

SCHEDULING DELLA CPU

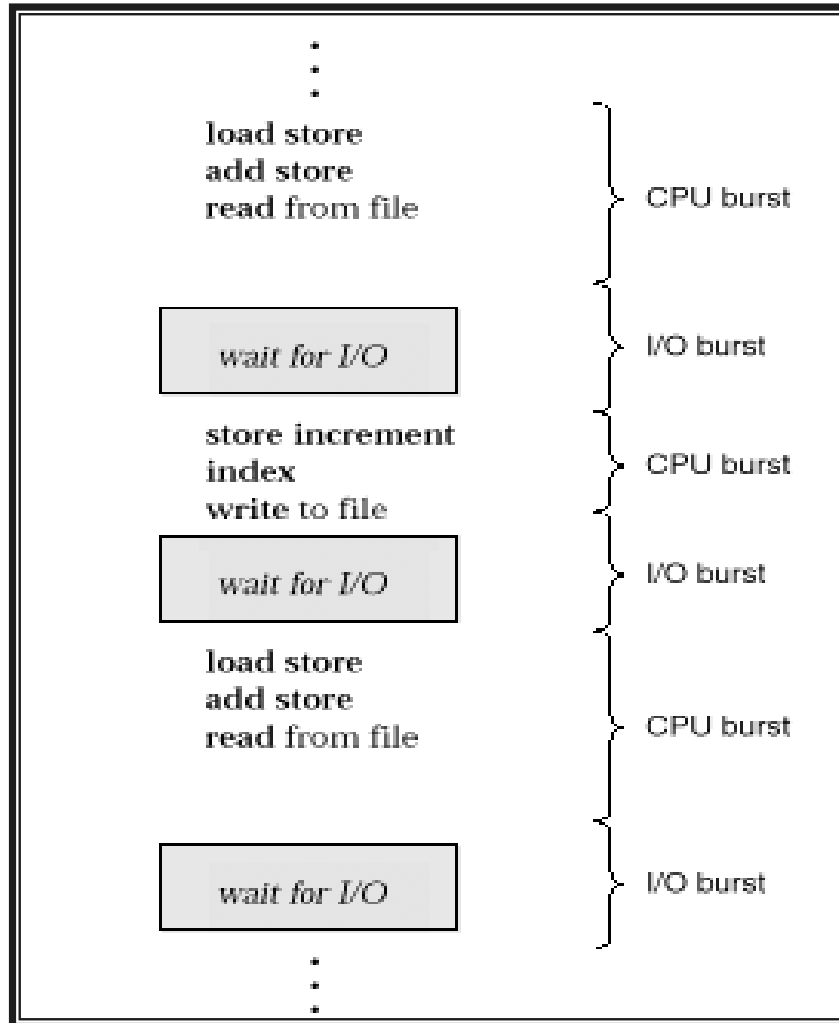
Scheduling della CPU

- Concetti di Base
- Criteri di Scheduling
- Algoritmi di Scheduling
 - FCFS, SJF, Round-Robin, A code multiple
- Scheduling in Multi-Processori
- Scheduling Real-Time
- Valutazione di Algoritmi

Concetti di Base

- Lo scheduling della CPU è l'elemento fondamentale dei sistemi operativi con multiprogrammazione.
- L'obiettivo dello scheduling è la massimizzazione dell'utilizzo della CPU.
- Questo si ottiene assegnando al processore processi che sono pronti per eseguire delle istruzioni.
- **Ciclo CPU – I/O Burst** : l'esecuzione di un processo consiste di un ciclo di esecuzione nella CPU e attesa di I/O.

