

# Sistemi Operativi

Laurea in Ingegneria Informatica

Università di Roma Tor Vergata



Docente: Francesco Quaglia

## CPU-scheduling

1. Tipi di scheduling
2. Metriche
3. Algoritmi di scheduling classici
4. Scheduling multiprocessore
5. Scheduling in sistemi UNIX/Windows

# Tipi di scheduling

## **A lungo termine**

Decisioni sull'aggiunta di un nuovo processo all'insieme dei processi attivi

---

## **A medio termine**

Decisioni sull'inserimento, totale o parziale di un processo attivo in memoria di lavoro

---

## **A breve termine (CPU-scheduling o dispatching)**

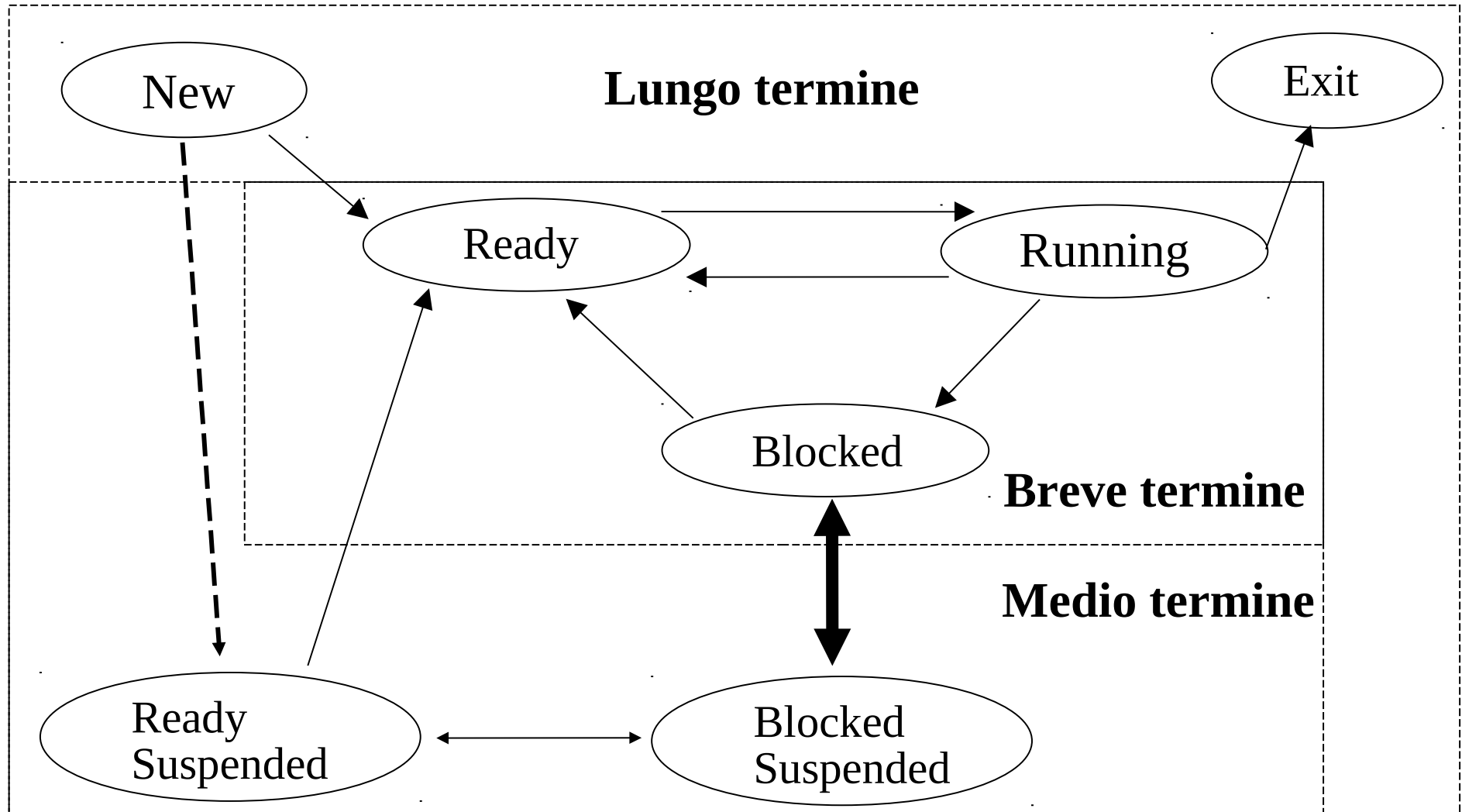
Decisioni su quale processo debba impegnare la CPU

---

## **I/O scheduling**

Decisioni sulla sequenzializzazione di richieste da servire sui dispositivi (logici e fisici)

# Tipi di scheduling e stati di processi



# Scheduling a lungo termine - alcuni dettagli

## Decisione di attivazione di processo

- raggiungimento di un dato livello di multiprogrammazione
- mistura conveniente di processi **I/O bound e CPU bound**

## Attivazione dello scheduler

- alla terminazione di un processo
- su richiesta
- quando la percentuale di utilizzo della CPU scende sotto valori specifici

---

Tipicamente non controlla  
applicazioni interattive



Attivazione di processo governata  
dalle condizioni di carico del sistema

---

Tipico di sistemi batch multiprogrammati

# **Criteri per il dispatching**

## **Orientamento all'utente**

- decisioni di dispatching funzione di come gli utenti percepiscono il comportamento del sistema (es. tempo di risposta)

## **Orientamento al sistema**

- decisioni di dispatching tese a ottimizzare il comportamento del sistema nella sua globalità (es. utilizzazione di risorse)
- 

## **Orientamento a metriche prestazionali**

- approccio quantitativo
- parametri facilmente misurabili (monitorabili), analizzabili

## **Orientamento a metriche non prestazionali**

- parametri tipicamente qualitativi o non facilmente misurabili

# Criteri orientati all'utente

## Prestazionali

- **tempo di risposta**                      ovvero il tempo necessario affinché' un processo inizi a produrre l'output
- **tempo di turnaround**                      ovvero del tempo totale intercorrente tra l'istante di creazione e l'istante di completamento di un processo
- **scadenze**                                      ovvero una deadline di completamento

## Altri

- **prevedibilita'**                                      possibilita' di supportare esecuzioni conformi a determinati parametri indipendentemente dal livello di carico del sistema

