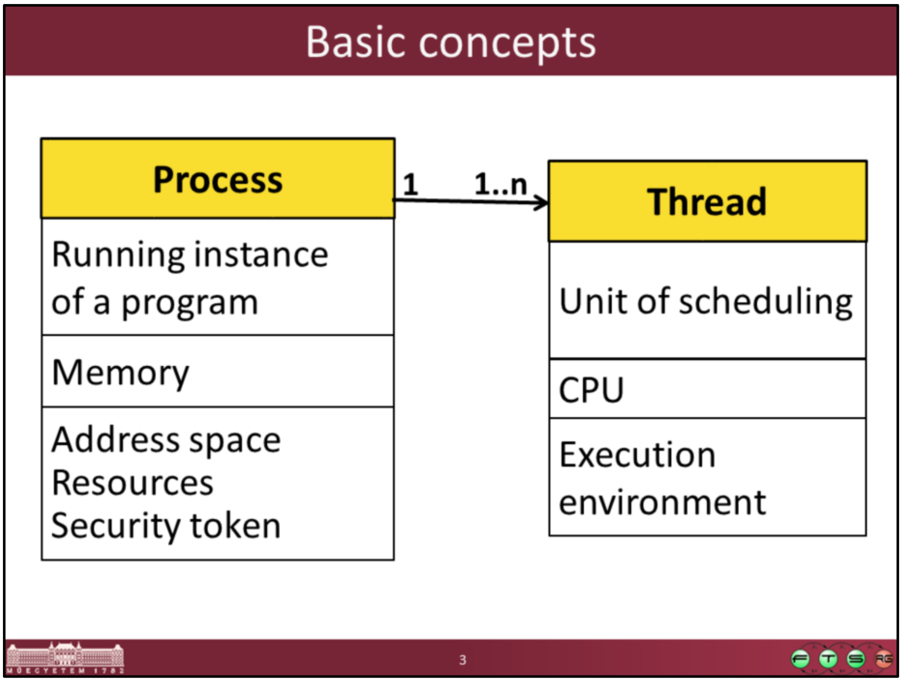
Scheduling in Windows

Zoltan Micskei

http://mit.bme.hu/~micskeiz

Copyright Notice



--From the *Windows Operating System Internals Curriculum Development Kit*

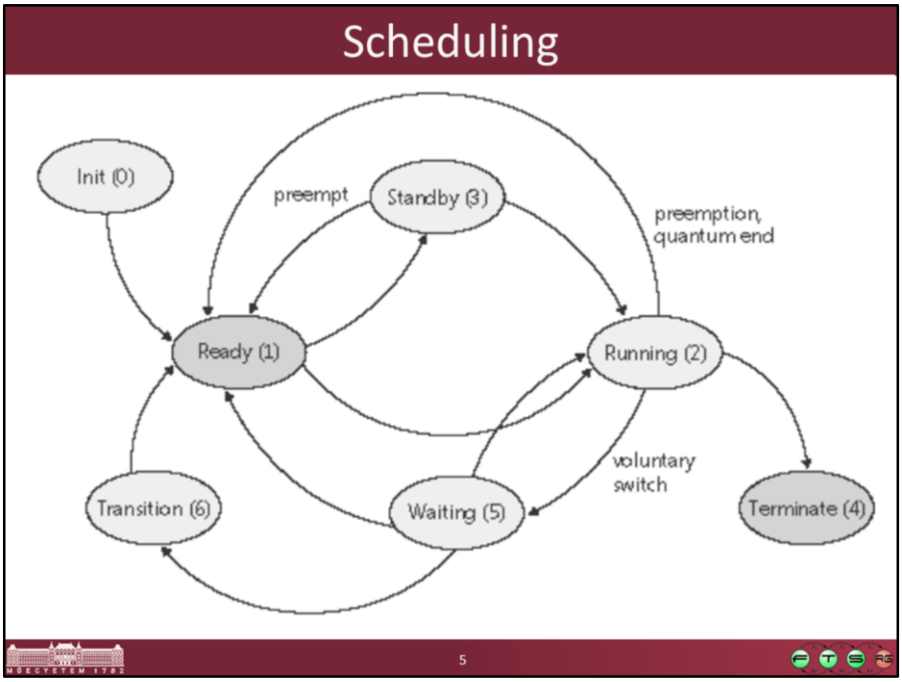
"Although programs and processes appear similar on the surface, they are fundamentally different. A *program* is a static sequence of instructions, whereas a *process* is a container for a set of resources used when executing the instance of the program. At the highest level of abstraction, a Windows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

