CAPITOLO 9

SCHEDULING MONOPROCESSORE

In un sistema in multiprogrammazione, i processi sono mantenuti nella memoria principale, e ogni processo alternativamente usa il processore o aspetta l'esecuzione dell'I/O o il completamento di un altro evento; infatti, il processore o i processori sono impegnati eseguendo un processo mentre gli altri processi sono in attesa.

Il punto centrale per la multiprogrammazione è lo scheduling: infatti, quattro tipi di scheduling sono generalmente coinvolti (Tabella 9.1). Uno di questi, l'I/O scheduling, è trattato più ampiamente nel Capitolo 11, dove l'I/O è approfondito. I tre tipi rimanenti di scheduling, che sono tipi di scheduling del processore, sono discussi in questo e nel successivo capitolo.

Il capitolo comincia con un esame dei tre tipi di scheduling del processore, mostrando come essi siano in relazione. Si può vedere che lo scheduling a lungo termine e lo scheduling a medio

Tabella 9.1 *Tipi di scheduling*

Scheduling a lungo termine	Si decide di aggiungere un processo all'insieme dei processi che devono essere eseguiti
Scheduling a medio termine	Si decide di aggiungere un processo all'insieme dei processi che sono parzialmente o completamente in memoria
Scheduling a breve termine	Si decide quale processo disponibile sarà eseguito dal processore
I/O scheduling	Si decide quale processo in attesa di I/O sarà gestito da un dispositivo di I/O disponibile

termine sono guidati principalmente da considerazioni sulle prestazioni relative al grado di multiprogrammazione. Questi argomenti sono stati trattati nel Capitolo 3, e più dettagliatamente nei Capitoli 7 e 8, perciò il resto di questo capitolo si concentra sullo scheduling a breve termine, in particolare nel caso di sistemi a singolo processore. Poiché l'uso di più processori aggiunge complessità all'argomento trattato, è meglio focalizzare l'attenzione sul caso di sistema monoprocessore, così da capire le differenze tra gli algoritmi di scheduling.

La Sezione 9.2 presenta i vari algoritmi che possono essere usati per realizzare lo scheduling a breve termine.

9.1 Tipi di scheduling

Lo scopo dello scheduling del processore è assegnare i processi che devono essere eseguiti al processore o ai processori, nel tempo, in modo da realizzare gli obiettivi del sistema, come il tempo di risposta, il throughput e l'efficienza del processore. In molti sistemi, quest'attività di scheduling è suddivisa in tre funzioni: scheduling a breve, medio o lungo termine. I nomi suggeriscono la frequenza relativa con cui le funzioni sono eseguite.

La Figura 9.1 collega le funzioni di scheduling al diagramma delle transizioni di stato del processo. Quando si crea un nuovo processo, si esegue lo scheduling a lungo termine: si decide di aggiungere un nuovo processo all'insieme dei processi che sono attivi al momento. Lo scheduling a medio termine è una parte della funzione di trasferimento su disco dei processi: si decide di aggiungere un processo a quelli che sono almeno parzialmente nella memoria princi-

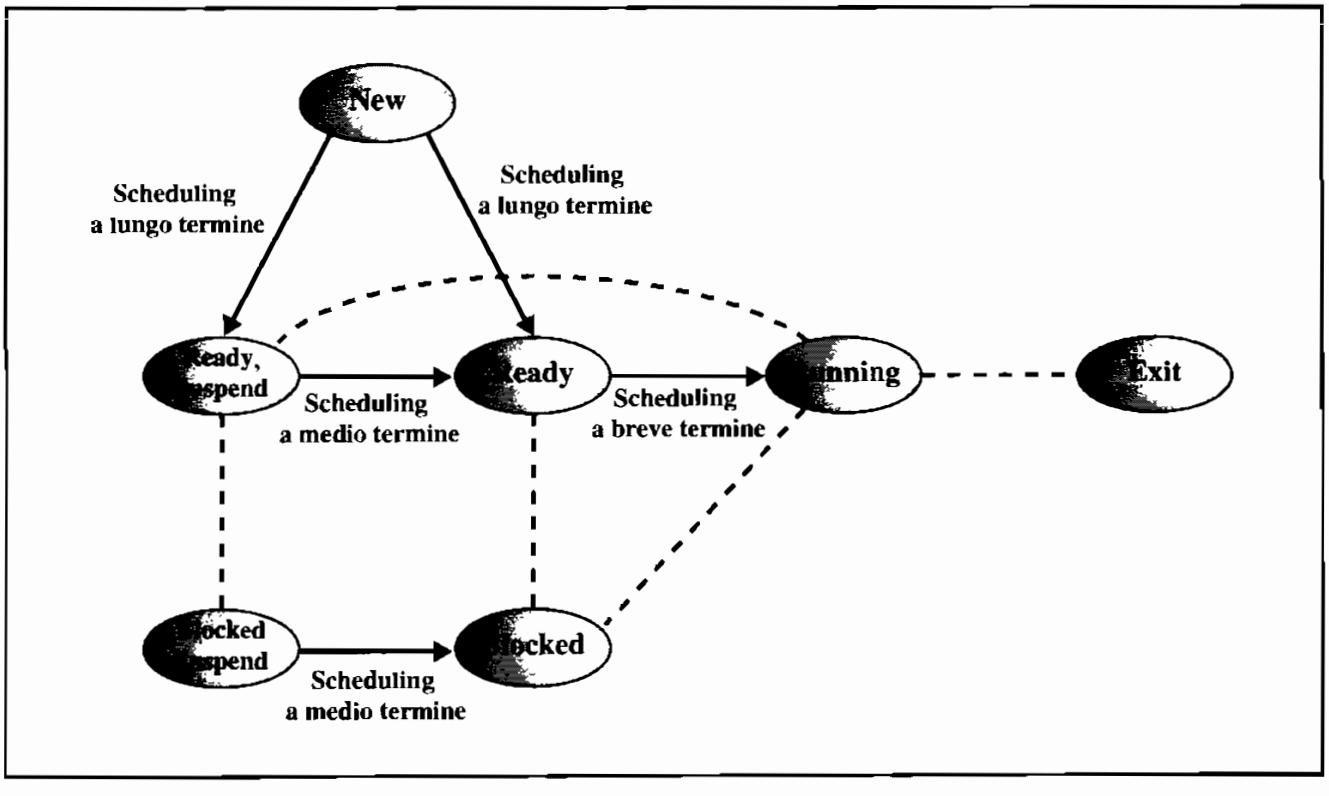

Figura 9.1 *Scheduling e transizioni di stato dei processi*

pale e perciò pronti ad essere eseguiti. Lo scheduling a breve termine decide quale processo Ready eseguire. La Figura 9.2 riorganizza il diagramma delle transizioni di stato per suggerire l'annidamento delle funzioni di scheduling.

