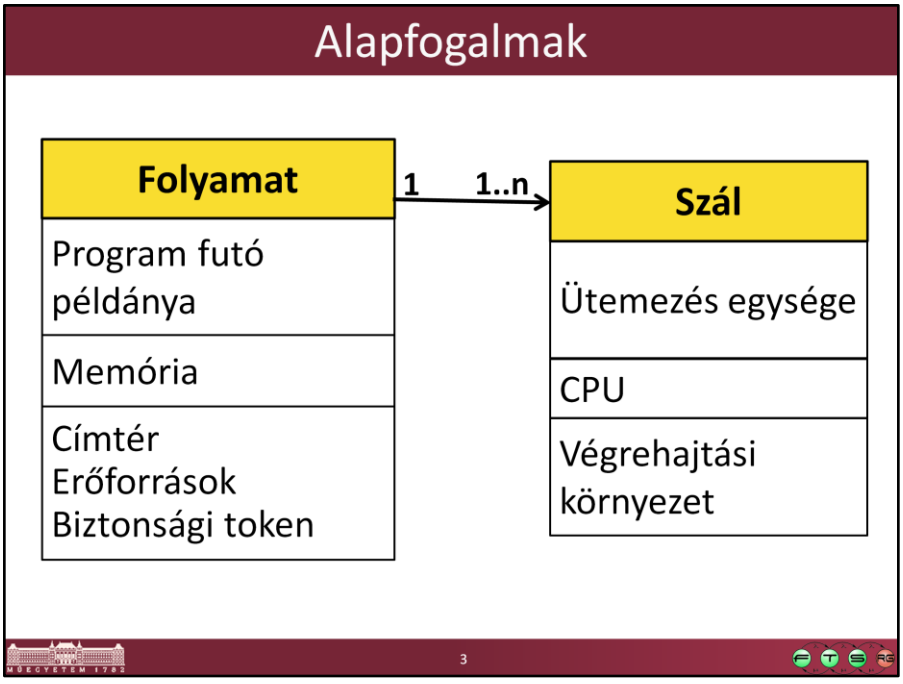
Utolsó módosítás: 2012. 03. 12.

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.



• A program maga a végrehajtható kód.
• A folyamat egy végrehajtás alatt lévő program.
• A folyamat egy szála az, ami éppen fut egy CPU-n, és nem maga a folyamat.
• Minden folyamathoz tartozik legalább egy szál, ami elinduláskor elkezdi futtatni a program main metódusát.

Although programs and processes appear similar on the surface, they are fundamentally different. A *program* is a static sequence of instructions, whereas a *process* is a container for a set of resources used when executing the instance of the program. At the highest level of abstraction, a Windows process comprises the following:

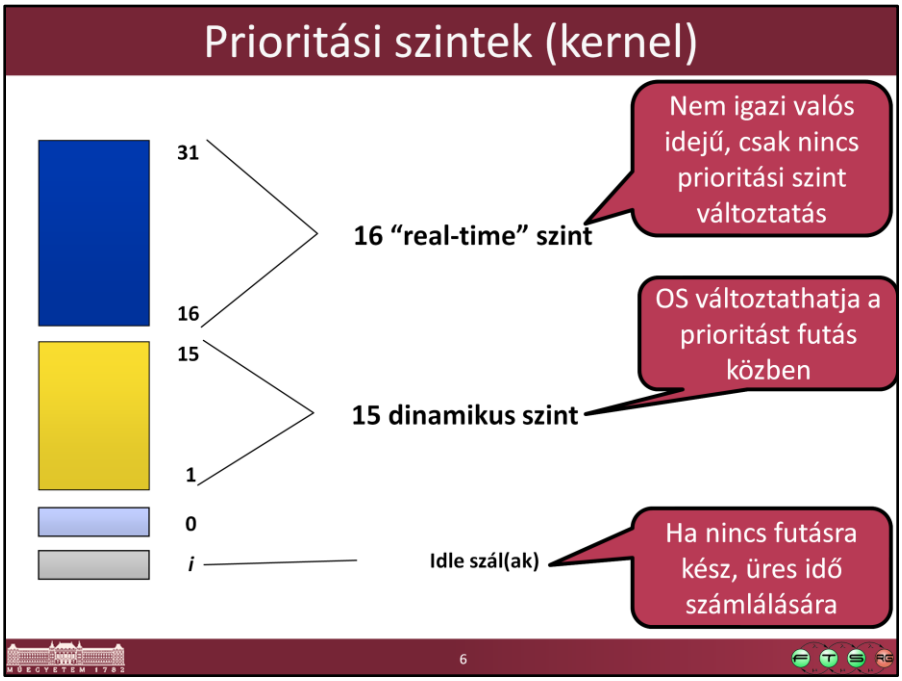
• A *private virtual address space*, which is a set of virtual memory addresses that the process can use
• An executable program, which defines initial code and data and is mapped into the process's virtual address space
• A list of open handles to various system resources, such as semaphores, communication ports, and files, that are accessible to all threads in the process
• A security context called an *access token* that identifies the user, security groups, and privileges associated with the process
• A unique identifier called a *process ID* (internally called a *client ID*)
• At least one thread of execution