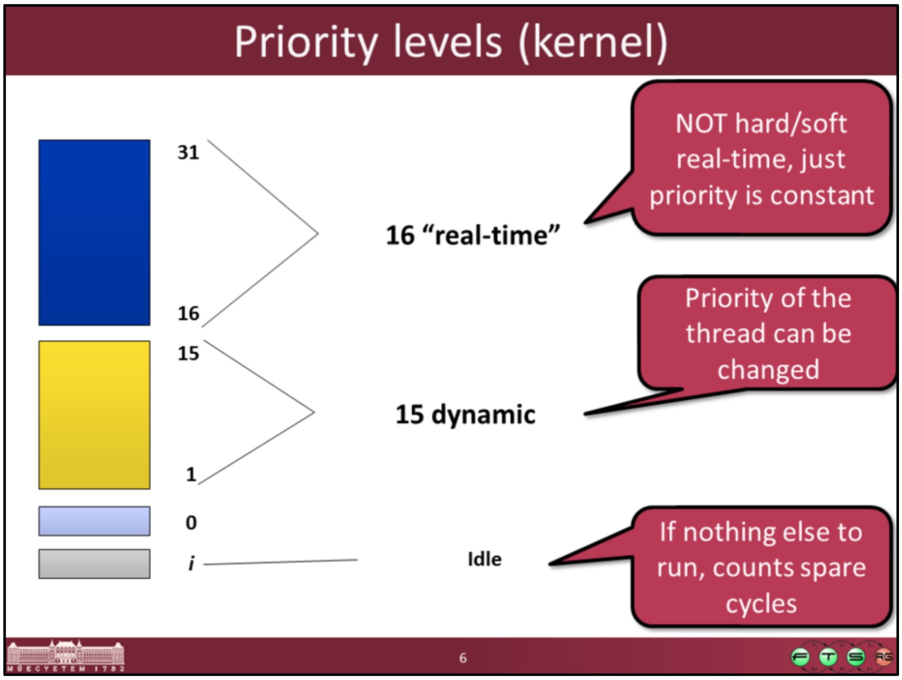
Principles of Windows scheduling

- Preemptive scheduler (both kernel and user!)
- 32 priority levels
  - (One of the) Thread with the highest priority runs
  - Round robin between threads with same priority
- Threads run for a fixed time (quantum)
- No central scheduler,
  scheduling is driven by events
- Priority of threads can change runtime

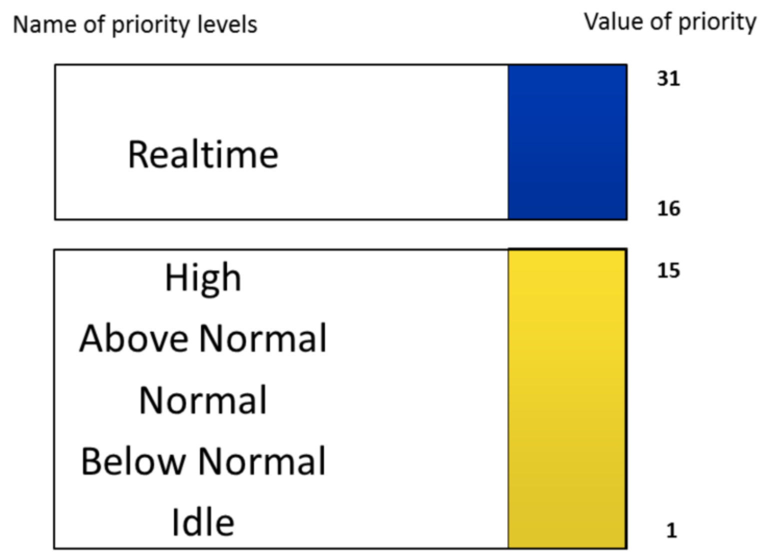


Ready A thread in the ready state is waiting to execute. When looking for a thread to execute, the dispatcher considers only the pool of threads in the ready state.

