WIN 32 API

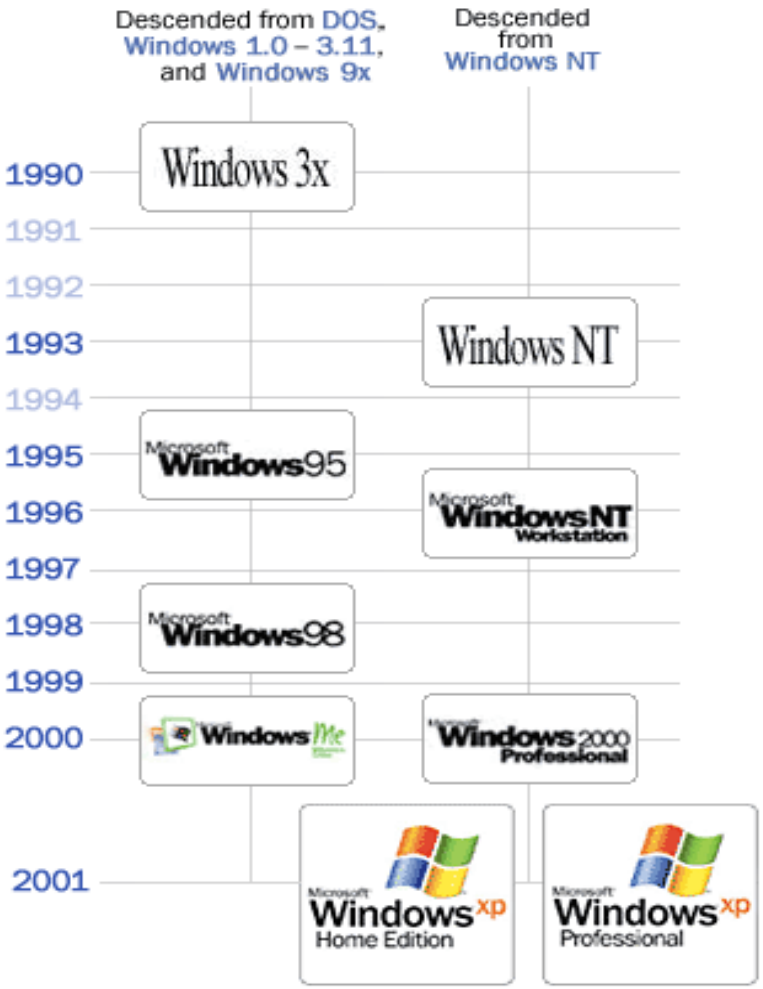
Úvod

- Něco málo historie
- Přehled architektury
  - Kernel mode vs User Mode
  - Native API
  - Win 32 API
- WIN 32 API
  - Soubory, synchronizace
  - Console applications

Historie - desktop

http://www.microsoft.com/windows/WinHistoryProGraphic.mspx

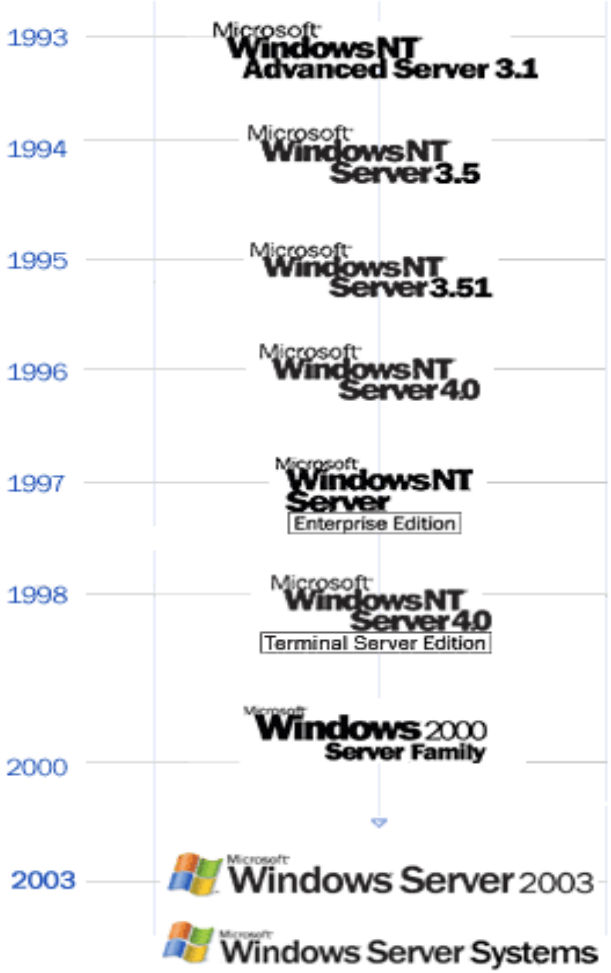
- 10. listopad 1983, Microsoft ohlásil první Microsoft Windows®



