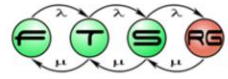
Operációs rendszerek (vimia219)

A Windows operációs rendszer

dr. Micskei Zoltán
http://mit.bme.hu/~micskeiz



Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

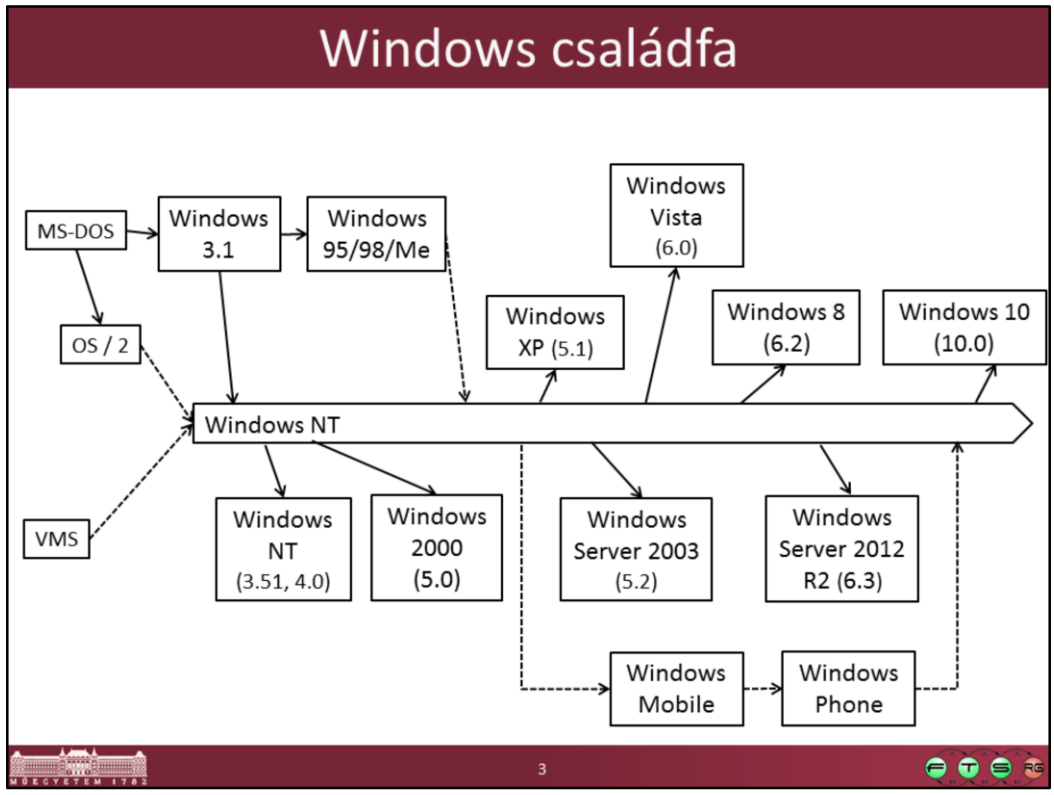
Utolsó módosítás: 2015. 02. 16.

A jegyzetek részben a ---- vonal alatti részek csak érdekességek, nem része a tananyagnak.

Copyright Notice

- These materials are part of the *Windows Operating System Internals Curriculum Development Kit,* developed by David A. Solomon and Mark E. Russinovich with Andreas Polze
- Microsoft has licensed these materials from David Solomon Expert Seminars, Inc. for distribution to academic organizations solely for use in academic environments (and not for commercial use)
- http://www.academicresourcecenter.net/curriculum/pfv.aspx?ID=6191
-

A fóliák részben a Windows Operating System Internals Curriculum Development Kit alapján készültek.



- Az ábra eléggé elnagyolt. Pontosabb családfa: http://www.levenez.com/windows/history.html
- A szerver és kliens verzióknál a forrás nagyjából ugyanaz, csak más az alap paraméterezés (maximális memória, alkalmazásokra vagy szolgáltatásokra van-e optimalizálva)
- További leágazások:
    - Windows Embedded, Windows XP Tablet PC, Xbox, Windows High Performance Computing Server…
- Windows Server 2008 is called SP1. Adventures in doing things right? (http://blogs.msdn.com/iainmcdonald/archive/2008/02/15/windows-server-2008-is-called-sp1-adventures-in-doing-things-right.aspx)
- Windows 7 azért lett állítólag belül 6.1, hogy ne legyen gond a major verzió változással, és a régi alkalmazásoknál ne legyen kompatibilitási gond.

A Windows NT története

- Új operációs rendszer írása 1988-ban
  - Eredetileg: OS/2 3.0
  - Változás: Windows 3.0 utódját elkészíteni
- Megalkotói:
  - Dave Cutler (Digitalnál a VMS tervezője)
- Windows NT név
  - NT = New Technology
  - Windows NT = WNT = ?

Microsoft WindowsNT

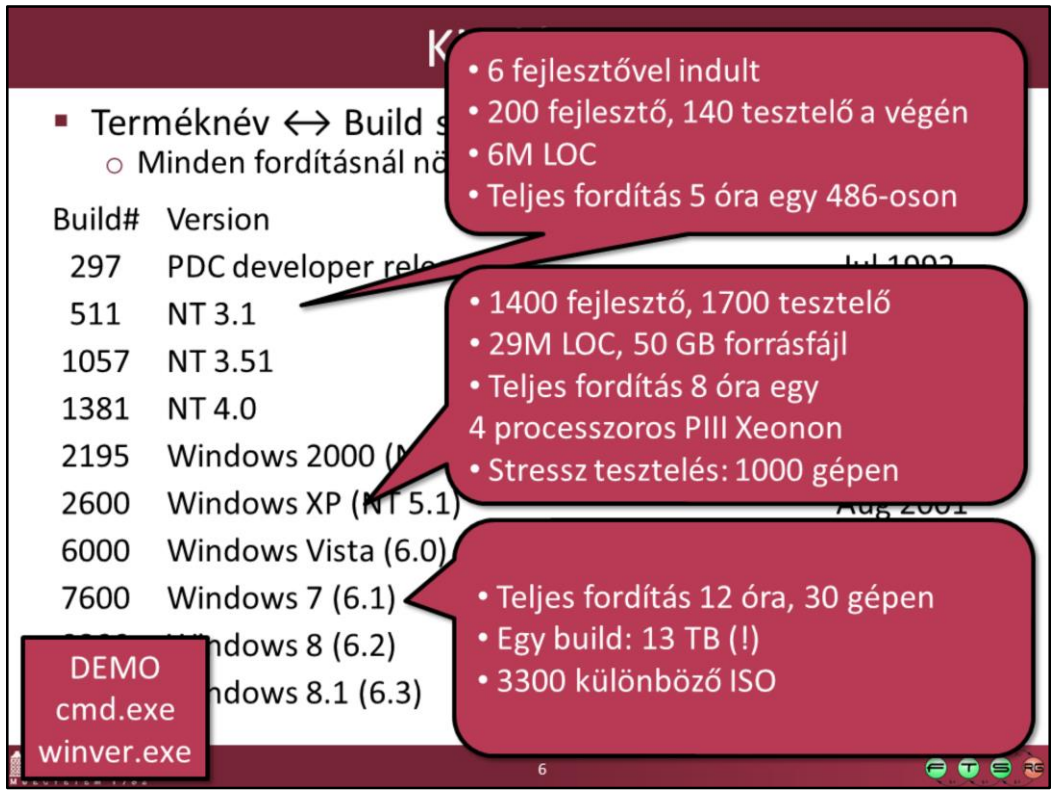
(Mostantól a "Windows" az NT alapú Windowsokat takarja)

VMS+1=WNT állítólag csak véletlen, az eredeti név az Intel i860 processzorából jött, melynek kódneve N10 ('N-Ten').

Kiadások

- Terméknév ↔ Build szám
  - Minden fordításnál növekszik (hetente 5-6 alkalom)

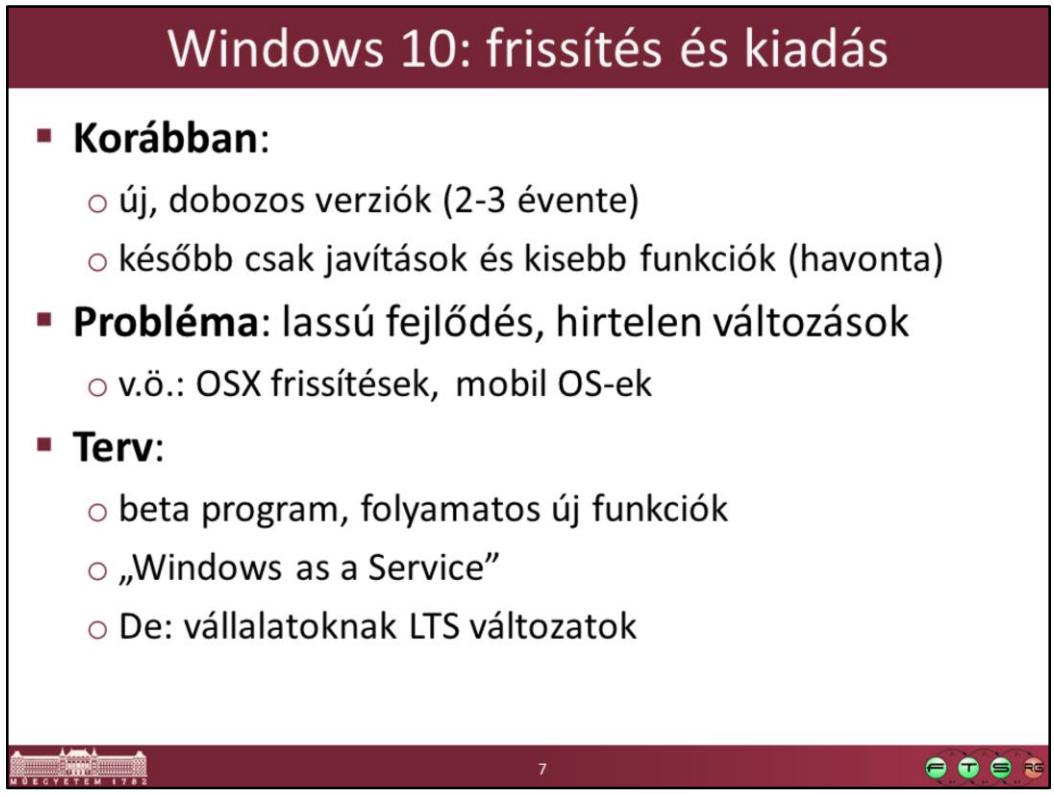
Build#	Version	Date
297	PDC developer release	Jul 1992
511	NT 3.1	Jul 1993
1057	NT 3.51	May 1995
1381	NT 4.0	Jul 1996
2195	Windows 2000 (NT 5.0)	Dec 1999
2600	Windows XP (NT 5.1)	Aug 2001
6000	Windows Vista (6.0)	Nov 2006
7600	Windows 7 (6.1)	July 2009
9200	Windows 8 (6.2)	August 2012
9600	Windows 8.1 (6.3)	October 2013



Nyissuk meg egy cmd.exe-t vagy indítsuk el a winver.exe programot:

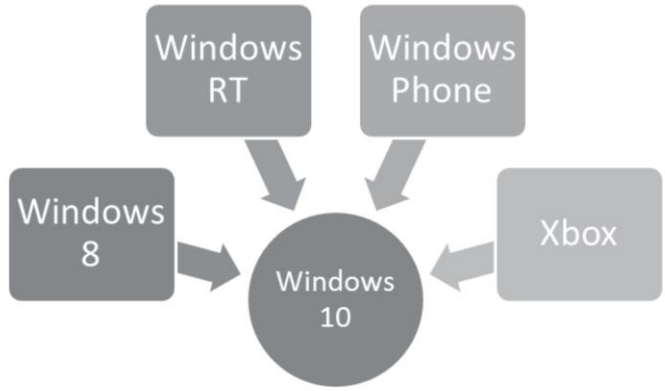
XP SP2:
Microsoft Windows XP [Version 5.1.2600]
Vista RTM:
Microsoft Windows [verziószám: 6.0.6000]
Windows 7 RTM:
Microsoft Windows [Version 6.1.7600]
Windows 8 RTM:
Microsoft Windows [Version 6.2.9200]
Windows 8.1 RTM:
Microsoft Windows [Version 6.3.9600]

- Ha egy termék elkészül és kiadják, akkor új ág (branch) készül hozzá, és a javítások abba mennek. A verziószáma nem változik, legfeljebb service pack telepítése esetén (pl. Windows 7 SP1: 6.1.7601)
- További részleteket találhatunk a BuildLabEx registry kulcsban, lásd:
  - „How to determine your Windows Server 2008 version”, http://blogs.dirteam.com/blogs/sanderberkouwer/archive/2007/11/01/how-to-determine-your-windows-server-2008-version.aspx
- Egy érdekes interjú a Windows 7 build folyamatáról:
  - „Building Win7 - interview with a Build Engineer”, http://edge.technet.com/Media/Building-Win7-interview-with-a-Build-Engineer/

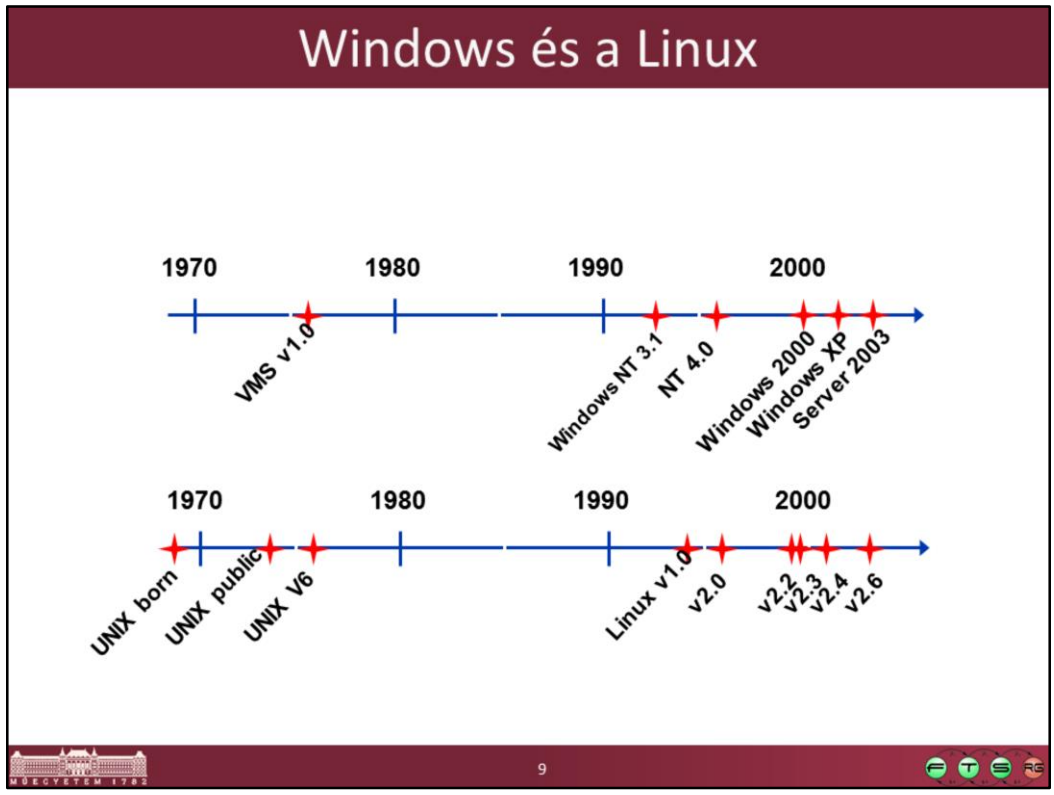


Bővebben lásd például itt:
Ars Technica. „What Windows as a Service and a “free upgrade” mean at home and at work”, 2015-01-31. URL: http://arstechnica.com/information-technology/2015/01/what-windows-as-a-service-and-a-free-upgrade-mean-at-home-and-at-work/

Windows 10: „One OS to rule them all…”



- Közös alapok, Universal Apps…



Mindkét operációs rendszernek az alapjai a ’70-es évekből származnak, és azóta párhuzamosan fejlődnek, folyamatosan hatva egymásra.