Lo scheduling influenza le prestazioni del sistema perché determina i processi che aspetteranno e quelli che procederanno. Questo punto di vista è presentato nella Figura 9.3, che mostra le code coinvolte nelle transizioni di stato di un processo. Fondamentalmente, lo scheduling gestisce queste code in modo da minimizzare il tempo d'attesa in coda ed ottimizzare le prestazioni in un ambiente a code.

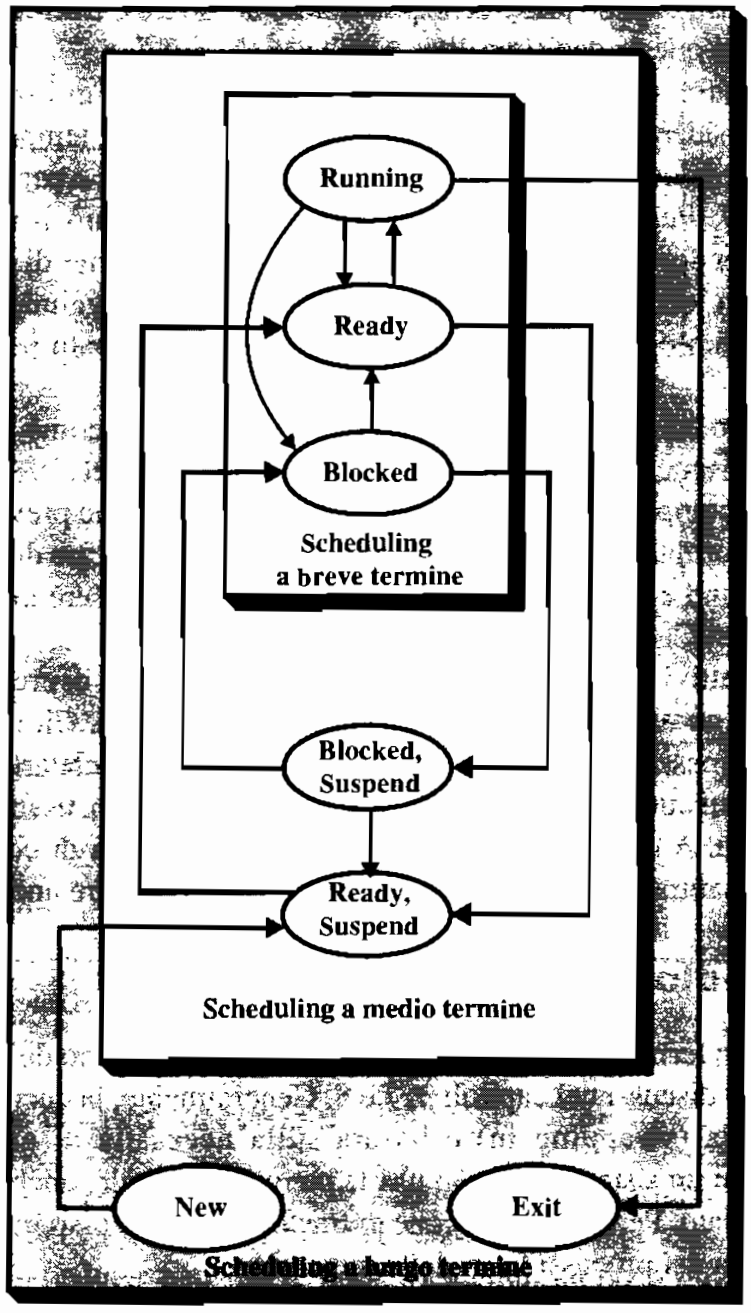

Figura 9.2 *Livelli di scheduling*

Scheduling a lungo termine

Lo scheduler a lungo termine determina quali programmi inserire nel sistema per l'esecuzione, perciò, controlla il grado di multiprogrammazione. Una volta inserito, un job, o un programma utente, diventa un processo e va ad aggiungersi alla coda dello scheduler a breve termine. In alcuni sistemi, un nuovo processo creato comincia in una condizione di trasferito su disco, nel qual caso è aggiunto alla coda dello scheduler a medio termine.

In un sistema batch, o per la parte batch di un sistema operativo generale, i nuovi job presentati sono diretti al disco e mantenuti in una coda batch. Lo scheduler a lungo termine preleva i processi dalla coda quando può; infatti, deve prendere due decisioni: se il sistema operativo può prendere uno o più processi, e quale job (anche più di uno) accettare e trasformarlo in processo. Consideriamo brevemente queste decisioni.

La decisione di creare un nuovo processo è generalmente guidata dal livello di multiprogrammazione desiderato. Più processi sono creati, minore è il tempo a disposizione di ogni processo per l'esecuzione (cioè, più processi sono in competizione per lo stesso tempo di utilizzo del processore). Perciò, lo scheduler a lungo termine può limitare il grado di multiprogrammazione per fornire un servizio soddisfacente all'insieme di processi che deve gestire. Ogni volta che un job termina, lo scheduler può decidere se aggiungere uno o più job, inoltre può essere chiamato ogni volta che il processore non è occupato per più di una certa percentuale di tempo.

Si può decidere che job ammettere con la semplice tecnica del first-come-first-served, oppure si può utilizzare uno strumento per la gestione delle prestazioni di un sistema. I criteri utilizzati nella scelta possono comprendere la priorità, il tempo di esecuzione previsto e i requisiti di I/O. Per esempio, se l'informazione è disponibile, lo scheduler può tentare di prendere un misto di processi processor-bound e I/O bound.[1] Inoltre la decisione può essere presa in dipendenza dalle risorse di I/O richieste, nel tentativo di bilanciare l'uso dell'I/O.

Per i programmi interattivi in un sistema time-sharing, una richiesta di creazione di processo è generata dall'azione di un utente che cerca di connettersi al sistema. Gli utenti time-sharing non sono semplicemente messi in coda in attesa che il sistema possa accettarli, il sistema operativo accetterà tutti gli utenti autorizzati fino a che il sistema non è saturo, usando qualche misura di saturazione predefinita. A quel punto, si risponde ad una richiesta di connessione con un messaggio che indica la saturazione del sistema e l'utente dovrà provare ancora.