A *thread* is the entity within a process that Windows schedules for execution. Without it, the process's program can't run. Although threads have their own execution context, every thread within a process shares the process's virtual address space (in addition to the rest of the resources belonging to the process), meaning that all the threads in a process can write to and read from each other's memory. Threads cannot accidentally reference the address space of another process, however, unless the other process makes available part of its private address space as a *shared memory section* (called a *file mapping object* in the Windows API) or unless one process has the right to open another process to use cross-process memory functions such as *ReadProcessMemory* and *WriteProcessMemory*.

Ütemezési alapelvek

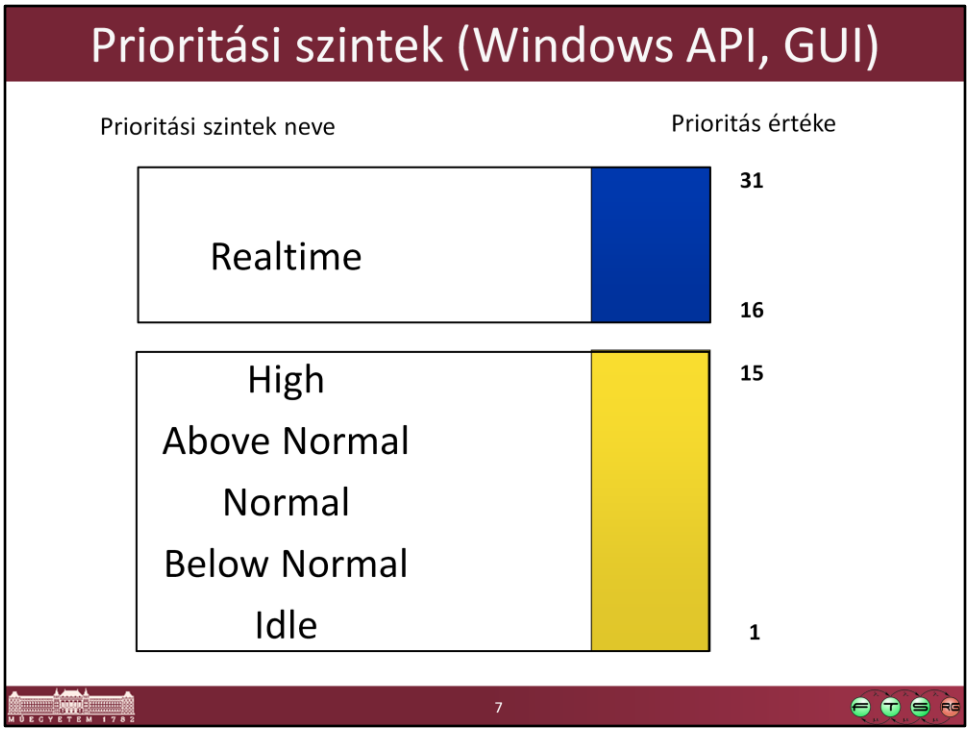
- Preemptív ütemező (kernel és user módban is!)
- 32 prioritási szint
  - Legmagasabb prioritású szál fut mindig
  - Azonos prioritásúak között Round Robin
- A szálak adott ideig futnak (quantum)
- Nincs mindig futó központi ütemező, ütemezést események indítják
- Szálak prioritása változhat a futás során