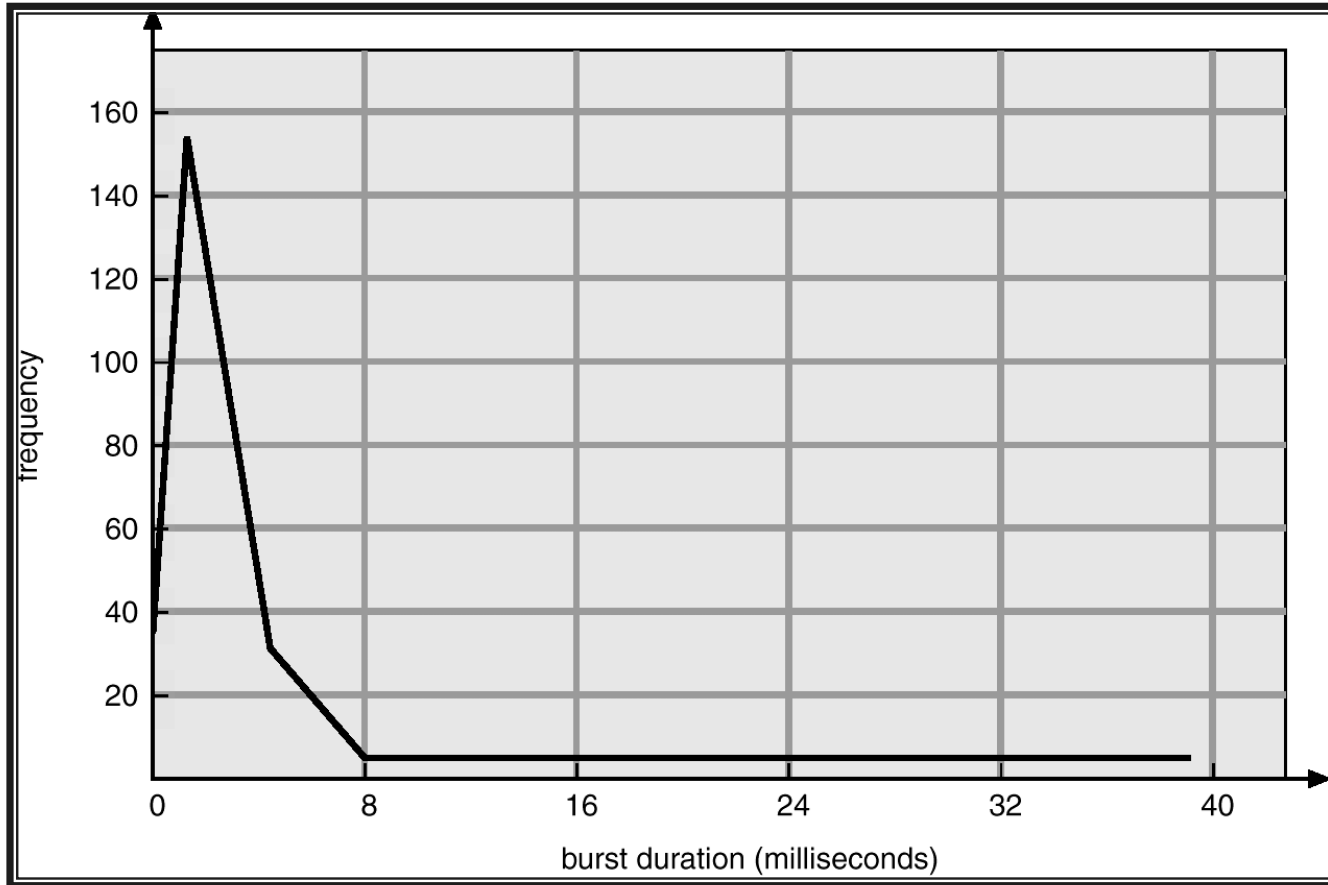
Sequenza Alternata di CPU e I/O Burst



Concetti di Base

- La distribuzione dei CPU burst dipende dalle attività dei diversi programmi.
- La frequenza dei CPU burst brevi è molto alta mentre la frequenza dei CPU burst lunghi è molto bassa.
- Differenza tra processi *CPU bound* e processi *I/O bound*.
- Queste caratteristiche sono considerate nella selezione delle strategie di scheduling.

Istogramma dei CPU burst



Scheduler della CPU

- Lo scheduler a breve termine seleziona uno tra i processi in memoria pronti per essere eseguiti (*ready queue*) e lo assegna alla CPU.
- Lo scheduler interviene quando un processo:
 1. Passa dallo stato **running** allo stato **waiting**.
 2. Passa dallo stato **running** allo stato **ready**.
 3. Passa dallo stato **waiting** allo stato **ready**.
 4. **Termina**.
- Nei casi 1 e 4 lo scheduling è ***nonpreemptive*** (*senza prelazione*).
- Negli altri casi è ***preemptive*** (*con prelazione*).

Dispatcher

- Il modulo **dispatcher** svolge il lavoro di passare il controllo ai processi selezionati dallo scheduler della CPU per la loro esecuzione. Esso svolge:
 - il context switch
 - il passaggio al modo utente
 - il salto alla istruzione da eseguire del programma corrente.
- Il **dispatcher** deve essere molto veloce.
- *Latenza di dispatch* – tempo impiegato dal dispatcher per fermare un processo e far eseguire il successivo.

Criteri di Scheduling

- Nella scelta di una strategia di scheduling occorre tenere conto delle diverse caratteristiche dei programmi.
- CRITERI da considerare:
 - **Utilizzo della CPU** – avere la CPU il più attiva possibile
 - **Throughput** – n° di processi completati nell'unità di tempo
 - **Tempo di turnaround** – tempo totale per eseguire un processo
 - **Tempo di waiting** – tempo totale di attesa sulla ready queue
 - **Tempo di risposta** – tempo da quando viene inviata una richiesta fino a quando si produce una prima risposta (non considerando il tempo di output).

Criteri di Ottimizzazione

CRITERI:

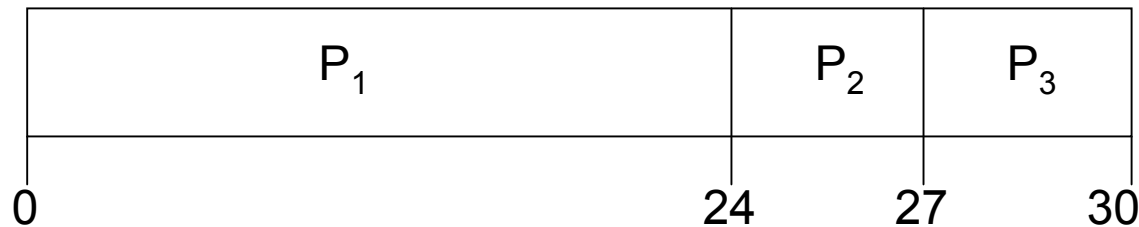
- Massimizzare l'utilizzo della CPU
 - Massimizzare il throughput
 - Minimizzare il tempo di turnaround
 - Minimizzare il tempo di waiting
 - Minimizzare il tempo di risposta
-
- Generalmente si tende ad ottimizzare i valori medi.
-
- Nei sistemi time-sharing è più importante **minimizzare la varianza del tempo di risposta.**

First-Come, First-Served (FCFS) Scheduling

- Primo arrivato, primo servito (gestito con coda FIFO).

<u>Processo</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

- Supponiamo che i processi arrivino nell'ordine: P_1 , P_2 , P_3
Lo schema di Gantt è:



- Tempo di waiting per: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Tempo di waiting medio: $(0 + 24 + 27)/3 = 17$

Scheduling FCFS

Supponiamo che i processi arrivino nell'ordine

P_2, P_3, P_1 .

- Lo schema di Gantt è:



- Tempo di waiting per $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Tempo di waiting medio: $(6 + 0 + 3)/3 = 3$
- Molto meglio che nel caso precedente.
- *Effetto convoglio*: i processi "brevis" attendono i processi "lunghe".