# Criteri orientati al sistema

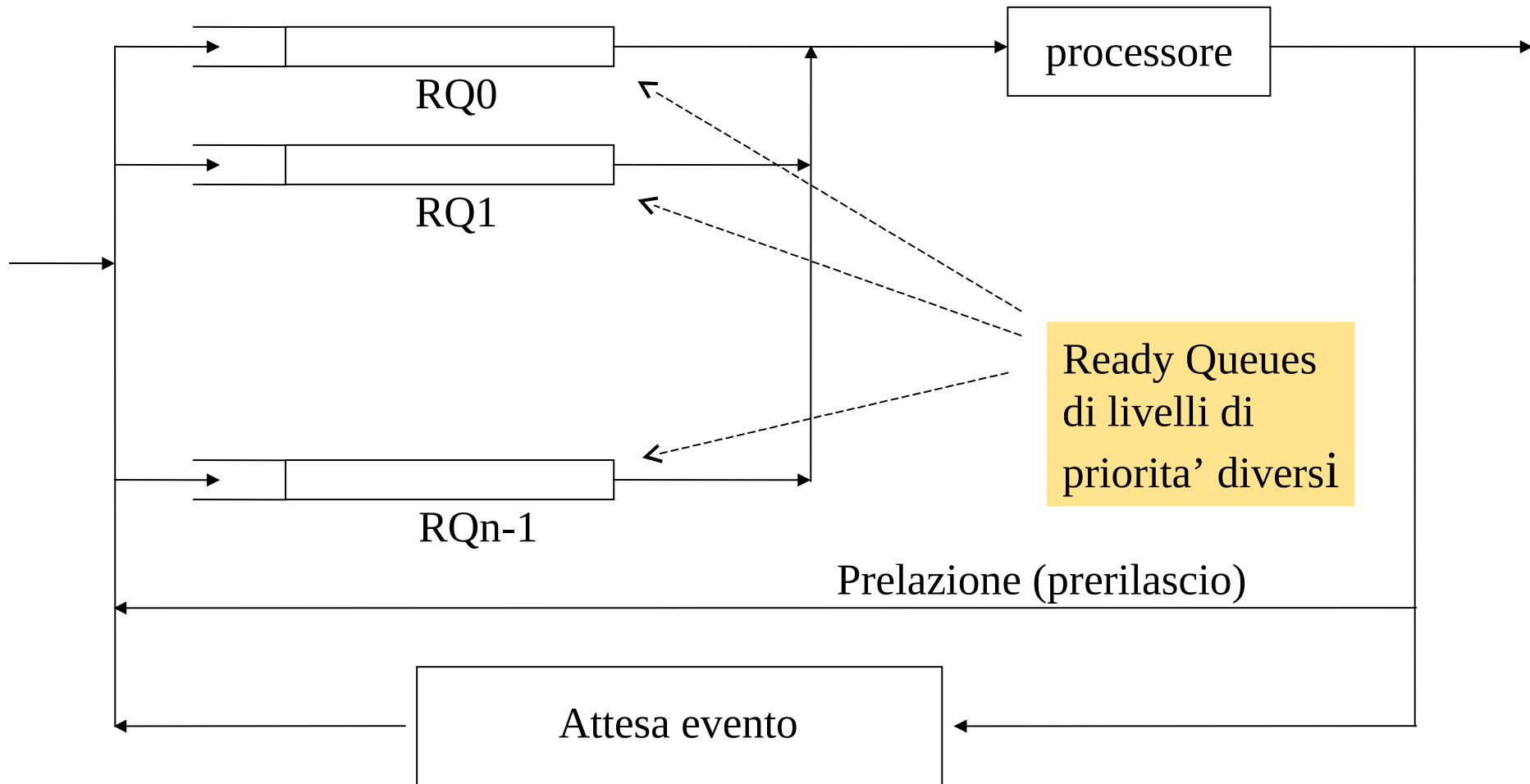
## Prestazionali

- throughput processi completati per unita' di tempo
- utilizzazione del processore percentuale del tempo in cui la CPU risulta impegnata

## Altri

- fairness capacita' di evitare starvation di processi attivi
- priorita' capacita' di distinguere tra livelli di priorita' multipli per i processi attivi
- bilanciamento delle risorse capacita' di equilibrare l'utilizzo delle risorse al fine di aumentarne lo sfruttamento

# Priorita', round-robin, prelazione e starvation





Storicamente il problema dello scheduling di CPU ha riguardato sistemi a processi ed e' nato nell'ambito di macchine uniprocessore ...

# Scheduling FCFS (First Come First Served)

## Caratteristiche

- i processi nello stato Ready vengono mandati in esecuzione secondo l'ordine di inserimento nella “Ready Queue”
- non vi è prelazione, quindi ogni processo rimane in esecuzione fino al suo completamento, oppure fino a che esso non rilascia la CPU spontaneamente

## Svantaggi

- non minimizza il tempo di attesa, e di conseguenza neanche il tempo di turnaround
- inadeguato per la gestione di processi interattivi
- può causare sottoutilizzo dei dispositivi di I/O a causa del fatto che i processi interattivi non necessariamente vengono favoriti

# Scheduling Round-Robin (Time-Slicing)

## Caratteristiche

- i processi nello stato Ready vengono mandati in esecuzione a turno per uno specifico **quanto di tempo**
- vi e' prelazione, quindi un processo puo' essere bloccato anche se non ha completato la sua traccia o non vuole rilasciare la CPU spontaneamente

## Svantaggi

- sfavorisce processi I/O bound rispetto a processi CPU bound
- non propriamente adeguato per la gestione di processi interattivi
- puo' causare sottoutilizzo dei dispositivi di I/O a causa del fatto che i processi I/O bound vengono sfavoriti

---

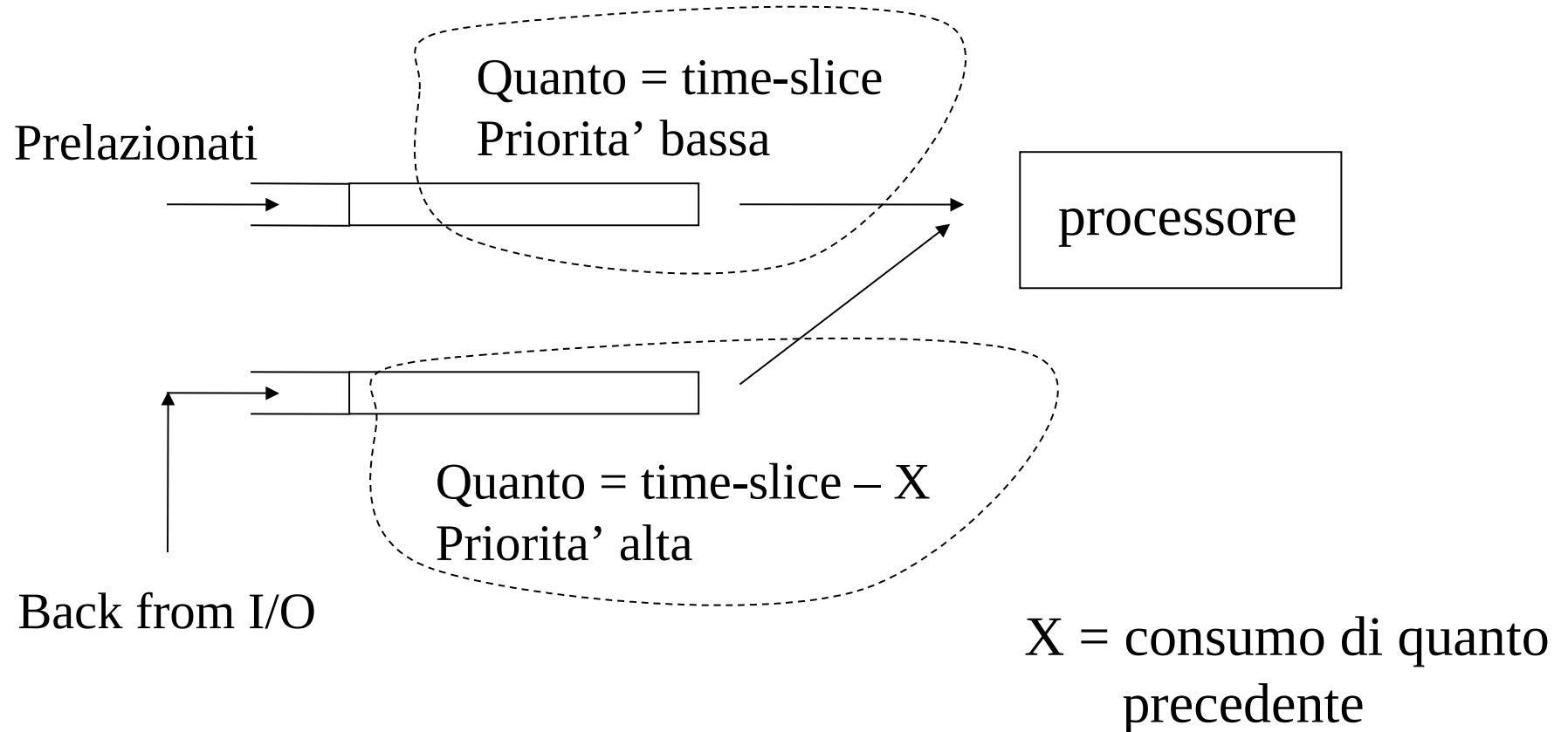
Criticita' della scelta  
del time-slice



