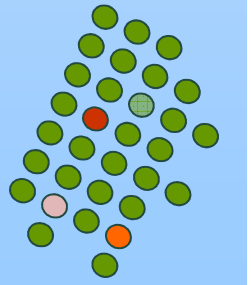
Scheduling della CPU

Capitolo 5 - Silberschatz



Concetti di base

- La multiprogrammazione cerca di ottenere la massima utilizzazione della CPU.
- L’esecuzione di un processo consiste in **cicli** d’esecuzione della CPU ed in attese di I/O – CPU burst e I/O burst

Sequenza Alternata di CPU Burst e I/O Burst

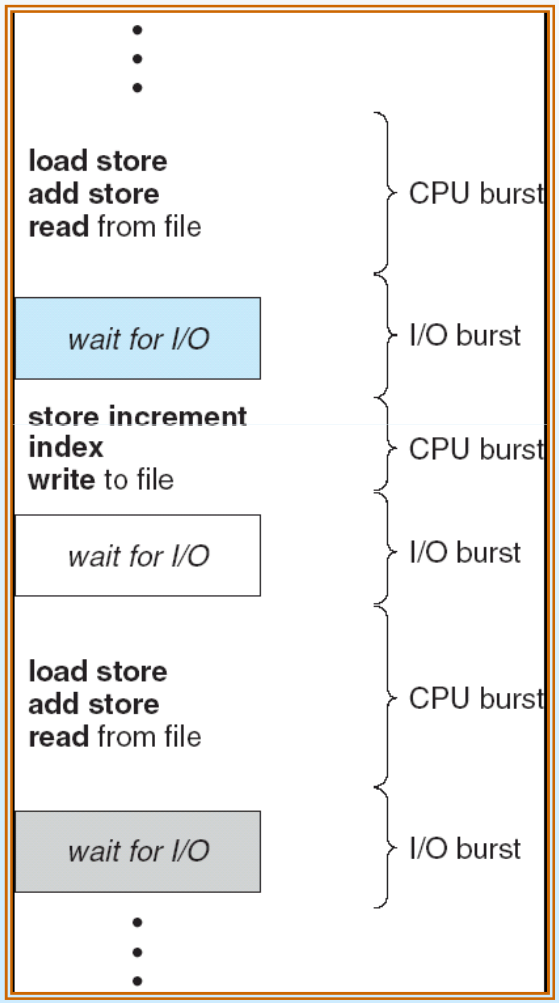
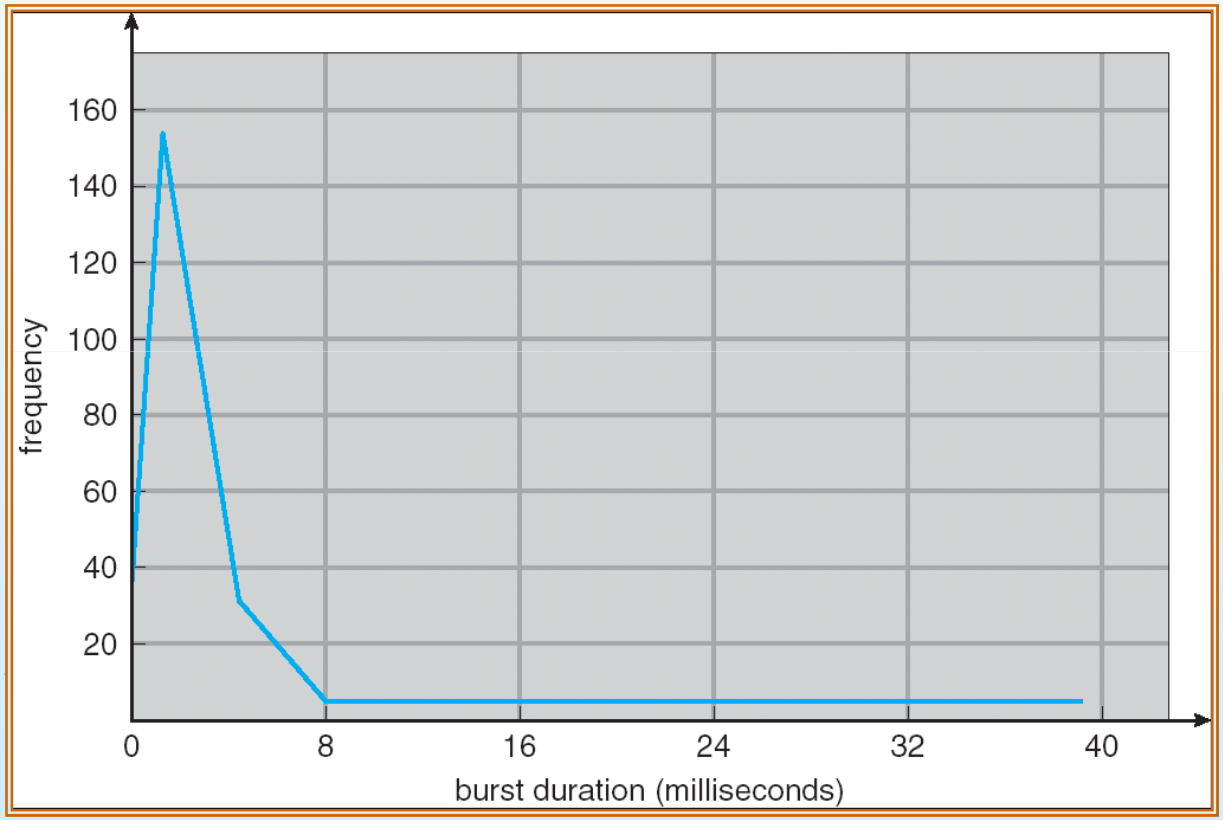


Diagramma della durata dei CPU-burst



Lo scheduler della CPU

- Il SO sceglie tra i processi in memoria che sono pronti per l'esecuzione ed assegna la CPU ad uno di essi.