Historie - server

http://www.microsoft.com/windows/WinHistorySrvrGraphic.mspx

- 1988 se zformovala vývojová skupina pro Microsoft Windows NT®
- 1993 Windows NT Server



Windows Vista

- Původní plán vydáním počátek 2006 - mezikrok mezi Windows XP a Windows “Vienna”
- Oficiální release - 30. ledna 2007
- Jako základ použité Windows 2003 Server
- Hlavní nové technologie
  - .NET Framework, nový grafický a zvukový subsystém
  - WPF – Windows Presentation Foundation – GUI používající XML, .Net a vektory
  - WCF - Windows communication Foundation – komunikační framework
  - WWF . Windows Workflow Foundation – podpora pro vytváření složitých workflow

Architektura Windows

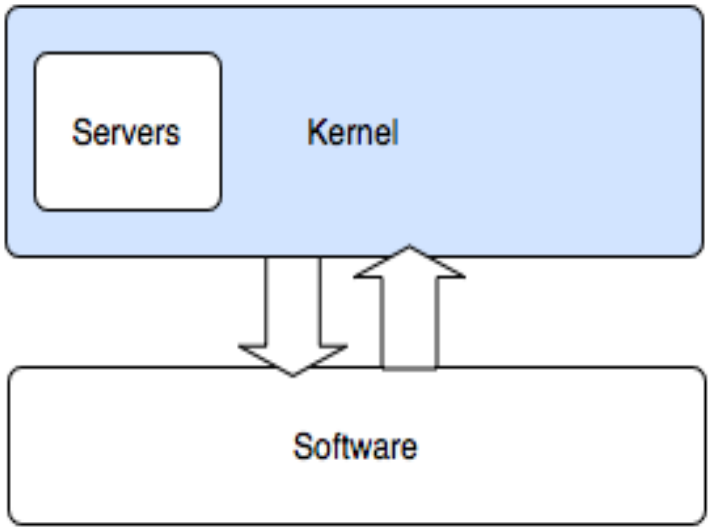
- Rodina Windows NT
  - Windows NT
  - Windows 2000
  - Windows XP
  - Windows Server 2003
- Další slajdy se zbývají pouze rodinou NT
- WIN32 API existuje i na Windows 95 a 98, ale nepodporuje všechny funkce
  - Architektura systému je jiná