Impatto sul numero di quanti per  
attivare una richiesta di I/O

# Scheduling Round-Robin Virtuale

## Separazione tra processi prelazionati e non



# Scheduling SPN (Shortest Process Next)

## Caratteristiche

- i processi nello stato Ready vengono mandati in esecuzione secondo ordine crescente della lunghezza del prossimo “CPU burst”
- puo' esservi (SRTN – Shortest Remaining Time Next) o non prelazione
- in caso negativo ogni processo rimane in esecuzione fino al suo completamento, oppure fino a che esso non rilascia la CPU spontaneamente, ovvero al termine del CPU Burst

## Vantaggi

- minimizza il tempo di attesa, e di conseguenza il tempo di turnaround
- relativamente adeguato per la gestione di processi interattivi in caso di prelazione
- in generale non causa sottoutilizzo dei dispositivi in caso di prelazione

## Svantaggi

- necessita di meccanismi di predizione della lunghezza dei CPU Burst
- puo' provocare starvation a causa del particolare trattamento della priorita'

# Stima dei CPU burst

**Media aritmetica**

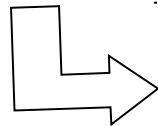
$$S_{n+1} = \frac{1}{n} \sum_{i=1}^n S_i$$

---

**Media esponenziale**

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_{n-1}$$

$\alpha$  vicino all'unità' determina maggior peso per osservazioni recenti



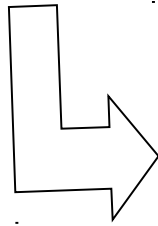
Impatto sulla stabilità in presenza di alta varianza

# Scheduling Highest Response Ratio Next (HRRN)

Processi selezionati in base al  
Rapporto di Risposta

$$RR = \frac{w + s}{s}$$

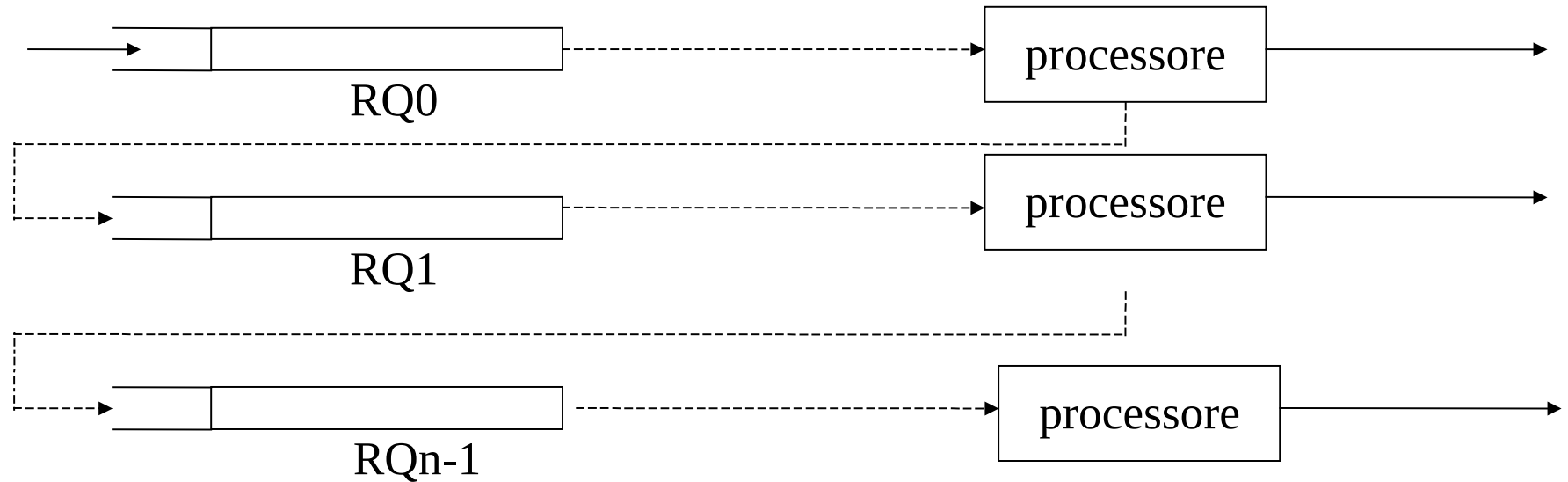
**Dove :**  $w$  = tempo di attesa (permanenza nello stato ready)  
 $s$  = tempo di servizio (di esecuzione)



- ✓ favorisce gli I/O bound (caratterizzati da piccoli valori di  $s$ )
- ✓ affronta il problema della starvation dovuto alle priorità

# Multi-level feedback-queue scheduling

- differentemente da SPN, SRTN e HRRN non necessita di informazioni (predette o monitorate)
- uso di code di priorit  multiple



Quanto di tempo  fisso per tutte le priorit  : starvation sui processi molto lunghi

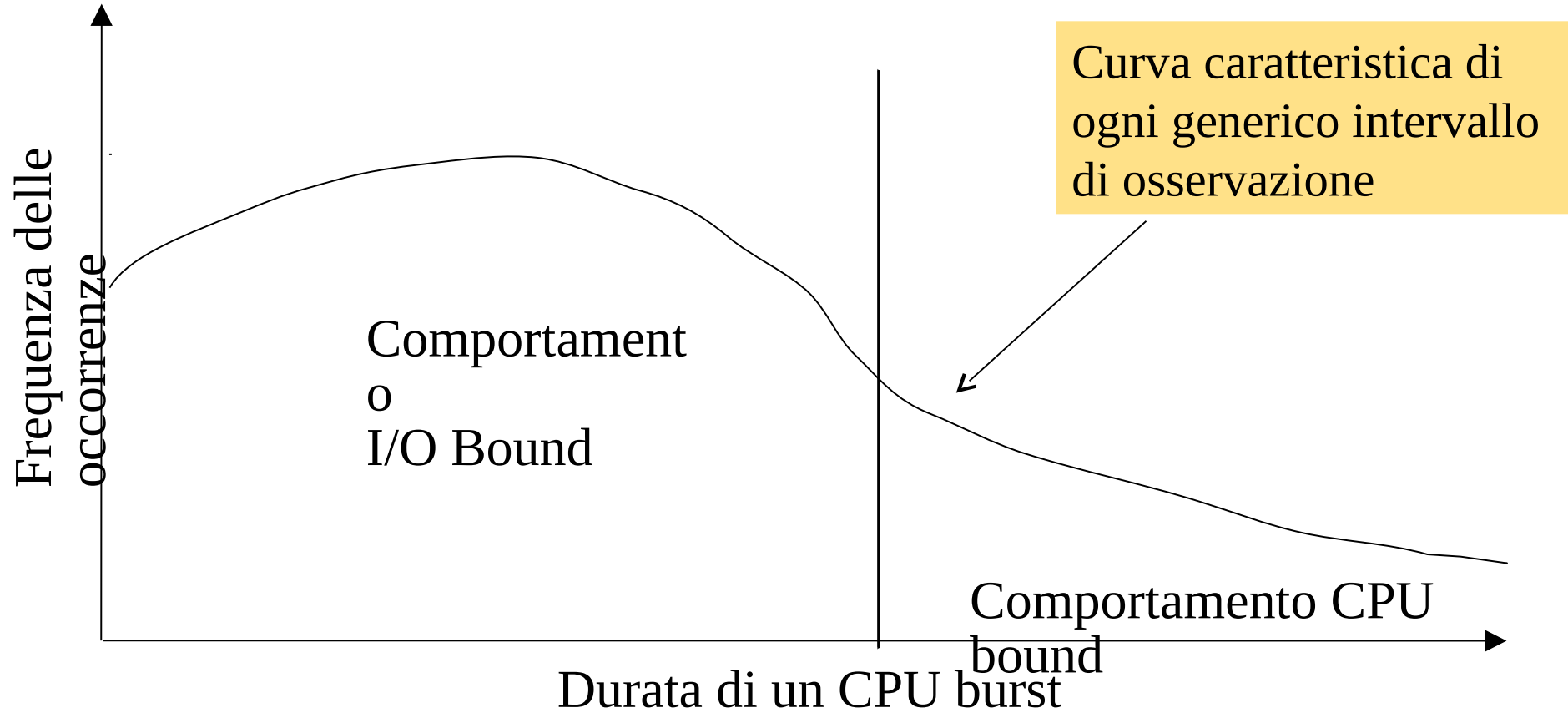


soluzione parziale:

quanto di tempo pari a  $2^i$  dove  $i$  e' la priorit 



# Caratteristiche del carico reale



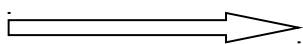
# Scheduling UNIX tradizionale (SVR3 – 4.3 BSD)