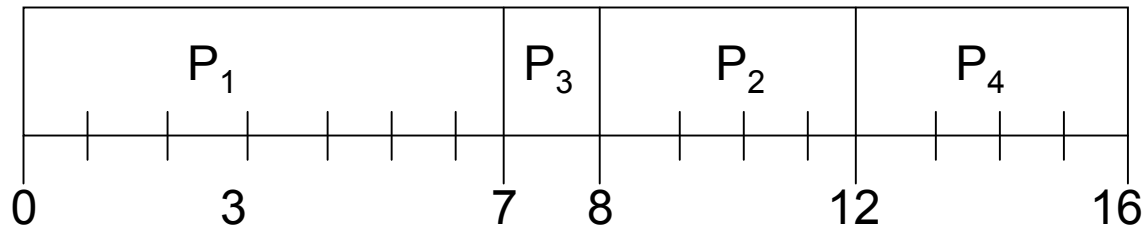
Scheduling Shortest-Job-First (SJF)

- Associa ad ogni processo la lunghezza del prossimo CPU burst. Use questi tempi per schedulare il processo con la lunghezza minima.
- Due schemi:
 - ***nonpreemptive*** – il processo assegnato non può essere sospeso prima di completare il suo CPU burst.
 - ***preemptive*** – se arriva un nuovo processo con un CPU burst più breve del tempo rimanente per il processo corrente, viene servito. Questo schema è conosciuto come ***Shortest-Remaining-Time-First (SRTF)***.
- *SJF è ottimale* – offre il minimo tempo medio di attesa per un insieme di processi.

Esempio di Non-Preemptive SJF

<u>Processo</u>	<u>Tempo Arrivo</u>	<u>Tempo di Burst</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

■ SJF (non-preemptive)

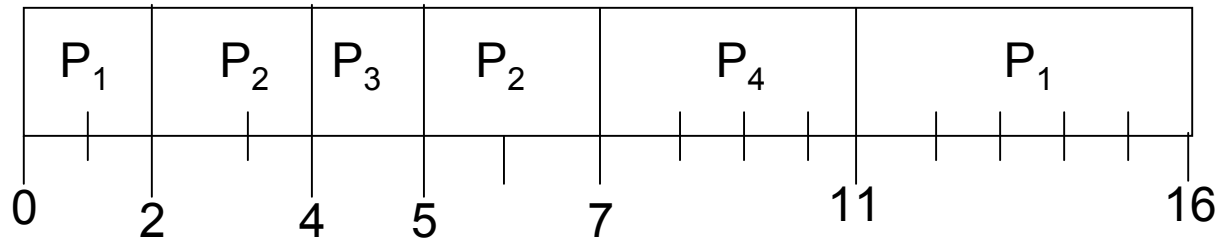


■ Tempo di attesa medio = $(0 + 6 + 3 + 7)/4 - 4$

Esempio di Preemptive SJF

<u>Processo</u>	<u>Tempo di Arrivo</u>	<u>Tempo di Burst</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

■ SJF (preemptive)



■ Tempo di attesa medio = $(9 + 1 + 0 + 2)/4 - 3$

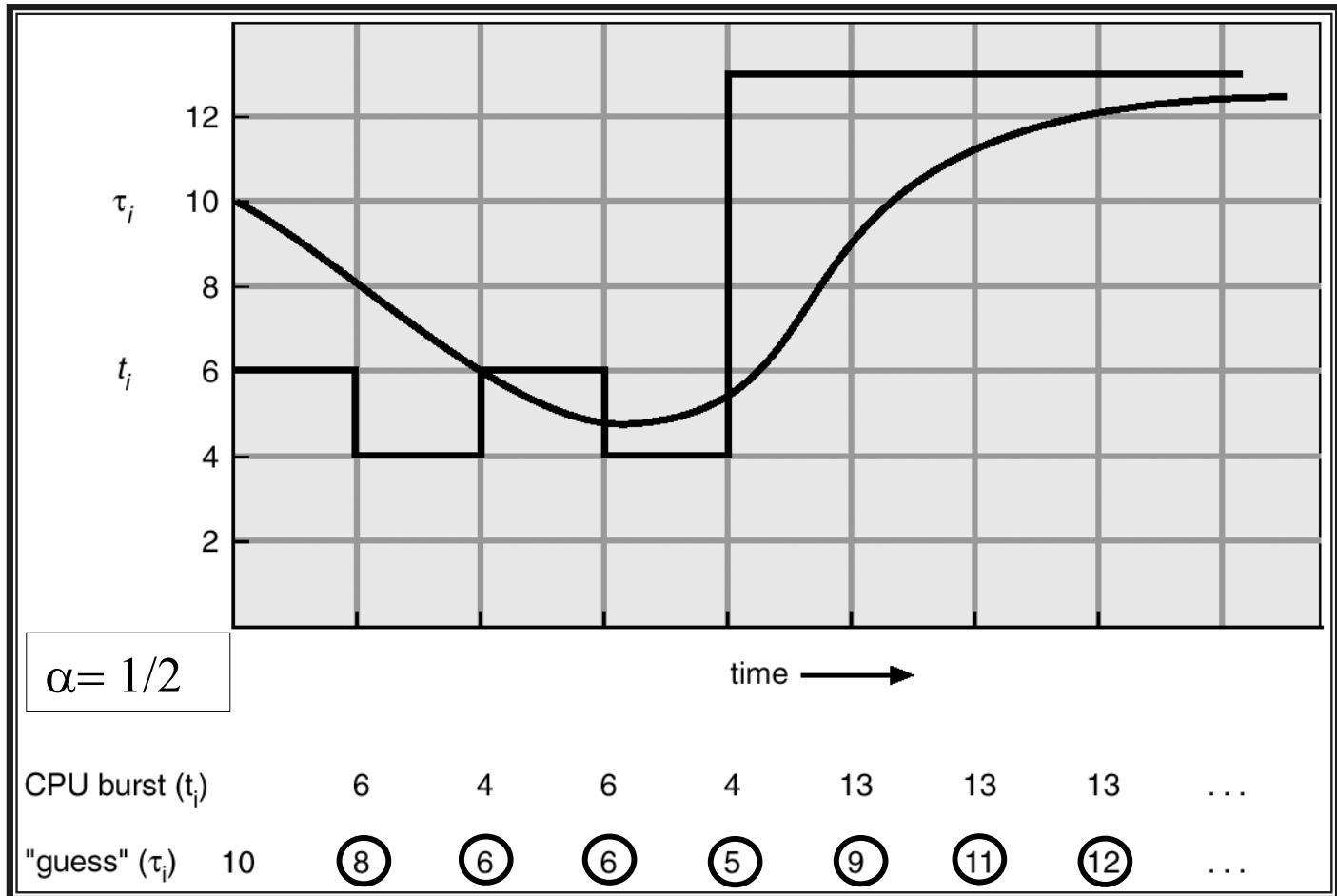
Lunghezza del prossimo CPU Burst ?