• A 32 prioritási szintnek megfelelően 32 sort (FIFO listát) tart nyilván a futásra kész szálaknak
• Egy CPU esetén: mindig a legmagasabb prioritású szál fut
• Több CPU esetén: valamelyik legmagasabb prioritású szál biztos fut valahol
• Nem törekszik arra, hogy a folyamatokat egyenlő arányban futassa, ha valakinek több szála van, akkor az több időt fog futni



Fontos, hogy a szintek között nincsenek rendszer és felhasználói szintek. Van olyan rendszer szál, aminek 16-nál kisebb a prioritása, és egy felhasználói szálnak is beállíthatunk (megfelelő joggal) 15-nél magasabb prioritást.

A 0-s prioritás a *zero page thread* nevű rendszerszálnak van fenntartva, aminek a feladata a felszabadított memórialapok kinullázása, mielőtt azt másnak odaadja az OS.



Ezeket a neveket láthatjuk például a Feladatkezelő felületén is.

Windows API vs. kernel prioritások

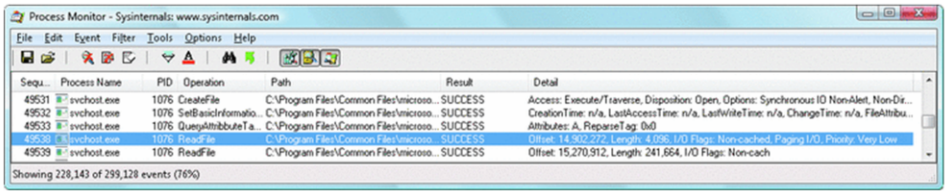
- Szálak: 7 féle relatív prioritás
- Leképezés:

		Win32 folyamat osztályok					
		Realtime	High	Above Normal	Normal	Below Normal	Idle
Win32 szál prioritások	Time-critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above-normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below-normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

(A pontos számokat nem kell megjegyezni, elég annyit tudni, hogy a szál és a prioritási szintből és a relatív prioritásból kapjuk meg a prioritási értéket.)

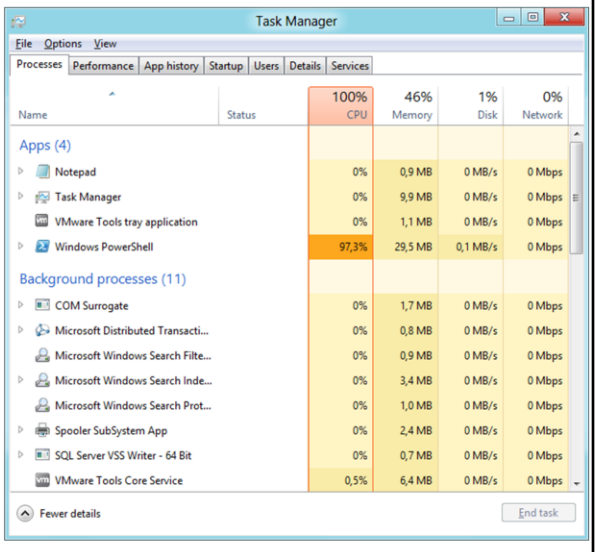
I/O prioritás

- Vista kernel módosítás
- 5 féle prioritás az I/O kéréseknek, pl.
  - Critical: Dirty page writer
  - Low: Desktop search indexer
- I/O sávszélesség foglalás



DEMO Windows 8 feladatkezelő

- „Heat map”
- Újratervezés telemetria alapján, pl.
  - 32%: kill process
- Csoportosítás
- „Friendly name”