## Caratteristiche

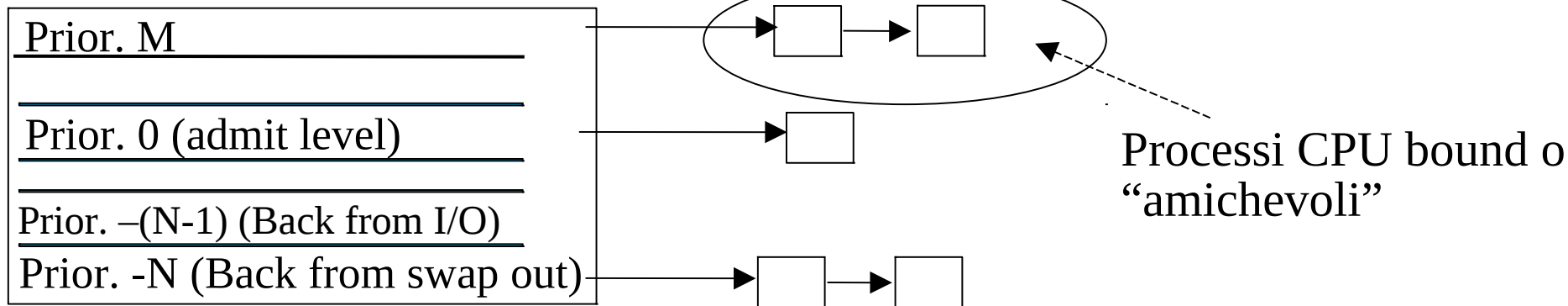
- code multiple con feedback
- un livello di priorit a' distinto per ciascuna coda
- gestione di tipo Round-Robin nell'ambito di ciascuna coda

## Passaggio da una coda all'altra (feedback)

- in caso di rientro nello stato Ready dopo un passaggio nello stato Sleep
- in caso di variazione della priorit a' imposto dal sistema  
(su base periodica)



$$P = base + \frac{CPU}{2} + nice$$



... tale scheduler di CPU e' tutt'ora riadattato per  
sistemi orientati ai threads ...

# Variazione della “niceness” su sistemi UNIX

**Baseline system call**    `int nice(int incr)`

The range of the nice value is +19 (low priority) to -20 (high priority).  
Attempts to set a nicevalue outside the range are clamped to the range

questa e' la system call che viene invocata dalla shell quando si passa sulla linea di comando il comando **nice [+number][command]**