- Lo scheduling della CPU può avvenire quando un processo:
  1. Passa dallo stato di esecuzione allo stato di attesa.
     - per una operazione I/O o per una wait
  2. Passa dallo stato di esecuzione allo stato di pronto.
     - per l'arrivo di un segnale di interruzione
  3. Passa dallo stato di attesa allo stato di pronto.
     - perché è terminata una operazione di I/O
  4. Termina.

- Quando lo scheduling della CPU interviene solo nei punti 1 e 4 allora la schedulazione è detta **nonpreemptive** (senza diritto di prelazione).

- Quando lo scheduling della CPU interviene anche nei punti 2 e 3, la schedulazione è detta **preemptive**.

Dispatcher

- Il **dispatcher** è il modulo del SO che dà il controllo della CPU ad un processo selezionato dallo scheduler. Questa funzione comprende:
  - cambio di contesto;
  - passaggio alla modalità utente;
  - salto alla corretta locazione nel programma utente per ricominciarne l'esecuzione.
- **Latenza del dispatcher** – tempo necessario al dispatcher per fermare un processo e cominciarne un altro.

Parametri

- **Utilizzo della CPU** – mantenere la CPU il più impegnata possibile.
- **Frequenza di completamento** (*throughput*) – numero di processi completati per unità di tempo.
- **Tempo di completamento** (*turnaround time*) – intervallo che va dal momento dell'immissione del processo nel sistema al momento del completamento.
- **Tempo di attesa** (*waiting time*)– somma dei tempi spesi in attesa nella coda dei processi pronti.
- **Tempo di risposta** – tempo che intercorre dalla formulazione della prima richiesta fino alla produzione della prima risposta, **non** l'output (per gli ambienti di time-sharing).

Ottimizzazione

- Massimizzare l'utilizzo della CPU.
- Massimizzare il throughput.
- Minimizzare il turnaround time.
- Minimizzare il tempo di attesa.
- Minimizzare il tempo di risposta.

Solitamente si ottimizzano i valori medi. A volte è opportuno ottimizzare i valori minimi o massimi (come ridurre il massimo tempo di risposta).

Tempo di risposta

Per i sistemi interattivi andrebbe minimizzata la varianza
(indica quanto i valori sono “vicini” al valore medio)

P1 = 1

P2 = 9

t.m. (1+9)/2= 5

var. (4+4)/2= 4

P1 = 4

P2 = 6

t.m. (4+6)/2= 5

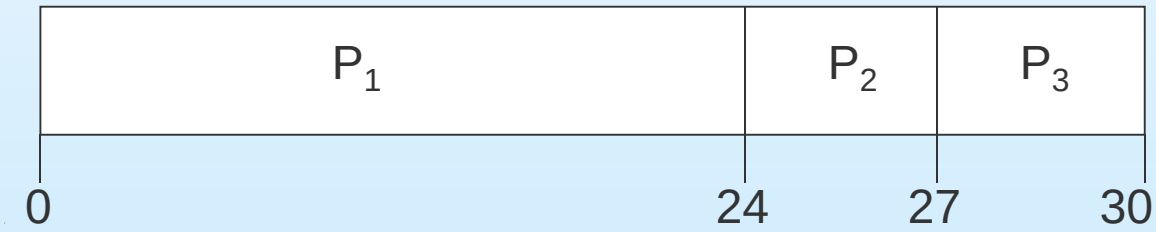
var. (1+1)/2 =1

Algoritmi di scheduling

First-Come, First-Served (FCFS)

Process	CPU Burst
P_1	24
P_2	3
P_3	3

- Se i processi arrivano nell'ordine: P_1 , P_2 , P_3
  si ottiene il risultato mostrato nel seguente diagramma di Gantt:



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

FCFS (Cont.)

Se I processi arrivano nell'ordine

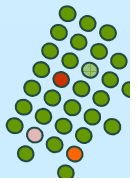
$$P_2 , P_3 , P_1$$

- Il diagramma di Gantt è:

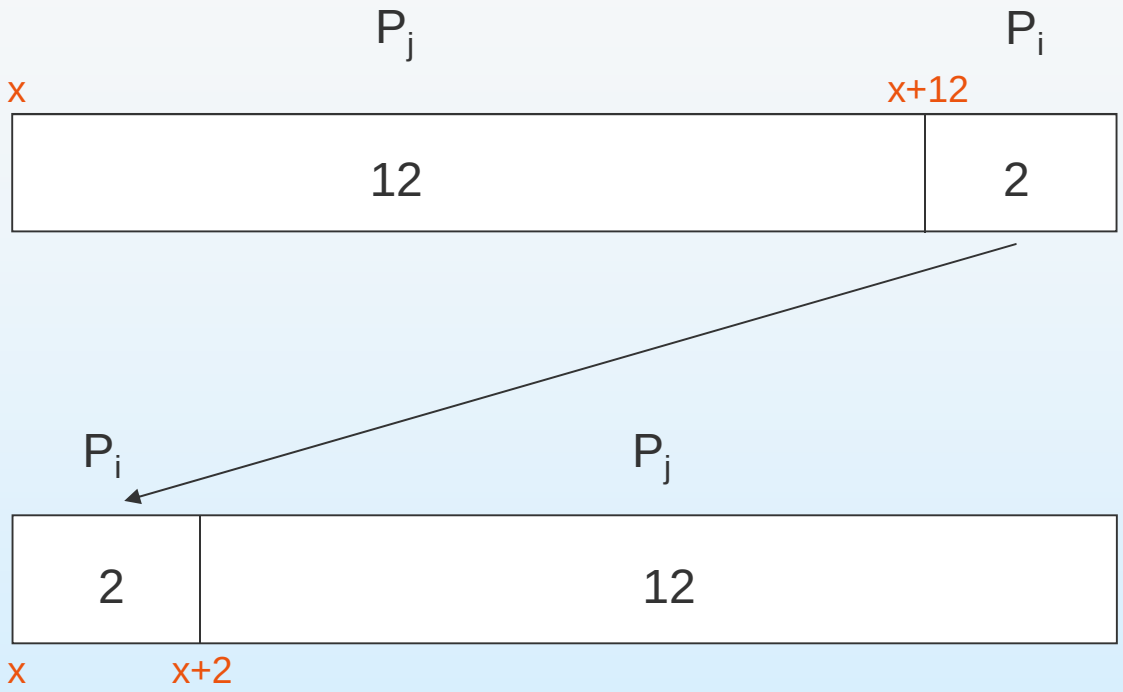
- Tempo di attesa per P1 = 6; P2 = 0; P3 = 3
- Tempo di attesa medio: (6 + 0 + 3)/3 = 3
- Molto meglio del caso precedente.
- C'è un effetto di ritardo a catena (*convoy effect*) mentre tutti i processi attendono che quello grosso rilasci la CPU.