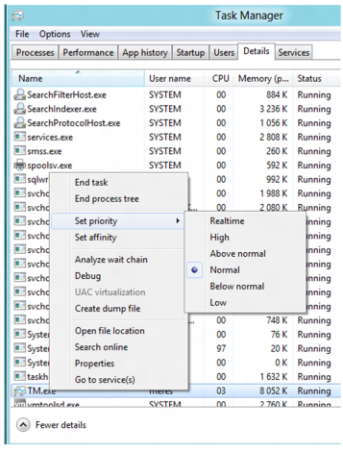


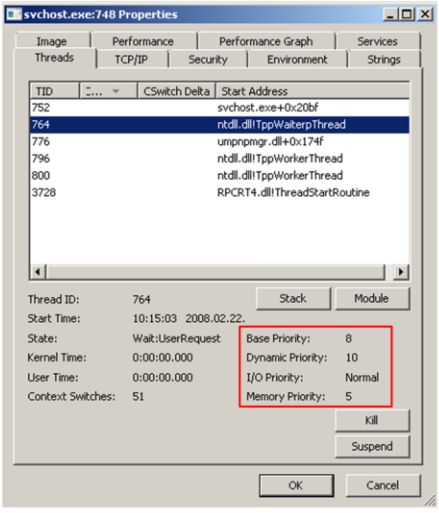
Részletes leírás a Windows 8 változtatásokról és az azok mögött meghúzódó tervezői döntésekről:

- MSDN Building Windows 8 Blog, „The Windows 8 Task Manager”, October 13, 2011. URL: http://blogs.msdn.com/b/b8/archive/2011/10/13/the-windows-8-task-manager.aspx

DEMO Prioritás állítása

- Feladatkezelő
  - Prioritás beállítása





Ahhoz, hogy a Realtime tartománynak megfelelő szintet beállíthassuk, megfelelő joggal kell rendelkeznünk.

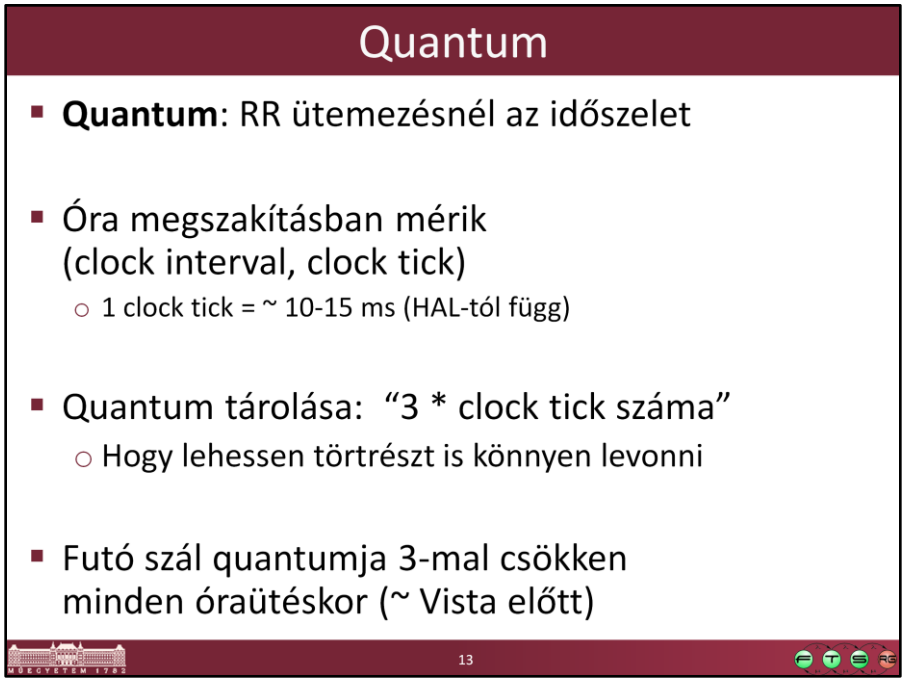
Nyissunk meg egy Notepad-et, nézzük meg Process Explorerben, és nézzük meg a prioritási értékeit. A Feladatkezelőben állítsuk át a prioritását, nézzük meg most is.