# System call addizionali

## SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

int getpriority(int which, int who);
int setpriority(int which, int who, int prio);
```

## DESCRIPTION

The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the `getpriority` call and set with the `setpriority` call. *Which* is one of `PRIO_PROCESS`, `PRIO_PGRP`, or `PRIO_USER`, and *who* is interpreted relative to *which* (a process identifier for `PRIO_PROCESS`, process group identifier for `PRIO_PGRP`, and a user ID for `PRIO_USER`). A zero value for *who* denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process. *Prio* is a value in the range -20 to 20 (but see the Notes below). The default priority is 0; lower priorities cause more favorable scheduling

# LINUX auto-grouping

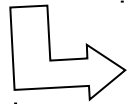
- In LINUX a partire dal kernel 2.6.38 (2010) viene offerto il supporto per l'auto-grouping
- Questo associa tutti i processi lanciati da uno stesso terminale (ed i loro thread) in un unico gruppo
- In tal caso il cambio del `nice` ha effetto solo all'interno del gruppo (per cui la ripartizione dell'uso della CPU tra gruppi rimane equa)
- Per escludere l'auto-grouping si può utilizzare il file di configurazione di sistema `/proc/sys/kernel/sched_autogroup_enabled`
- Valore 0 esclude l'auto-grouping

... scheduling multiprocessore e per sistemi basati  
su threads

# Sistemi multiprocessore

## Caratteristiche architetturali

- unita' di calcolo (CPU/CPU-core) multiple che condividono una memoria principale comune
- i processori sono controllati da un unico sistema operativo



tightly coupled system (sistema strettamente accoppiato)

---

## Problematiche

- assegnazione dei processi ai processori
- uso (o non) di politiche classiche di multiprogrammazione sui singoli processori
- selezione dell'entita' schedulabile da mandare in esecuzione



# Assegnazione di processi/thread alle unità di calcolo

## Statica

- overhead ridotto poiche' l'assegnazione e' unica per tutta la durata del processo
- possibilita' di sottoutilizzo dei processori

## Dinamica (convenzionalmente utilizzata in sistemi moderni)

- overhead superiore dovuto a riassegnazioni multiple
  - migliore utilizzo dei processori
- 

## Approccio master/slave

- il sistema operativo viene eseguito su una specifica unita' di calcolo
- richiesta esplicita di accesso a strutture del kernel da parte delle altre
- semplicita' di progetto (estensione di kernel classici per uniprocessori)

## Approccio peer (convenzionalmente utilizzato in sistemi moderni)

- il sistema operativo viene eseguito su tutte le unita' di calcolo
- problemi di coerenza dei dati, inclusi quelli di gestione dello scheduling
- complessita' di progetto

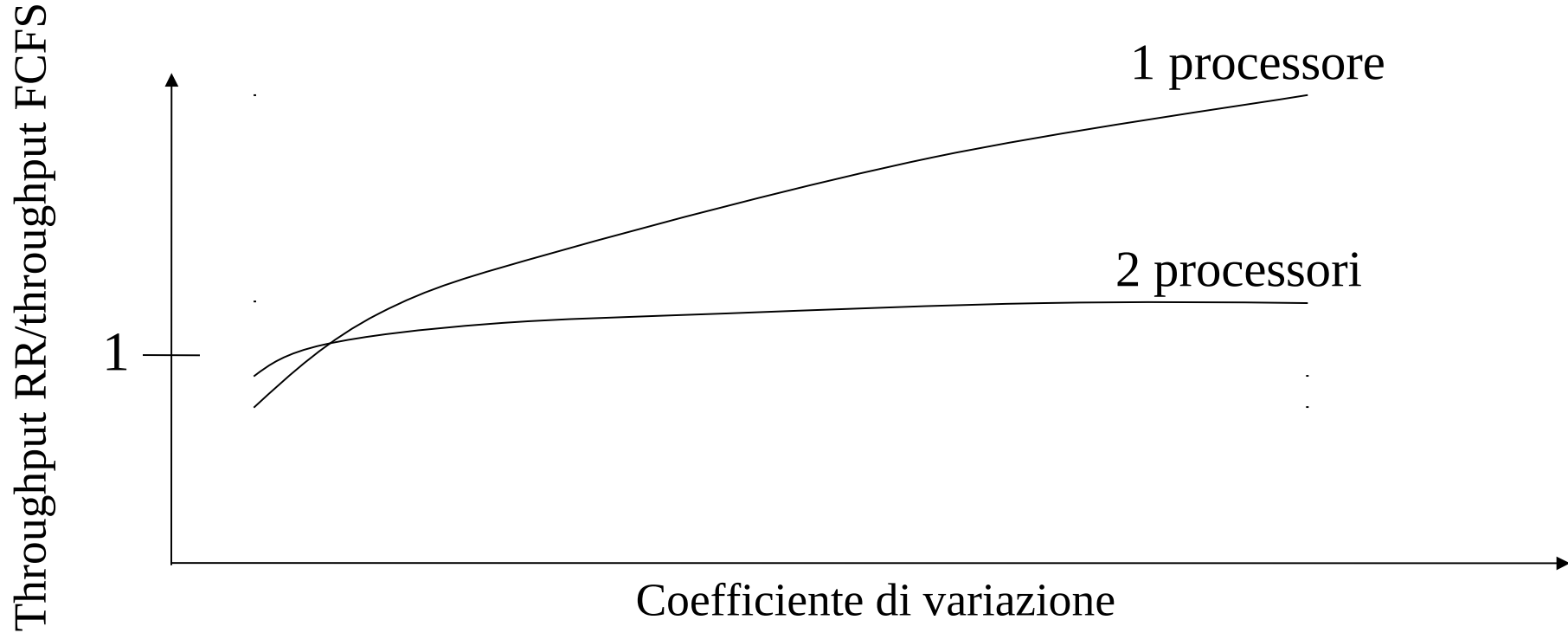
# Multiprogrammazione sui singoli processori?

## Non piu' mandatorio?

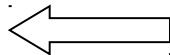
- quando sono disponibili molti processori, il livello di utilizzo del processore non e' piu' un fattore cosi' critico (dato il costo proporzionalmente ridotto del processore rispetto a quello dell'intera architettura)
- rientra in gioco la metrica del **tempo di turnaround** delle applicazioni

# Di nuovo sullo scheduling di processi

Limitato impatto della politica di selezione



$$C_s = \frac{\sigma_s}{s}$$

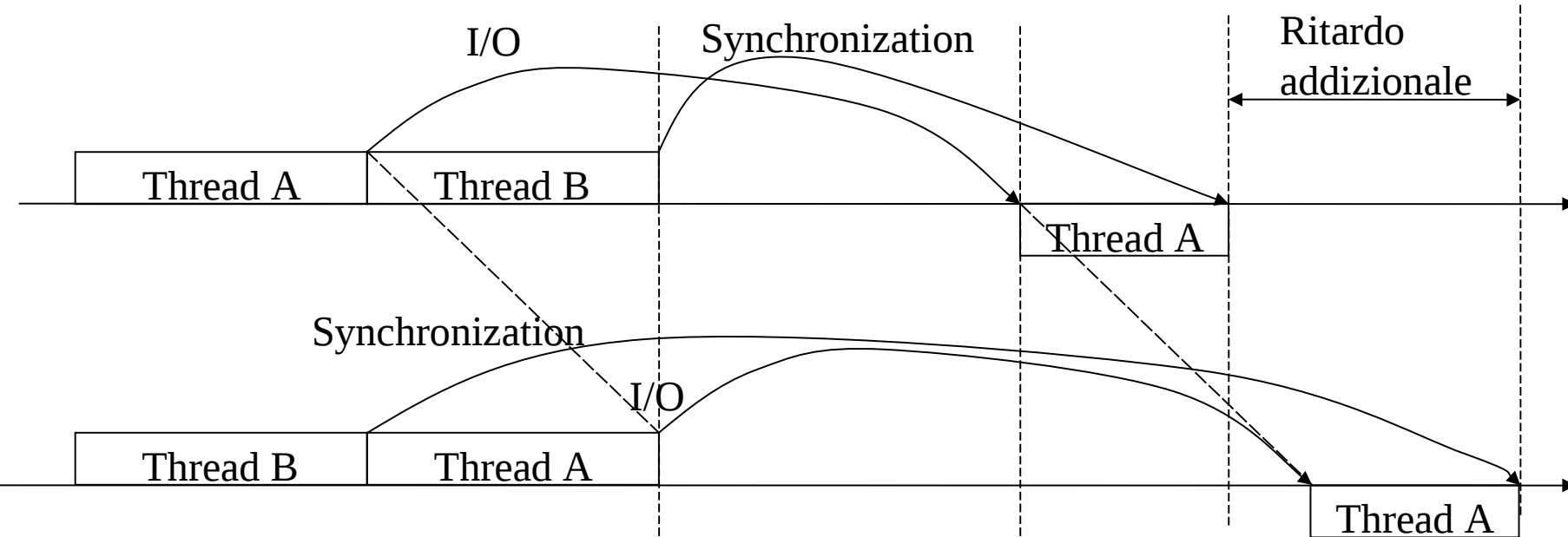


$\frac{\text{Deviazione standard tempo di servizio}}{\text{Tempo di servizio medio}}$

# Scheduling di threads

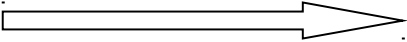
## Fattori nuovi

- la decomposizione di applicazioni in threads introduce criteri di selezione innovativi rispetto alle priorità classiche (interattività)
- un processo può essere sia CPU che I/O bound dipendendo dal comportamento dei singoli thread che lo compongono



# Politiche di scheduling di threads

## Load sharing (e.g. Linux 2.4)

- coda globale di threads pronti ad eseguire
- possibilita' di gestire priorit  
- distribuzione uniforme del carico
- problemi di scalabilit  nell'accesso alla coda globale in caso di macchine altamente parallele
- ridotta efficienza del caching in caso di cambio di processore da parte dei thread

FCFS

SNTF (smallest number of threads first), con e senza Preemption

.....

## Load balancing (e.g. Linux 2.6 e successivi)

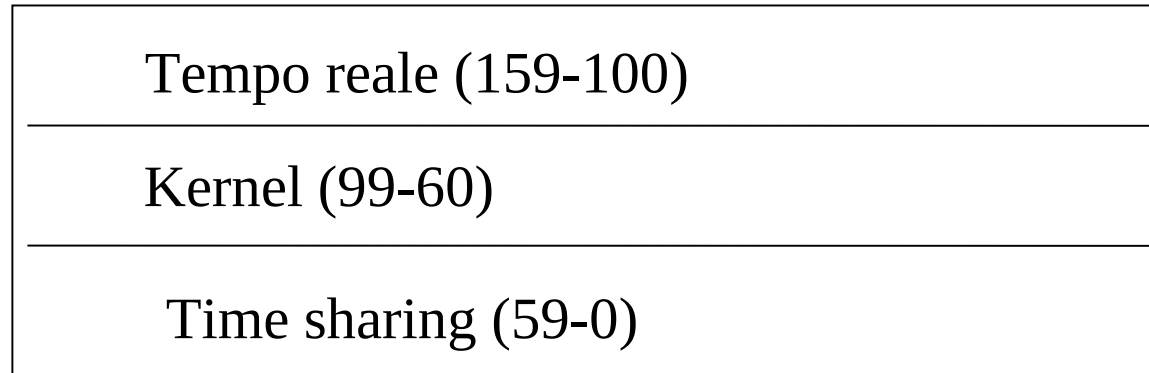
- code di threads pronti ad eseguire separate, una per ogni CPU-core
- migliore scalabilit  delle operazioni di scheduling su macchine altamente parallele
- spostamento periodico di thread da una coda ad un'altra in caso di sbilanciamento del carico

# Scheduling UNIX SVR4

## Caratteristiche

- 160 livelli di priorit 
- 3 classi di priorit : Tempo Reale (159-100), Kernel (99-60), Time-Sharing (59-0)
- **kernel preemptabile** (identificazione di safe places)
- **bitmap** per determinare i livelli non vuoti
- quanto di tempo variabile in funzione della classe e, in alcune classi, del livello

Sequenza di scheduling



# LINUX extended priority scheme