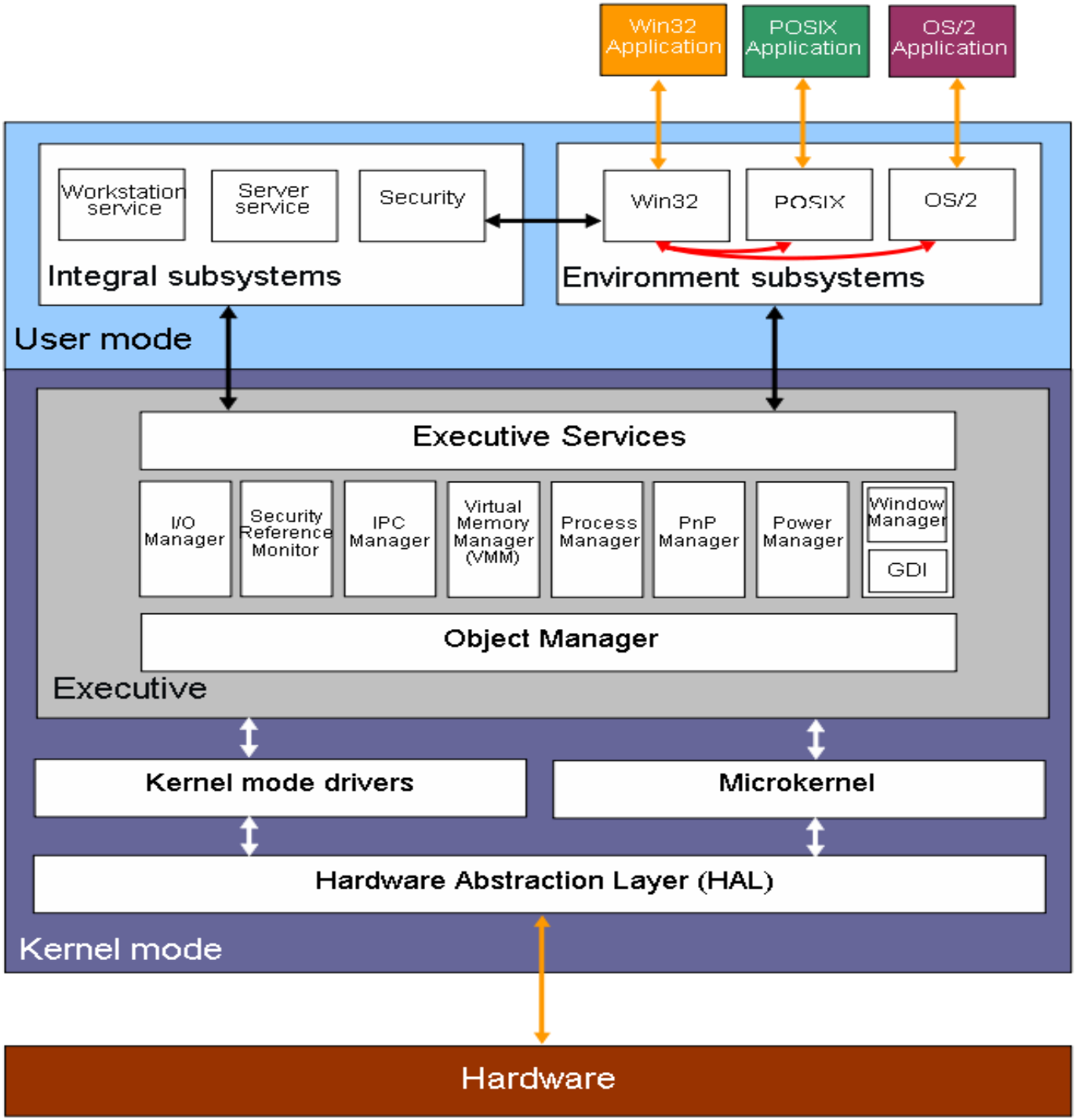
Architektura Windows

- Preemptivní
- Reentrantní
- Uniprocessor i Symmetrical Multi Processor (SMP)
- Windows XP – podpora pro 64 bitové CPU
- Modulární
- Dvě základní vrstvy - User a Kernel mode