- **Standby** A thread in the standby state has been selected to run next on a particular processor. When the correct conditions exist, the dispatcher performs a context switch to this thread. Only one thread can be in the standby state for each processor on the system. Note that a thread can be preempted out of the standby state before it ever executes (if, for example, a higher priority thread becomes runnable before the standby thread begins execution).
- **Running** Once the dispatcher performs a context switch to a thread, the thread enters the running state and executes. The thread's execution continues until its quantum ends (and another thread at the same priority is ready to run), it is preempted by a higher priority thread, it terminates, it yields execution, or it voluntarily enters the wait state.
- **Waiting** A thread can enter the wait state in several ways: a thread can voluntarily wait for an object to synchronize its execution, the operating system can wait on the thread's behalf (such as to resolve a paging I/O), or an environment subsystem can direct the thread to suspend itself. When the thread's wait ends, depending on the priority, the thread either begins running immediately or is moved back to the ready state.
- **Transition** A thread enters the transition state if it is ready for execution but its kernel stack is paged out of memory. Once its kernel stack is brought back into memory, the thread enters the ready state.
- **Terminated** When a thread finishes executing, it enters the terminated state. Once the thread is terminated, the executive thread block (the data structure in nonpaged pool that describes the thread) might or might not be deallocated. (The object manager sets policy regarding when to delete the object.)
- **Initialized** This state is used internally while a thread is being created.


Priority levels (kernel)
31
16
16 “real-time”
NOT hard/soft real-time, just priority is constant
15
1
15 dynamic
Priority of the thread can be changed
0
i
Idle
If nothing else to run, counts spare cycles
6

Priority levels (Windows API, GUI)



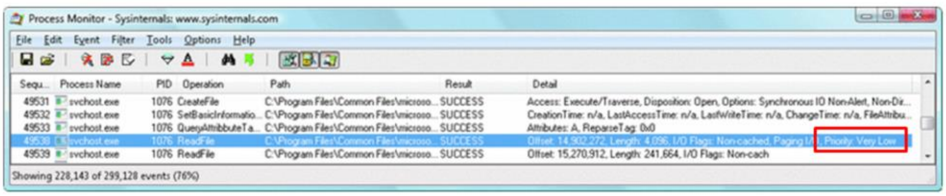
Windows API vs. kernel names

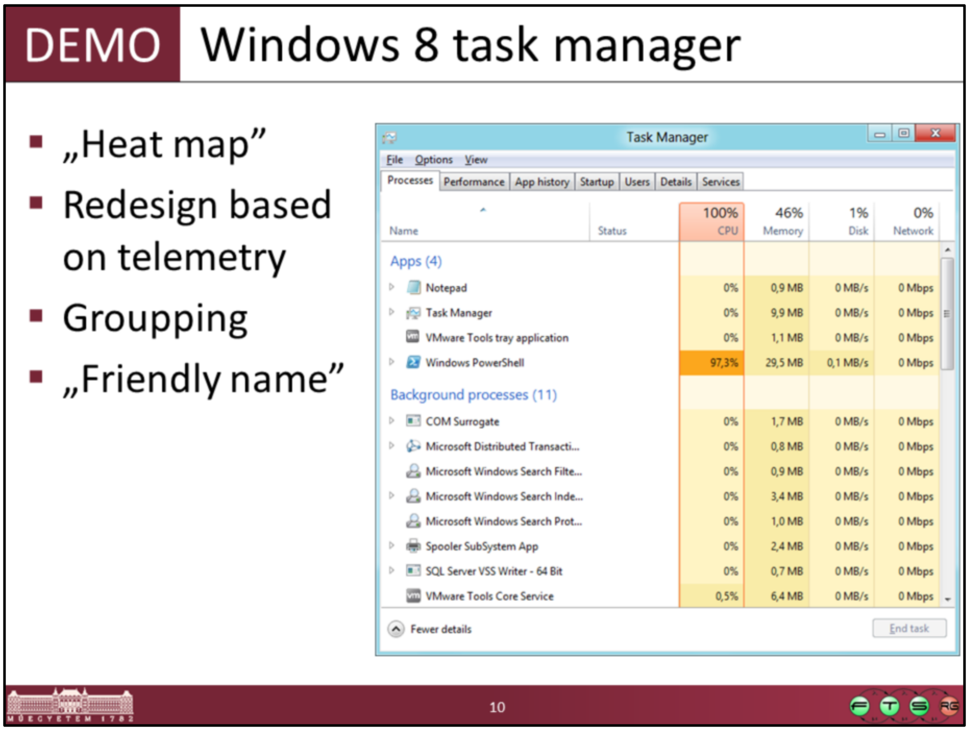
- Threads: 7 different relative priority
- Mapping:

		Win32 process priority levels					
		Realtime	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	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

I/O priority

- Since Vista
- 5 different I/O priority for requests, e.g.
  - Critical: Dirty page writer
  - Low: Desktop search indexer
- I/O bandwidth allocation

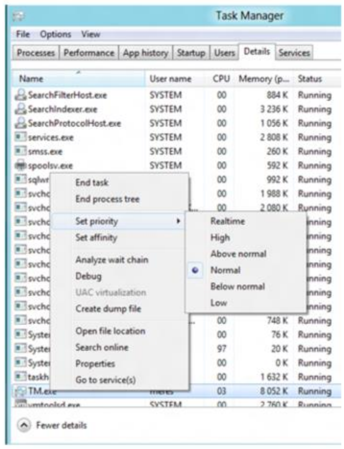


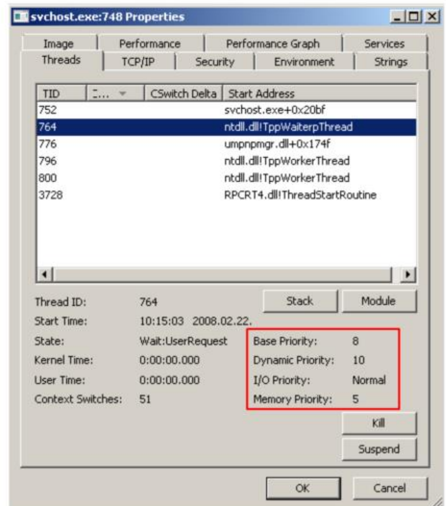


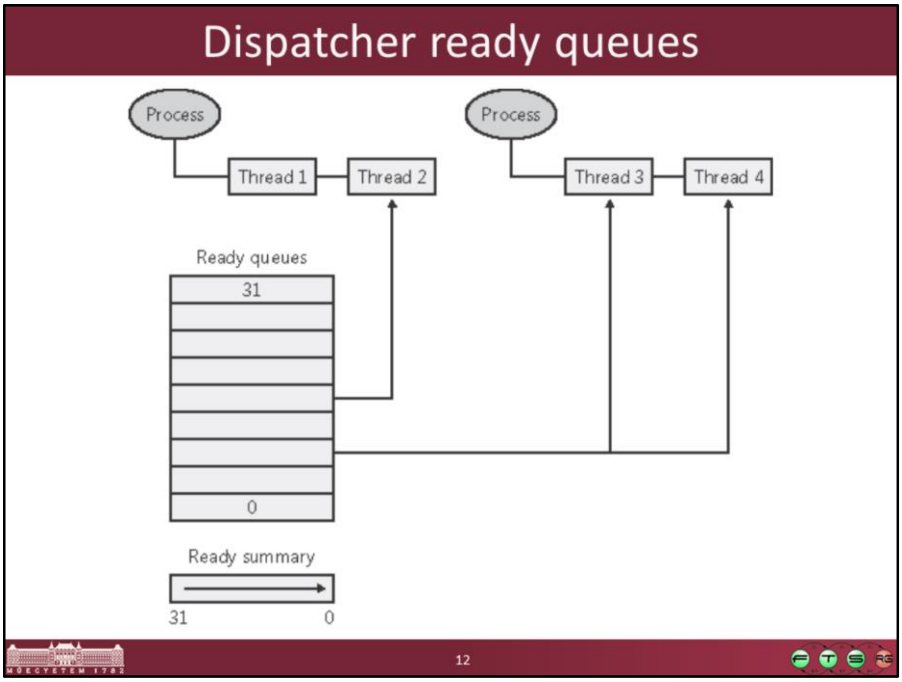
See> MSDN Building Windows 8 Blog, „The Windows 8 Task Manager”, October 13, 2011. URL: http://blogs.msdn.com/b/b8/archive/2011/10/13/the-windows-8-task-manager.aspx

DEMO Changing priority

- Task manager







From Windows Internals curriculum:
"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.)"