Shortest-Job-First (SJF)

- Schedula il processo con il prossimo CPU burst più breve.
- L'algoritmo SJF può essere:
  - **nonpreemptive** – quando un processo ha ottenuto la CPU, non può essere prelazionato fino al completamento del suo cpu-burst.
  - **preemptive** – quando un nuovo processo è pronto, ed il suo CPU-burst è minore del tempo di cui necessita ancora il processo in esecuzione, c'è prelazione. Questa schedulazione è anche detta *shortest-remaining-timefirst.*
- SJF è ottimale – fornisce il minor tempo di attesa medio per un dato gruppo di processi.



SJF ottimale

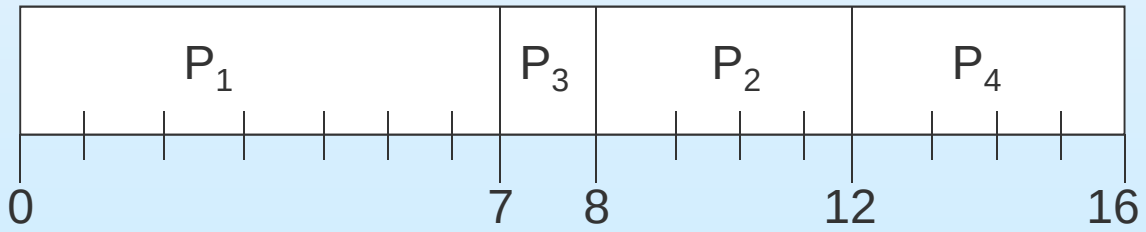


Spostando un processo breve prima di un processo lungo, il tempo di attesa del processo breve diminuisce più di quanto aumenti il tempo d'attesa per il processo lungo. Di conseguenza il tempo d'attesa medio diminuisce.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



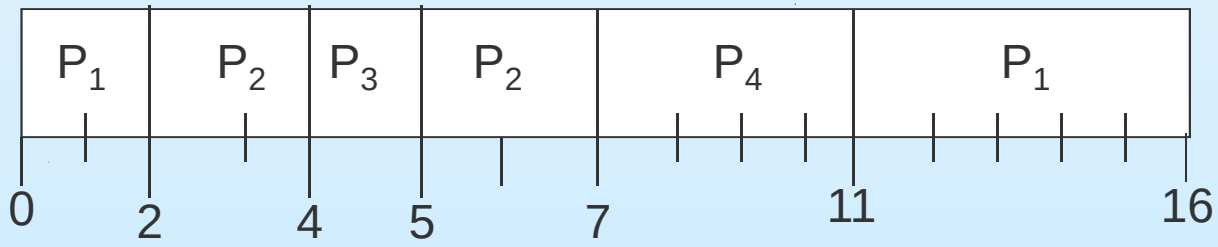
- Average waiting time = (0 + 3 + 6 + 7)/4 = 4

7-4 8-2 12-5

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



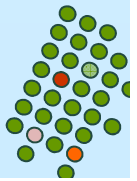
- Average waiting time = (9 + 1 + 0 +2)/4 = 3