Architektura Windows

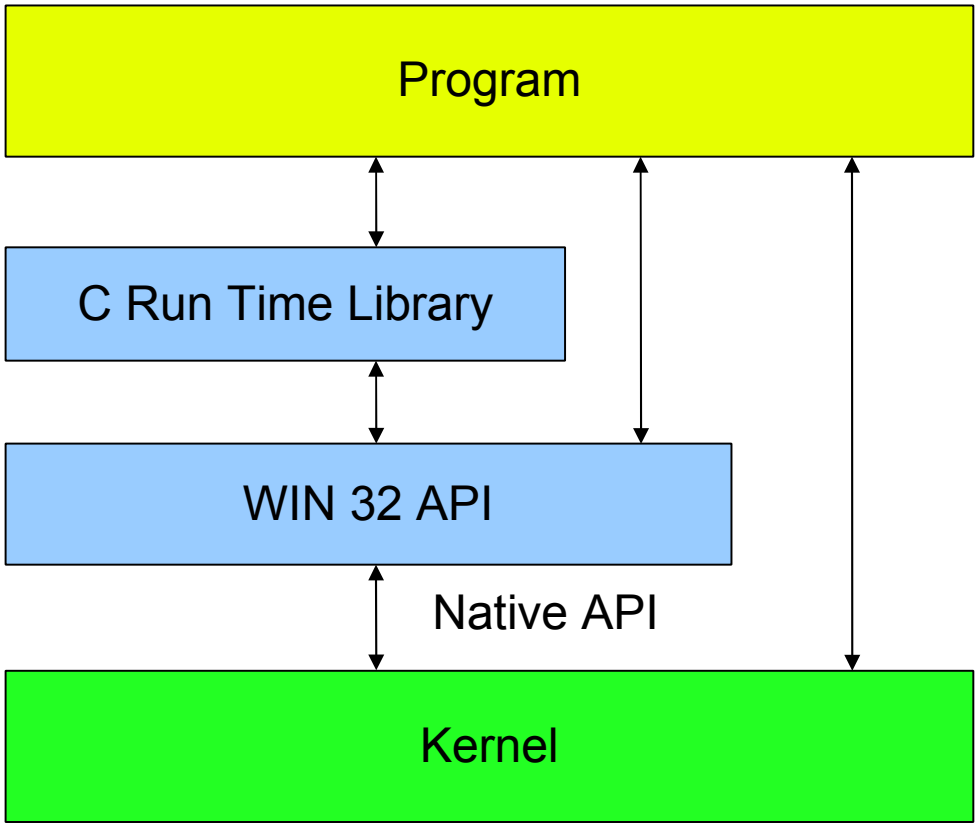
- Hybridní kernel
  - Obsahuje prvky mikro-kernelu
  - Má i rysy monolitického kernelu (v kernel mode jsou i některé subsystémy)




Win32 Application
POSIX Application
OS/2 Application
Workstation service
Server service
Security
Integral subsystems
Win32
POSIX
OS/2
Environment subsystems
User mode
Executive Services
I/O Manager
Security Reference Monitor
IPC Manager
Virtual Memory Manager (VMM)
Process Manager
PnP Manager
Power Manager
Window Manager
GDI
Object Manager
Executive
Kernel mode drivers
Microkernel
Hardware Abstraction Layer (HAL)
Kernel mode
Hardware

Architektura Windows

- Kam patří Win32 API z hlediska programu?



Windows API

- API pro programování Windows (v C/C++)
- Existuje několik verzí
  - Win16 – v 16 bitových verzích Windows
  - Win32 – od Windows NT (ořezaná verze ve Windows 96/98)
  - Win32s – podmnožina pro Windows 3.11
  - Win64 – Win32 for 64-bit Windows – stejné fce jako ve Win32, ale v 64 bitech
  - Win FX – Windows Vista, objektové API, .Net (rozšiřuje a využívá)

Windows API

- Base services (kernel32.dll, advapi.dll) – file systém, device drivers, procesy, thready, atd.
- Graphics Device Interface (gdi32.dll) – grafika, tiskárny, atd.
- User Interface (user32.dll, comctl32.dll) – řízení oken a základní prvky (tlačítka, atd.)
- Common Dialog Box Library (comdlg32.dll) – základní dialogy (výběr souboru, fontu, atd.)

Windows API

- Windows shell (shell32.dll, shlwapi.dll) – funkce shellu
- Network Services (různé dll dle služby) – síťové služby včetně WinSocku

- DDE->OLE->COM (interakce mezi programy)
- DirectX
- A další....