Tartalom

- Bevezető
- **Tervezési célok**
- Egyszerűsített architektúra
- (Kevésbé) egyszerűsített architektúra

Kvíz

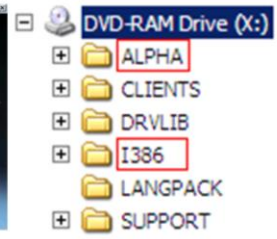
Mi az smss.exe, és miért fut a gépemen?

Miért voltak az XP telepítő CD-n a fájlok nagyrésze egy i386 könyvtárban?

Minek a rövidítése a WoW?

Tervezési célok

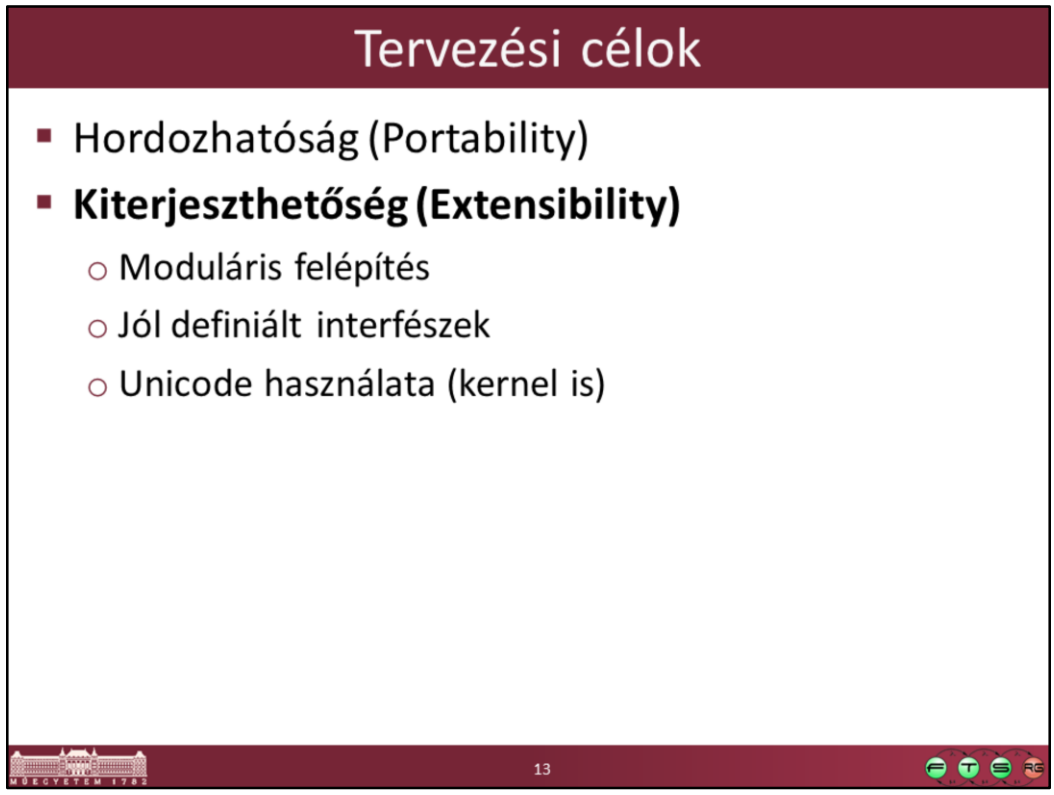
- **Hordozhatóság (Portability)**
  - Többféle processzor architektúra:
    - Kezdetben: Intel x86, MIPS, Alpha, PowerPC
    - Windows XP: Intel x86
    - Windows Server 2003: x86, x64, IA64 (Itanium)
    - Windows 8: x86, x64, ARM*
  - HW specifikus rész elkülönítve
  - Kernel: C nyelven



A Windows NT készítésekor fontos cél volt, hogy többféle architektúrán is fusson az új OS.

• Ezért HW-specifikus rész csak a HAL-ban és a kernel alsó részében (és persze az eszközkezelőkben) van. Assembly nyelven csak a teljesítménykritikus részeket vagy a nagyon alacsony szintű funkciókat írták (pl. megszakítások kezelése, környezetváltás).

• A Windows NT még többféle architektúrára készült, ezért van egy i386 (Intel 386) könyvtár a Windows telepítő CD-n.



Az operációs rendszer belül is modulokból épül fel, amik jól definiált interfészeken keresztül érik el egymás funkcióit.

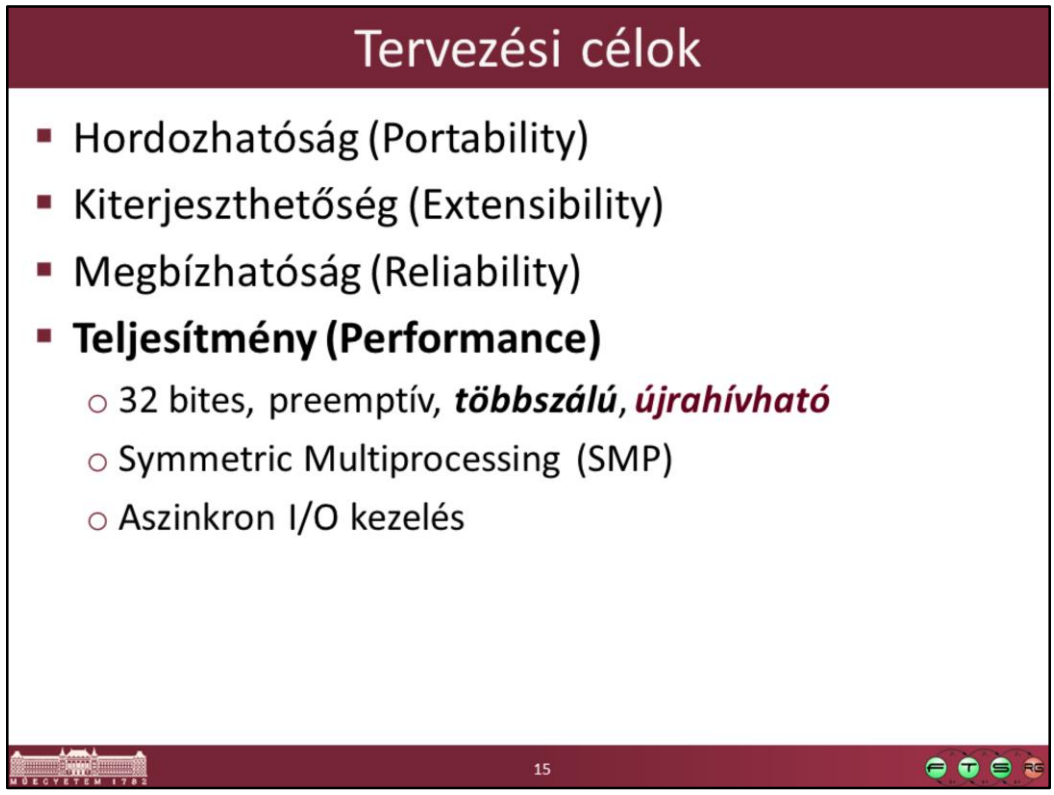
A belső függvények is Unicode stringeket használnak, minden függvénynek két változata van, pl.:

CreateFileA: ANSI karakter paramétert használ (ansi)
CreateFileW: Unicode karakter paramétert használ (wide)

- CreateFile-ból pedig define segítségével vagy az egyik vagy a másik lesz
- CreateFileA belül átalakítja a karaktereket és a W-s változatot hívja, mert a kernel már Unicode stringekkel dolgozik.

Tervezési célok

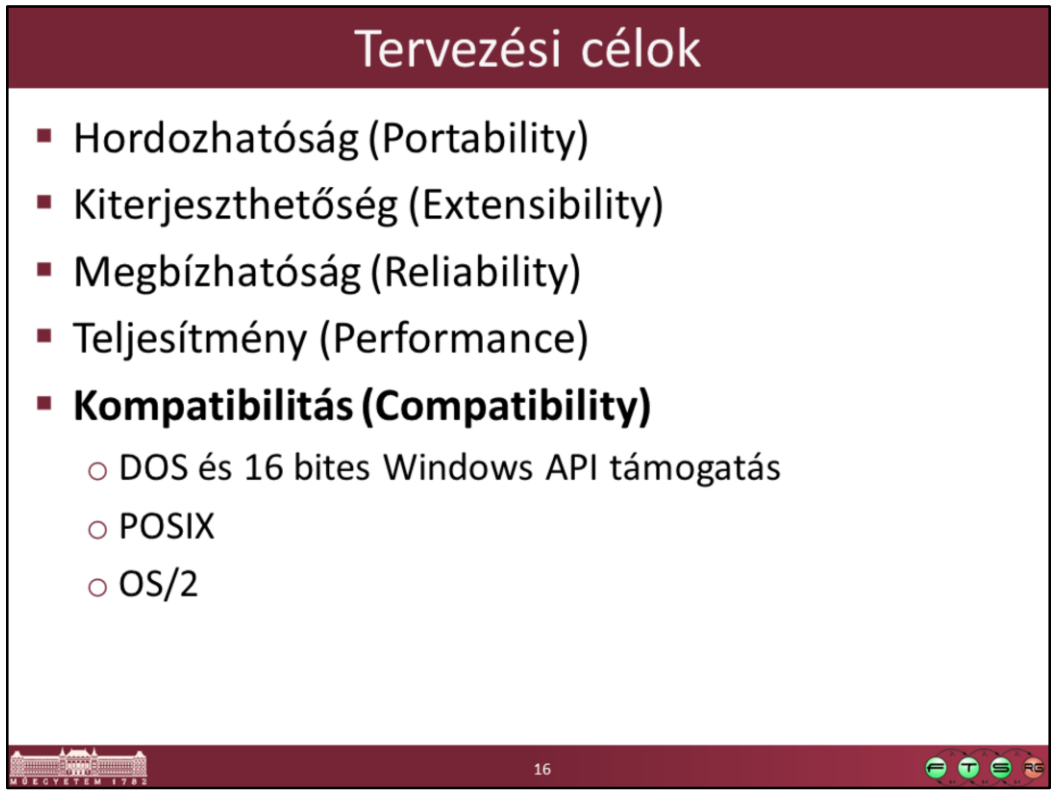
- Hordozhatóság (Portability)
- Kiterjeszthetőség (Extensibility)
- **Megbízhatóság (Reliability)**
  - Windows 3.0: közös címtér mindenkinek
  - Biztonsági szabványok támogatása



Újrahívható (reentrant): a rendszerhívásokat több alkalmazás is meghívhatja egyszerre, nem blokkódnak, ha már valakit éppen kiszolgál az adott rendszerhívás.

Preemptív: egy szálat fel lehet függeszteni futás közben. A Windows kernel **teljesen preemptív**, a kernel szálak is felfüggeszthetőek.

Az OS **eleve többszálúra** lett tervezve, a szál alapvető koncepció a Windowsban.



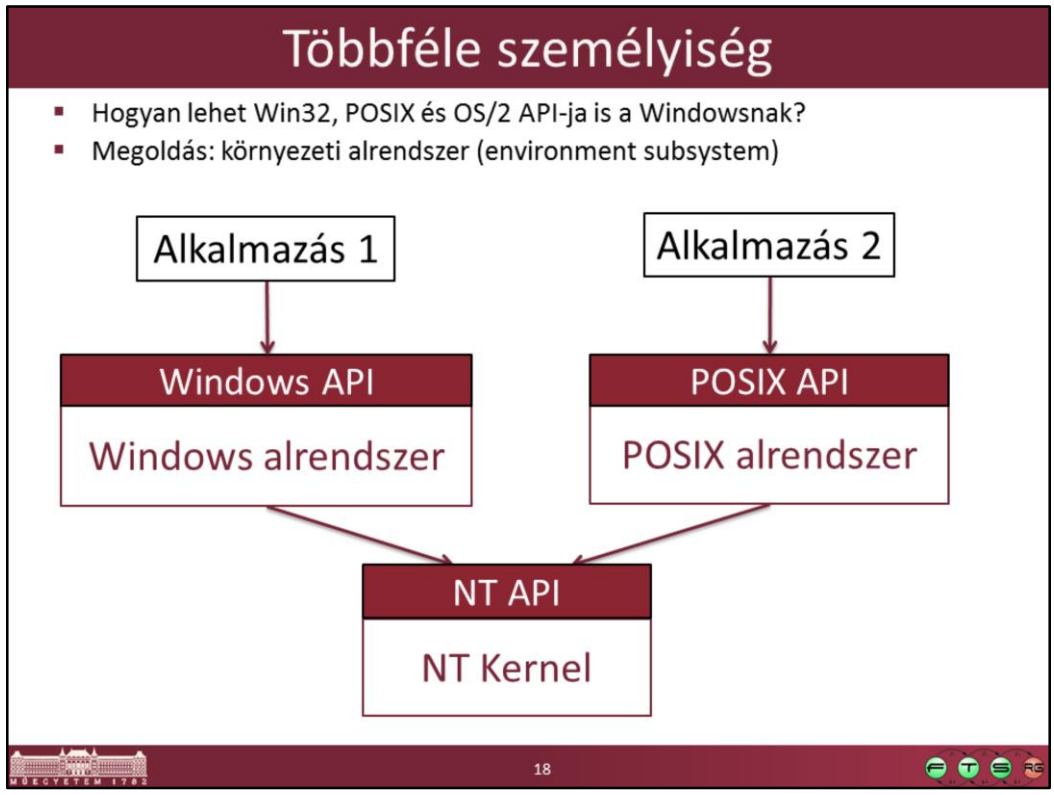
POSIX: azért, hogy megfeleljen a Department of Defense Federal Information Processing Standard (FIPS) 151-2 szabványának

OS/2: IBM-mel együtt kezdték az NT-t kidolgozni, sőt az OS/2 volt az elsődleges API a Windows NT tervezésének kezdetén. (Csak aztán a Windows 3.0 nagy siker lett, a Microsoft és az IBM összeveszett, és a Microsoft az NT-t fejlesztette tovább, az IBM az OS/2-t, és a Windows lett az elsődleges felület.)

