Unit 2: Windows 2000 Architecture

2.2. Win32 API – Naming Conventions, Types

Win32 System Interfaces

- Additional Reading:
  Johnson M. Hart, "**Win32 System Programming**",
  Addison-Wesley Advanced Windows Series

- Introduction to Win32 programming, style, conventions
- Comparison of Win32 and UNIX system calls
- Lots of examples – no GUI covered

Overview

- Win32 Principles
- 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, Win32 Synchronization
- Comparison of Win32, UNIX, and C Libraries

Win32 Principles

- System resources are *kernel objects* referenced by a *handle* (handle vs. UNIX file descriptors & PIDs)
- *Kernel objects* must be manipulated via Win32 API
- Objects – files, processes, threads, IPC pipes, memory mappings, events – have security attributes
- Win32 is rich & flexible: convenience functions
- Thread is unit of executions (instead of UNIX process)
- Function names are long and descriptive (as in VMS)
  - *WaitForSingleObject()*
  - *WaitForMultipleObjects()*

Win32 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\VC98\)

Differences from UNIX

- HANDLEs are opaque (no short integers)
  - No analogy to file descriptors 0,1,2 in Win32
- No distinctions between HANDLE and process ID
  - Most functions treat file, process, event, pipe identically
- Win32 processes have no parent-child relationship
- No process groups
- Windows text files have CR-LF instead of LF (UNIX)
- Anachronisms: “long pointer“ (32 bit)
  - LPSTR, LPVOID

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 256

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,0666);
   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 Win32
- 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 <windows.h>
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256

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 Win32

- <windows.h> imports all Win32 function definitions and data types
- Access to Win32 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()
- NT security is complex and difficult to program

Basic cp file copy program. Win32 Implementation

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

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 Win32 Convenience Functions

- Convenience functions may improve performance
- No need to think about arbitrary buffer sizes

```
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

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;
}
```