11-2 …

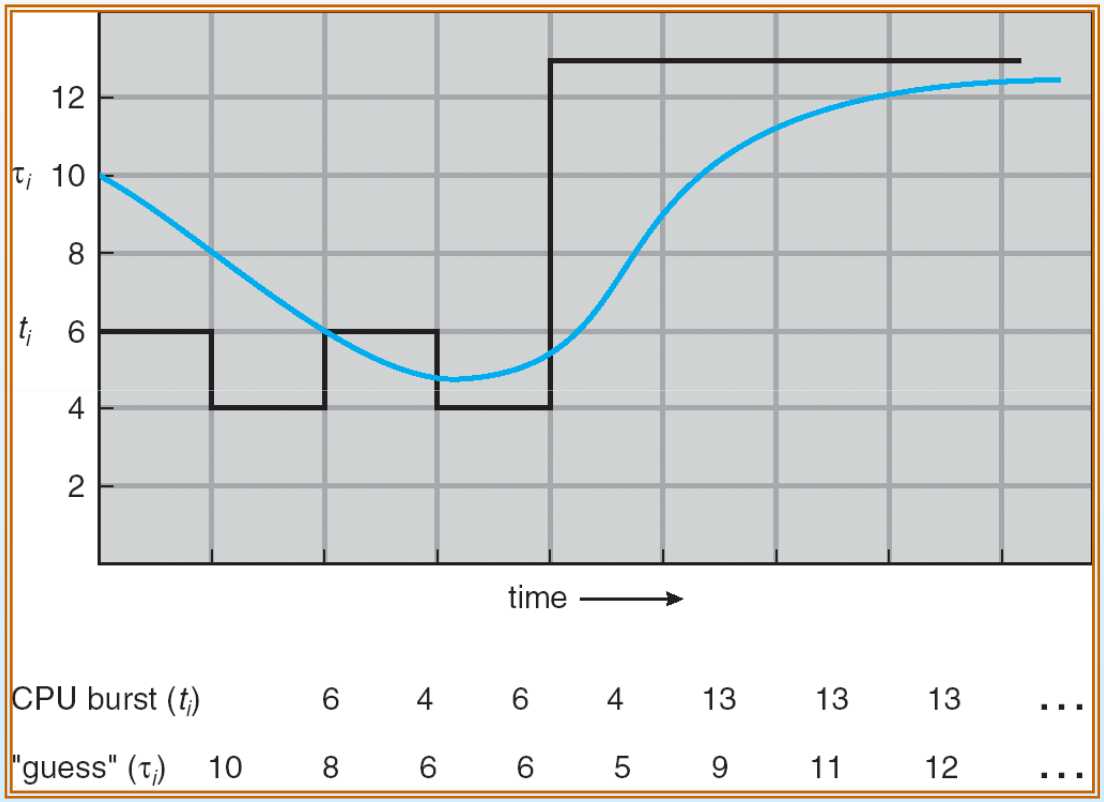
Stima del prossimo CPU burst

- É possibile fare solo una stima della lunghezza.
- Può essere fatta utilizzando la lunghezza dei precedenti CPU burst, usando una media esponenziale.
  - t_n = valore reale dell'ennesimo CPU burst
  - τ_{n+1} = valore previsto per il prossimo CPU burst
  - α = parametro con valore $0 \leq \alpha \leq 1$
  - τ_1 = previsione 1° CPU burst (valore di default)

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



Previsione della durata del prossimo CPU Burst



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12 ...

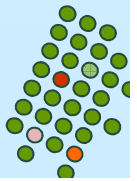
$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$ con $\alpha=1/2$ e $\tau_1 = 10$

Analisi della formula $\tau_{n+1} = \alpha\, t_n + (1- \alpha)\, \tau_n$

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La storia recente non ha alcun effetto
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Conta solo il CPU burst più recente
- Se sviluppiamo la formula otteniamo:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots + (1 - \alpha)^j \alpha\, t_{n-j} + \ldots + (1 - \alpha)^{n+1} \tau_0$$

- Poichè sia α che $(1 - \alpha)$ sono minori o uguali a 1, ciascun termine successivo ha un peso inferiore rispetto a quello precedente



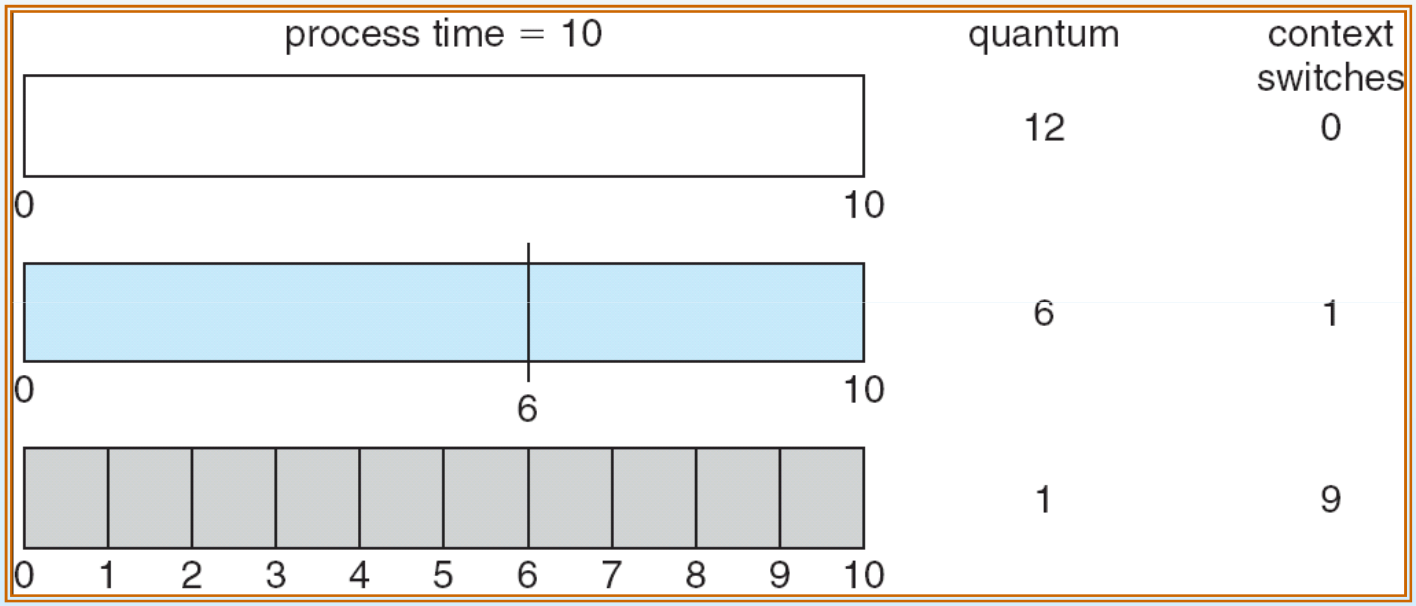
Scheduling a priorità

- Si associa una priorità numerica a ciascun processo.
- La CPU viene allocata al processo con priorità più alta.
  - *preemptive*
  - *nonpreemptive*
- SJF è un algoritmo a priorità dove la priorità è l'inverso della lunghezza del prossimo CPU burst (previsto).
- Problema ≡ Blocco indefinito (starvation) – processi a bassa priorità potrebbero non essere mai eseguiti.
- Soluzione ≡ Invecchiamento (aging) – accrescere gradualmente le priorità dei processi nel sistema.

Round Robin (RR)

- Ogni processo riceve la CPU per una piccola unità di tempo (time quantum), generalmente 10-100 millisecondi. Se entro questo arco di tempo il processo non lascia la CPU, viene interrotto e rimesso nella coda dei processi pronti.
- Se ci sono *n* processi nella coda dei processi pronti e il quanto di tempo è *q*, ciascun processo non deve attendere più di *(n – 1) x q* unità di tempo.
- Prestazioni:
  - *q* grande ⇒ FIFO
  - *q* piccolo ⇒ *q* deve essere grande rispetto al tempo di un context switch, altrimenti l'overhead diventa troppo elevato.