Application Programming Interface (API)

- Kívülről milyen függvényeken, adatstruktúrákon keresztül érhető el egy alkalmazás
- Pl. Google Maps API
  - GMap2, GDirections…
  - google.maps.GMap2: setCenter(), zoomIn()…
- Operációs rendszer esetén:
  - tipikusan a rendszerhívások elérésére szolgál
  - POSIX: fopen(), fork(), select()…
  - Win32: CreateFile(), ReadFile()…
  - …

Fontos: egy API az csak egy interfész, nem pontos azt mondani, hogy az API kiszolgál kéréseket. Kell lennie mögötte egy komponensnek, aki megvalósítja az API-ban definiált függvényeket, és az a komponens az, aki nyújtja az API-ban definiált szolgáltatást.



A három különböző alrendszer nem csak különböző nevű függvényeket jelent (fopen vs. CreateFile), hanem teljesen **eltérő szemantikájuk** is van. Például POSIX esetén a fájlnévben számít a kis és nagybetű, Windows esetén nem; POSIX szál vs. Windowsos szál, más tulajdonságaik vannak.

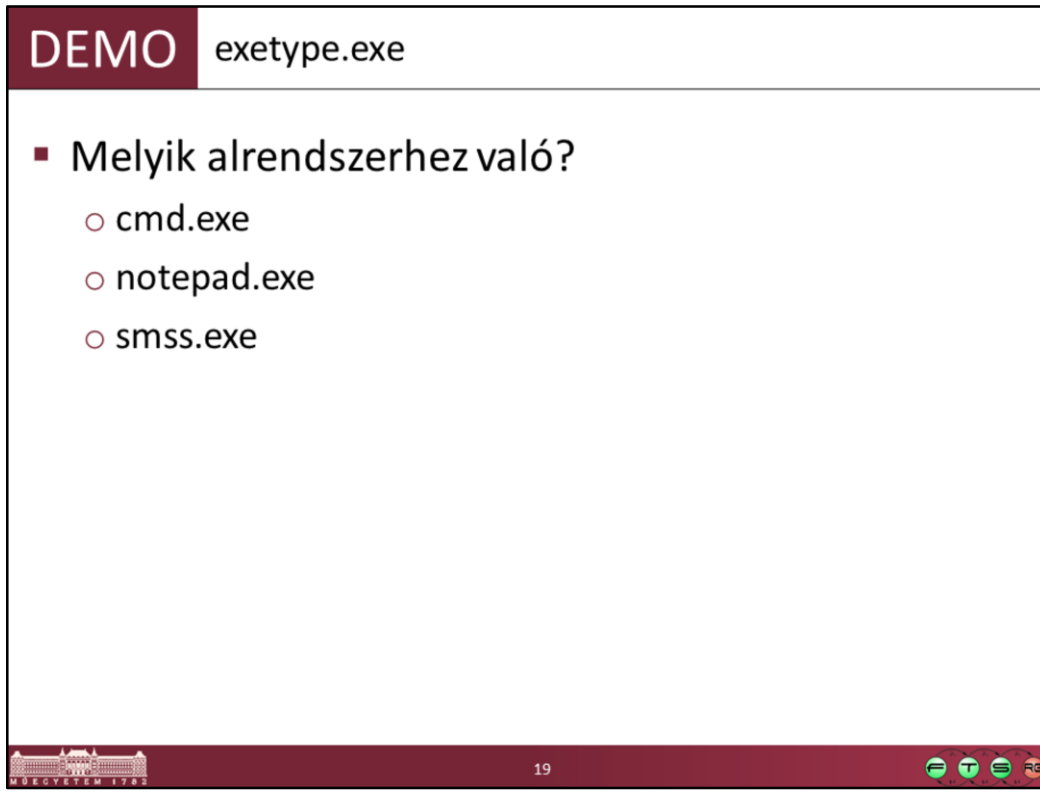
• Windows API és POSIX: teljesen **dokumentált**
• NT API: nem dokumentált, csak egyes részei az eszközmeghajtó fejlesztéshez (Driver Development Kit). Ami nem dokumentált, az változhat az egyes verziók között.

Windows API: régi neve Win32 API volt, de a 64 bites Windows miatt átkeresztelték Windows API-re, és ez az új közös név

Az alkalmazások viszont **nem keverhetik** az alrendszereket, mindegyik csak egyet használhat; ezt linkeléskor kell eldönteni.

- Eredetileg: Windows, POSIX, and OS/2
- Windows 2000: kikerült az OS/2
- Windows XP: csak a Windows maradt
    - POSIX alrendszer helyett Services for Unix kiegészítés
- Windows Server 2003 R2-ben visszakerült a POSIX: **Windows Subsystem for UNIX-based Applications (SUA) néven**

Kiegészítő: *Egyszerű POSIX alkalmazás Windows alatt* **(http://micskeiz.wordpress.com/2010/06/14/egyszeru-posix-alkalmazas-windows-alatt/)**



Exetype letöltése:
http://www.petri.co.il/download_free_reskit_tools.htm

Eredmény:
C:\tools\exetype>EXETYPE.EXE c:\Windows\System32\cmd.exe
File "c:\Windows\System32\cmd.exe" is of the following type:
Windows NT
32 bit machine
Built for the Intel 80386 processor
Runs under the Windows character-based subsystem

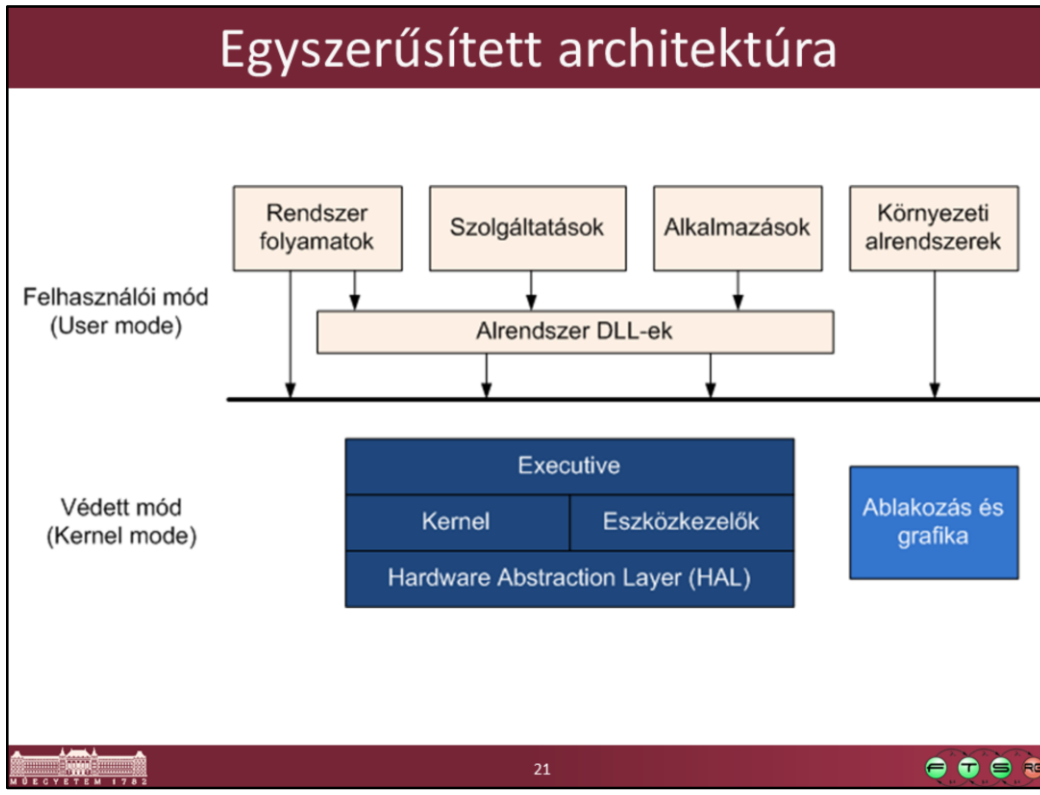
C:\tools\exetype>EXETYPE.EXE c:\Windows\notepad.exe
File "c:\Windows\notepad.exe" is of the following type:
Windows NT
32 bit machine
Built for the Intel 80386 processor
Runs under the Windows GUI subsystem

C:\tools\exetype>EXETYPE.EXE c:\Windows\System32\smss.exe
File "c:\Windows\System32\smss.exe" is of the following type:
Windows NT
32 bit machine
Built for the Intel 80386 processor
Requires no subsystem to run (Native to Windows NT)

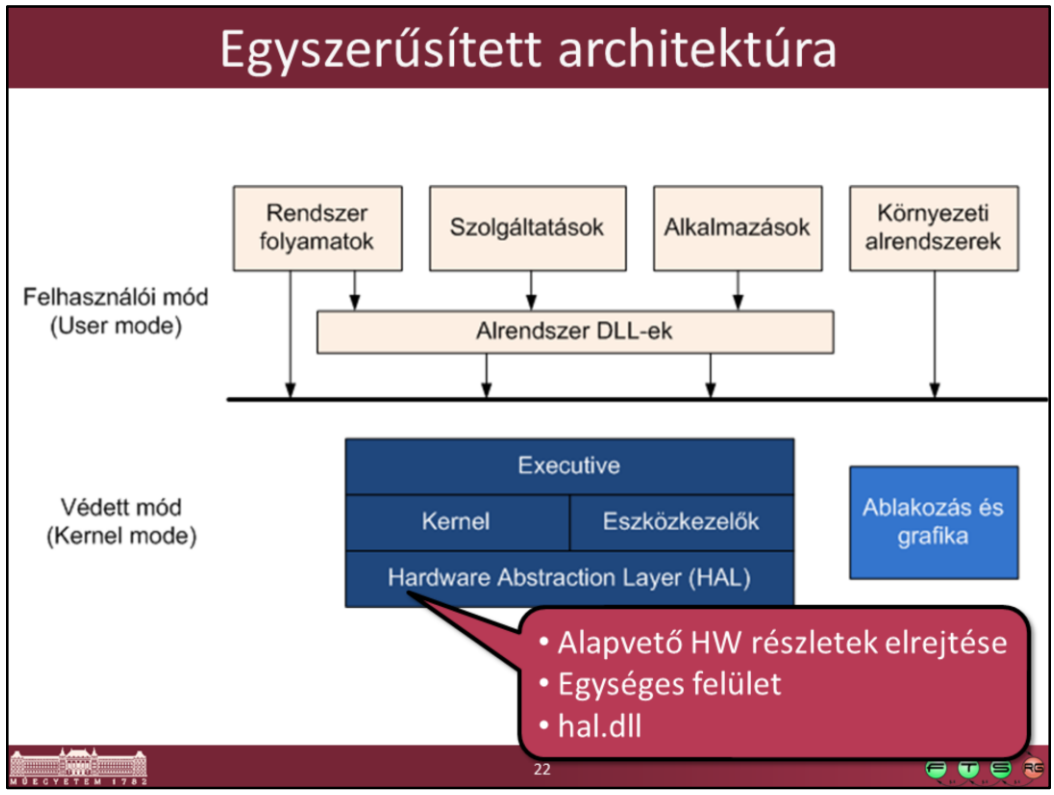
64 bites Windowson (az exetype 32 bites, így annál a c:\windows\system32 automatikusan a c:\windows\sysWOW64 könyvtárra irányítódik át):
C:\temp\exetype>EXETYPE.EXE c:\Windows\sysnative\smss.exe
File "c:\Windows\sysnative\smss.exe" is of the following type:
Windows NT
Requires no subsystem to run (Native to Windows NT)

Tartalom

- Bevezető
- Tervezési célok
- **Egyszerűsített architektúra**
- (Kevésbé) egyszerűsített architektúra

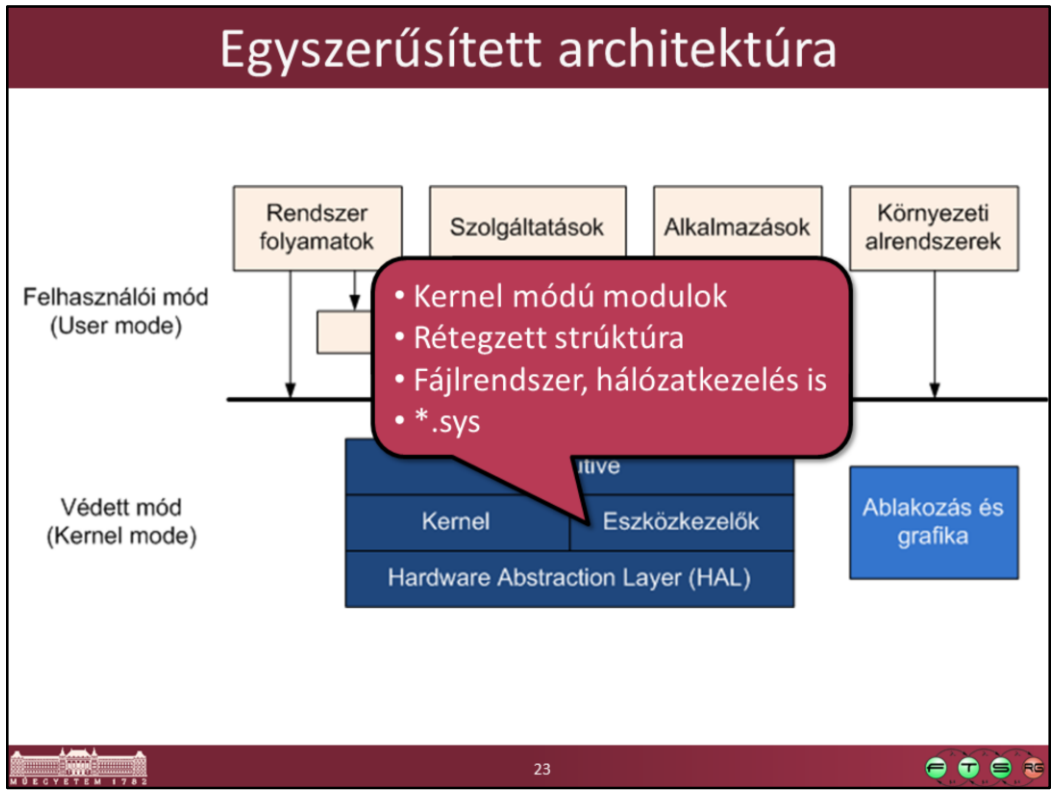


Az NT architektúrája 100 km-ről.