- La lunghezza del prossimo CPU burst non si conosce.
- Ma può essere stimato.
- Usando la lunghezza dei precedenti CPU burst e usando una media esponenziale:

t_n	lunghezza dell'n-esimo CPU burst
τ_{n+1}	valore predetto del prossimo CPU burst
α	$0 \leq \alpha \leq 1$

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

Predizione della lunghezza del prossimo CPU Burst



Esempi di media esponenziale

- $\alpha = 0$

$$\tau_{n+1} = \tau_n$$

- La storia recente non conta.

- $\alpha = 1$

$$\tau_{n+1} = t_n$$

- Conta solo l'ultimo valore reale del CPU burst.

- Se espandiamo la formula, si ha:

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots \\ & + (1 - \alpha)^2 \alpha t_{n-2} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Poiché sia α che $(1 - \alpha)$ sono minori o uguali ad 1, ogni termine successivo da un contributo sempre più piccolo.

Scheduling con Priorità

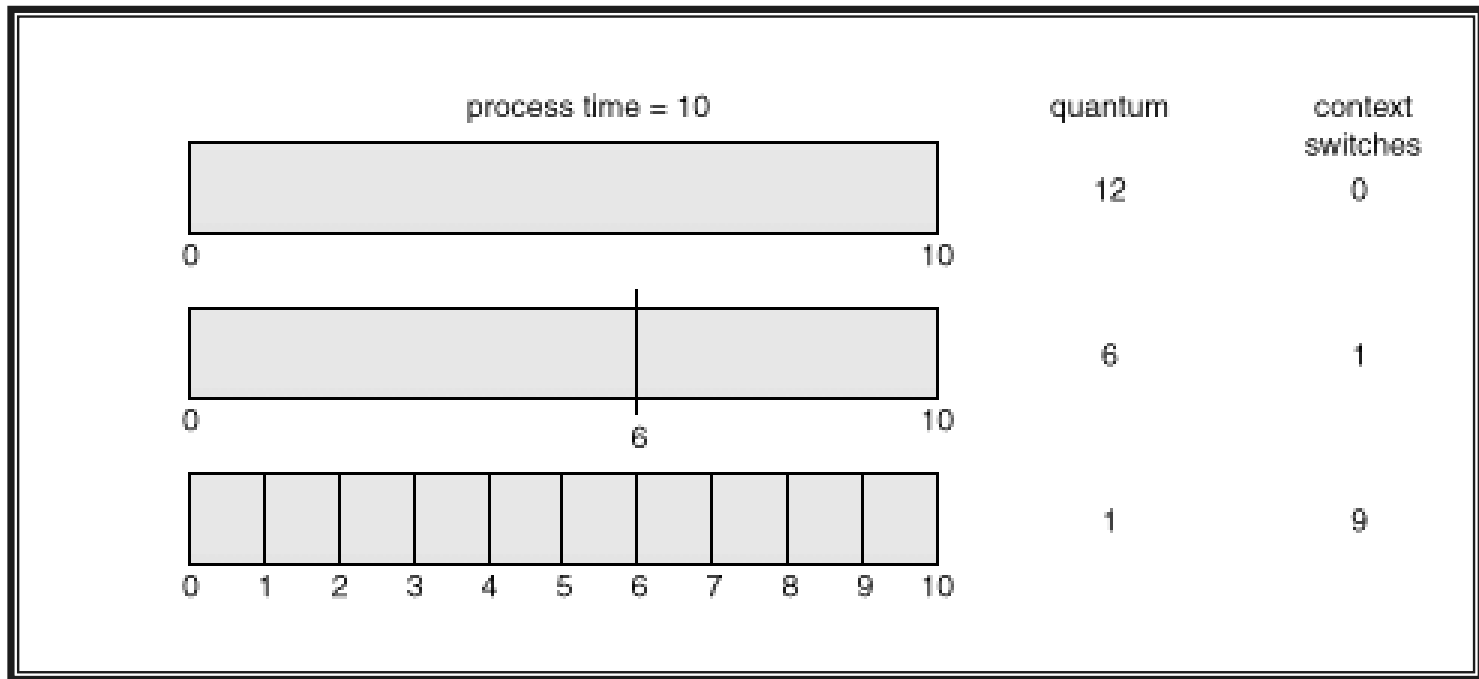
- Una priorità (numero intero) è assegnata ad ogni processo.
- La CPU è assegnata al processo con più alta priorità (es: il più piccolo intero \equiv la più alta priorità).
 - Preemptive
 - nonpreemptive
- SJF è uno scheduling con priorità stabilita dal valore del tempo del prossimo CPU burst.
- Problema \equiv **Starvation** – i processi a più bassa priorità potrebbero non essere mai eseguiti.
- Soluzione \equiv **Aging** – al trascorrere del tempo di attesa si incrementa la priorità di un processo che attende.

Scheduling Round Robin (RR)

- Ogni processo è assegnato alla CPU per un intervallo temporale fissato (*quanto di tempo*), ad es: 10-100 millisecondi.
- Quando il tempo è trascorso il processo viene tolto dalla CPU e inserito nella ready queue.
- Se ci sono **N** processi nella ready queue e il quanto di tempo è **Q**, ogni processo ottiene **1/N** del tempo della CPU a blocchi di lunghezza **Q**. Nessun processo attende più di **(N-1)Q** unità di tempo.
- Prestazioni
 - *Q* grande \Rightarrow FIFO
 - *Q* piccolo \Rightarrow *Q* deve essere molto più grande del tempo di *context switch*, altrimenti il costo è troppo alto.