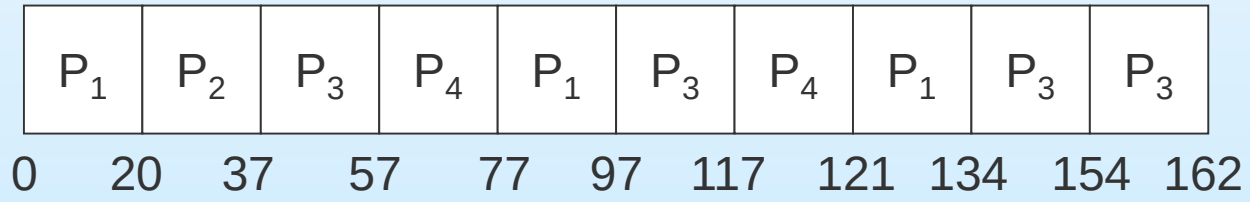
Quanto di tempo e Context Switch



Esempio di RR con quanto di tempo pari a 20 millisecondi

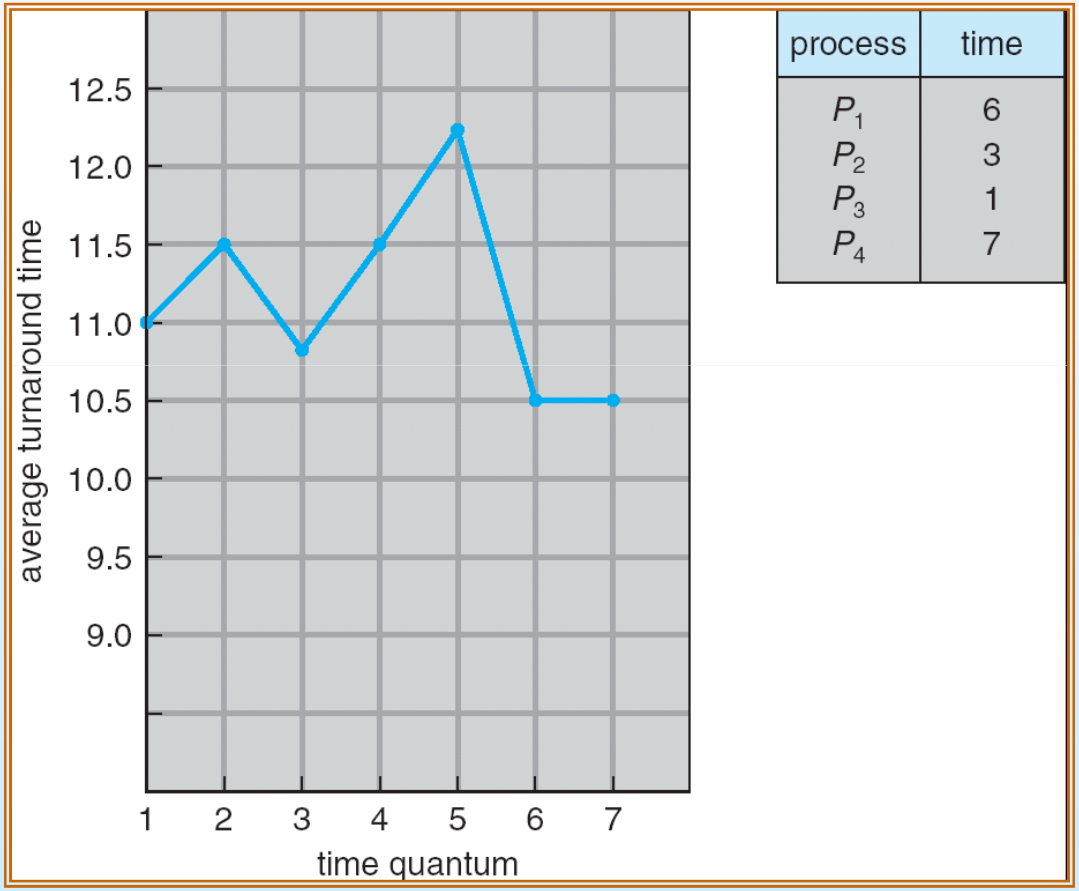
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Il diagramma di Gantt è:



- Tipicamente, il tempo medio di turnaround è più alto di SJF, ma il tempo di risposta è più breve.