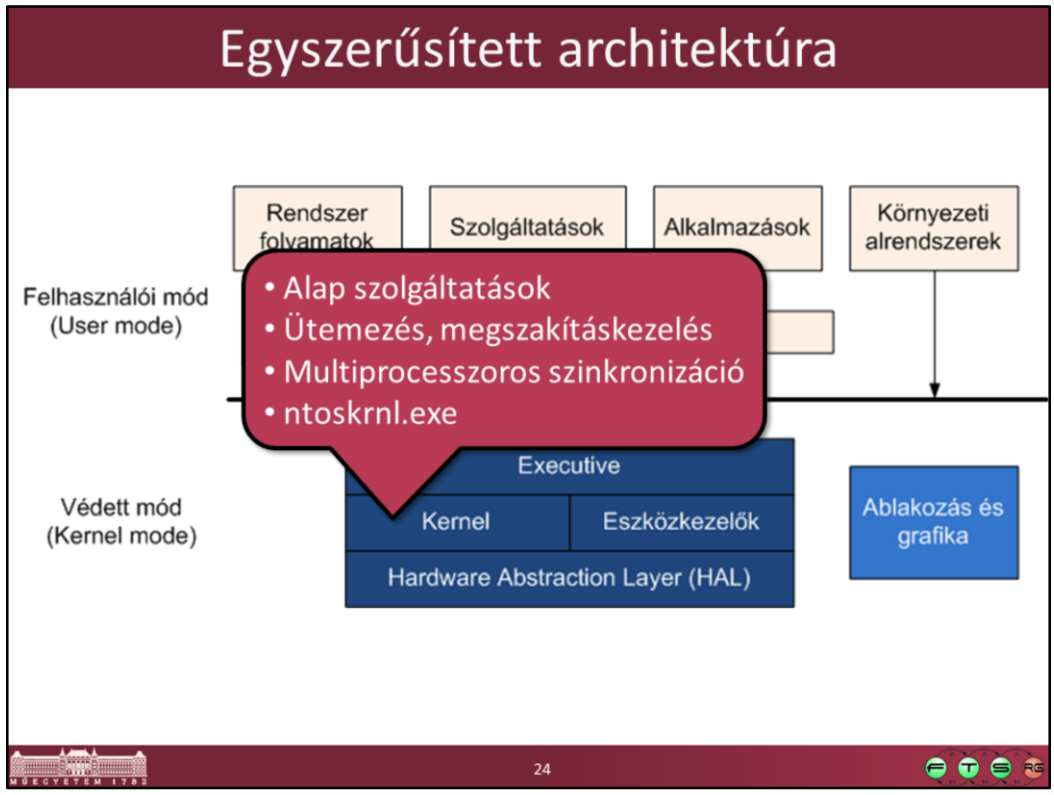


HAL: A felsőbb rétegek a HAL-on keresztül érik el az alap HW szolgáltatásokat (pl. HalGetInterruptVector), a HAL szerepe, hogy elfedje az alapvető HW elemek megvalósításának részleteit, és egy egységes felületet biztosítson.

Megjegyzés: a HAL csak a legalapvetőbb erőforrásokkal és műveletekkel foglalkozik (pl. megszakításkezelő, firmware, CPU-k detektálása és beállítása). Az egyes további eszközök és perifériák kezelését a további rétegekben kell megvalósítani. A HAL függvényei csak abban segítenek, hogy az alap erőforrások kezelését már egyszerűbb rutinok segítségével tudják elvégezni ezek a komponensek.



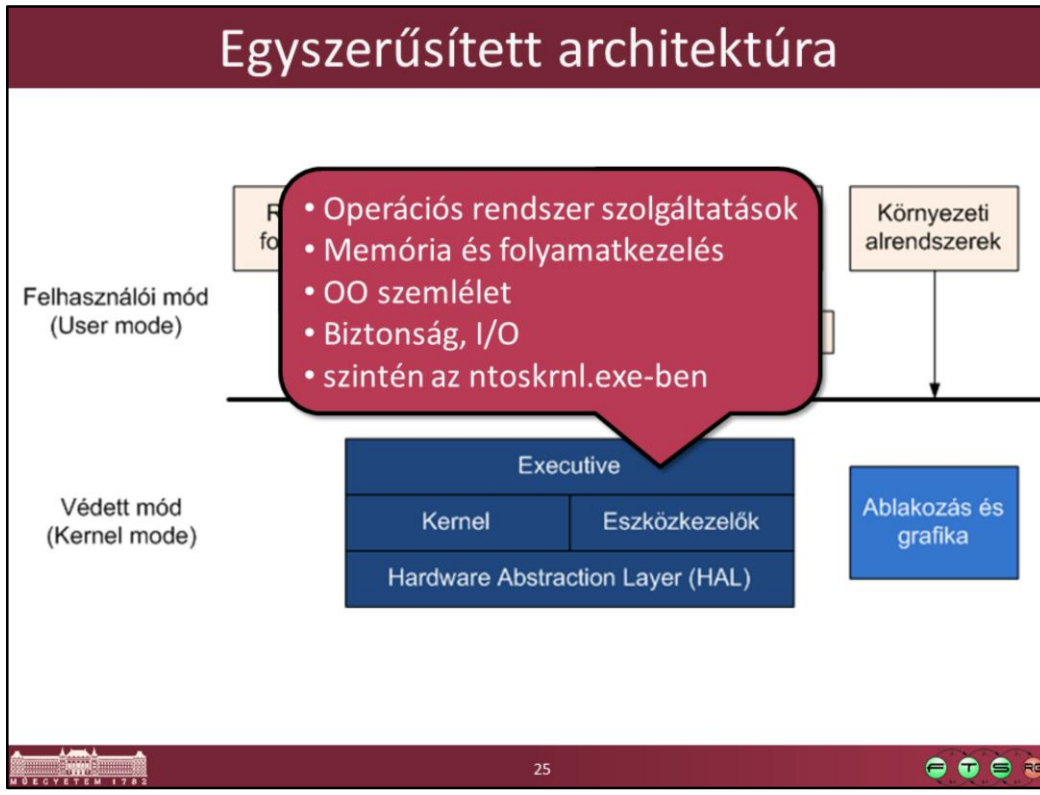
Eszközkelezők (Device driver): kernel módú modulok, melyek az általános kéréseket lefordítják a konkrét eszköznek szóló parancsokra. A Windowsban rétegzett struktúrájú eszközkezelő modell van, az egyes eszközkezelők láncot alkotnak (például, az NTFS fájlrendszer és a merevlemez eszközkezelője közé beilleszthető egy modul, ami hibatűrést, különböző RAID struktúrákat, valósíthat meg transzparensen). Az eszközkezelők sys kiterjesztésű fájlok.



Kernel: az operációs rendszer alapfunkcióit nyújtó komponense (pl. ütemezés, megszakításkezelés). Még ebben a részben is lehetnek hardver specifikus kódrészletek, hisz például a környezetváltás megvalósításához ismerni kell, hogy milyen regiszterei vannak a processzornak. Az ntoskrnl.exe fájl tartalmazza. (Szokás a kernel névvel az összes védett módú komponensre is együtt hivatkozni.)

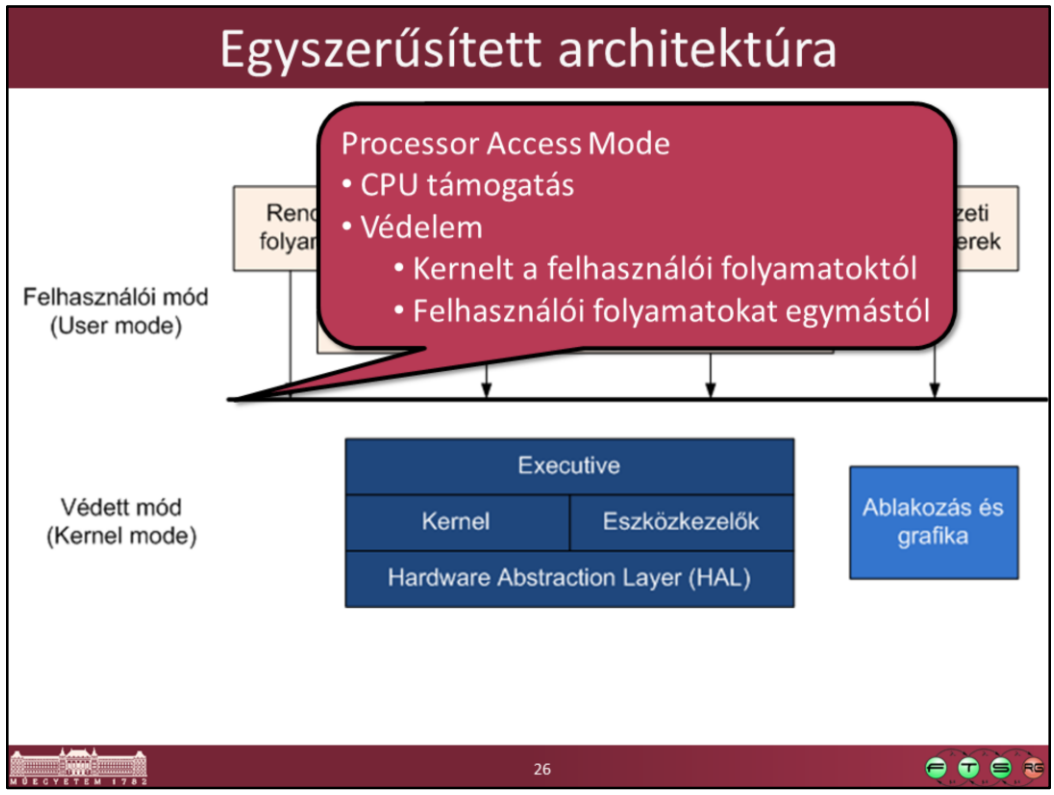
Main services

- Thread waiting, scheduling & context switching
- Exception and interrupt dispatching
- Operating system synchronization primitives (different for MP vs. UP)
- A few of these are exposed to user mode



Executive: Az operációs rendszer magasabb szintű funkcióit szolgáltató rétege (memóriakezelés, biztonság, stb.). Az adatokat objektumokban tárolja, melyeket leírókkal (handle) lehet csak elérni, jól definiált interfészeken keresztül. Bár a kernel funkcióit csak a kernel interfészén keresztül éri el, szintén az ntoskrnl.exe tartalmazza. A legtöbb rendszerhívás itt van megvalósítva.

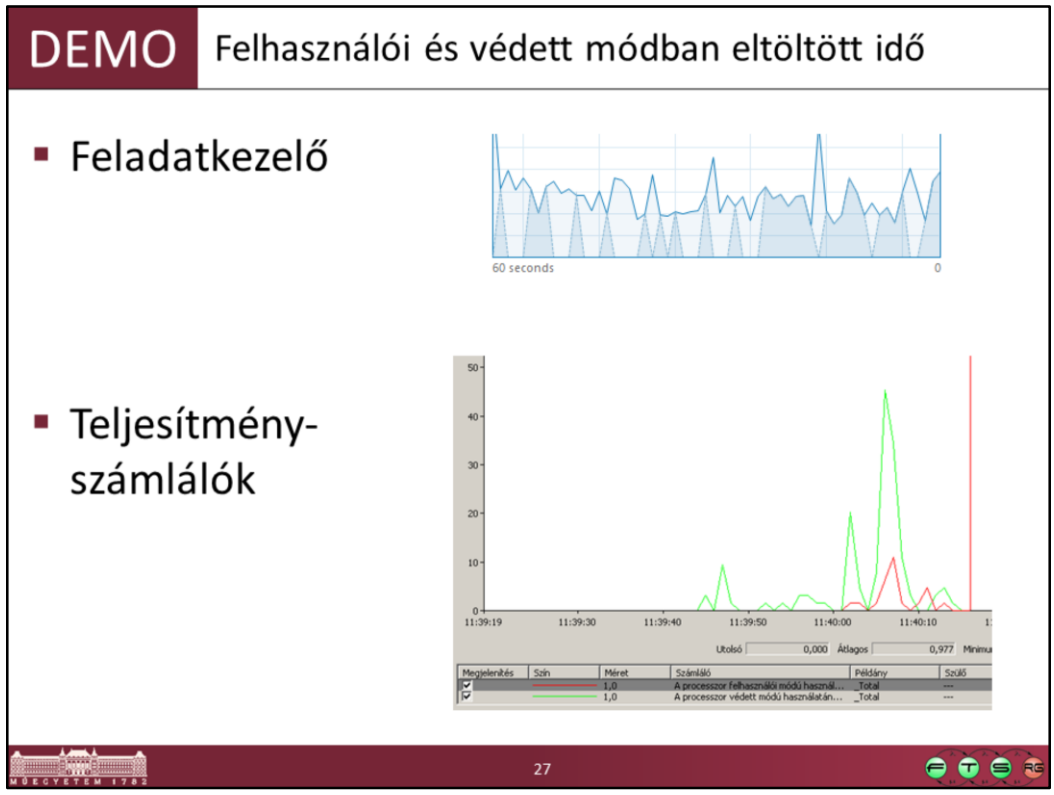
- Komponenseit lásd később
- Nincs igazán elfogadott magyar neve, így Executive-ként fogunk rá hivatkozni



A CPU különböző hozzáférési vagy védelmi módokban működhet, bizonyos CPU utasítások csak a processzor privilegizált módjában adhatóak ki. A kernel a legtöbb jogot biztosító módban fut, a felhasználói alkalmazások pedig egy olyanban, ami sokkal kevesebb mindent enged meg (például nem tudnak hardvereszközökhöz közvetlenül hozzáférni).

• Miért csak **két hozzáférési szint** van:
  • az Intel 386 4 szintet, úgynevezett ring-et, támogat, de
  • a kezdeti architektúrák közül a Compaq Alpha és a Silicon Graphics MIPS csak 2 szintet támogatott,
  • így végül a Windows NT-ben csak két szintet használnak (ennek az a hátránya, hogy pl. egy külső gyártótól származó eszközkezelő vagy kernelmodul is hozzáfér mindenhez).

Fontos: **ez más, mint a környezetváltás** (context switch), ahol elmentjük az éppen futó szál adatait (regiszterek, program számláló, stb.), és betöltünk, majd futtatunk egy újat; itt nem változik, hogy melyik szálat hajtjuk végre.