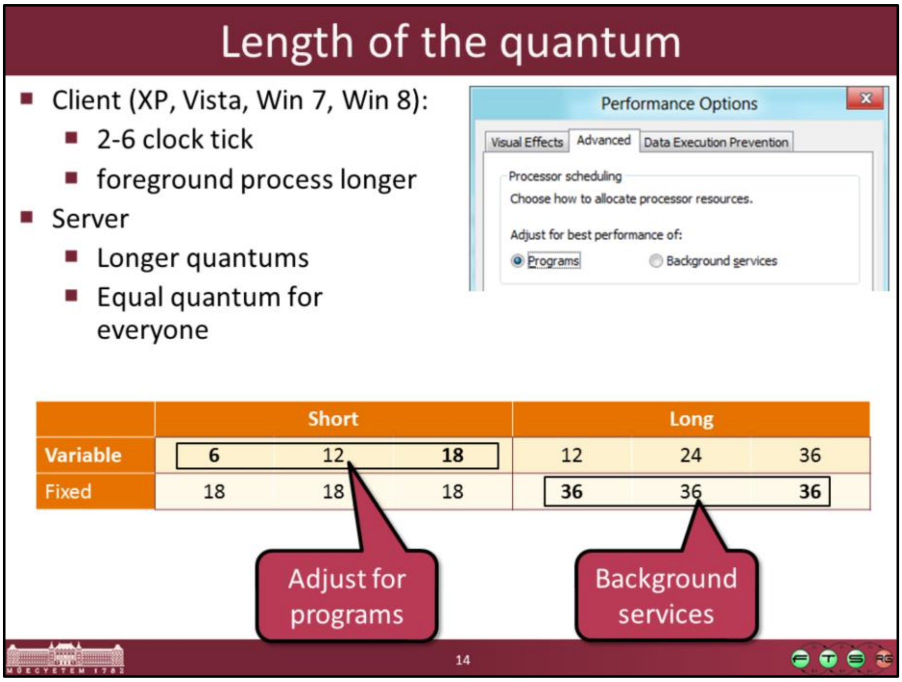
Quantum

- **Quantum**: time slice for Round Robin
- Measured in clock interval (clock tick)
  - 1 clock tick = ~ 10-15 ms (before Windows 8)
  - 1 clokc tick = 0.5–15.6 ms (after Windos 8)
- Storing quantum: "3 * number of clock tick "
  - Easy to subtract fractions
- At each tick quantum of the running thread decreases by 3 (~ before Vista)

13

See> 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.