Variazione del tempo medio di turnaround in funzione del quanto di tempo

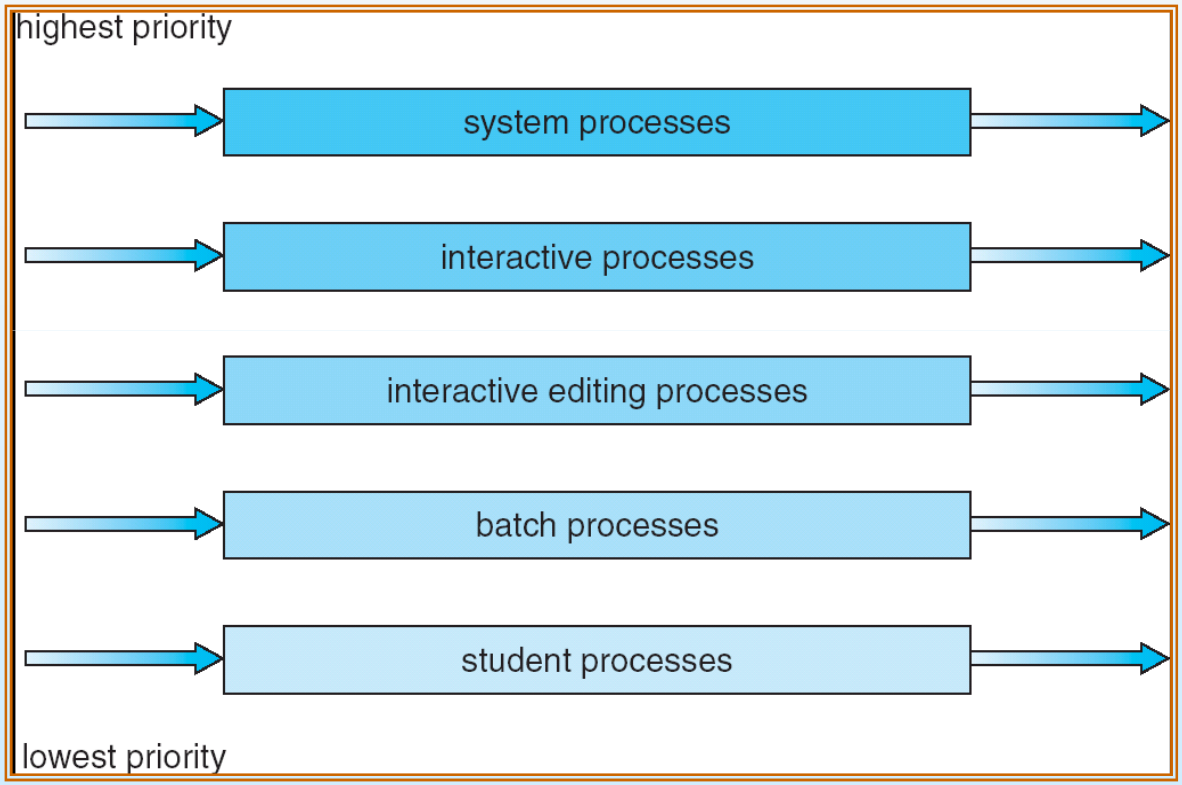


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Scheduling a code multiple

- La coda dei processi pronti è partizionata in code separate:
  - foreground (interattivi),
  - background (batch – sullo sfondo).
- Ciascuna coda ha il proprio algoritmo di scheduling:
  - foreground – RR
  - background – FCFS
- Ci deve essere una schedulazione tra le code
  - A priorità fissa; (e.g., tutti i processi in foreground, poi quelli in background). Possibilità di starvation.
  - Time slice – ciascuna coda ha una certa quantità di tempo di CPU, che può schedulare fra i processi in essa contenuti;e.g., 80% del tempo di CPU per la coda foreground (RR), 20% background in FCFS.

Scheduling a code multiple



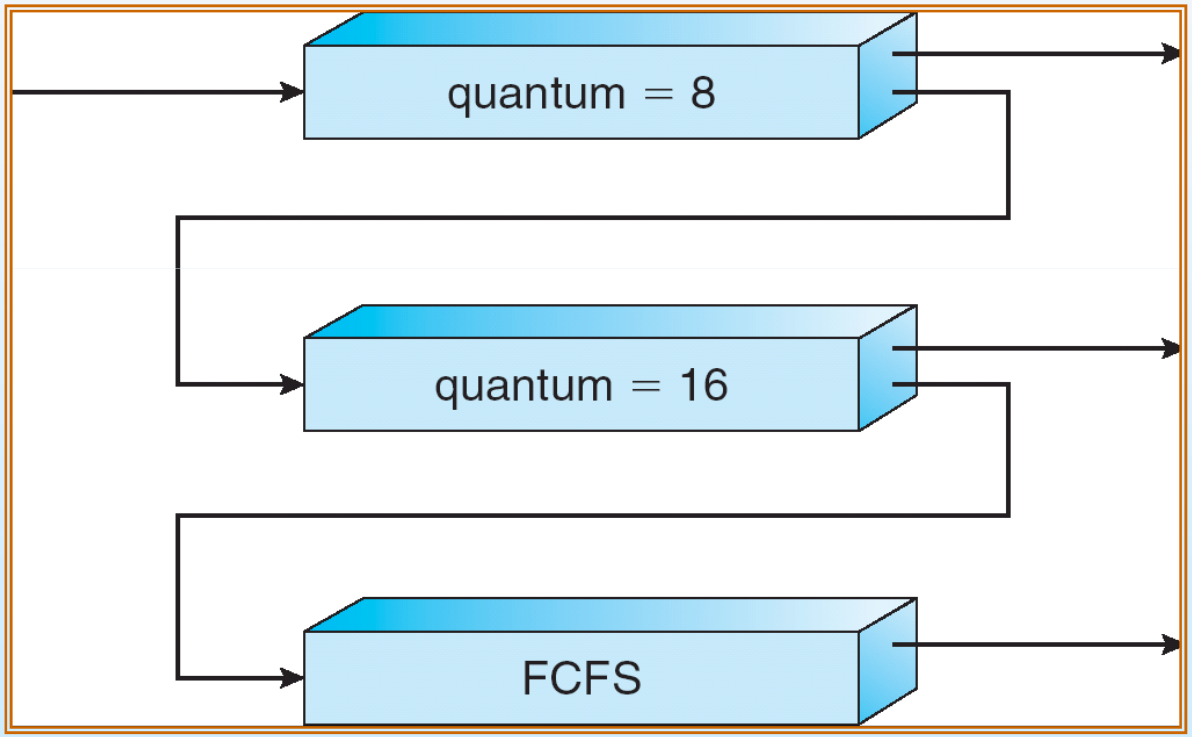
Code multiple con feedback

- Un processo può muoversi tra le varie code; l'aging potrebbe essere implementato in questo modo.
- Uno schedulatore a code multiple con feedback è definito dai seguenti parametri:
  - numero di code;
  - algoritmo di schedulazione per ciascuna coda;
  - metodo utilizzato per "far salire" un processo verso una coda a priorità più alta;
  - metodo utilizzato per spostare un processo in una coda a più bassa priorità;
  - metodo utilizzato per determinare in quale coda entrerà un processo quando avrà bisogno di un servizio.

Esempio di code multiple con feedback

- Tre code:
  - Q_0 – RR con quanto di tempo: 8 millisecondi
  - Q_1 – RR con quanto di tempo: 16 millisecondi
  - Q_2 – FCFS
- Schedulazione
  - Un nuovo processo entra nella coda Q_0. Quando schedulato ottiene la CPU per 8 millisecondi. Se non termina in 8 millisecondi, viene spostato nella coda Q_1.
  - In Q_1 il processo, quando schedulato, riceve la CPU per 16 millisecondi. Se non completa entro i 16 millisecondi, viene spostato nella coda Q_2.