```
#include <sched.h>
```

```
int sched_setscheduler(pid_t pid, int policy, const struct sched_param *p);
```

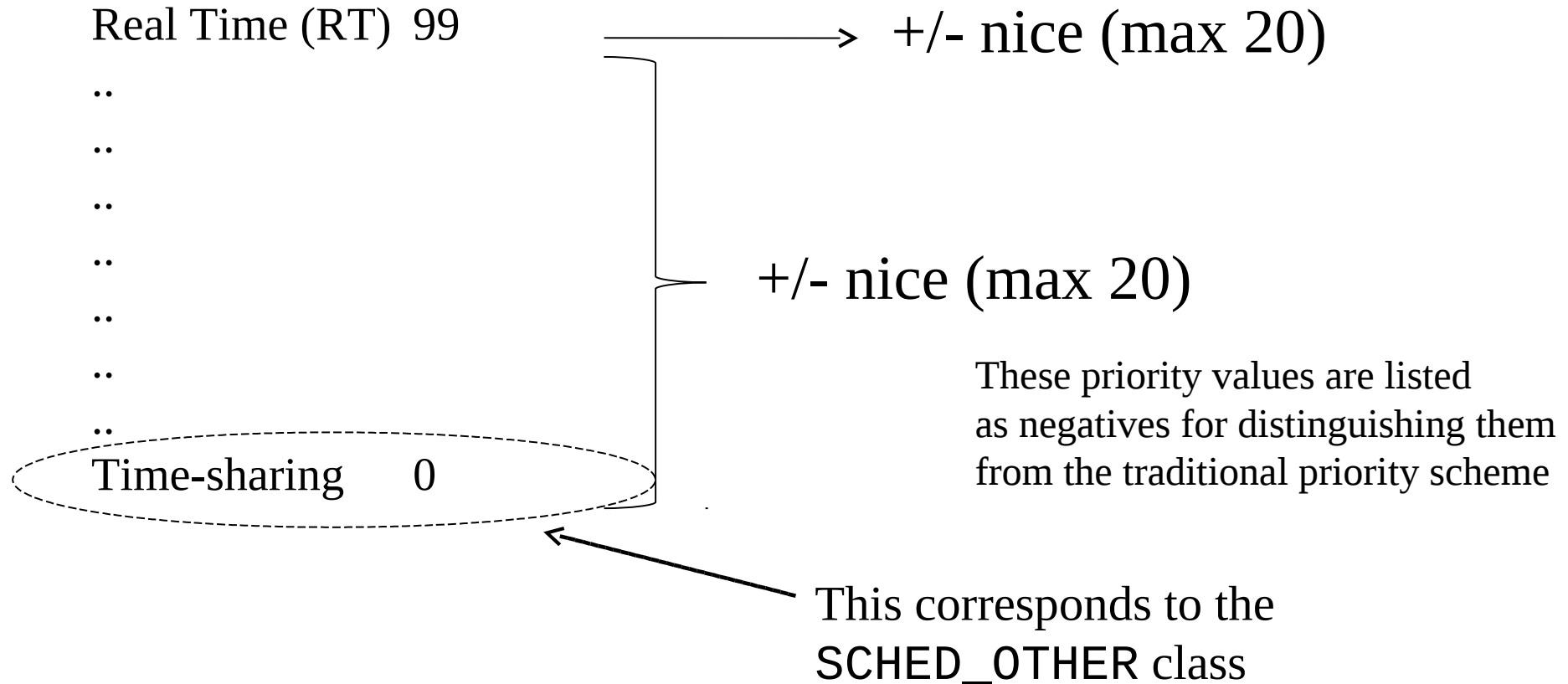
```
int sched_getscheduler(pid_t pid);
```

```
struct sched_param {  
    ...  
    int sched_priority;  
    ...  
};
```

---

From the shell we have the **chrt** command

# Actual LINUX priorities with the extended scheme





# Epoche di scheduling in LINUX

- LINUX utilizza il concetto di **“epoca di scheduling”** (in realta' gia' presente in LINUX prima dello schema esteso)
- Un'epoca e' un periodo di tempo per l'operativita' del sistema
- All'inizio di ogni epoca ad ogni thread attivo viene assegnato un numero di **“quanti di tempo”** da poter spendere
- Ad ogni nuovo thread nato durante un'epoca viene data parte del “budget” di tempo di CPU del padre
- Questo permette di avere la possibilita' di eseguire comunque thread di classi di priorita' basse, se pur con minimi quanti di tempo assegnati ad essi per ogni epoca

# Scheduling in sistemi Windows

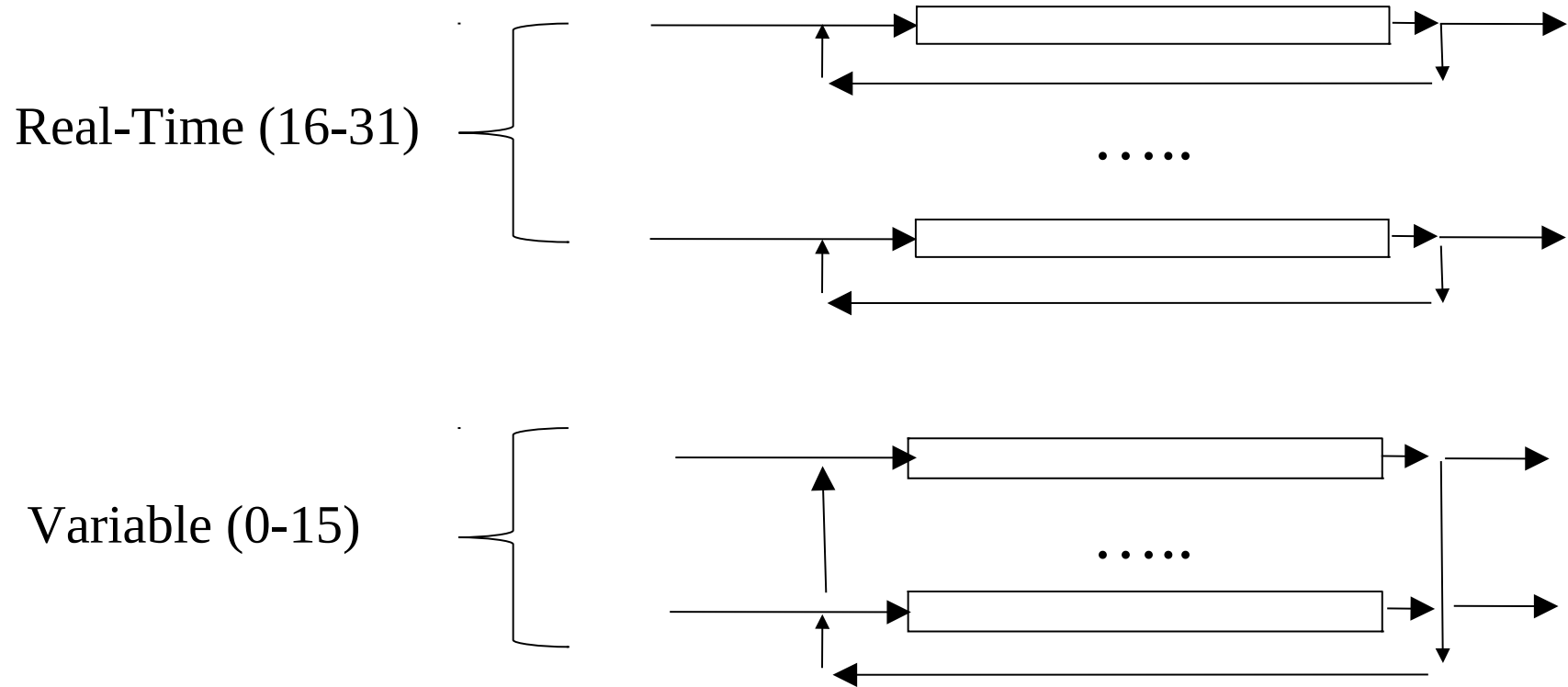
## Caratteristiche

- code multiple distinte in due fasce: Real-Time e Variable
- un livello di priorit  distinto per coda (0-15 fascia Variable – 16-31 fascia Real Time)
- gestione di tipo Round-Robin nell'ambito di ciascuna coda
- priorit  base per i processi
- priorit  dinamica entro vincoli per i threads
- prerilascio basato su priorit 

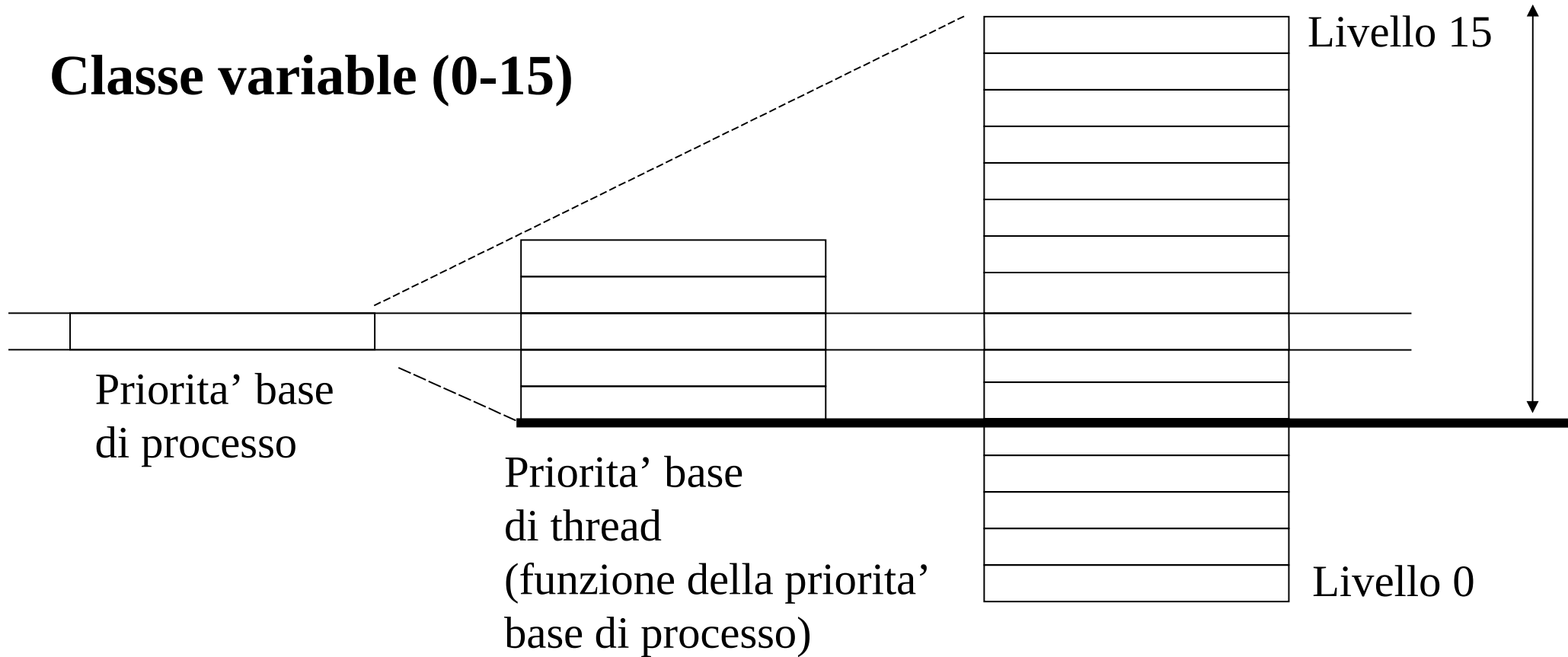
## Passaggio da una coda all'altra (feedback)

- non ammesso nella fascia Real-Time
- ammesso nella fascia Variable (rilascio della CPU allo scadere del quanto
- provoca diminuzione della priorit , rilascio anticipato provoca incremento)

# Uno schema



# Priorita' dei threads in Windows



# System calls (i)