Win32 API

- Sídlí v dll souborech
  - Sdílené
- Běží v user mode
- Dobře dokumentované – viz např. MSDN
  - http://msdn1.microsoft.com
- Při kompilaci v C potřebujete
  - **include** soubory – především Windows.h
  - **lib** soubory – součást SDK, např. kernel32.lib, user32.lib, gdi32.lib, advapi32.lib .....
  - **DLL** soubory – součásti Windows :-)

Win32 API
Jak se volá kernel?

- Na Linuxu – volání kernelu přes interrupt
  - Int 80H
- Jak je to na Windows?
  - Při volání funkcí jádra je také potřeba přepnout do ringu0
  - Nedělá se to přímo
  - Existuje tzv. Native API
    - Není tak dobře dokumentováno
    - Mělo by být voláno jenom přes Windows API
    - Podobně jako na Linuxu přes int -> INT 2Eh

Native API

- Volání přes INT je náročné (viz dokumentace k Intelu, brána přerušení a context switch)
- Novější CPU mají speciální instrukce pro urychlení volání (zrychlení až o 250%)
  - SYSENTER – Intel Pentium II
  - SYSCALL – AMD K7
- Podpora ve Windows XP
  - Umí INT/SYSCALL/SYSENTER
  - Dle CPU se vybere správná verze

Native API

- INT 2Eh /SYSCALL/SYSENTER
- Číslo funkce je v EAX
- Parametry funkce jsou na zásobníku a ukazuje na ně registr EDX
- POZOR – Čísla funkcí nejsou dokumentována jako na Unixech, protože
  - Jsou generována, když se staví nová verze OS
  - Jsou různá pro různé verze OS
    - I např. pro různé service packy

Win32 API

Jak se volá kernel?

Příklad implementace jedné fce

NtQuerySystemInformation

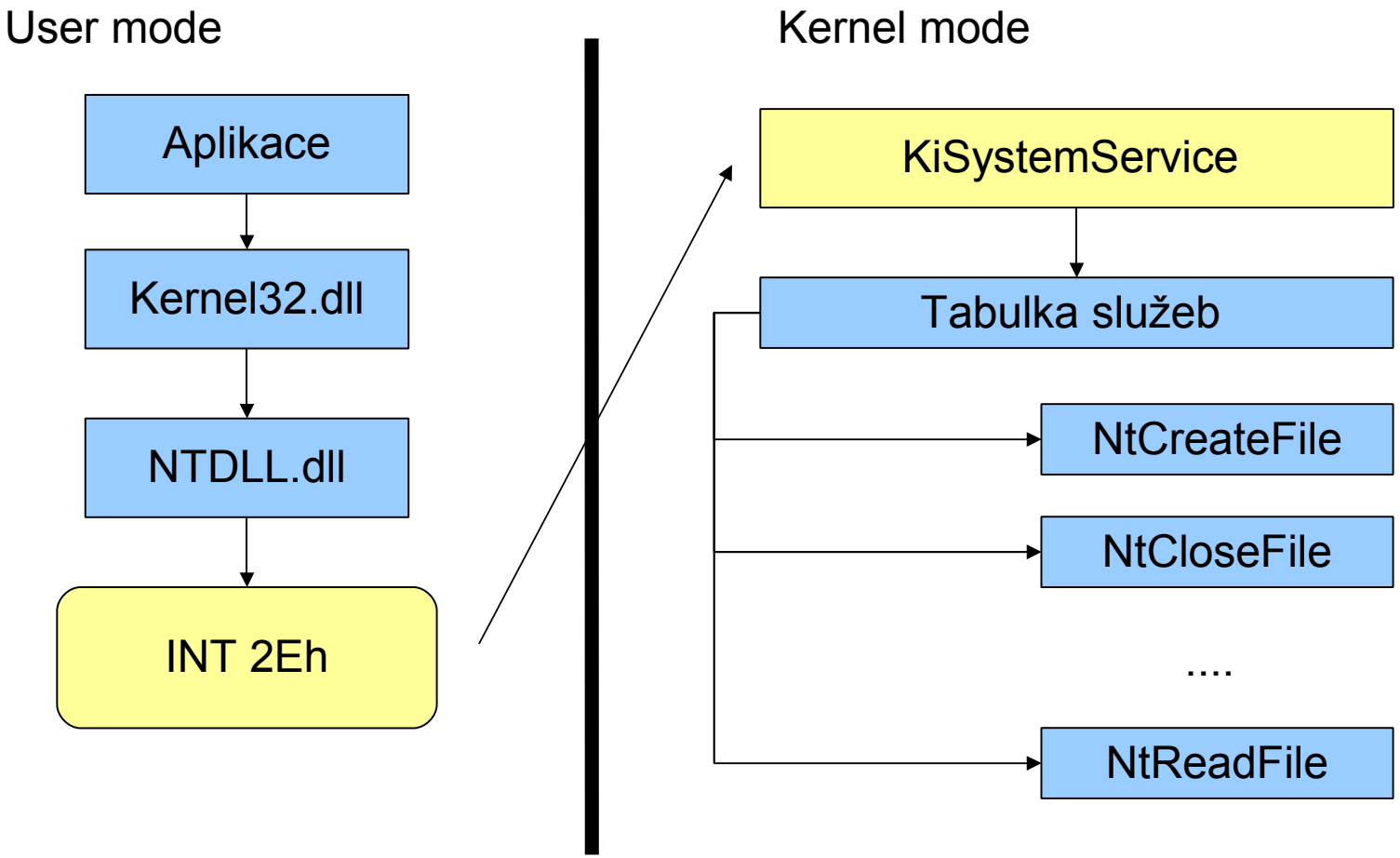
```
mov   eax, 97h
lea   edx, [esp+4]
int   2Eh
ret   10h
```