Code multiple con Feedback



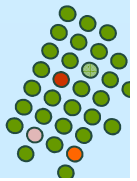
Time sharing

In un sistema time sharing puro l' esecuzione viene ripartita a turno tra i processi della stessa priorità.

In un sistema senza priorità un processo non è mai forzato nell'uscita.

In un sistema con priorità i processi più prioritari hanno precedenza.

Quindi se un processo a priorità p è in esecuzione e durante il suo quanto di tempo un processo di priortià q > p entra nello stato di pronto (ad es. perché ha terminato un'attesa su evento esterno) il processo in esecuzione viene interrotto.



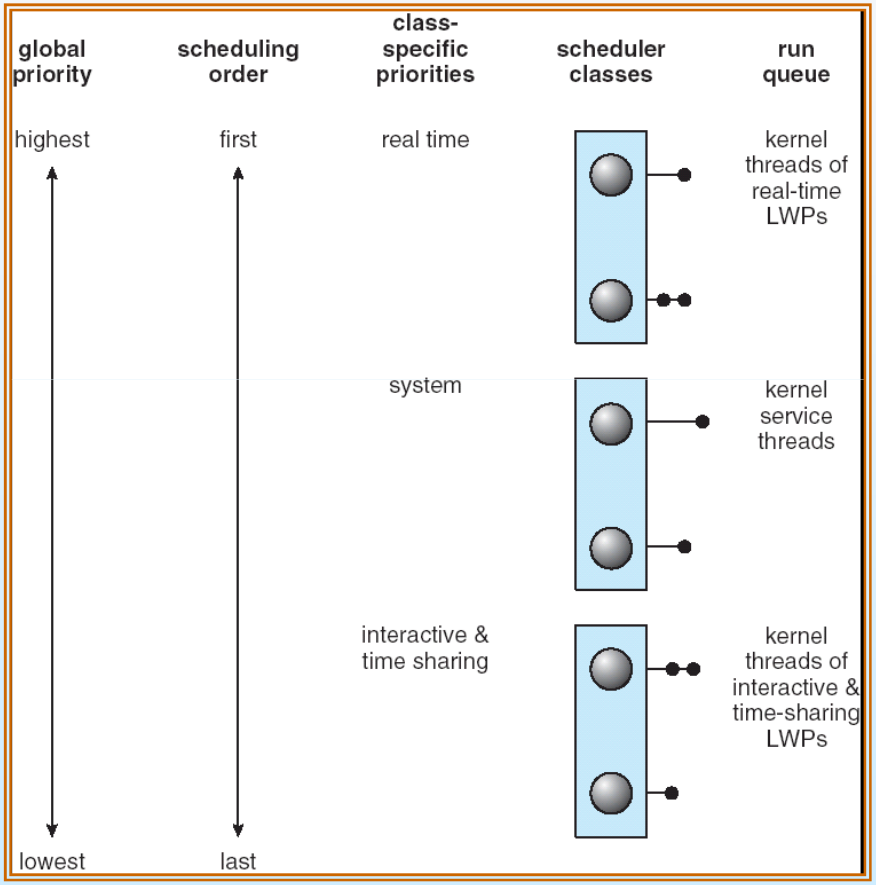
Algoritmi di scheduling usati in sistemi operativi reali

Scheduling in Solaris

Usa un algoritmo preempitive con priorità

Processi real-time sono processi che necessitano una risposta in un tempo determinato e quindi devono essere serviti prima di altri

I processi che arrivano vengono classificati a seconda che siano
real-time (a priorità più alta),
di sistema (riservata ai processi del kernel-priorità immediatamente più bassa)
time-sharing (classe di scheduling predefinita – essa è strutturata in diverse code a feedback ed i processi possono variare la loro priorità passando da una coda all'altra)
interattivi (hanno una priorità maggiore rispetto a quelli che richiedono più tempo di cpu)



Solaris Dispatch Table dei processi interattivi e time sharing

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Scheduling in Windows XP

Usa un algoritmo preempitive con priorità a 32 livelli.
Le priorità son divise in 2 classi: variabile (1-15) e real-time (16-31), 0 a processi che gestiscono la memoria.
C'è una coda per ogni priorità.