Scheduling a medio termine

Lo scheduling a medio termine è parte della funzione di trasferimento dei processi su disco. I temi coinvolti sono stati discussi nei capitoli 3, 7 e 8. Generalmente, la decisione di introdurre un processo in memoria (swapping in) è basata sulla necessità di gestire il grado di multiprogrammazione. Su un sistema che non usa la memoria virtuale, anche la gestione della memoria è un tema correlato. Perciò, la decisione di trasferire un processo in memoria considererà i requisiti di memoria dei processi fuori dalla memoria.

[1] Un processo è detto processor bound se esegue prevalentemente lavoro computazionale e se usa solo occasionalmente i dispositivi di I/O. Un processo è detto I/O bound se passa la maggior parte del tempo usando i dispositivi di I/O piuttosto che il processore.

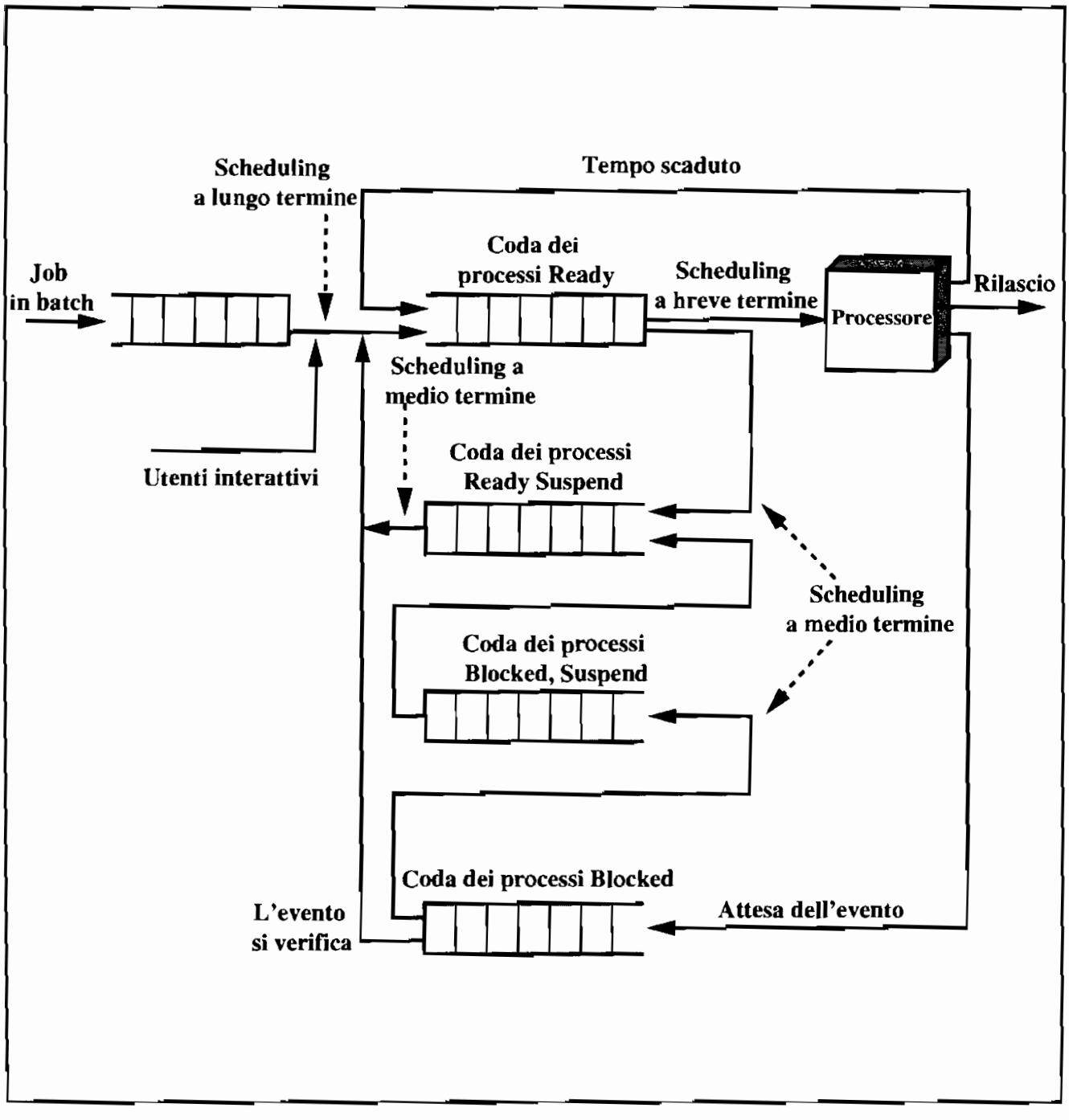

Figura 9.3 *Diagramma delle code usate per lo scheduling*

Scheduling a breve termine

In termini di frequenze di esecuzione, lo scheduler a lungo termine è eseguito poco frequentemente e prende la decisione grossolana se caricare un nuovo processo in memoria e, nel caso, quale caricare. Lo scheduler a medio termine è eseguito un poco più spesso per decidere se è necessario un trasferimento su disco. Lo scheduler a breve termine, detto anche allocatore o dispatcher, è eseguito più frequentemente e prende la decisione più fine di quale processo eseguire per prossimo.

Lo scheduler a breve termine è chiamato ogni volta che si verifica un evento che può portare alla sospensione del processo corrente o che consente di prerilasciare il processo attualmente in esecuzione in favore di un altro. Di seguito, alcuni esempi di tali eventi:

- Interruzioni di clock
- Interruzioni di I/O
- Chiamate a sistema operativo
- Segnali.

9.2 Algoritmi di scheduling

I criteri dello scheduling a breve termine

Generalmente, si valutano le strategie di scheduling sulla base dei seguenti criteri.

I criteri solitamente usati possono essere classificati in due dimensioni: la prima consiste nel fare una distinzione tra criteri orientati agli utenti e criteri orientati al sistema. I criteri **orientati all'utente** sono relativi a come il singolo utente o processo percepisce il comportamento del sistema. Un esempio è il tempo di risposta in un sistema interattivo, cioè il tempo trascorso tra l'invio di una richiesta e l'inizio dell'uscita della risposta. Questa quantità è visibile all'utente e naturalmente gli interessa. Si vorrebbe una strategia di scheduling che fornisca un buon servizio ai vari utenti: nel caso del tempo di risposta, si può definire una soglia (ad esempio, 2 secondi). Lo scopo del meccanismo di scheduling consiste nel massimizzare il numero di utenti che sperimentano un tempo medio di risposta di non più di 2 secondi.