Native API

- Hodně funkcí se speciálními prefixy
  - Nt, Zw - deklarovány v ntdll/ntoskrnl skok do kernelu.
  - Rtl - rozšiřují C runtime, většinou nepotřebují přepnutí do kernel módu.
  - Csr – Client-Server, komunikace s csrss démonem.
  - Dbg – pomocné funkce pro debugging.
  - Ki – volání z kernel-mode (např. APC dispatching).
  - Ldr – funkce loaderu pro PE soubory.
  - Nls – Native Language Support
  - C runtime – popdpora pro základní fce C runtime - malloc(), strlen(), sprintf() and floor().

Native API

Většina v ntoskrnl.exe a voláná přes nt.dll



WIN32 API

- USER MODE
  - Export funkcí WIN32 API
  - CSRSS.EXE (Client/Server Runtime SubSystem) – např. konzole
  - Client side DLLs (slouží k poslání zprávy serveru nebo přímému zavolání native API)
- KERNEL MODE
  - win32k.sys
  - Implementace daných funkcí

win32k.sys

- GUI a infrastruktura OS
- Zahrnuje kernelovou část od
  - User (GUI)
  - GDI (Grafika)
  - DirectX (akcelerovaná grafika, zvuk, atd.)
  - atd.

WIN32
Soubory

```
HANDLE CreateFile(
  LPCTSTR lpFileName,                             // file name
  DWORD dwDesiredAccess,                          // access mode
  DWORD dwShareMode,                              // share mode
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,     // SD
  DWORD dwCreationDisposition,                    // how to create
  DWORD dwFlagsAndAttributes,                     // file attributes
  HANDLE hTemplateFile                            // handle to template
file
);

BOOL CloseHandle(
  HANDLE hObject   // handle to object
);

BOOL CopyFile(
  LPCTSTR lpExistingFileName, // name of an existing file
  LPCTSTR lpNewFileName,      // name of new file
  BOOL bFailIfExists          // operation if file exists
);

BOOL DeleteFile(
  LPCTSTR lpFileName   // file name
);
```