```
BOOL WINAPI SetPriorityClass(  
    _In_   HANDLE hProcess,  
    _In_   DWORD dwPriorityClass  
);
```

| Priority   | Meaning  |
|--|--|
| <b>ABOVE_NORMAL_PRIORITY_CLASS</b><br>0x00008000   | Process that has priority above <b>NORMAL_PRIORITY_CLASS</b> but below <b>HIGH_PRIORITY_CLASS</b> .  |
| <b>BELOW_NORMAL_PRIORITY_CLASS</b><br>0x00004000   | Process that has priority above <b>IDLE_PRIORITY_CLASS</b> but below <b>NORMAL_PRIORITY_CLASS</b> .  |
| <b>HIGH_PRIORITY_CLASS</b><br>0x00000080           | Process that performs time-critical tasks that must be executed immediately. The threads of the process preempt the threads of normal or idle priority class processes. An example is the Task List, which must respond quickly when called by the user, regardless of the load on the operating system. Use extreme care when using the high-priority class, because a high-priority class application can use nearly all available CPU time.                               |
| <b>IDLE_PRIORITY_CLASS</b><br>0x00000040           | Process whose threads run only when the system is idle. The threads of the process are preempted by the threads of any process running in a higher priority class. An example is a screen saver. The idle-priority class is inherited by child processes.  |
| <b>NORMAL_PRIORITY_CLASS</b><br>0x00000020         | Process with no special scheduling needs.  |
| <b>PROCESS_MODE_BACKGROUND_BEGIN</b><br>0x00100000 | Begin background processing mode. The system lowers the resource scheduling priorities of the process (and its threads) so that it can perform background work without significantly affecting activity in the foreground.<br>This value can be specified only if <i>hProcess</i> is a handle to the current process. The function fails if the process is already in background processing mode.<br><b>Windows Server 2003 and Windows XP:</b> This value is not supported. |
| <b>PROCESS_MODE_BACKGROUND_END</b><br>0x00200000   | End background processing mode. The system restores the resource scheduling priorities of the process (and its threads) as they were before the process entered background processing mode.<br>This value can be specified only if <i>hProcess</i> is a handle to the current process. The function fails if the process is not in background processing mode.<br><b>Windows Server 2003 and Windows XP:</b> This value is not supported.                                    |
| <b>REALTIME_PRIORITY_CLASS</b><br>0x00000100       | Process that has the highest possible priority. The threads of the process preempt the threads of all other processes, including operating system processes performing important tasks. For example, a real-time process that executes for more than a very brief interval can cause disk caches not to flush or cause the mouse to be unresponsive.   |

# System calls (ii)