Gli altri criteri sono **orientati al sistema** il cui punto focale sta nell'efficiente ed efficace utilizzazione del processore. Un esempio è il throughput, che è la frequenza di completamento dei processi. Questa è certamente una valida misura delle prestazioni del sistema, ed è una misura che deve essere massimizzata; però si concentra sulle prestazioni del sistema piuttosto che sui servizi prestati all'utente: perciò, è di competenza dell'amministratore di sistema ma non degli utenti.

Mentre i criteri orientati all'utente sono praticamente importanti su tutti i sistemi, i criteri orientati al sistema sono generalmente di importanza minore sui sistemi a singolo utente, dove non è importante raggiungere un'alta utilizzazione del processore o un alto throughput fintanto che la prontezza di risposta del sistema alle applicazioni dell'utente è accettabile.

Un'altra dimensione di classificazione consiste nel distinguere i criteri che sono in relazione con le prestazioni da quelli che non sono in diretta relazione con essa. I criteri che sono **relativi alle prestazioni** sono quantitativi e generalmente possono essere prontamente misurati. Alcuni esempi sono il tempo di risposta e il throughput. I criteri **non relativi alle prestazioni** o sono qualitativi, o non sono facilmente misurabili e analizzabili: un esempio di tali criteri è la prevedibilità. Si vorrebbe che alcune caratteristiche dei servizi forniti agli utenti siano indipendenti dagli altri lavori eseguiti dal sistema; in una qualche misura, questo criterio può essere misurato, calcolando le variazioni come funzione del carico di lavoro. Comunque, questo non è

diretto quanto misurare il throughput o il tempo di risposta come funzione del carico di lavoro.

La Tabella 9.2 riassume i criteri chiave dello scheduling. Questi sono interdipendenti e non è possibile ottimizzarli tutti allo stesso tempo. Per esempio, fornire un buon tempo di risposta può richiedere un algoritmo di scheduling che cambia frequentemente i processi, ma questo aumenta l'overhead nel sistema, riducendo il throughput. Perciò la progettazione di una strategia di

Tabella 9.2 *Criteri di scheduling*

	Orientati all'utente, Relativi alle prestazioni
Tempo di risposta	Per un processo interattivo, questo è il tempo che trascorre dall'invio di una richiesta fino a quando la risposta non comincia ad essere ricevuta. Spesso un processo può cominciare a produrre qualche output per l'utente, mentre continua l'esecuzione. Perciò, dal punto di vista dell'utente, questa è una misura migliore del tempo di turnaround. La disciplina di scheduling dovrebbe tentare di raggiungere bassi tempi di risposta e di massimizzare il numero di utenti interattivi che hanno un tempo di risposta accettabile.
Tempo di turnaround	È l'intervallo di tempo tra l'invio di un processo e il suo completamento. Comprende il tempo di esecuzione e il tempo speso in attesa delle risorse, compreso il processore. Questa è una misura appropriata per i job batch.
Scadenze	Quando si può specificare il termine di completamento di un processo, la disciplina di scheduling può subordinare altri obiettivi oltre a quello di massimizzare la percentuale di scadenze raggiunta.
	Orientati all'utente, Altri
Prevedibilità	Un dato job può essere eseguito nello stesso tempo e allo stesso costo, indipendentemente dal carico del sistema. Un'ampia variazione nel tempo di risposta o nel tempo di turnaround è fastidiosa per l'utente: potrebbe segnalare un'ampia oscillazione nel carico di lavoro del sistema oppure la necessità di adattare il sistema per evitare l'instabilità
	Orientati al sistema, Relativi alle prestazioni
Throughput	La strategia di scheduling dovrebbe tentare di massimizzare il numero di processi completati per unità di tempo. Così si misura la quantità di lavoro eseguito, che dipende chiaramente dalla lunghezza media di un processo, ma che è anche influenzata dalla strategia di scheduling che può avere effetti sull'utilizzazione
Utilizzazione del processore	Questa è la percentuale del tempo in cui il processore è occupato. Per un sistema costoso e condiviso, questo è un criterio significativo, mentre in sistemi a singolo utente e in alcuni altri sistemi, come i sistemi a tempo reale, questo criterio è meno importante.
	Orientati al sistema, Altri
Fairness	In assenza di indicazioni da parte dell'utente, oppure del sistema, i processi devono essere trattati allo stesso modo, e nessun processo deve subire starvation.
Priorità	Quando si assegna una priorità ai processi, la strategia di scheduling dovrebbe favorire i processi a più alta priorità.
Bilanciamento delle risorse	La strategia di scheduling deve tenere impegnate le risorse del sistema, favorendo i processi che sottoutilizzano le risorse più impegnate. Questo criterio coinvolge anche lo scheduling a medio e lungo termine.