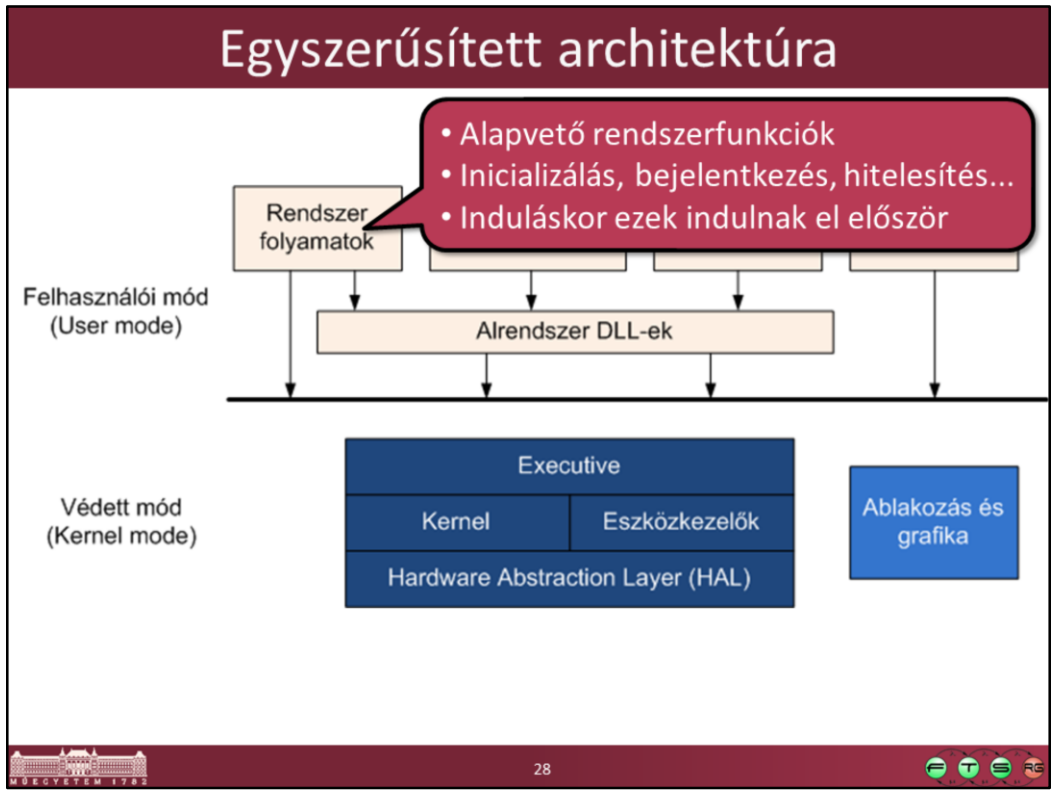


Indítsunk el programokat, pl. WinDbg, Visual Studio
Felhasználói módban végzett műveletekre jó példa egy hosszú számolás, pl. a következő PowerShell kód:

```
$sum = 0; 1..1000000 | % {$sum+=$_}; $sum
```

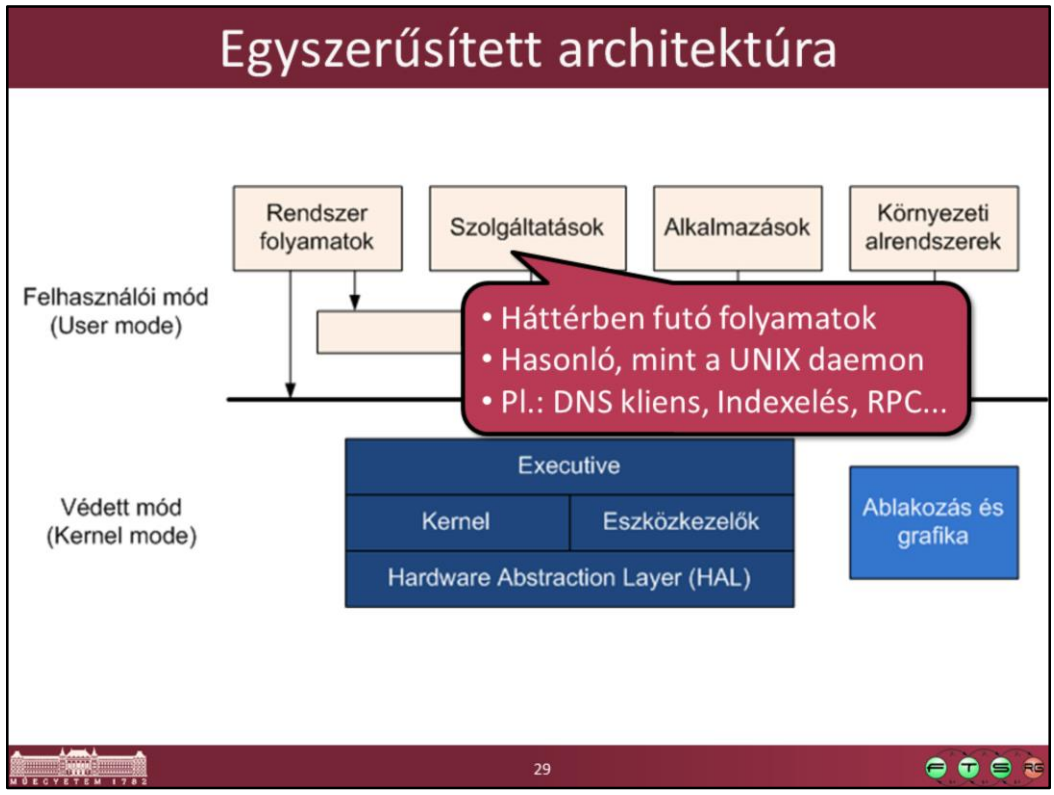
Eszközök:

- Feladatkezelő: Teljesítmény fül, CPU grafikonon jobb gomb / Kernelidők mutatása
- Teljesítményszámlálók (Felügyeleti eszközök / Teljesítményfigyelő):
  - Processzor / A processzor felhasználói módú használatának aránya (%)
  - Processzor / A processzor védett módú használatának aránya (%)



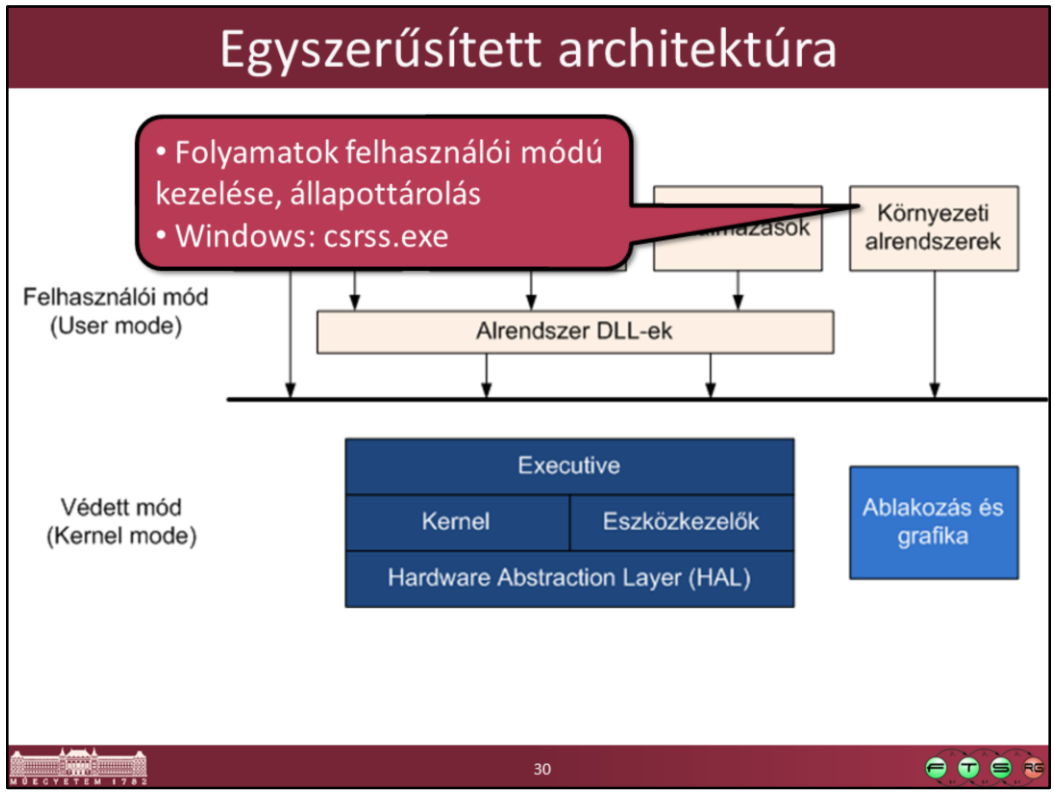
Rendszerfolyamatok (System processes): felhasználói módban futó olyan beépített folyamatok, amik a rendszer futásához szükséges funkciókat valósítanak meg.

Szerepüket lásd később
- Session Manager
- Winlogon
- LSASS (Local Security Authority Subsystem)
- Service Control Manager



Szolgáltatások (Services): olyan folyamatok, amik a felhasználói felülettől és belépéstől függetlenül a háttérben futnak, és kibővítik az operációs rendszer alap szolgáltatásait.

• Szolgáltatások listája: Vezérlőpult / Felügyeleti eszközök / Szolgáltatások

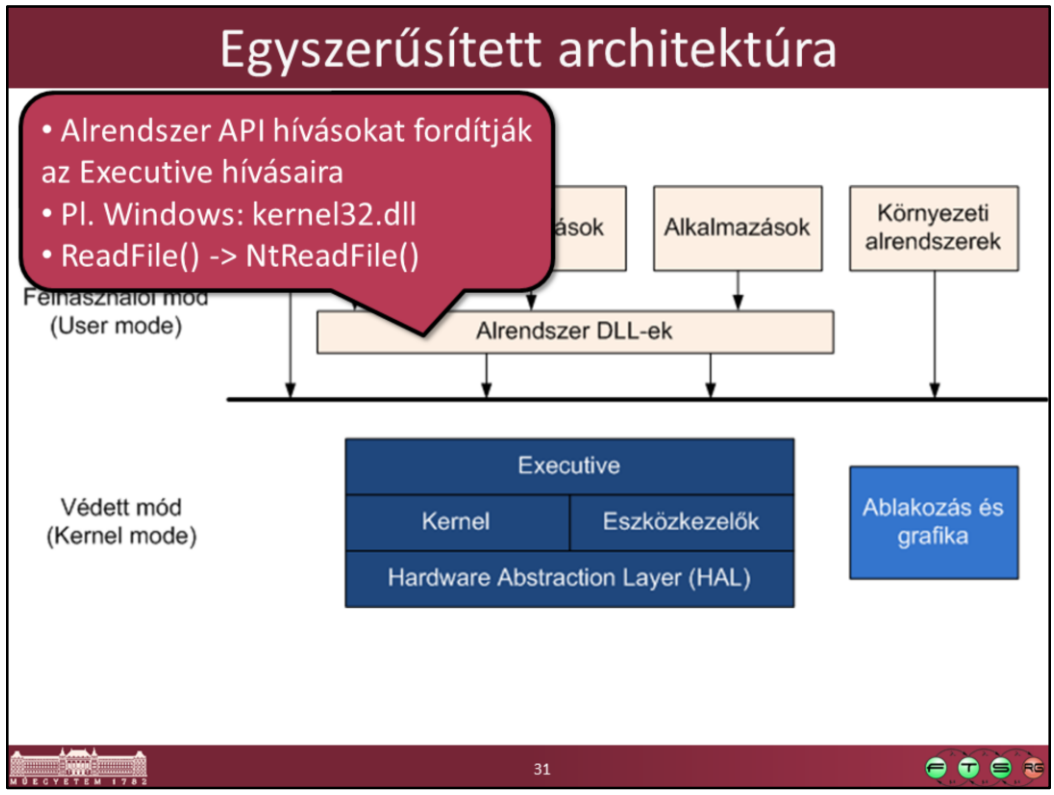


Környezeti alrendszerek (environment subsystems): a felhasználónak vagy programozónak nyújtott környezet, személyiség egy részét a környezeti alrendszer folyamatok valósítják meg, minden egyes környezet külön API-t mutat (Windows, POSIX…), az operációs rendszer rendszerhívásainak egy részét kínálja a felhasználói alkalmazások számára.

• HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems kulcsban tárolódik, hogy milyenek vannak és azok beállításai
• Windows: csrss.exe – **Client/Server Run-Time Subsystems**

  • A Windows alrendszer **kötelező** a rendszer futásához.
  • Miért ez a név: mert eredetileg ez egy általános folyamat volt, ami szálakként futtatta az egyes alrendszereket. Később kikerült a Posix és az OS/2 innen, a Windows maradt, és megtartotta ezt a nevet.
  • Funkciói: konzolos ablak kezelése, folyamat és szál létrehozás/törlés, apróbb függvények (pl. GetTempFile, DefineDosDevice, ExitWindowsEx, hálózati meghajtó csatlakoztatása)

• POSIX (SUA): psxss.exe

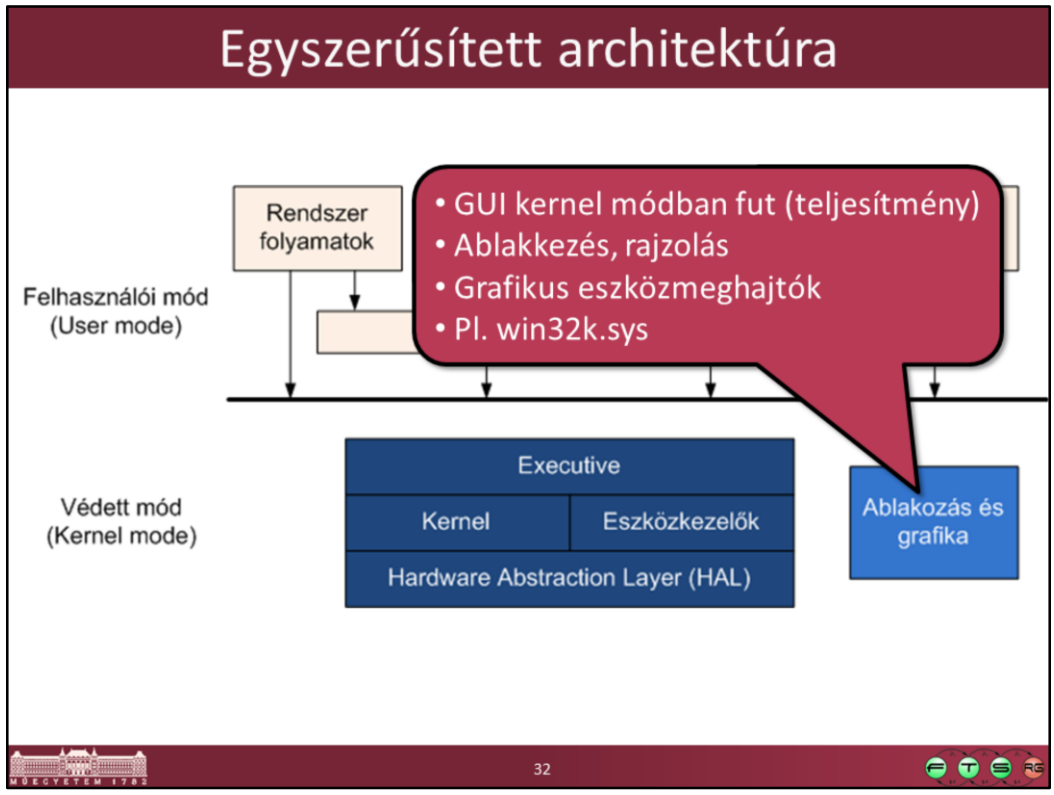


Alrendszerk DLL-ek: az alkalmazások nem hívják közvetlenül az Executive rendszerhívásait, hanem az alrendszer DLL-ek által mutatott függvényeken keresztül érik el azokat.

• Windows: Kernel32.dll, Advapi32.dll, User32.dll, and Gdi32.dll
• Posix: Psxdll.dll

• Nem minden Executive funkció érhető el az alrendszerekből, és az egyes alrendszerek különböző részhalmazt valósítanak meg (pl. a Windowsban nincsen fork).