Quanto di tempo e Context Switch

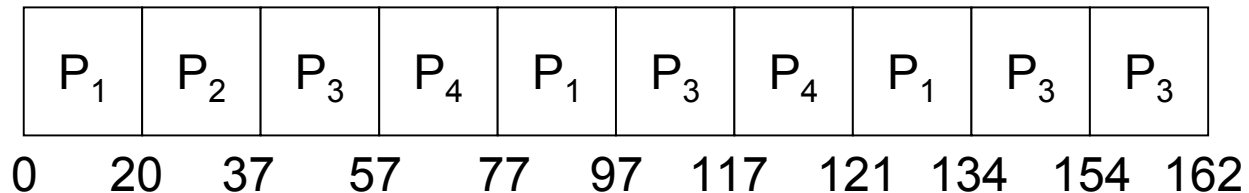
- Il quanto di tempo deve essere molto più grande del tempo di context switch, altrimenti il costo è troppo alto.



Esempio di RR con $Q = 20$

<u>Processi</u>	<u>tempo di burst</u>
P_1	53
P_2	17
P_3	68
P_4	24

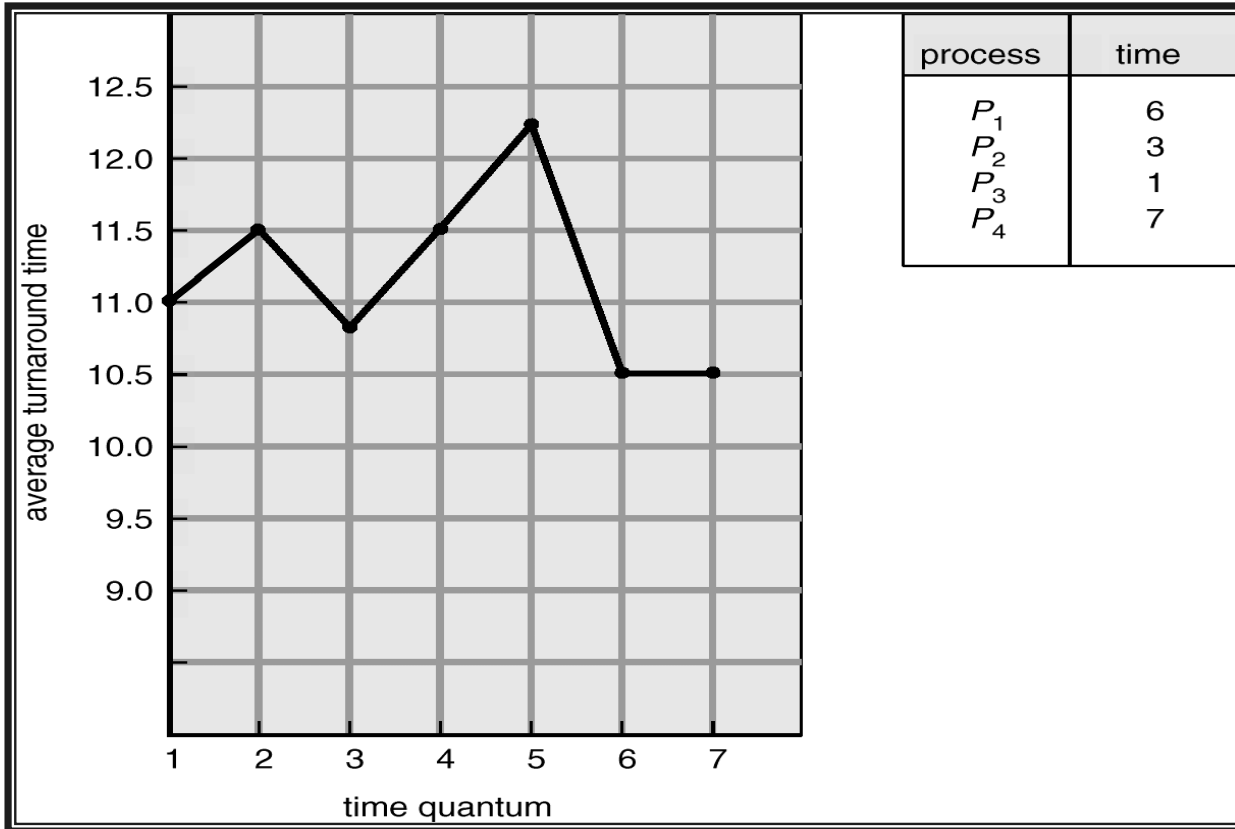
■ Gantt:



- Tempo di *turnaround* maggiore di SJF, ma migliore *tempo di risposta*.