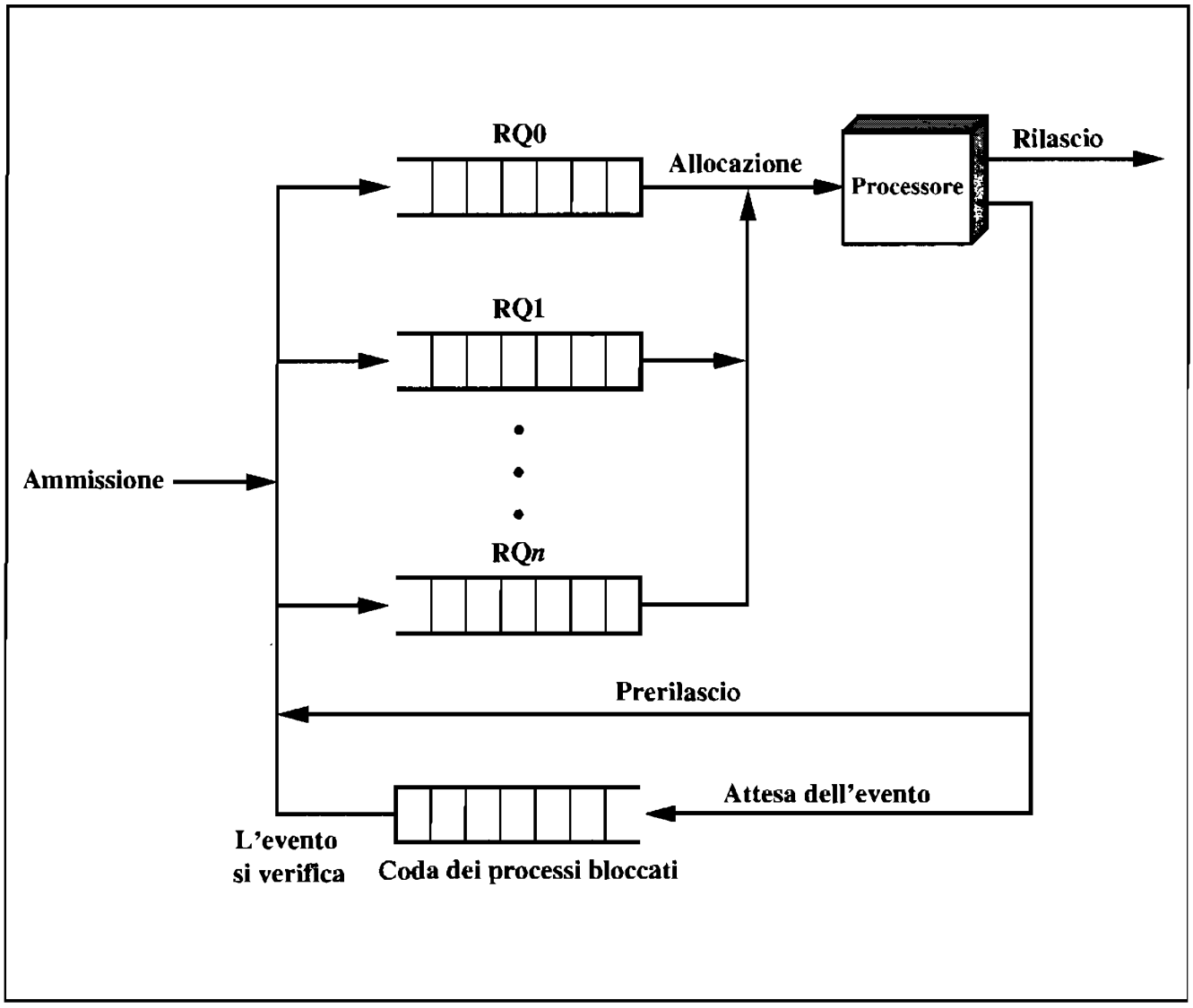

Figura 9.4 *Code di priorità*

scheduling richiede un compromesso tra i requisiti in competizione; i pesi relativi dati ai vari requisiti dipenderanno dalla natura e dall'uso del sistema.

Nella maggior parte dei sistemi operativi interattivi, siano essi a singolo utente o in time-sharing, il requisito critico è il tempo di risposta adeguato. Per l'importanza di questo requisito, e poiché la definizione di adeguatezza varierà da un'applicazione all'altra, l'argomento è approfondito nell'appendice di questo capitolo.

L'uso delle priorità

In molti sistemi, ad ogni processo è assegnata una priorità e lo scheduler deve sempre scegliere un processo a più alta priorità a discapito di processi con più bassa priorità. La Figura 9.4 illustra l'uso delle priorità. Per chiarezza, il diagramma a code è semplificato, ignorando l'esistenza di più code di processi bloccati e di stati Suspended (confrontare con la Figura 3.6a). Invece di una sola coda di processi Ready, si usa un insieme di code, in ordine di priorità discen-

dente: RQ0, RQ1, ..., RQn, con priorità [RQi] > priorità [RQj] per $i < j$. Quando deve essere fatta una selezione di scheduling, lo scheduler comincerà dalla coda di processi Ready con priorità più alta (RQ0): se ci sono uno o più processi nella coda, una strategia di scheduling ne seleziona uno, se, invece, RQ0 è vuota, allora si esamina RQ1 e così via.

Il problema con uno schema di scheduling basato solamente sulla priorità è che i processi con la priorità più bassa possono soffrire di starvation. Questo succederà se sono presenti stabilmente processi Ready a più alta priorità. Per superare questo problema, la priorità di un processo può cambiare in dipendenza del tempo di presenza in una coda o del corso della sua esecuzione. In seguito sarà presentato un esempio di ciò.

Strategie di scheduling alternative

La Tabella 9.3 presenta alcune informazioni riassuntive sulle varie strategie di scheduling esaminate in questa sezione. La **funzione di selezione** determina quale processo, tra i processi Ready, selezionare per la prossima esecuzione. La funzione può basarsi sulla priorità, sulle richieste di risorsa o sulle caratteristiche di esecuzione del processo. Nell'ultimo caso, tre sono le quantità significative:

w = tempo fin qui speso nel sistema, aspettando e eseguendo

e = tempo speso fin qui in esecuzione

s = tempo totale di servizio richiesto dal processo, compreso e

Per esempio, la funzione di selezione max[w] indica la strategia del first-come-first-served (FCFS).

Il **modo di decisione** specifica gli istanti in cui si esegue la funzione di selezione. Ci sono due categorie generali:

- **Senza prerilascio:** in questo caso, una volta che il processo è in esecuzione, continua la sua esecuzione fino alla terminazione o finché si blocca in attesa di I/O o se richiede altri servizi del sistema operativo.
- **Con prerilascio:** Il processo al momento in esecuzione può essere interrotto e spostato in stato Ready dal sistema operativo. La decisione di prerilascio può essere presa quando arriva un nuovo processo, quando avviene un interrupt che trasferisce un processo dallo stato Blocked allo stato Ready, oppure periodicamente in base all'interruzione di clock.

Le strategie con prerilascio provocano un overhead maggiore di quelle senza prerilascio, ma possono fornire un servizio migliore all'intero insieme dei processi, perché impediscono ad ogni processo di monopolizzare il processore per molto tempo. Inoltre, il costo del prerilascio può essere mantenuto relativamente basso usando meccanismi efficienti per il cambio di processo (si ricerca l'aiuto maggiore possibile da parte dell'hardware) e fornendo tanta memoria principale da contenere una gran percentuale dei programmi.