```
BOOL WINAPI SetThreadPriority(  
    _In_   HANDLE hThread,  
    _In_   int nPriority  
);
```

| Priority  | Meaning  |
|---|--|
| <b>THREAD_MODE_BACKGROUND_BEGIN</b><br>0x00010000 | Begin background processing mode. The system lowers the resource scheduling priorities of the thread so that it can perform background work without significantly affecting activity in the foreground.<br>This value can be specified only if <i>hThread</i> is a handle to the current thread. The function fails if the thread is already in background processing mode.<br><b>Windows Server 2003 and Windows XP:</b> This value is not supported. |
| <b>THREAD_MODE_BACKGROUND_END</b><br>0x00020000   | End background processing mode. The system restores the resource scheduling priorities of the thread as they were before the thread entered background processing mode.<br>This value can be specified only if <i>hThread</i> is a handle to the current thread. The function fails if the thread is not in background processing mode.<br><b>Windows Server 2003 and Windows XP:</b> This value is not supported.                                     |
| <b>THREAD_PRIORITY_ABOVE_NORMAL</b><br>1          | Priority 1 point above the priority class.   |
| <b>THREAD_PRIORITY_BELOW_NORMAL</b><br>-1         | Priority 1 point below the priority class.   |
| <b>THREAD_PRIORITY_HIGHEST</b><br>2               | Priority 2 points above the priority class.  |
| <b>THREAD_PRIORITY_IDLE</b><br>-15                | Base priority of 1 for <b>IDLE_PRIORITY_CLASS</b> , <b>BELOW_NORMAL_PRIORITY_CLASS</b> , <b>NORMAL_PRIORITY_CLASS</b> , <b>ABOVE_NORMAL_PRIORITY_CLASS</b> , or <b>HIGH_PRIORITY_CLASS</b> processes, and a base priority of 16 for <b>REALTIME_PRIORITY_CLASS</b> processes.  |
| <b>THREAD_PRIORITY_LOWEST</b><br>-2               | Priority 2 points below the priority class.  |
| <b>THREAD_PRIORITY_NORMAL</b><br>0                | Normal priority for the priority class.  |
| <b>THREAD_PRIORITY_TIME_CRITICAL</b><br>15        | Base priority of 15 for <b>IDLE_PRIORITY_CLASS</b> , <b>BELOW_NORMAL_PRIORITY_CLASS</b> , <b>NORMAL_PRIORITY_CLASS</b> , <b>ABOVE_NORMAL_PRIORITY_CLASS</b> , or <b>HIGH_PRIORITY_CLASS</b> processes, and a base priority of 31 for <b>REALTIME_PRIORITY_CLASS</b> processes.   |

If the thread has the **REALTIME\_PRIORITY\_CLASS** base class, this parameter can also be -7, -6, -5, -4, -3, 3, 4, 5, or 6. For more information, see [Scheduling Priorities](#).

# Schema completo delle priorit 

| Process priority class      | Thread priority level         | Base priority |
|-----------------------------|-------------------------------|---------------|
| IDLE_PRIORITY_CLASS         | THREAD_PRIORITY_IDLE          | 1             |
|                             | THREAD_PRIORITY_LOWEST        | 2             |
|                             | THREAD_PRIORITY_BELOW_NORMAL  | 3             |
|                             | THREAD_PRIORITY_NORMAL        | 4             |
|                             | THREAD_PRIORITY_ABOVE_NORMAL  | 5             |
|                             | THREAD_PRIORITY_HIGHEST       | 6             |
|                             | THREAD_PRIORITY_TIME_CRITICAL | 15            |
| BELOW_NORMAL_PRIORITY_CLASS | THREAD_PRIORITY_IDLE          | 1             |
|                             | THREAD_PRIORITY_LOWEST        | 4             |
|                             | THREAD_PRIORITY_BELOW_NORMAL  | 5             |
|                             | THREAD_PRIORITY_NORMAL        | 6             |
|                             | THREAD_PRIORITY_ABOVE_NORMAL  | 7             |
|                             | THREAD_PRIORITY_HIGHEST       | 8             |
|                             | THREAD_PRIORITY_TIME_CRITICAL | 15            |

|                             |                               |    |
|-----------------------------|-------------------------------|----|
| NORMAL_PRIORITY_CLASS       | THREAD_PRIORITY_IDLE          | 1  |
|                             | THREAD_PRIORITY_LOWEST        | 6  |
|                             | THREAD_PRIORITY_BELOW_NORMAL  | 7  |
|                             | THREAD_PRIORITY_NORMAL        | 8  |
|                             | THREAD_PRIORITY_ABOVE_NORMAL  | 9  |
|                             | THREAD_PRIORITY_HIGHEST       | 10 |
|                             | THREAD_PRIORITY_TIME_CRITICAL | 15 |
| ABOVE_NORMAL_PRIORITY_CLASS | THREAD_PRIORITY_IDLE          | 1  |
|                             | THREAD_PRIORITY_LOWEST        | 8  |
|                             | THREAD_PRIORITY_BELOW_NORMAL  | 9  |
|                             | THREAD_PRIORITY_NORMAL        | 10 |
|                             | THREAD_PRIORITY_ABOVE_NORMAL  | 11 |
|                             | THREAD_PRIORITY_HIGHEST       | 12 |
|                             | THREAD_PRIORITY_TIME_CRITICAL | 15 |



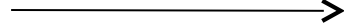
|                         |                               |    |
|-------------------------|-------------------------------|----|
| HIGH_PRIORITY_CLASS     | THREAD_PRIORITY_IDLE          | 1  |
|                         | THREAD_PRIORITY_LOWEST        | 11 |
|                         | THREAD_PRIORITY_BELOW_NORMAL  | 12 |
|                         | THREAD_PRIORITY_NORMAL        | 13 |
|                         | THREAD_PRIORITY_ABOVE_NORMAL  | 14 |
|                         | THREAD_PRIORITY_HIGHEST       | 15 |
|                         | THREAD_PRIORITY_TIME_CRITICAL | 15 |
| REALTIME_PRIORITY_CLASS | THREAD_PRIORITY_IDLE          | 16 |
|                         | THREAD_PRIORITY_LOWEST        | 22 |
|                         | THREAD_PRIORITY_BELOW_NORMAL  | 23 |
|                         | THREAD_PRIORITY_NORMAL        | 24 |
|                         | THREAD_PRIORITY_ABOVE_NORMAL  | 25 |
|                         | THREAD_PRIORITY_HIGHEST       | 26 |
|                         | THREAD_PRIORITY_TIME_CRITICAL | 31 |

# Affinita' di processore

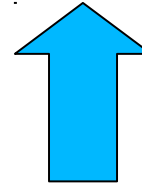
- porta un qualsiasi thread ad essere eseguito temporaneamente solo su specifici CPU-core
- la scelta puo' dipendere da politiche interne del sistema operativo (e.g. Linux 2.6 e piu' recenti)
- oppure puo' essere determinata da utenti e/o amministratori di sistema
- e' una facility molto utile, per esempio permette di simulare scenari dove le applicazioni vengono attivate su CPU-core risultanti come sottoinsieme di quelli realmente disponibili nell'architettura sottostante (scale down)
- permette anche il reserving di CPU-core per, e.g. mansioni (e quindi applicazioni) critiche

# Supporto per l'on-demand affinity

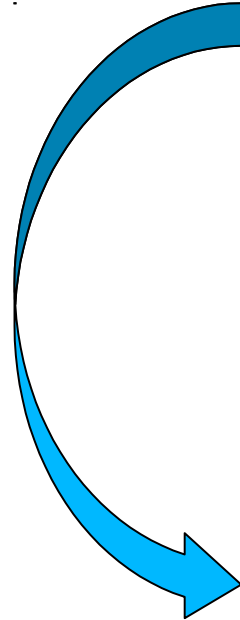
Per thread  
CPU mask



0011..0...11...0...101



Allowed/denied CPUs



La 'mask' del parent viene ereditata da ogni thread child (ma e' poi modificabile)

# Affinita' in sistemi Unix

SCHED\_SETAFFINITY(2)

Linux Programmer's Manual

SCHED\_SETAFFINITY(2)

## NAME [top](#)

`sched_setaffinity`, `sched_getaffinity` - set and get a thread's CPU affinity mask

## SYNOPSIS [top](#)

```
#define _GNU_SOURCE                /* See feature_test_macros(7) */
#include <sched.h>

int sched_setaffinity(pid_t pid, size_t cpusetsize,
                      const cpu_set_t *mask);

int sched_getaffinity(pid_t pid, size_t cpusetsize,
                      cpu_set_t *mask);
```

## DESCRIPTION [top](#)

These are the backend of the taskset shell command



# Affinita' processo/CPU-cores in sistemi Windows

## Syntax

C++

```
BOOL WINAPI SetProcessAffinityMask(  
    _In_ HANDLE      hProcess,  
    _In_ DWORD_PTR   dwProcessAffinityMask  
);
```

## Parameters

*hProcess* [in]

A handle to the process whose affinity mask is to be set. This handle must have the PROCESS\_SET\_INFORMATION access right. For more information, see [Process Security and Access Rights](#).

*dwProcessAffinityMask* [in]

The affinity mask for the threads of the process.

On a system with more than 64 processors, the affinity mask must specify processors in a single [processor group](#).

# Affinita' thread/CPU-cores in sistemi Windows

Sets a processor affinity mask for the specified thread.

## Syntax

C++

```
DWORD_PTR WINAPI SetThreadAffinityMask(  
    _In_ HANDLE      hThread,  
    _In_ DWORD_PTR   dwThreadAffinityMask  
);
```

↘ Within current processor group