WIN32 soubory

```
BOOL FlushFileBuffers(
  HANDLE hFile  // handle to file
);

BOOL ReadFile(
  HANDLE hFile,                // handle to file
  LPVOID lpBuffer,             // data buffer
  DWORD nNumberOfBytesToRead,  // number of bytes to read
  LPDWORD lpNumberOfBytesRead, // number of bytes read
  LPOVERLAPPED lpOverlapped    // overlapped buffer
);

BOOL WriteFile(
  HANDLE hFile,                    // handle to file
  LPCVOID lpBuffer,                // data buffer
  DWORD nNumberOfBytesToWrite,     // number of bytes to write
  LPDWORD lpNumberOfBytesWritten,  // number of bytes written
  LPOVERLAPPED lpOverlapped        // overlapped buffer
);

DWORD SetFilePointer(
  HANDLE hFile,                // handle to file
  LONG lDistanceToMove,        // bytes to move pointer
  PLONG lpDistanceToMoveHigh,  // bytes to move pointer
  DWORD dwMoveMethod           // starting point
);
```

WIN32 Adresáře

```
HANDLE FindFirstFile(
  LPCTSTR lpFileName,                // file name
  LPWIN32_FIND_DATA lpFindFileData  // data buffer
);

BOOL FindNextFile(
  HANDLE hFindFile,                 // search handle
  LPWIN32_FIND_DATA lpFindFileData // data buffer
);

BOOL FindClose(
  HANDLE hFindFile   // file search handle
);

BOOL CreateDirectory(
  LPCTSTR lpPathName,                          // directory name
  LPSECURITY_ATTRIBUTES lpSecurityAttributes  // SD
);

BOOL RemoveDirectory(
  LPCTSTR lpPathName   // directory name
);
```

WIN32
Adresáře

```
DWORD GetCurrentDirectory(
  DWORD nBufferLength,  // size of directory buffer
  LPTSTR lpBuffer       // directory buffer
);

BOOL SetCurrentDirectory(
  LPCTSTR lpPathName   // new directory name
);

BOOL MoveFile(
  LPCTSTR lpExistingFileName, // file name
  LPCTSTR lpNewFileName       // new file name
);

DWORD GetTempPath(
  DWORD nBufferLength,  // size of buffer
  LPTSTR lpBuffer       // path buffer
);

UINT GetWindowsDirectory(
  LPTSTR lpBuffer,  // buffer for Windows directory
  UINT uSize        // size of directory buffer
);
```

WIN32 Procesy

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName,                 // name of executable module
  LPTSTR lpCommandLine,                      // command line string
  LPSECURITY_ATTRIBUTES lpProcessAttributes, // SD
  LPSECURITY_ATTRIBUTES lpThreadAttributes,  // SD
  BOOL bInheritHandles,                      // handle inheritance option
  DWORD dwCreationFlags,                     // creation flags
  LPVOID lpEnvironment,                      // new environment block
  LPCTSTR lpCurrentDirectory,                // current directory name
  LPSTARTUPINFO lpStartupInfo,               // startup information
  LPPROCESS_INFORMATION lpProcessInformation // process information
);

HANDLE GetCurrentProcess(VOID);

DWORD GetCurrentProcessId(VOID);

BOOL TerminateProcess(
  HANDLE hProcess, // handle to the process
  UINT uExitCode   // exit code for the process
);
```

WIN32 Thready

```
DWORD GetLastError(VOID);

HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
  SIZE_T dwStackSize,                       // initial stack size
  LPTHREAD_START_ROUTINE lpStartAddress,    // thread function
  LPVOID lpParameter,                       // thread argument
  DWORD dwCreationFlags,                    // creation option
  LPDWORD lpThreadId                        // thread identifier
);
```

Funkce z knihovny jazyka C – lépe používat tuto

```
uintptr_t _beginthreadex(
   void *security,
   unsigned stack_size,
   unsigned ( __stdcall *start_address )( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);
```

WIN32 Thready

```
VOID ExitThread(
  DWORD dwExitCode   // exit code for this thread
);

HANDLE GetCurrentThread(VOID);

DWORD GetCurrentThreadId(VOID);

VOID Sleep(
  DWORD dwMilliseconds   // sleep time
);

DWORD SuspendThread(
  HANDLE hThread   // handle to thread
);

DWORD ResumeThread(
  HANDLE hThread   // handle to thread
);
```

WIN32
Thready - TLS

TLS – Thread Local Storage

