O
- No control of file security via C library

Basic cp file copy program. C library Implementation

```
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 512  /* or 4096 */

int main (int argc, char *argv []) {
   FILE *in_file, *out_file;
   char rec [BUF_SIZE];
   size_t bytes_in, bytes_out;
   if (argc != 3) {
      printf ("Usage: cp file1 file2\n");
      return 1;
   }
   in_file = fopen (argv [1], "rb");
   if (in_file == NULL) {
      perror (argv [1]);
      return 2;
   }
   out_file = fopen (argv [2], "wb");
   if (out_file == NULL) {
      perror (argv [2]);
      return 3;
   }
```

```
/* Process the input file a record
   at a time. */

   while ((bytes_in =
      fread (rec, 1, BUF_SIZE, in_file)) > 0) {
        bytes_out =
           fwrite (rec, 1, bytes_in, out_file);
        if (bytes_out != bytes_in) {
           perror ("Fatal write error.");
           return 4;
        }
   }

   fclose (in_file);
   fclose (out_file);
   return 0;
}
```

File Copying with Windows API

- <windows.h> imports all Windows API function definitions and data types
- Access Windows objects via variables of type HANDLE
- Generic CloseHandle() function works for most objects
- Symbolic constants and flags
  - INVALID_HANDLE_VALUE, GENERIC_READ
- Functions return boolean values
- System error codes obtained via GetLastError()
- Windows security is complex and difficult to program

Basic cp file copy program. Windows API Implementation

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 512  /* or 4096 */

int main (int argc, LPTSTR argv []) {
   HANDLE hIn, hOut;
   DWORD nIn, nOut;
   CHAR Buffer [BUF_SIZE];
   if (argc != 3) {
      printf("Usage: cp file1 file2\n");
      return 1;
   }
   hIn = CreateFile (argv [1],
         GENERIC_READ,
         FILE_SHARE_READ, NULL,
         OPEN_EXISTING,
         FILE_ATTRIBUTE_NORMAL,
         NULL);
   if (hIn == INVALID_HANDLE_VALUE) {
      printf ("Input file error:%x\n",
             GetLastError() );
      return 2;
   }
   hOut = CreateFile (argv [2],
            GENERIC_WRITE, 0, NULL,
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            NULL);
   if (hOut == INVALID_HANDLE_VALUE) {
      printf("Output file error: %x\n",
             GetLastError() );
      return 3;
   }
   while (ReadFile (hIn, Buffer,
               BUF_SIZE, &nIn, NULL)
               && nIn > 0) {
      WriteFile (hOut, Buffer, nIn, &nOut, NULL);
      if (nIn != nOut) {
         printf ("Fatal write error: %x\n",
                 GetLastError() );
         return 4;
      }
   }
   CloseHandle (hIn);
   CloseHandle (hOut);
   return 0;
}
```

File Copying with Windows API Convenience Functions

- Convenience functions may improve performance
  - Programmer does not need to be concerned about arbitrary buffer sizes
  - OS manages speed vs. space tradeoffs at runtime

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

int main (int argc, LPTSTR argv [])
{
    if (argc != 3) {
        printf ("Usage: cp file1 file2\n"); return 1;
    }
    if (! CopyFile (argv [1], argv [2], FALSE)) {
        printf ("CopyFile Error: %x\n", GetLastError() );
        return 2;
    }
    return 0;
}
```

Further Reading

- Johnson M. Hart, Win32 System Programming: A Windows® 2000 Application Developer's Guide, 2nd Edition, Addison-Wesley, 2000.
  - (This book discusses select Windows programming problems and addresses the problem of portable programming by comparing Windows and Unix approaches).
- Jeffrey Richter, Programming Applications for Microsoft Windows, 4th Edition, Microsoft Press, September 1999.
  - (This book provides a comprehensive discussion of the Windows API – suggested reading).