-- From *Windows Internals Curriculum*

"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

DEMO Length of the quantum

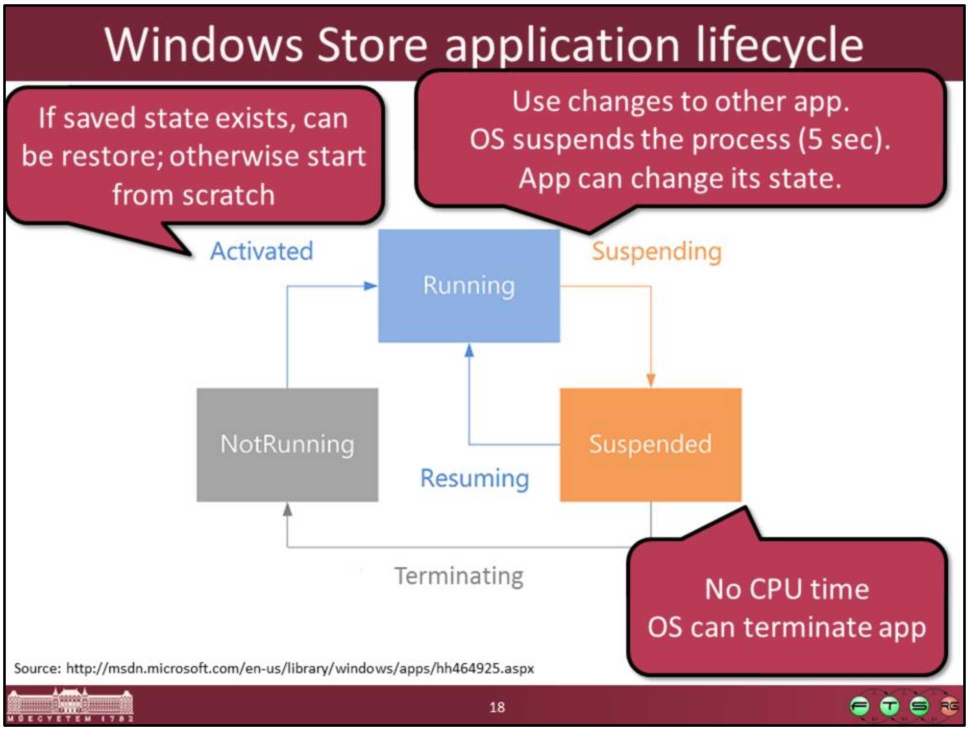
- Clockres.exe utility
  - Length of Clock tick
- Adjust quantum length
- Perfmon
- Windows Performance Analyzer
  - Timeline by Process, Thread view

Contents

- Windows scheduling (basics)
- **Windows 8: Windows Store applications**
- Windows scheduling (advanced)

Windows 8: new application model

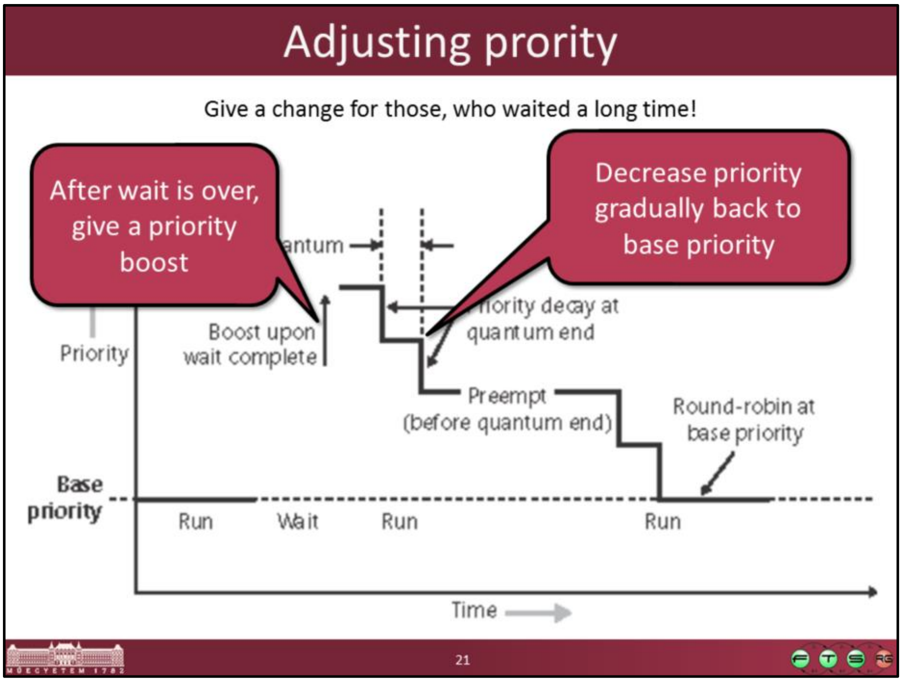
- Design goals:
  - Low powers and resource-consumption
  - Easier deployment and upgrade
  - Separated applications (security, reliability)
  - …
- Solution:
  - New API: WinRT
  - App store: Windows Store
  - New application lifecycle
  - …


Windows Store application lifecycle
If saved state exists, can be restore; otherwise start from scratch
Use changes to other app.
OS suspends the process (5 sec).
App can change its state.
Activated
Running
Suspending
NotRunning
Suspended
Resuming
Terminating
No CPU time
OS can terminate app
Source: http://msdn.microsoft.com/en-us/library/windows/apps/hh464925.aspx
18