Il tempo di Turnaround dipende da Q

- Anche il tempo di Turnaround dipende dal quanto di tempo

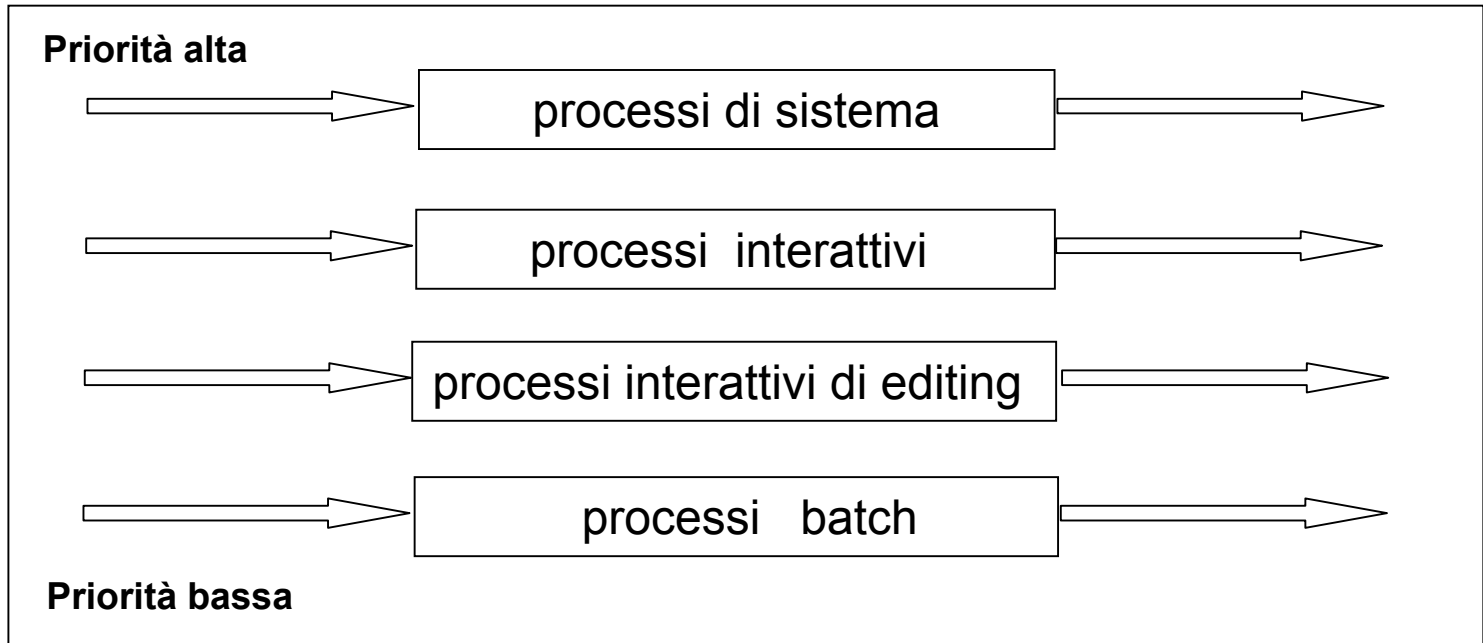


- Circa l'80% dei CPU burst devono essere più brevi di Q.

Scheduling a code multiple

- La ready queue è partizionata in più code. Ad esempio:
 - **foreground** (processi interattivi)
 - **background** (processi batch)
- Ogni coda è gestita da un proprio algoritmo di scheduling. Ad esempio:
 - **foreground** – RR
 - **background** – FCFS
- E' necessario uno scheduling tra le code.
 - **Scheduling a priorità fissa** : Possibilità di starvation.
 - **Quanto di tempo** : ogni coda ha un certo ammontare di tempo di CPU che usa per i suoi processi. Ad esempio:
 - ❖ 80% ai processi interattivi con RR
 - ❖ 20% ai processi batch con FCFS.

Scheduling a code multiple : Esempio



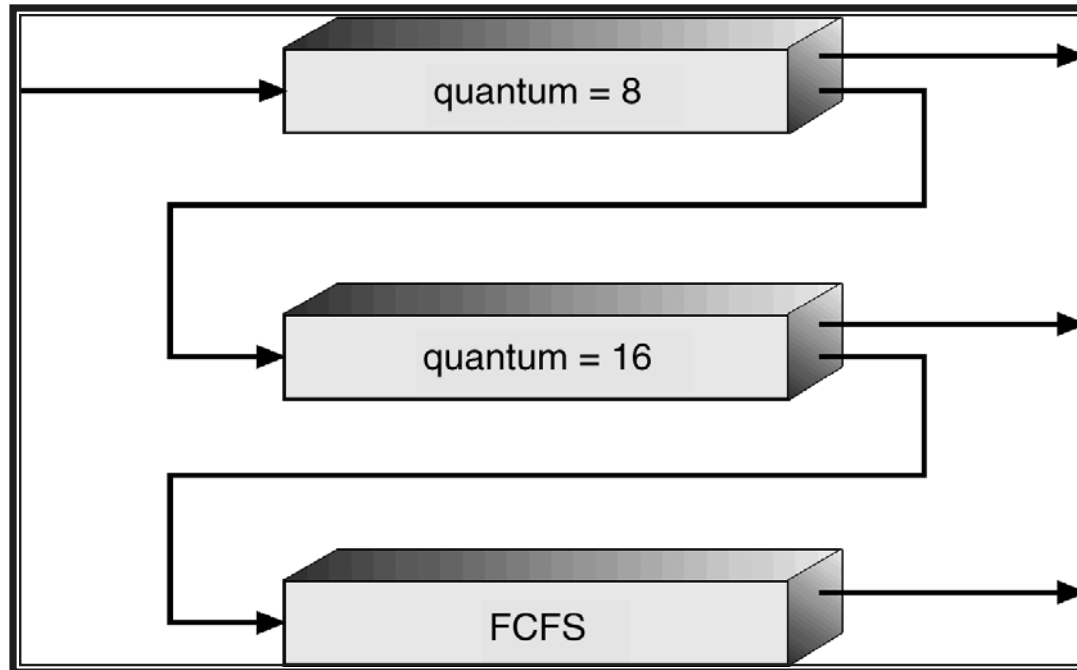
Scheduling a code multiple con feedback

- Un processo si può spostare tra le code. Questo evita situazioni di starvation o di eccessivo utilizzo della CPU.
- Lo scheduler che usa code multiple con feedback usa i seguenti parametri:
 - numero di code
 - algoritmi di scheduling per ogni coda
 - un metodo per "promuovere" un processo (--> maggiore priorità)
 - un metodo per "degradare" un processo (--> minore priorità)
 - un metodo per decidere in quale coda inserire un processo quando questo chiede un servizio.

Scheduling a code multiple con feedback

■ Esempio con tre code:

- Q_0 – quanto di tempo di 8 millisecondi
- Q_1 – quanto di tempo di 16 millisecondi
- Q_2 – FCFS



Scheduling a code multiple con feedback

- Si servono prima i processi della coda 0, quindi quelli della coda 1 e solo dopo quelli della coda 2.
- Esempio di scheduling
 - un nuovo processo arriva nella coda Q_0 e quando è servito dalla CPU gli viene assegnato un quanto di tempo di 8 millisecondi. Se non completa in questo tempo va in Q_1 .
 - Nella coda Q_1 il processo riceve 16 millisecondi. Se non completa l'esecuzione, dopo questo tempo viene tolto dalla CPU e assegnato alla coda Q_2 .