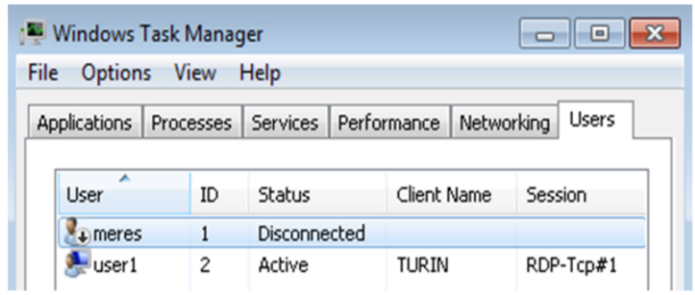
• **Három lehetséges üzemmód**. A hívott alrendszer API függvény
    • teljesen az alrendszer DLL-ben van megvalósítva, így nem kell további hívás.
    • Executive hívást igényel, így tovább kell hívni (az NTDLL.DLL-en keresztül) a rendszerhívások felé
    • az alrendszer folyamat munkáját igényli, ilyenkor üzenetet küld az alrendszer folyamatnak, és megvárja a válaszát



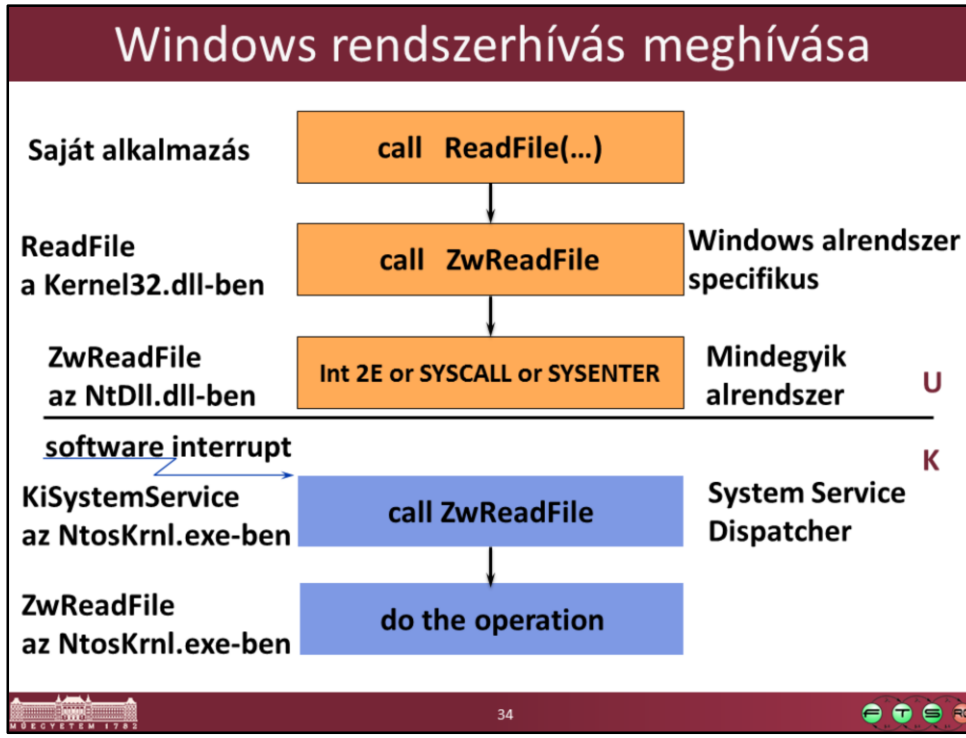
- **Windows NT 4.0-ban került le** kernel szintre ez a komponens, hogy kevesebb környezet és módváltás legyen (Ne kelljen mindig visszaváltani a csrss.exe-be, majd onnan átváltani kernel módba, utasítani a hardvert, visszaváltani felhasználói módba, majd visszaváltani a felhasználói folyamatba, aki kezdeményezte a változtatást.)
- A felhasználói módú folyamatban (csrss.exe) csak a **konzol kezelés** maradt
- Hozzá tartozó alrendszer dll-ek:
  - User32.dll: menük, űrlap vezérlők, ablakok
  - Gdi32.dll: Graphics Device Interface, rajzolás
- **Instabilabb-e a** Windows a kernel módú grafikus komponens miatt? Örök vita tárgya☺
  - Ha hiba van benne, akkor az egész rendszer magával rántja. Ugyanúgy egy hiba a grafikus eszközmeghajtóban is kritikus, nagyon sok kék halál oka a rosszul megírt grafikus eszközmeghajtó.
  - Viszont amíg a csrss.exe-ben volt, egy hiba akkor is tönkretette a teljes GUI-t. A windowsos folyamatok pedig túlságosan építenek a GUI elemekre (Esetleg a Windows Server 2008 Server Core változatában lesz változás:)
  - Azonban emiatt lehetett gyors DirectX-et írni és játszani a Windowson☺

Munkamenet

- Munkamenet (session)
- Több felhasználó bejelentkezve egy gépen
- 0-s munkamenet: rendszerszolgáltatások



Munkamenet (session): egy felhasználói bejelentkezéshez tartozó folyamatokat és rendszerobjektumokat fogja össze.



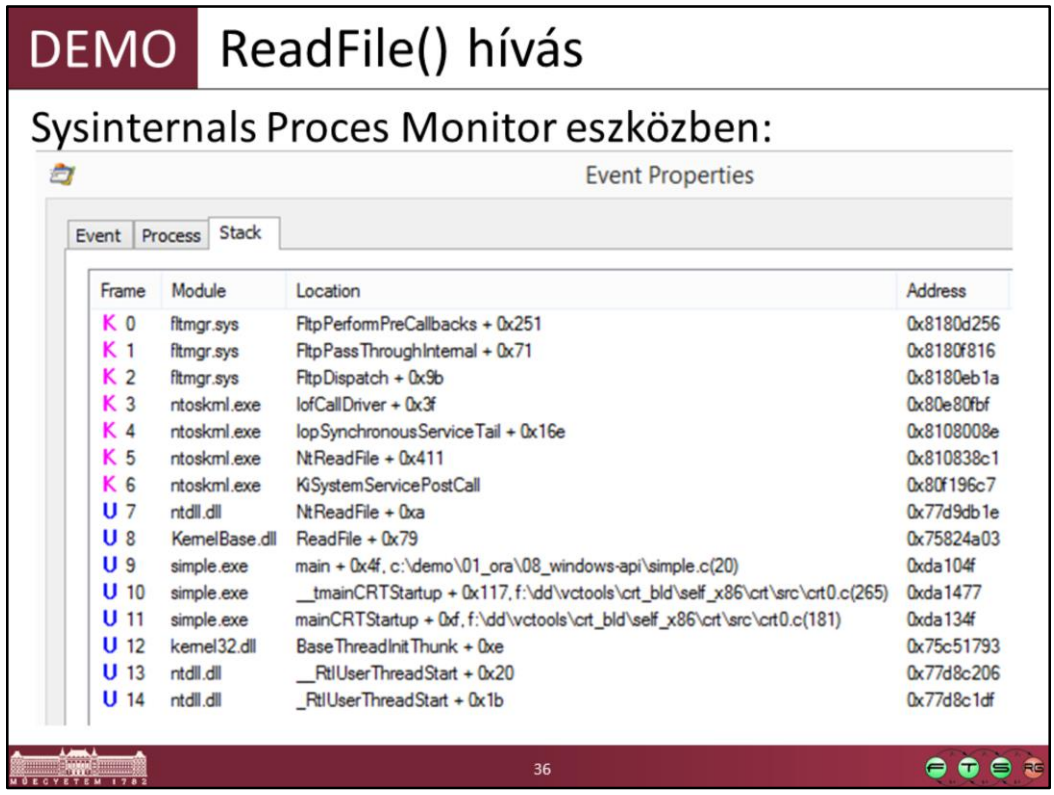
Hogyan is zajlik pontosan egy **rendszerhívás meghívása**:

- Kernel-mode functions ("services") are invoked from user mode via a protected mechanism
  - x86: INT 2E (as of XP, faster instructions are used where available: SYSENTER on x86, SYSCALL on AMD)
  - i.e., on a call to an OS service from user mode, the last thing that happens in user mode is this "change mode to kernel" instruction
  - Causes an exception or interrupt, handled by the system service dispatcher (KiSystemService) in kernel mode
  - Return to user mode is done by dismissing the interrupt or exception
- The desired system function is selected by the "system service number"
  - Every Windows function exported to user mode has a unique number
  - This number is stored in a register just before the "change mode" instruction (after pushing the arguments to the service)
  - This number is an index into the system service dispatch table
  - Table gives kernel-mode entry point address and argument list length for each exported function
- All validity checks are done after the user to kernel transition
  - KiSystemService probes argument list, copies it to kernel-mode stack, and calls the executive or kernel routine pointed to by the table
  - Service-specific routine checks argument values, probes pointed-to buffers, etc.
  - Once past that point, everything is "trusted"
- This is safe, because:
  - The system service table is in kernel-protected memory; and
  - The kernel mode routines pointed to by the system service table are in kernel-protected memory; therefore:
    - User mode code can't supply the code to be run in kernel mode; it can only select from among a predefined list
  - Arguments are copied to the kernel mode stack before validation; therefore Other threads in the process can't corrupt the arguments "out from under" the service

DEMO Windows API használata

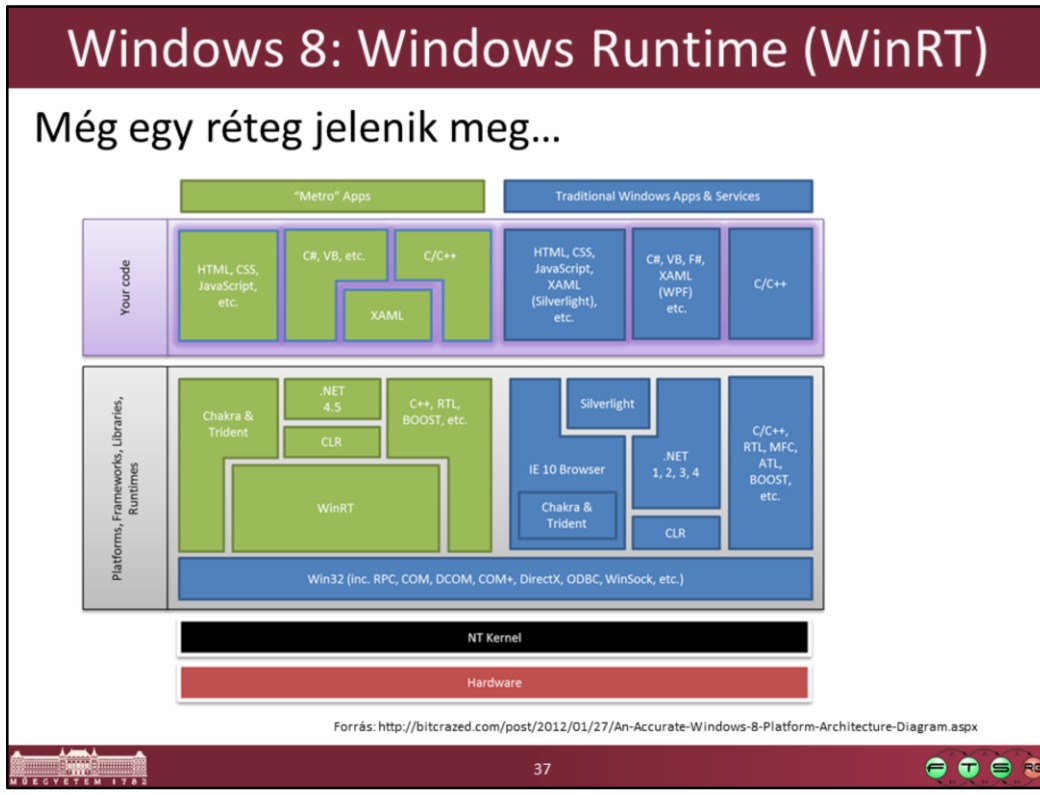
- Windows API függvény
  - Pl. ReadFile
  - SDK-ban dokumentált
- Rendszer szolgáltatás
  - Executive felhasználói módból hívható függvényei
- Windows belső függvények
  - Csak védett módból hívható
- Hívás követése:
  - alkalmazás → kernel32.dll → ntdll.dll
  - Dependency Walker eszköz

- Példák:
  - Windows API: *CreateProcess, CreateFile, GetMessage*
  - Windows system services: *NtCreateProcess*
  - Windows internal routines: *ExAllocatePool*



- Indítsunk el egy Process Monitort
- Futassuk le a példaalkalmazást, amit vizsgálni akarunk
- Váltsunk vissza a Process Monitorra,
    - állítsuk le a rögzítést (mert másodpercenként több ezer eseményt rögzít, így elég sok erőforrást fogyaszt)
    - állítsunk be egy szűrőt, amiben a példaalkalmazás nevére szűrűnk
    - vizsgáljuk meg az alkalmazásunk által generált eseményeket (de már egy egyszerű Hello world típusú alkalmazás esetén több száz lesz)
    - ha kiválasztunk egy eseményt, akkor annak a tulajdonságainál a Stack fülön tudjuk megnézni az adott eseményhez tartozó hívási láncot.

Mit kell az ábrán látni:
- a vermet (stack) lentről felfelé olvasva a legrégebbi hívásoktól haladunk a legfrissebbek felé
- egy-egy elem egy függvényhíváshoz jelzi, hogy hova kell majd visszatérni a hívás után
    - pl. a 8-as: a KernelBase.dll nevű könyvtárban lévő ReadFile függvénynek a függvény kezdetétől számított 0x79 offszeten lévő utasításából hívtuk meg a 7-es sorban lévő függvényt, és amikor az véget ér majd (return utasítás), akkor innen kell folytatni a végrehajtást
    - Ebben a példában most mindenhol látszanak a függvények nevei. Ezt az információt azonban a fordító az optimalizáció során eltávolíthatja, így a bináris fájlból ezt már nem feltétlenül tudjuk megállapítani. Most azért látszik, mert ezokhoz a könyvtárakhoz és fájlokhoz van úgynevezett debug szimbólum fájlunk (pdb kiterjesztésű fájl Windows esetén), ami a függvény címe és neve közötti összerendeléseket tartalmazza.
- 14-12: az operációs rendszer elindítja a simple.exe programot és létrehozza a szálat, amiben majd futnak az utasításai
- 11-10: elindul a vizsgált alkalmazás is, a C Run Time (CRT) végez némi inicializálást
- 9: az alkalmazásnak fut a main függvénye, a simple.c 20. sorában lesz egy függvényhívás, ami
- 8: meghívja a ReadFile Windows API hívást, mivel a program a Windows alrendszert használja, így itt a Windows alrendszer dll-jében van ez (KernelBase.dll)
- 7: az alrendszer dll továbbhív az NT API megfelelő függvényébe, ami az ntdll.dll fájlban van megvalósítva
- 7-6: itt történik meg a processzor módváltása, innentől a K betű jelzi, hogy kernel módban hajtódnak végre azok az utasítások
- 6: a kernelben a SystemServiceDispatcher kezeli a bejövő rendszerhívást
- 5: meghívódik az a függvény, ami a kernelen belül tartalmazza ezt a funkcionalitást (az ntsoskrnl.exe-ben lévő NtReadFile)
- 4-3: előkészítjük a megfelelő I/O hívást
- 2-0: az I/O kezelésért felelős egyik kernelmodul végrehajtja az I/O-műveletet



Windows 8-ban egy új fogalom a Windows Store stílusú alkalmazások, amik alapvetően más grafikus felülettel rendelkeznek és más koncepció szerint kell megtervezni és fejleszteni őket. Ehhez egy teljesen új fejlesztői- és futtatókörnyezet tartozik.

A WinRT nem egy új környezeti alrendszer, hanem a meglévő Windows API-t használó új API, ami annak egy átstrukturált változata, kibővítve mindenféle plusz szolgáltatással (metaadatok, verziózás stb.)