Kétszeresen láncolt listák a futásra kész szálakról. Lista elejéről veszi le az ütemező a következő szálat -> nem O(n)-es komplexitás

---

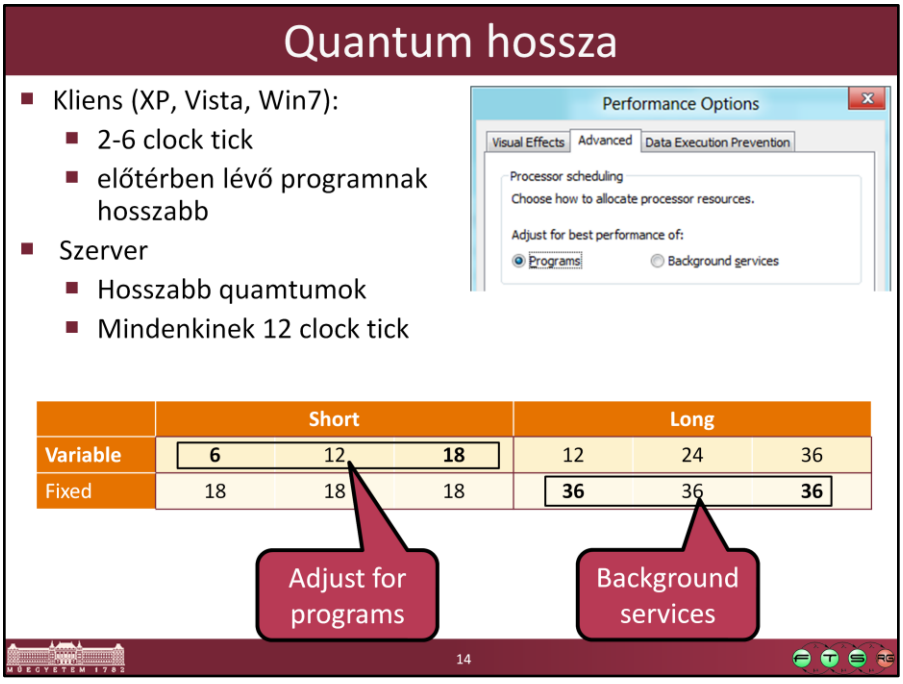
Futásra kész szálak listája: !ready parancs a kernel debuggerben

The dispatcher ready queues (KiDispatcherReadyListHead) contain the threads that are in the ready state, waiting to be scheduled for execution. There is one queue for each of the 32 priority levels. To speed up the selection of which thread to run or preempt, Windows maintains a 32-bit bit mask called the ready summary (KiReadySummary). Each bit set indicates one or more threads in the ready queue for that priority level. (Bit 0 represents priority 0, and so on.)



Vista óta már nem óraütés alapján mérik az időszeletet, hanem pontosabb módszereket alkalmaznak (hogy pl. a megszakítások kiszolgálásával eltöltött időt ne számolja bele), lásd Windows Vista Cycle-Based Scheduling (http://technet.microsoft.com/en-us/magazine/2007.02.vistakernel.aspx?pr=blog).

The length of the clock interval varies according to the hardware platform. The frequency of the clock interrupts is up to the HAL, not the kernel. For example, the clock interval for most x86 uniprocessors is about 10 milliseconds and for most x86 and x64 multiprocessors it is about 15 milliseconds.