Scheduling per Multiprocessori

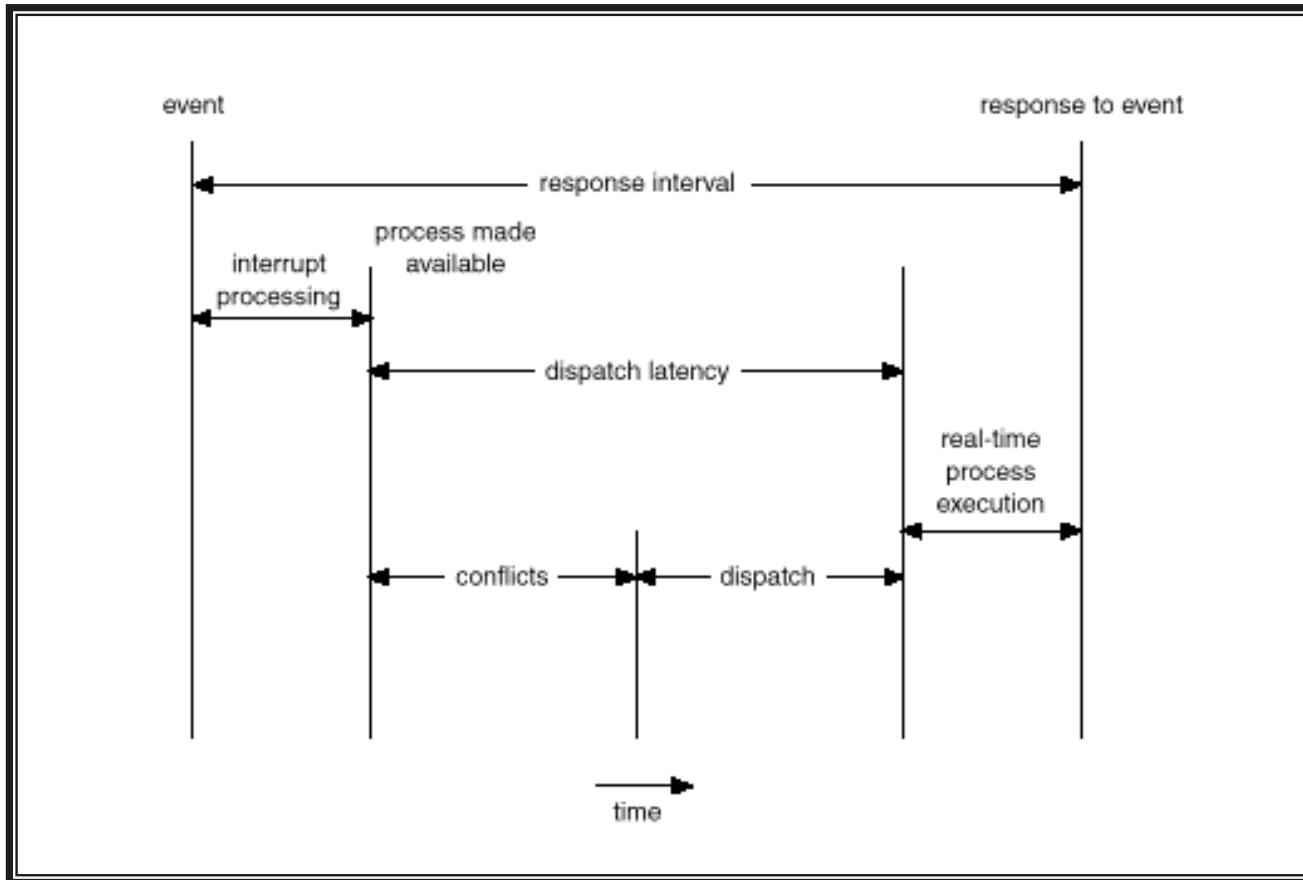
- Lo scheduling nei sistemi multiprocessore è più complesso.
- Si possono avere **processori omogenei** (tutti uguali) o **disomogenei** (diversi processori).
- Problema del **bilanciamento del carico** (*load balancing*).
- *Multiprocessing Asimmetrico* – solo un processore (master) accede alle strutture del sistema, gli altri (slave) eseguono programmi utente.

Scheduling Real-Time

- ***Sistemi hard real-time*** – i processi devono completare l'esecuzione entro un tempo fissato.
 - *Prenotazione delle risorse*: il processo viene accettato con una indicazione di tempo di completamento
 - Se il sistema non può soddisfare la richiesta rifiuta l'esecuzione del processo.

- ***Sistemi soft real-time*** – i processi "critici" ricevono una maggiore priorità rispetto ai processi "normali".
 - Priorità non decrescente
 - Prelazione delle system call.

Latenza di Dispatch



Valutazione di algoritmi di scheduling

■ Come scegliere un algoritmo di scheduling adatto/ottimale ?

⇒ Fissare i criteri di ottimizzazione.

✿ Usare metodi di valutazione:

➤ **Modellazione Deterministica** (valutazione analitica)

❖ Fissati i diversi carichi di lavoro definisce le prestazioni dei diversi algoritmi per ognuno dei carichi analizzati.