Egy nagyon részletes, érdekes leírás, hogy hogyan jutottunk el Windowson a WinRT-ig:
- Peter Bright. Turning to the past to power Windows' future: An in-depth look at WinRT. URL: http://arstechnica.com/features/2012/10/windows-8-and-winrt-everything-old-is-new-again/

(Kiegészítés: MinWin)

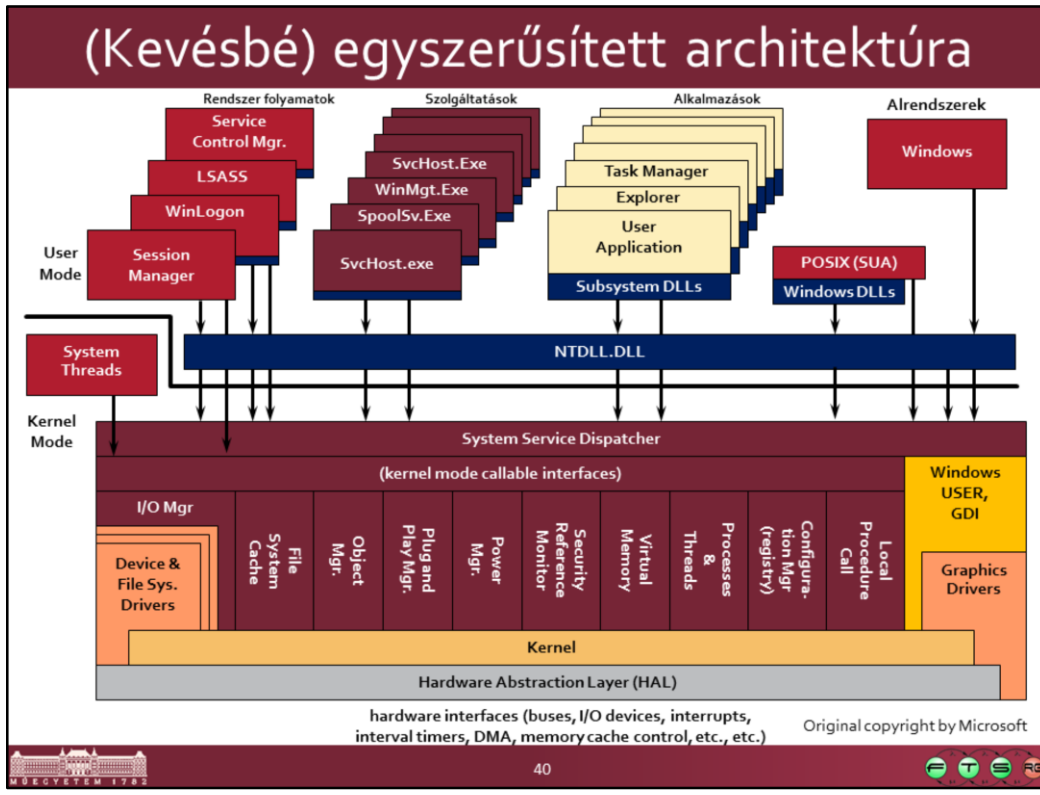
- Monolitikus kernel „rendbe rakása”
  - Belső változtatások, refactoring
  - Függőségek felderítése és átszervezése
  - MinWin: minimális mag, ami önállóan bootolható
- Látható eredmény:
  - kernel32.dll feldarabolása, „virtuális dll-ek”
    - Pl. api-ms-win-core-file-l1-1-0.dll (L1 – layer 1)

További információ:

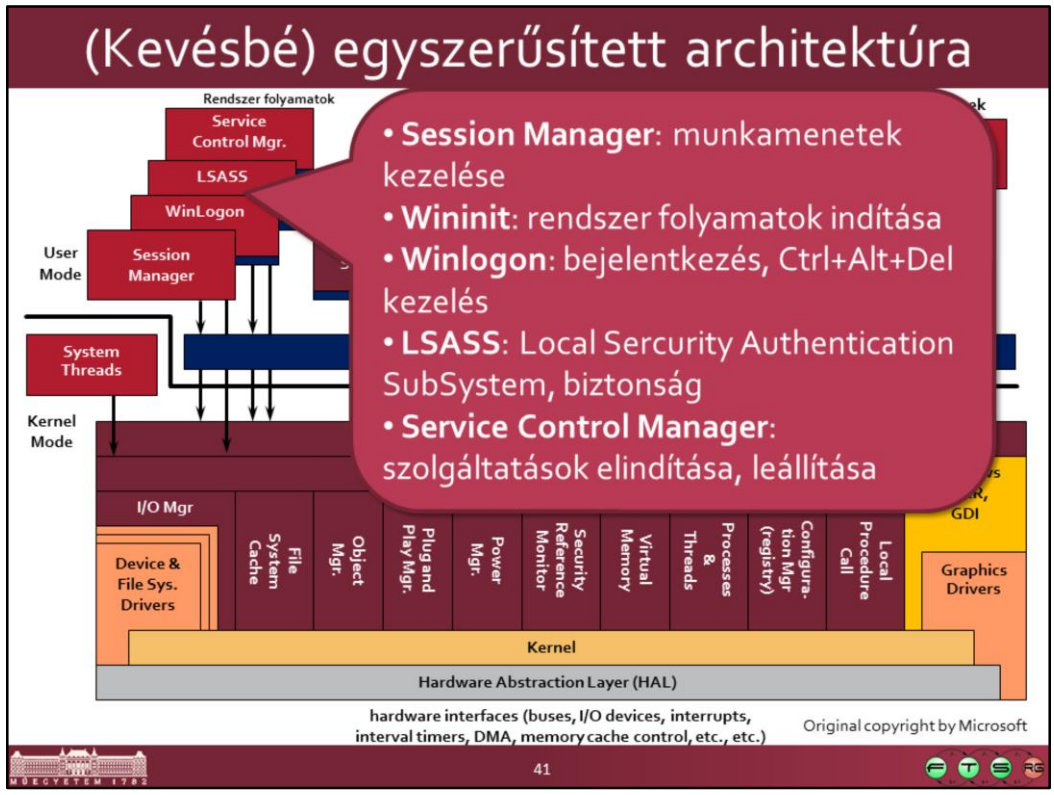
- Mark Russinovich. „Windows 7 and Windows Server 2008 R2 Kernel Changes” URL: http://download.microsoft.com/download/8/C/2/8C21BAFE-3432-48D1-962A-F7A9DD54A2AC/Windows%207%20and%20Windows%20Server%202008%20R2%20Kernel%20Changes.pptx
- Nir Sofer. „Windows 7 Kernel Architecture Changes - api-ms-win-core files”, URL: http://www.nirsoft.net/articles/windows_7_kernel_architecture_changes.html

Tartalom

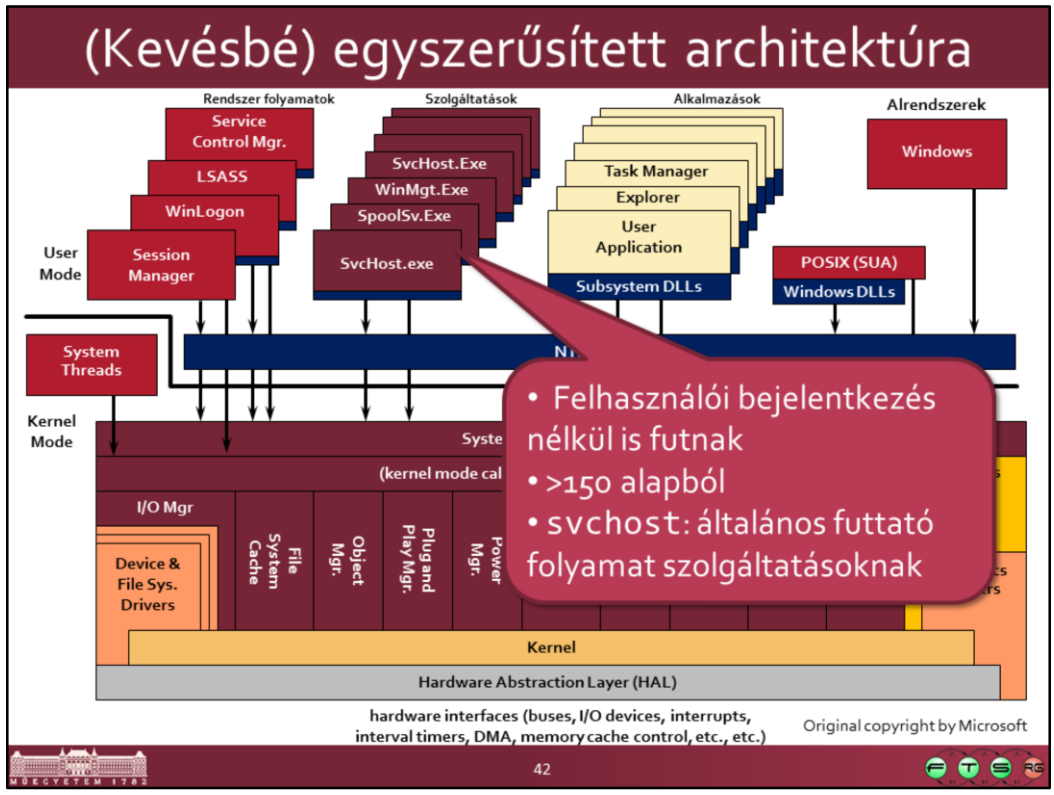
- Bevezető
- Tervezési célok
- Egyszerűsített architektúra
- **(Kevésbé) egyszerűsített architektúra**



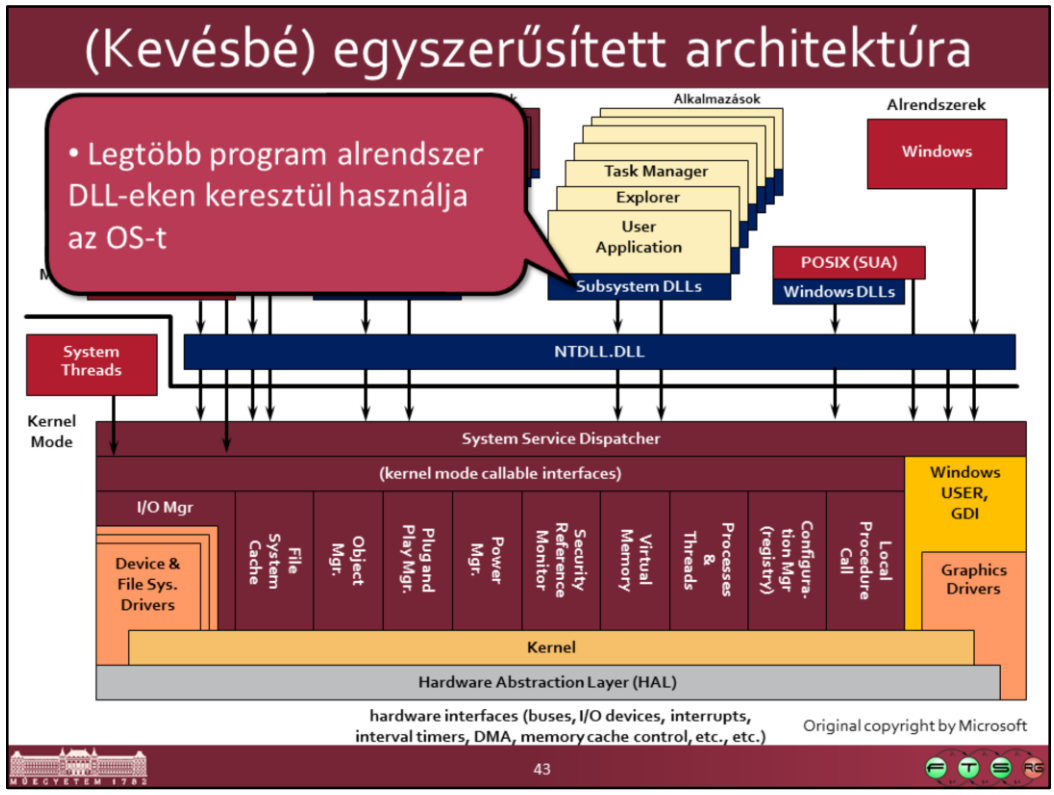
Az NT architektúrája 10 km-ről.



Ezek a folyamatok felelősek azért, hogy miután elindult az operációs rendszer a felhasználók tudják is használni, be tudjanak lépni, stb.

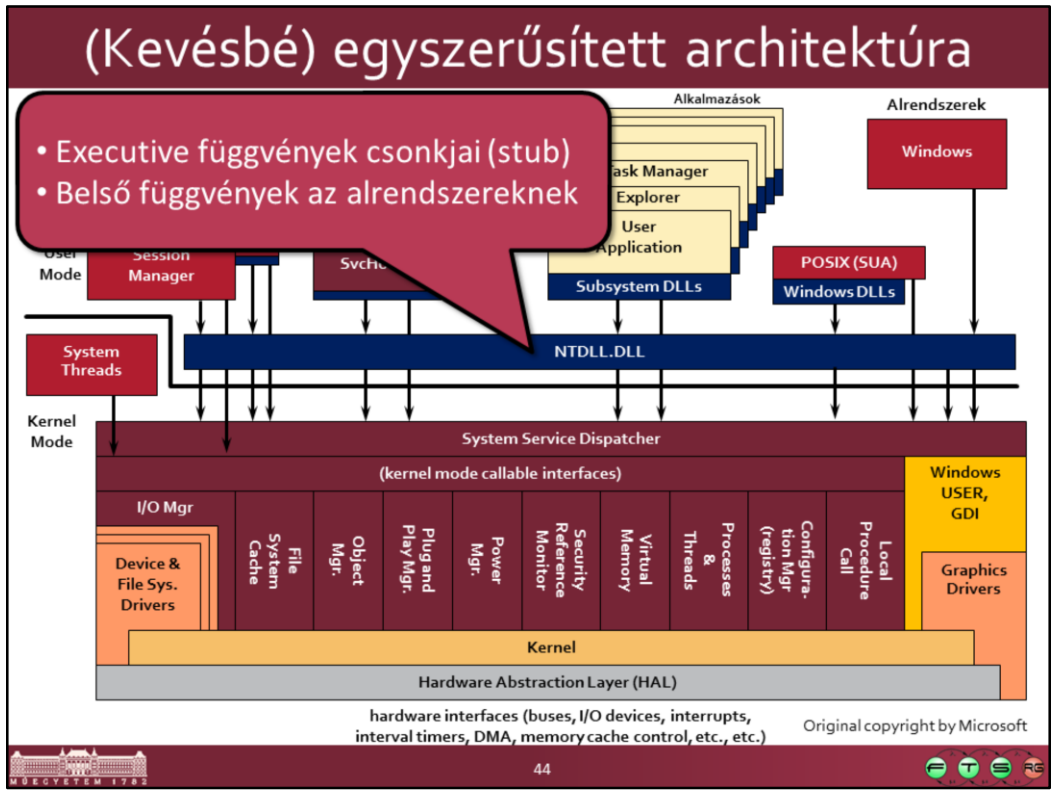


• Nem feltétlenül szükségesek az alap operációs rendszer működéséhez, a felhasználó ezek nélkül is be tud lépni, tud programokat indítani (nem úgy, mint a rendszerfolyamatok esetén)
• Ezek csak olyan programok, amiknek akkor is kell futnia, ha éppen nincs felhasználó bejelentkezve, aki elindítaná őket. Pl. ha elindult a gép, de nem jelentkezett be rajta senki, akkor is kapcsolódni tudjunk a fájlmegosztásaihoz (Server nevű szolgáltatás) vagy be tudjunk távolról lépni rá (Terminal Services szolgáltatás)



• Csak nagyon kevés program van, ami közvetlenül az NT rendszer belső API-ját hívja (az Executive függvényeit az NTDLL.DLL-en keresztül). Ilyen a Session Manager, hisz ő indítja el az alrendszereket, és ilyen maga a windowsos alrendszer. A többiek mind valamilyen alrendszeren keresztül látják csak a rendszerhívásokat.