DEMO Windows Store applications

- Windows Store
- Changing between Windows Store apps
- Process Explorer: Suspended state

Contents



•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

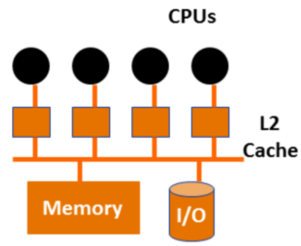
So that threads that get boosted after I/O completion won't keep running and never experiencing quantum end
Prevents I/O bound threads from getting unfair preference over CPU bound threads

Preventing starvation

- OS checks runnable threads (every 1 sec)
- If a thread has not been executed since 300 sec
  - change priority to 15 (max in dynamic),
  - increase quantum,
  - for 1 quantum.

Symmetric Multiprocessing (SMP)

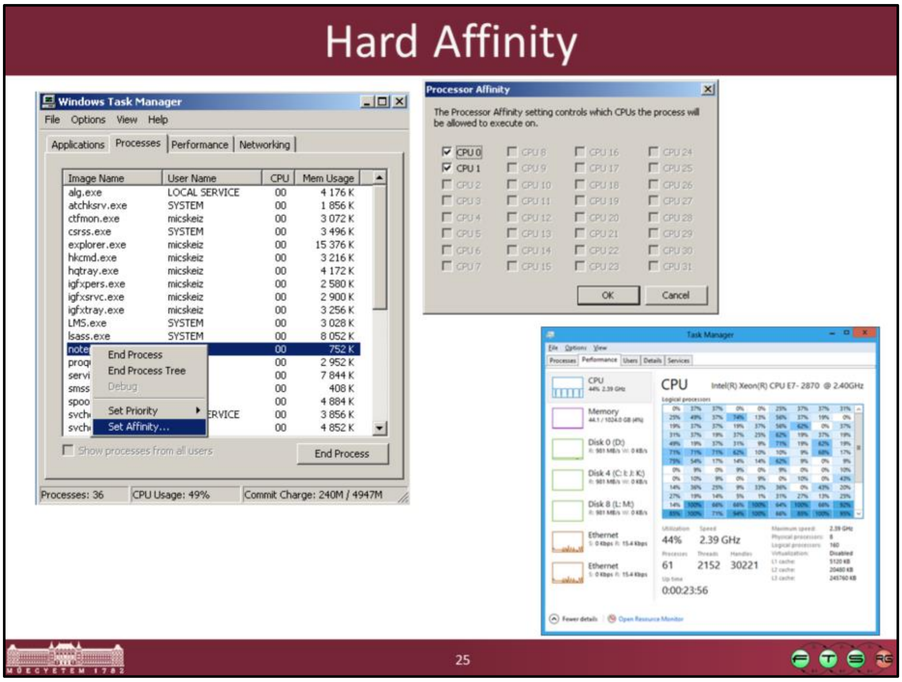
- All CPU is equal
  - Shared address space
  - Interrupts can be served by an CPU
- Implementation limit (length of a bit vector):
  - 32 CPUs on 32 bit systems
  - 64 CPUs on 64 bit systems
- Change in Windows 7 / Server 2008 R2
  - Groups of logical processors
  - Supporting 4 * 64 CPU
  - NUMA support



SMP

Multiprocessor scheduling

- Threads can run on any CPU by default, but
  - OS tries to keep on CPU, where it run (“soft affinity”)
  - Can be set to use only selected CPUs (“hard affinity”)
- No “master processor”
- Dispatcher queues:
  - Before Windows Server 2003 : one global queue
  - Windows Server 2003: per CPU queues



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 changes

- Core Parking (server)
  - Use fewer processor cores
  - Not used cores going to standby
- Time coalescing
  - Timers with same periodicity are merged
- Dynamic Fair Share Scheduling (DFSS)
  - for Remote Desktop
  - Every session gets a share
  - If share is exhausted, thread cannot run
- Eliminating global locks in scheduler

Summary

- Process ↔ Thread
- Scheduling:
  - Priority levels
  - Round robin / quantum