Per descrivere le varie strategie di scheduling, si userà il seguente insieme di processi come esempi:

Tabella 9.3 *Caratteristiche delle varie strategie di scheduling*

	FCFS	Round Robin	SPN	SRT	HRRN	Feedback
Funzione di selezione	max[w]	costante	min[s]	min[$s - e$]	max $(\frac{w+s}{s})$	(vedere il testo)
Modalità di decisione	Senza prerilascio	Con prerilascio (allo scadere del quanto di tempo)	Senza prerilascio	Con prerilascio (all'arrivo)	Senza prerilascio	Con prerilascio (allo scadere del quanto di tempo)
Throughput	Non enfatizzato	Può essere basso se il quanto è troppo breve	Alto	Alto	Alto	Non enfatizzato
Tempo di risposta	Può essere alto, specialmente se c'è una grande varianza nel tempo di esecuzione del processo	Fornisce un buon tempo di risposta per i processi brevi	Fornisce un buon tempo di risposta per i processi brevi	Fornisce un buon tempo di risposta	Fornisce un buon tempo di risposta	Non enfatizzato
Overhead	Minimo	Basso	Può essere alto	Può essere alto	Può essere alto	Può essere alto
Effetto sui processi	Penalizza i processi brevi e quelli I/O bound	Trattamento fair	Penalizza i processi lunghi	Penalizza i processi lunghi	Buon bilanciamento	Può favorire i processi I/O bound
Starvation	No	No	Possibile	Possibile	No	Possibile

w = tempo trascorso complessivamente nel sistema, in attesa e in esecuzione

e = tempo trascorso complessivamente in esecuzione

s = tempo di servizio totale richiesto dal processo, compreso e.

Processo	Tempo di arrivo	Tempo di servizio
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Si supponga di considerarli job in batch, e si supponga che il tempo di servizio sia il tempo totale richiesto per l'esecuzione. Alternativamente, è possibile considerare i processi elencati come processi in esecuzione che richiedono un uso alternato del processore e dell'I/O ripetutamente; in quest'ultimo caso, i tempi di servizio rappresentano il tempo di processore richiesto per un ciclo. In entrambi i casi, in termini di modello a code (vedere appendice A), questa quantità corrisponde al tempo di servizio.

First Come First Served

La più semplice strategia di scheduling è il first-come-first-served (primo arrivato primo servito - FCFS), o first-in-first-out (FIFO). Non appena un processo diventa Ready, s'inserisce nella coda dei processi Ready. Quando il processo al momento in esecuzione si sospende, il più vecchio processo nella coda dei processi Ready è selezionato per l'esecuzione.

La Figura 9.5 mostra il cammino di esecuzione del nostro esempio per un ciclo, e la Tabella 9.4 fornisce alcuni risultati chiave. Dapprima si determina il tempo a cui si sospende un processo, dal quale si può determinare il tempo di turnaround. Nei termini del modello a code, questo è il tempo che il processo trascorre in coda, o il tempo totale che un elemento trascorre nel sistema (tempo di attesa più tempo di servizio). Un numero più utile è il tempo di turnaround normalizzato, che è il rapporto tra il tempo di turnaround e il tempo di servizio. Questo valore indica il ritardo relativo subito da un processo. Tipicamente, più lungo è il tempo di esecuzione di un processo, maggiore è il ritardo assoluto totale che può essere tollerato. Il valore minimo di questo rapporto è 1.0; aumentare i valori significherebbe decrescere il livello del servizio.
FCFS offre migliori prestazioni per i processi lunghi che per quelli brevi. Si considerino i seguenti esempi, basati su un esempio di [FINK88]:

Processo	Tempo di arrivo	Tempo di servizio (T_s)	Tempo di inizio	Tempo di fine	Tempo di turnaround (T_q)	T_q/T_s
A	0	1	0	1	1	1
B	1	100	1	101	100	1
C	2	1	101	102	100	100
D	3	100	102	202	199	1.99
Media					100	26