POSIX: miért szerepel alatta a Windows DLL? Mert az ablakkezelő és GUI függvényeket nem akarták két helyen megvalósítani, így a Windowsosba került a megvalósítás, és a többi csak azt hívja.



NTDLL.DLL tartalma:

•Az executive által kiajánlott függvényeknek megfelelő függvény csonkok:

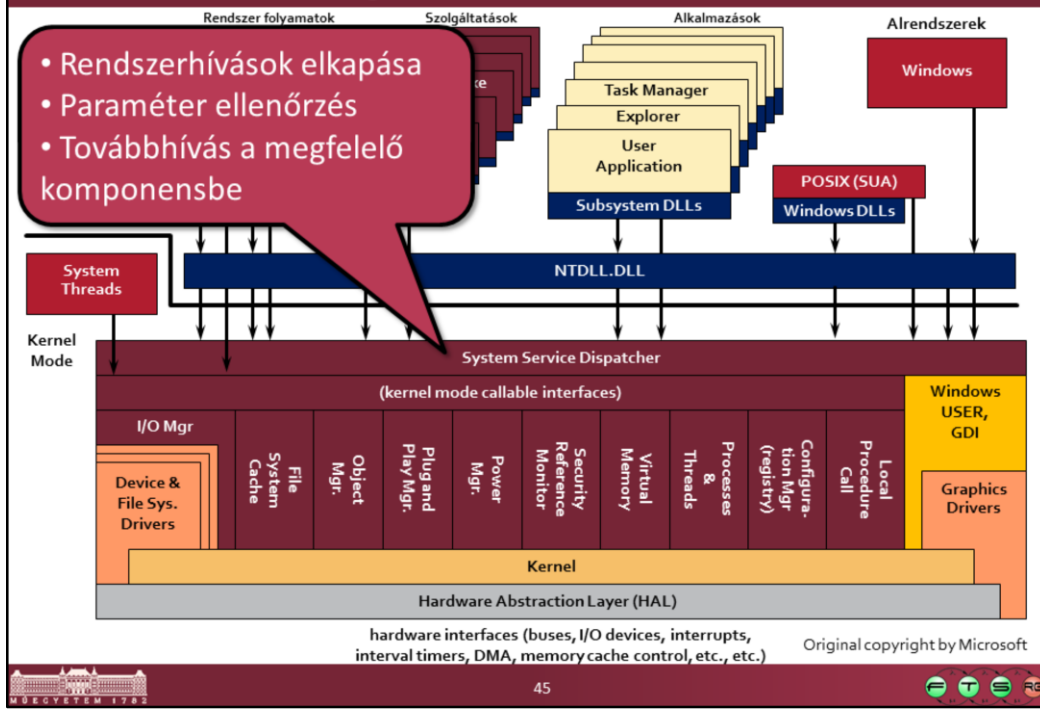
- Ugyanolyan a paraméterezésük, mint az executive-ban lévő párjuknak,
- Elvégzik az átváltást védett módba,
- Átadják a hívást a system service dispatchernek,
- Az ellenőrzi a hívási paramétereket, majd meghívja az executive függvényt.

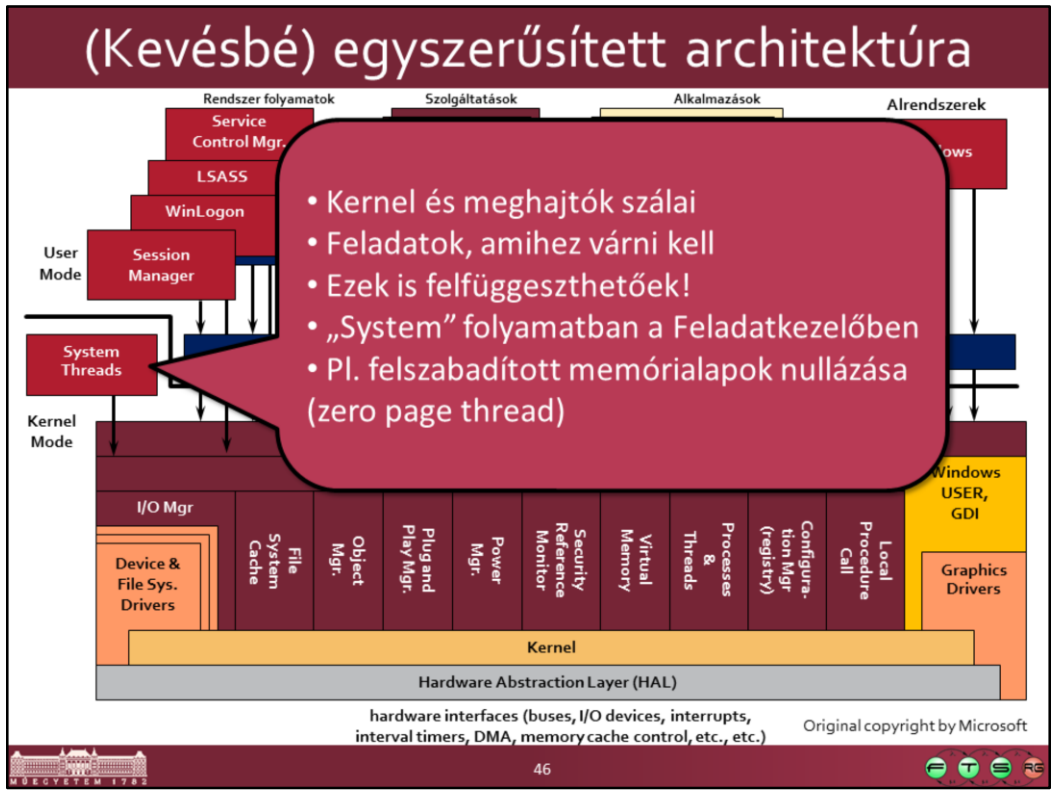
• Ezen kívül van számos függvény az alrendszerek támogatására, pl.

- heap kezelés, image loader.

• NTDLL jó néhány nem dokumentált függvénye:
http://undocumented.ntinternals.net/

(Kevésbé) egyszerűsített architektúra

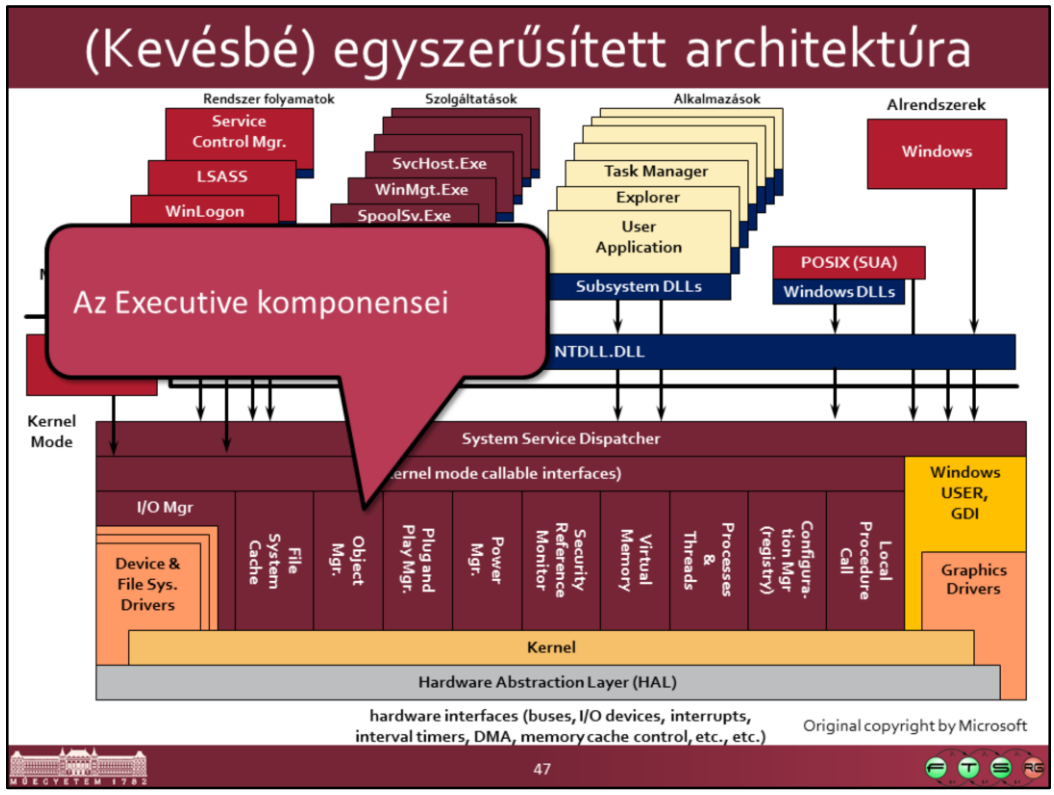




•“System” process (NT4: PID 2, W2K: PID 8, XP: PID 4)

Példák rendszer szálakra:

- Memory Manager: Modified Page Writer for mapped files, Modified Page Writer for paging files, Balance Set Manager, Swapper (kernel stack, working sets), Zero page thread (thread 0, priority 0)
- Security Reference Monitor: Command Server Thread
- Network: Redirector and Server Worker Threads
- Threads created by drivers for their exclusive use: Examples: Floppy driver, parallel port driver
- Pool of Executive Worker Threads: Used by drivers, file systems, …, Work queued using ExQueueWorkItem System thread (ExpWorkerThreadBalanceManager) manages pool



Kiegészítésképp, hogy miket is kell az operációs rendszernek megvalósítania:

• The configuration manager is responsible for implementing and managing the system registry.

• The process and thread manager creates and terminates processes and threads. The underlying support for processes and threads is implemented in the Windows kernel; the executive adds additional semantics and functions to these lower-level objects.

• The security reference monitor (or SRM) enforces security policies on the local computer. It guards operating system resources, performing run-time object protection and auditing.

• The I/O manager implements device-independent I/O and is responsible for dispatching to the appropriate device drivers for further processing.

• The Plug and Play (PnP) manager determines which drivers are required to support a particular device and loads those drivers. It retrieves the hardware resource requirements for each device during enumeration. Based on the resource requirements of each device, the PnP manager assigns the appropriate hardware resources such as I/O ports, IRQs, DMA channels, and memory locations. It is also responsible for sending proper event notification for device changes (addition or removal of a device) on the system.

• The power manager coordinates power events and generates power management I/O notifications to device drivers. When the system is idle, the power manager can be configured to reduce power consumption by putting the CPU to sleep. Changes in power consumption by individual devices are handled by device drivers but are coordinated by the power manager.

• The WDM Windows Management Instrumentation routines enable device drivers to publish performance and configuration information and receive commands from the user-mode WMI service. Consumers of WMI information can be on the local machine or remote across the network.

• The cache manager (explained in Chapter 11) improves the performance of file-based I/O by causing recently referenced disk data to reside in main memory for quick access (and by deferring disk writes by holding the updates in memory for a short time before sending them to the disk). As you'll see, it does this by using the memory manager's support for mapped files.

• The memory manager implements virtual memory, a memory management scheme that provides a large, private address space for each process that can exceed available physical memory. The memory manager also provides the underlying support for the cache manager.

• The logical prefetcher accelerates system and process startup by optimizing the loading of data referenced during the startup of the system or a process.

In addition, the executive contains four main groups of support functions that are used by the executive components just listed. About a third of these support functions are documented in the DDK because device drivers also use them. These are the four categories of support functions:

• The object manager, which creates, manages, and deletes Windows executive objects and abstract data types that are used to represent operating system resources such as processes, threads, and the various synchronization objects.

• The LPC facility passes messages between a client process and a server process on the same computer. LPC is a flexible, optimized version of remote procedure call (RPC), an industry-standard communication facility for client and server processes across a network.

• A broad set of common run-time library functions, such as string processing, arithmetic operations, data type conversion, and security structure processing.

• Executive support routines, such as system memory allocation (paged and nonpaged pool), interlocked memory access, as well as two special types of synchronization objects: resources and fast mutexes.

DEMO Process Explorer

Alap OS komponensek:

- NTOSKRNL.EXE: Executive és a kernel
- HAL.DLL: Hardware abstraction layer
- NTDLL.DLL: Executive hívások csonkjai, belső függvények

Rendszer folyamatok:

- SMSS.EXE: Session manager process
- WINLOGON.EXE: Logon folyamat
- SERVICES.EXE: Service controller process
- LSASS.EXE: Local Security Authority Subsystem

Windows alrendszer, ablakkezelés:

- CSRSS.EXE: Windows subsystem process
- WIN32K.SYS: USER and GDI kernel-mode components
- KERNEL32/USER32/GDI32.DLL: Windows subsystem DLLs

Indítsunk el egy Sysinternals Process Explorert, és azonosítsuk be, hogy milyen folyamatok futnak a gépen.

Ha találtunk olyat, ami nincs a fenti listában, annak próbáljuk kideríteni a szerepét!

Windows verziók

- Ugyanaz a kernel forrás skálázódik
  - 1 CPU, 1 GB memóriától (Starter)
  - 64 CPU, 4 TB memóriáig (Server Datacenter Edition)
- Registry beállítástól függ:
  - Szerver vagy munkaállomás
  - Szerver típusa
- Különbségek
  - Memóriakezelés, ütemezés alap értékei
  - Licencelési korlátok

További témák a félév folyamán

- Windows hibakeresés
- Windows ütemezés
- Windows memóriakezelése
- Windows biztonsági alrendszere

- (Nem Windows specifikus) Virtualizáció

Tools to dig in..

- Windows SDK
  - Platform SDK, .NET Framework SDK utódja
  - C/C++ header, API leírás, fordítók
- Windows Driver Kit
  - Windows DDK utódja
  - C/C++ header, dokumentáció, statikus ellenőrzők
- Windows Debugging Tools
  - User és kernel módú debugger
- **Sysinternals**
  - Mark Russinovich cége (MS megvette)
  - Process Explorer, Filemon, liveKd...
- Windows Support Tools, Windows Resource Kit
- ...

- http://www.microsoft.com/whdc/devtools/WDK/default.mspx
- http://www.microsoft.com/whdc/devtools/debugging/
- http://www.sysinternals.com/

Ajánlott házi feladat

- Töltsük le a **Sysinternals Process Explorer**t
  - Nézzük meg milyen folyamatok futnak
  - Ami nem ismerős, keressünk rá
- Írjunk egy kis **Windows API**-t használó programot
  - Beolvas egy fájlt
  - Windows SDK-ban megtalálható a C fordító és az API leírás
- Lásd a kapcsolódó **virtuális laborokat**!

- Mark Russinovich webcasts: http://technet.microsoft.com/en-us/sysinternals/bb963887.aspx

- Building Windows 8: http://blogs.msdn.com/b/b8/

- Mérés labor 4 segédlet: http://www.mit.bme.hu/4-meres-windows (belépés után érhető el)