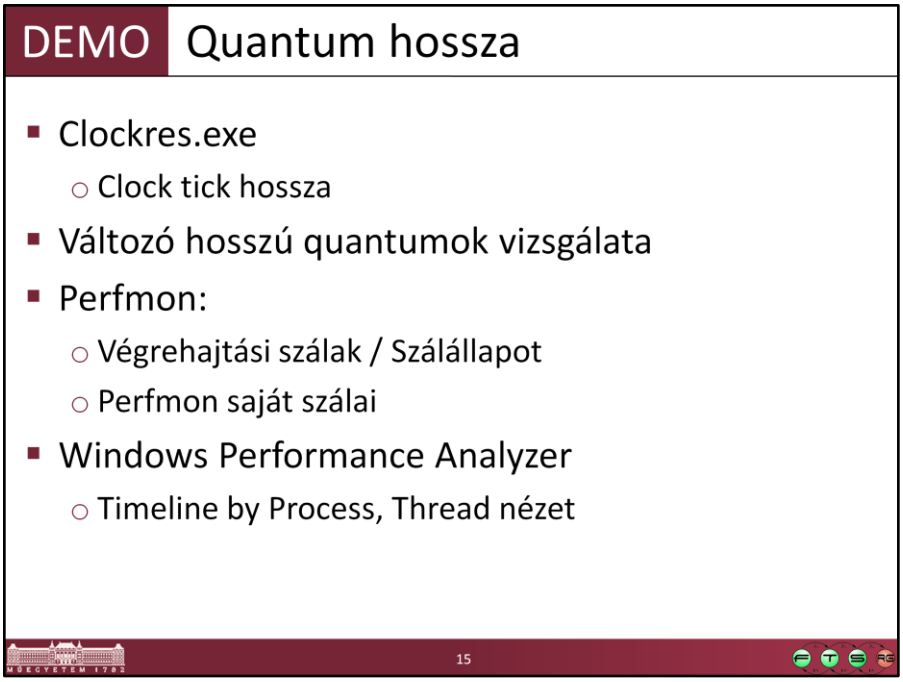


Egy cellában lévő elemek értéke:
• az első érték a háttérben lévő folyamatok szálainak quantum hossza
• az előtérben lévőkhöz pedig a harmadik érték tartozik általában (átállítható, hogy a második értéket kapja)

On Windows Vista, threads run by default for 2 clock intervals; on Windows Server systems, by default, a thread runs for 12 clock intervals. The rationale for the longer default value on server systems is to minimize context switching. By having a longer quantum, server applications that wake up as the result of a client request have a better chance of completing the request and going back into a wait state before their quantum ends.

Threads in the foreground process run with a quantum of 6 clock ticks, whereas threads in other processes have the default workstation quantum of 2 clock ticks. In this way, when you switch away from a CPU-intensive process, the new foreground process will get proportionally more of the CPU, because when its threads run they will have a longer turn that background threads (again, assuming the thread priorities are the same in both the foreground and background processes).

Short or Long, Variable or Fixed:
HKLM\SYSTEM\CurrentControlSet\Control\PriorityControl\Win32PrioritySeparation
Leírás: http://www.microsoft.com/mspress/books/sampchap/4354c.aspx

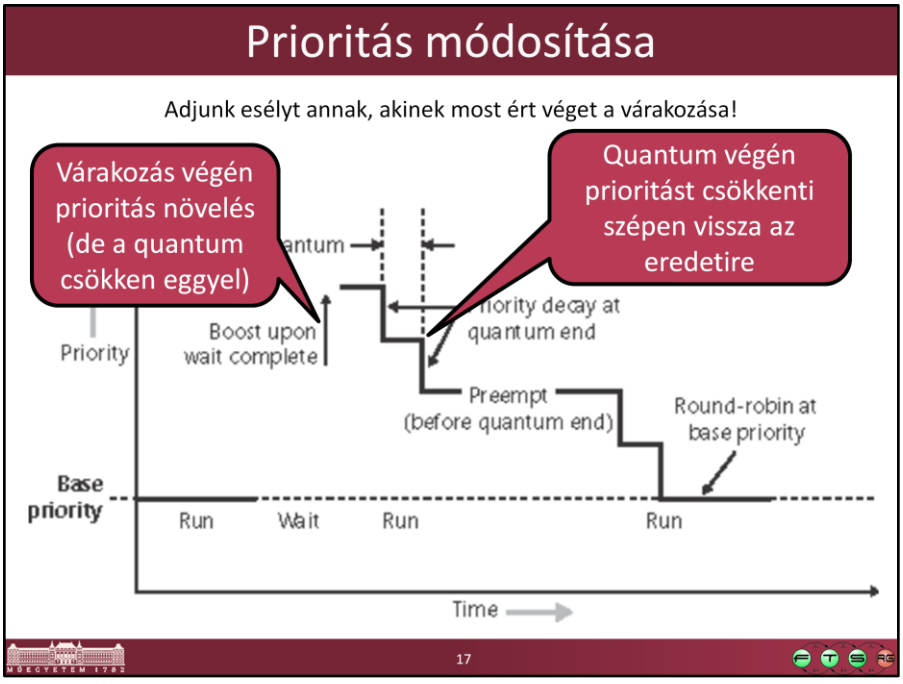


1. clockres:
- Sysinternals segédeszköz, csak le kell futtatni és kiírja a rendszeróra felbontását

2. Változó hosszú quantumok
- Könnyen kipróbálhatjuk, hogy az előtérben lévő folyamat szálainak hosszabb a quantumja: készítsünk egy rövid programot, ami valami hosszabb, de determinisztikus futási idejű számítást végez, és indítsuk el egyprocesszoros rendszeren két példányban. Mérjünk a programban a futási időt, és hasonlítsuk össze az előtérben és a háttérben futó példány futási idejét.