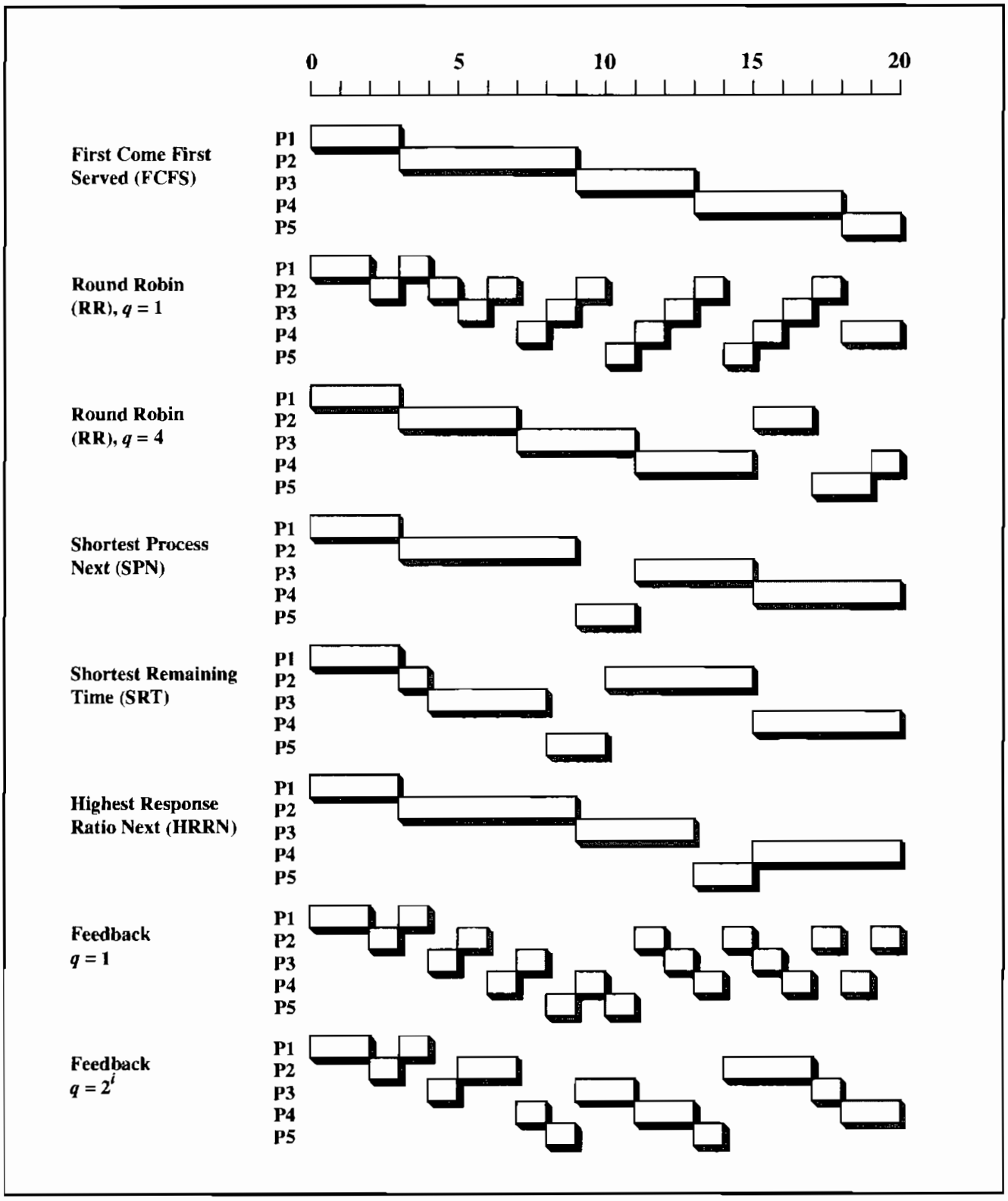

Figura 9.5 *Confronto tra le strategie di scheduling*

Il tempo di turnaround normalizzato per il processo C è intollerabile: il tempo totale che trascorre nel sistema è 100 volte il tempo di esecuzione richiesto, e questo accade ogni volta che un processo breve arriva subito dopo un processo lungo. D'altra parte, anche in quest'esempio estremo, i processi lunghi non vanno male: il processo D ha un tempo di turnaround che è quasi

Tabella 9.4 *Confronto tra le strategie di scheduling*

	Processo	1	2	3	4	5	Media
	Tempo di arrivo	0	2	4	6	8	
	Tempo di servizio (T_s)	3	6	4	5	2	
FCFS	Tempo di fine	3	9	13	18	20	
	Tempo di turnaround (T_q)	3	7	9	12	12	8.60
	T_q/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR q = 1	Tempo di fine	4	18	17	20	15	
	Tempo di turnaround (T_q)	4	16	13	14	7	10.80
	T_q/T_s	1.33	2.67	3.25	2.80	3.50	2.71
RR q = 4	Tempo di fine	3	17	11	20	19	
	Tempo di turnaround (T_q)	3	15	7	14	11	10.00
	T_q/T_s	1.00	2.5	1.75	2.80	5.50	2.71
SPN	Tempo di fine	3	9	15	20	11	
	Tempo di turnaround (T_q)	3	7	11	14	3	7.6
	T_q/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRT	Tempo di fine	3	15	8	20	10	
	Tempo di turnaround (T_q)	3	13	4	14	2	7.20
	T_q/T_s	1.00	2.17	1.00	2.80	1.00	1.59
HRRN	Tempo di fine	3	9	13	20	15	
	Tempo di turnaround (T_q)	3	7	9	14	7	8.00
	T_q/T_s	1.00	1.17	2.25	2.80	3.5	2.14
FB q = 1	Tempo di fine	4	20	16	19	11	
	Tempo di turnaround (T_q)	4	18	12	13	3	10.00
	T_q/T_s	1.33	3.00	3.00	2.60	1.5	2.29
FB q = $2^{(i-1)}$	Tempo di fine	4	17	18	20	14	
	Tempo di turnaround (T_q)	4	15	14	14	6	10.60
	T_q/T_s	1.33	2.50	3.50	2.80	3.00	2.63

il doppio di quello di C, ma il suo tempo di attesa normalizzato è inferiore a 2.0.

Il problema di questa strategia consiste nel fatto che tende a favorire i processi processor-bound a discapito di quelli I/O-bound. Si consideri un insieme di processi, uno dei quali usa prevalentemente il processore (processor bound) mentre gli altri preferiscono l'I/O (I/O bound). Quando un processo processor-bound è in esecuzione, tutti i processi I/O-bound devono aspettare; alcuni di questi possono essere nelle code di I/O (in stato Blocked) ma possono tornare nella coda dei processi Ready mentre il processo processor-bound è in esecuzione. A questo punto, la maggior parte dei dispositivi di I/O può essere in ozio, anche se c'è del lavoro potenziale per loro. Quando il processo al momento in esecuzione esce dallo stato Running, i processi I/O-bound si spostano rapidamente in stato Running, ma si bloccano sui rispettivi eventi di I/O. Se anche il processo processor-bound è bloccato, il processore rimane in ozio. Perciò, FCFS può usare inefficientemente sia il processore sia i dispositivi di I/O.

FCFS non è di per se stesso un'alternativa attraente per un sistema a singolo processore, ma è spesso combinato con uno schema di priorità per fornire uno scheduler efficiente. Pertanto, lo scheduler può mantenere alcune code, una per ogni livello di priorità, e distribuire i processi all'interno di ogni coda utilizzando il first-come-first-served. Un esempio di tale sistema si vedrà in seguito, discutendo dello scheduling con feedback.

Round robin

Un semplice modo per ridurre le penalità che i processi brevi subiscono con FCFS, è usare il prerilascio basato sul clock. La più semplice di tali strategie è il round robin. Un'interruzione di clock è generata ad intervalli periodici. Quando avviene un'interruzione, il processo in esecuzione al momento è inserito nella coda dei processi Ready e il successivo job Ready è selezionato con FCFS. Questa tecnica è anche conosciuta con il nome di **time slicing**, perché ad ogni processo si dà un po' di tempo prima di essere prerilasciato.

Con il round robin, la principale preoccupazione in fase di progettazione è la lunghezza del quanto di tempo, o slice, da usare. Se il quanto di tempo è molto breve, i processi brevi passeranno abbastanza velocemente all'interno del sistema. D'altra parte, c'è un tempo aggiuntivo di elaborazione dovuto alla gestione dell'interruzione di clock ed all'esecuzione delle funzioni di scheduling ed allocazione, perciò devono essere evitati quanti di tempo troppo brevi. Un criterio utile è di avere un quanto di tempo leggermente maggiore del tempo richiesto per una tipica interazione: se fosse minore, la maggior parte dei processi richiederebbe almeno due quanti di tempo. La Figura 9.6 illustra gli effetti che questo ha sul tempo di risposta: notare che nel caso limite di un quanto di tempo più lungo del più lungo processo Running, il round robin degenera in FCFS.

La Figura 9.5 e la Tabella 9.4 mostrano i risultati dell'esempio precedente usando un quanto di tempo q di 1 e 4 unità di tempo, rispettivamente. Bisogna notare che il processo 5, che è il job più breve, gode di un significativo miglioramento per un quanto di tempo di un'unità.