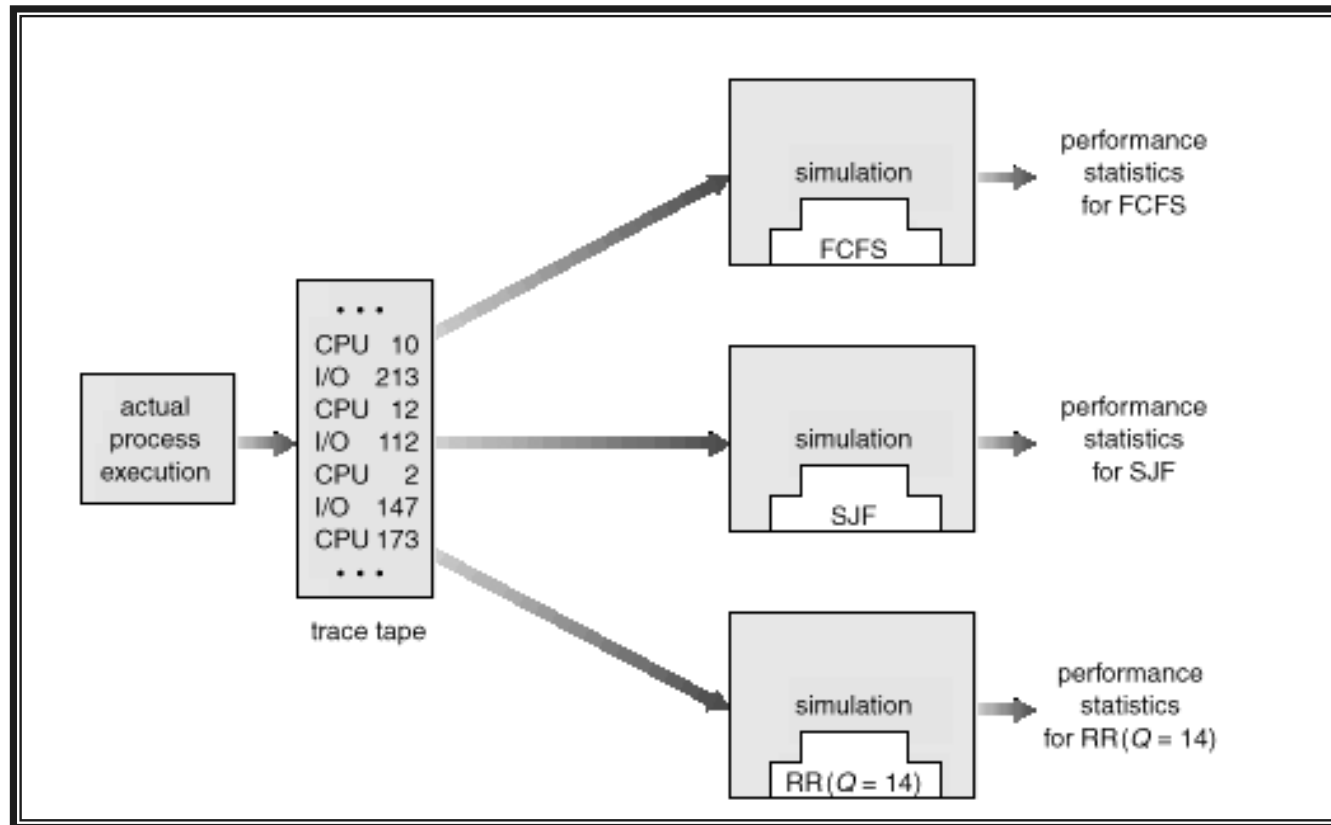
➤ **Modelli di code**

❖ Il sistema viene modellato come un insieme di server con le code associate; date le distribuzioni degli arrivi delle richieste si calcola la lunghezza media delle code, il tempo medio di attesa, etc.

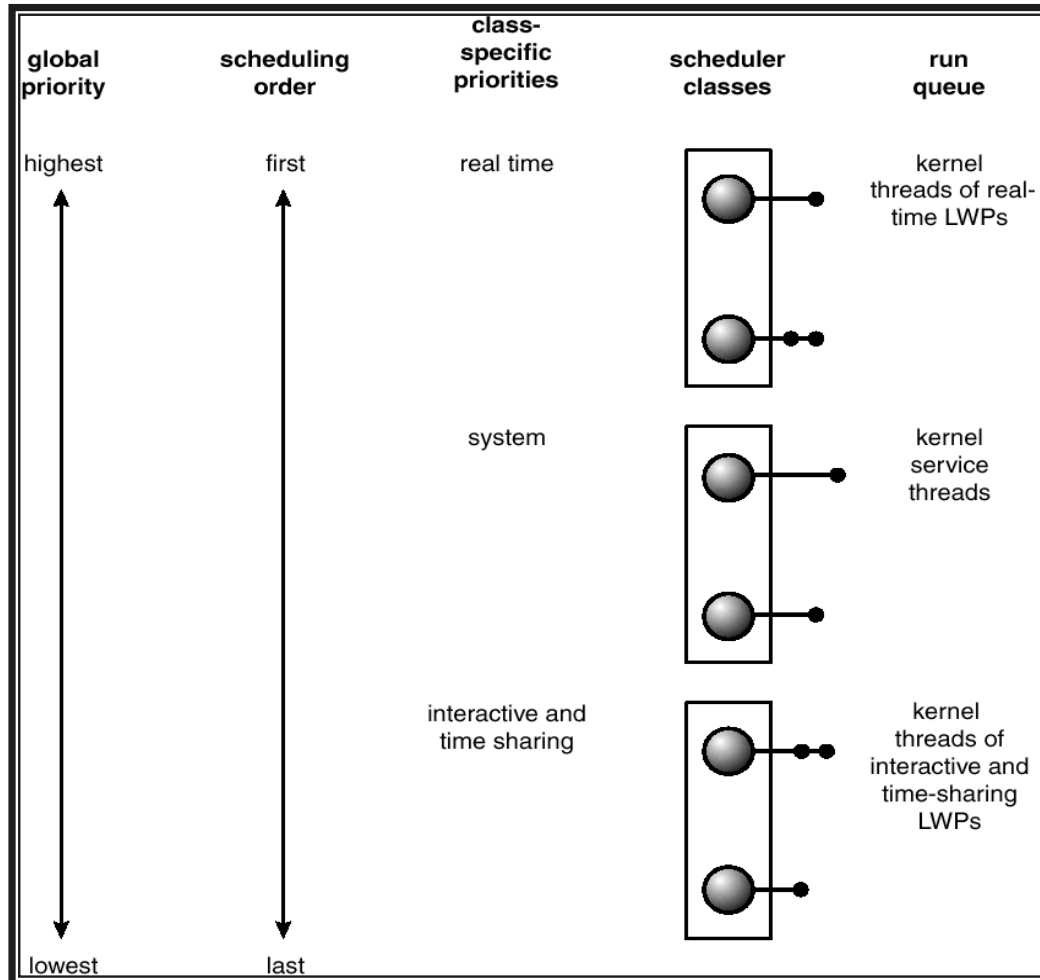
➤ **Realizzazione**

➤ **Simulazione**

Valutazione tramite simulazione



Scheduling di Solaris 2



Priorità di Windows 2000

	real-time	high	above normal	normal	below normal	idle priority
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

Domande

- Valutare i diversi algoritmi di scheduling sugli esempi usati per gli scheduling FCFS, SJF e RR.
- Come dovrebbe essere un algoritmo di scheduling per processi di tipo I/O bound ?
- Valutare l'effetto di quanti di tempo differenti sull'algoritmo RR.
- Spiegare l'effetto della priorità dinamica sugli algoritmi di scheduling con priorità.