	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

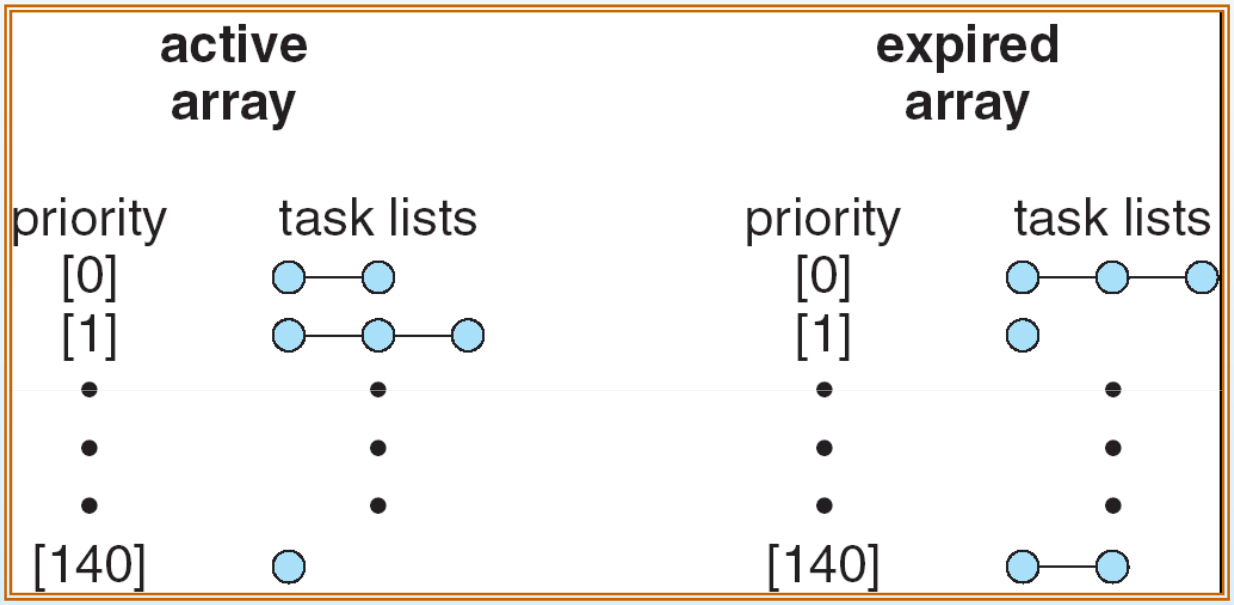
Scheduling in Linux

- Lo scheduler di Linux è preemptive e basato su priorità. Usa due range di valori di priorità – real time [0,99] e nice [100, 140].
- I valori di questi due range sono mappati in un valore di priorità globale. Numeri bassi rappresentano priorità alta.
- Il kernel mantiene una lista di tutti i task in una runqueue
- La runqueue contiene due array di priorità –
  - active array : tutti I task che hanno ancora tempo da sfruttare
  - expired array : tutti I task il cui tempo è scaduto

Relazione tra le priorità e la lunghezza del time-slice

numeric priority	relative priority		time quantum
0	highest	real-time tasks	200 ms
•			
•			
•			
99			
100		other tasks	
•			
•			
•			
140	lowest		10 ms

Liste dei task indicizzate in base alla priorità

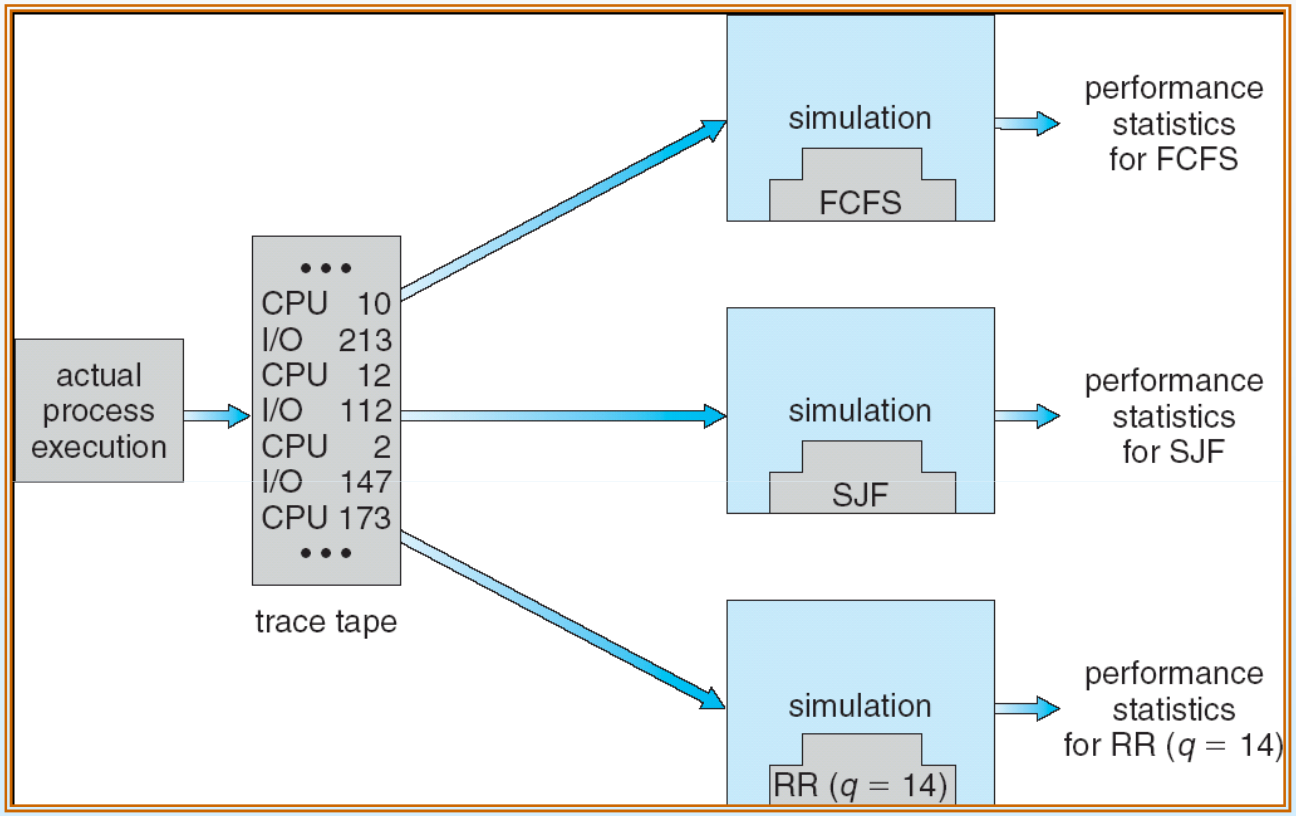


Una volta che l'array dei task attivi diventa vuoto, i due array si scambiano il ruolo

Valutazione degli algoritmi di scheduling

- Modello deterministico
  - Prende un particolare carico di lavoro e definisce le prestazioni di ciascun algoritmo per quel carico attraverso una formula o un numero.
- Reti di code
  - Determina le distribuzioni delle sequenze dei CPU burst e degli arrivi nel sistema e calcola la produt. media, il tempo medio di attesa, etc …
- Simulazione
  - Programmazione di un modello di sistema. Generatori casuali o trace-tape di esecuzioni reali.

Valutazione degli algoritmi di scheduling



- Implementazione
  - Problemi per gli utenti del sistema.