3. Perfmon
- A perfmon saját szálait nézzük, hisz ő csak a saját szálát láthatja futónak, ugyanis amikor ő fut, akkor más nem futhat egyprocesszoros rendszeren. Többmagos/többszálas gépen már érdekesebb megnézni.
- *Végrehajtási szálak / Szálállapot*: „A végrehajtási szál jelenlegi állapotát mutatja. A 0 az inicializált, az 1 az működésre kész, a 2 a futó, a 3 a készenléti, a 4 a befejezett, az 5 a várakozó, a 6 az átmeneti, a 7 pedig az ismeretlen állapotot jelöli. A futó szál éppen processzort használ; a készenléti processzorhasználatra készül. A működésre kész szál processzort használna, de várakoznia kell, mert éppen nincs szabad processzor. Az átmeneti szál valamilyen eszközre vagy erőforrásra vár a végrehajtás megkezdéséhez (például arra, hogy a végrehajtási vermet a rendszer belapozza a lemezről). A várakozó szál nem használja a processzort, mivel egy perifériaművelet befejeződésére, vagy egy erőforrás felszabadulására vár.”

4. Xperf
- Windows Performance Tools csomag része, http://msdn.microsoft.com/en-us/performance/cc825801.aspx
- Xperf Quickstart: http://msdn.microsoft.com/en-us/library/ff190971%28v=VS.85%29.aspx
  xperf -on DiagEasy
  (vizsgálandó program futtatása)
  xperf -d trace.etl
  xperf trace.etl
- A CPU Scheduling nézetet kell utána nézni
  - Mezők leírása: CSwitch Class, http://msdn.microsoft.com/en-us/library/aa964744%28v=vs.85%29.aspx



- Five types:
  - I/O completion
  - Wait completion on events or semaphores
  - When threads in the foreground process complete a wait
  - When GUI threads wake up for windows input
  - For CPU starvation avoidance

- Quantum decremented by 1 when you come out of a wait
  - So that threads that get boosted after I/O completion won't keep running and never experiencing quantum end
  - Prevents I/O bound threads from getting unfair preference over CPU bound threads

- Vista újítás: Multimedia Class Scheduler Service
  - Média lejátszó alkalmazások használhatnak pár új API függvényt, és akkor a rendszer az ő prioritásukat is megnöveli néha 21-re, hogy folyamatos legyen a média lejátszás

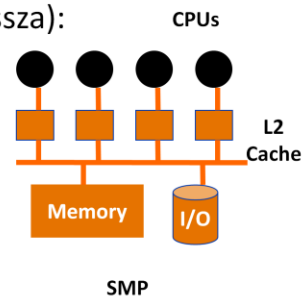
Éhezés elkerülése

- Az OS másodpercenként megnézi a futásra kész szálakat
- Aki nem futott már 300 óraütés óta, annak
  - 15-ös prioritást ad,
  - megnöveli a quantumját,
  - egy quantumnyi futásig.
  - (15-nél nagyobb prioritású szálakra nem vonatkozik)

Ez a *Balance Set Manager* feladata (rendszer szál, 16-os prioritással)

Symmetric Multiprocessing (SMP)

- Minden CPU egyenrangú
  - Közös memória címtér
  - Megszakításokat bármelyik CPU kiszolgálhatja
- CPU-k maximális száma a registry-ben tárolva
- Implementációs limit (bitvektor hossza):
  - 32 processzor 32-bites rendszereken
  - 64 processzor 64-bites rendszereken
- Windows 7 / Server 2008 R2
  - Logikai processzor csoportok
  - 4 * 64 CPU támogatása
  - NUMA támogatás