```
DWORD TlsAlloc(VOID);

BOOL TlsFree(
  DWORD dwTlsIndex   // TLS index
);

LPVOID TlsGetValue(
  DWORD dwTlsIndex   // TLS index
);

BOOL TlsSetValue(
  DWORD dwTlsIndex,  // TLS index
  LPVOID lpTlsValue  // value to store
);
```

WIN32 Synchronizace

Critical Section

```
VOID InitializeCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection  // critical section
);

VOID EnterCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection  // critical section
);

VOID LeaveCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection   // critical section
);

BOOL TryEnterCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection  // critical section
);

VOID DeleteCriticalSection(
  LPCRITICAL_SECTION lpCriticalSection   // critical section
);
```

WIN32 Synchronizace

Event

```
HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES lpEventAttributes, // SD
  BOOL bManualReset,                       // reset type
  BOOL bInitialState,                      // initial state
  LPCTSTR lpName                           // object name
);

BOOL SetEvent(
  HANDLE hEvent   // handle to event
);

BOOL ResetEvent(
  HANDLE hEvent   // handle to event
);

BOOL PulseEvent(
  HANDLE hEvent   // handle to event object
);
```

WIN32 Synchronizace - další

- Mutex
- Semaphore
- Timed Queue
- Interlocked functions
  - Pro jednoduchou práci s čísly/pointery bez nutností synchronizačních objektů
  - Provádí atomické operace
  - Např.

```
LONG InterlockedCompareExchange(
  LPLONG volatile Destination,  // destination address
  LONG Exchange,                // exchange value
  LONG Comperand                // value to compare
);
```

WIN32 Console Applications

- Emulace konzole
- Má k dispozici dva buffery
  - Screen buffer (to co je vidět na obrazovce)
    - Jsou zde uloženy znaky a jejich atributy (defakto kopie struktury dat VGA karet v textovém režimu)
    - Okno konzole může mít více screen bufferů ale pouze jeden je viditelný
  - Input buffer - fronta pro události od myši, klávesnice, atd.

Console Applications

- API je ve dvou verzích non-Unicode a Unicode
  - Non-Unicode Windows 95 a 98
    - Screen buffer 2 byty na znak (vlastní znak a jeho attributy)
  - Unicode pro Windows NT/2000/XP
    - Screen buffer 4 byty na znak (vlastní znak má 2 byty – UCS2 – dva byty jeho atributy)
  - Non-Unicode verzi lze použít na NT/2000/XP
  - Unicode verzi na 95/98 použít nelze

Console Applications

- I aplikace s GUI může mít konzoli
- Je ji potřeba jen vytvořit
  - Pro použití standardních fcí pro I/O (např. printf) je potřeba asociovat handly (AllocConsole)
- Několik důležitých funkcí
  - `BOOL AllocConsole(void);`
  - `HANDLE GetStdHandle(DWORD nStdHandle);`
  - `BOOL FreeConsole(VOID);`
  - `BOOL GetConsoleScreenBufferInfo(HANDLE hConsoleOutput, PCONSOLE_SCREEN_BUFFER_INFO lpConsoleScreenBufferInfo);`
  - `BOOL GetConsoleMode(HANDLE hConsoleHandle, LPDWORD lpMode);`
  - `BOOL WriteConsole(HANDLE hConsoleOutput,CONST VOID *lpBuffer,DWORD nNumberOfCharsToWrite,LPDWORD lpNumberOfCharsWritten,LPVOID lpReserved);`

Console Applications

ukázka

- Vypíše žlutý text na pozici 10,10

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

void main(void)
{
  HANDLE outH=GetStdHandle(STD_OUTPUT_HANDLE);
  SetConsoleTextAttribute(outH,
   FOREGROUND_INTENSITY |
   FOREGROUND_RED |
   FOREGROUND_GREEN);
  COORD coord;
  coord.X=10;
  coord.Y=10;
  SetConsoleCursorPosition(outH,coord);
  WriteConsole(outH,"Hello!",6,NULL,NULL);
}
```