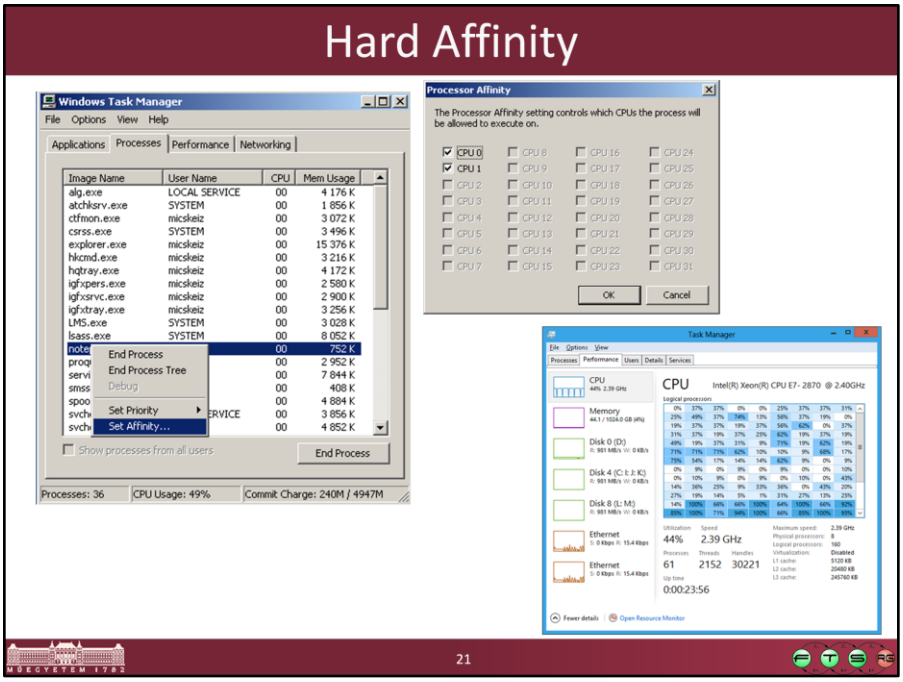


SMP

HKLM\System\CurrentControlSet\Control\Session Manager\LicensedProcessors

Multiprocesszoros ütemezés

- Szálak bármelyik CPU-n futhatnak, de
  - Megpróbálja az OS ugyanazon a CPU-n tartani (“soft affinity”)
  - Beállítható, hogy melyiken futhat (“hard affinity”)
- Elosztott (nincs “master processor”)
  - Bármelyik CPU bármelyik futását megszakíthatja
- Ütemezési adatok tárolása:
  - Windows Server 2003 előtt: rendszerszintű futásra kész sorok
  - Windows Server 2003-tól: CPU-nkénti sorok



Affinity is a bit mask where each bit corresponds to a CPU number

- Hard Affinity specifies where a thread is permitted to run
  - Defaults to all CPUs
- Thread affinity mask must be subset of process affinity mask, which in turn must be a subset of the active processor mask

Functions to change: SetThreadAffinityMask, SetProcessAffinityMask, SetInformationJobObject

Windows 7 módosítások

- Core Parking (szerver)
  - Minél kevesebb logikai CPU használata
  - Nem használt socketek standby üzemmódba
- Time coalescing
  - Azonos periódusú időzítők összevonása
- Dynamic Fair Share Scheduling (DFSS)
  - Remote Desktop szerverekhez
  - Minden munkamenet „CPU keretet” kap
  - Ha elhasználja, csak idle CPU-t kaphat
- Globális zárak megszüntetése

Összefoglalás

- Folyamat ↔ Szál
- Ütemezés:
  - Prioritási szintek
  - Prioritáson belül RR: quantum

Olvasnivaló

- Inside the Windows Vista Kernel, 1. rész (Technet magazin)
  - CPU Cycle Counting
  - Multimedia Class Scheduler
  - I/O Priority
  - …
- *Windows 7 and Windows Server 2008 R2 Kernel Changes*

- Inside the Windows Vista Kernel, http://www.microsoft.com/technet/technetmag/issues/2007/02/VistaKernel/default.aspx
- Windows 7 and Windows Server 2008 R2 Kernel Changes, http://download.microsoft.com/download/8/C/2/8C21BAFE-3432-48D1-962A-F7A9DD54A2AC/Windows%207%20and%20Windows%20Server%202008%20R2%20Kernel%20Changes.pptx