Il round robin è particolarmente efficiente in un sistema generale time-sharing, o in un sistema orientato alle transazioni; un inconveniente del round robin è il trattamento che riserva ai processi processor-bound e I/O-bound. Generalmente, un processo I/O-bound ha un burst di processo (tempo speso in esecuzione fra due operazioni di I/O) minore di quello di un processo processor-bound. Se ci sono sia processi I/O-bound sia processi processor-bound, un processo I/O-bound userà il processore per un breve tempo, poi sarà bloccato per l'I/O; aspetterà il completamento dell'operazione di I/O e poi s'inserirà nella coda dei processi Ready. Invece, un processo processor-bound generalmente usa un intero quanto di tempo e poi torna nella coda dei processi Ready. Perciò, i processi processor-bound tendono a ricevere una porzione del tempo di processore non equa, il che provoca scarse prestazioni dei processi I/O-bound, un uso inefficiente dei dispositivi di I/O e un aumento della varianza del tempo di risposta.

[HAL91] suggerisce un miglioramento del round robin, chiamato round robin virtuale (VRR), che elimina questa scorrettezza. La Figura 9.7 illustra lo schema: i nuovi processi arrivano e sono inseriti nella coda dei processi Ready, che è gestita con FCFS. Quando scade il tempo del processo in esecuzione, esso è reinserito nella coda dei processi Ready. Quando un processo è bloccato per l'I/O, s'inserisce in una coda di processi in attesa dell'I/O: finora tutto è come al solito. La nuove caratteristica è una coda ausiliaria FCFS nella quale sono inseriti i processi,

dopo essere stati rilasciati da un blocco di I/O. Quando si deve decidere un'allocazione dei processi, i processi che sono nella coda ausiliaria hanno la precedenza su quelli della coda dei processi Ready. Quando un processo è prelevato dalla coda ausiliaria, viene eseguito per un quanto, lungo al più come il quanto di tempo di base, meno il tempo totale trascorso in esecuzione dall'ultima volta che è stato selezionato dalla coda principale dei processi Ready. Studi sulle prestazioni condotti dagli autori indicano che quest'approccio è davvero superiore, in termini di fairness, al round robin.

Shortest Process Next

Un altro approccio per ridurre il vantaggio in favore dei processi lunghi, intrinseco per FCFS, è la strategia Shortest Process Next (SPN: il prossimo processo è il più breve). Questa è una strategia senza prerilascio, nella quale si sceglie come successivo processo da eseguire il proces-

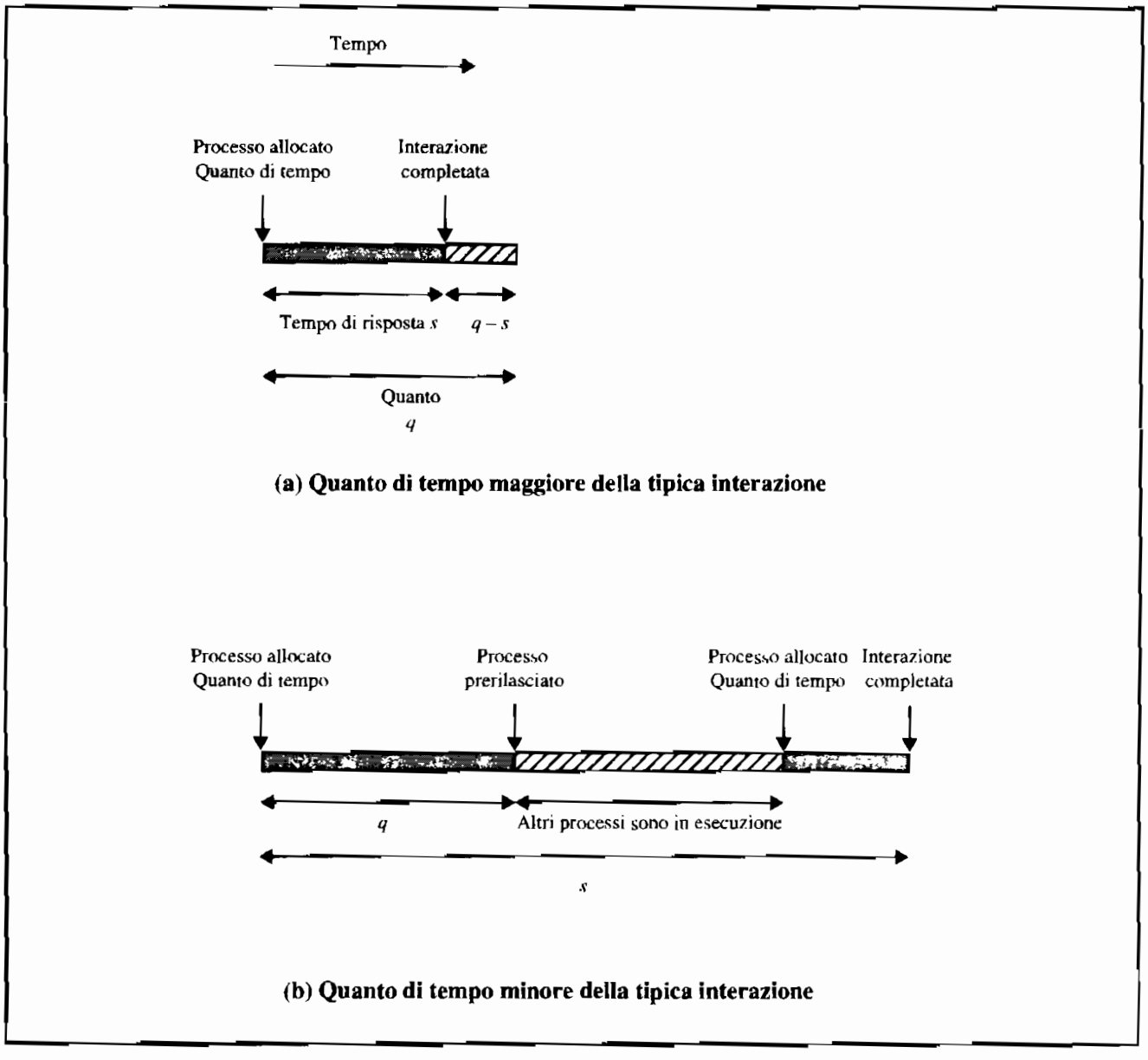

Figura 9.6 *Effetti della dimensione del quanto di tempo di prerilascio*

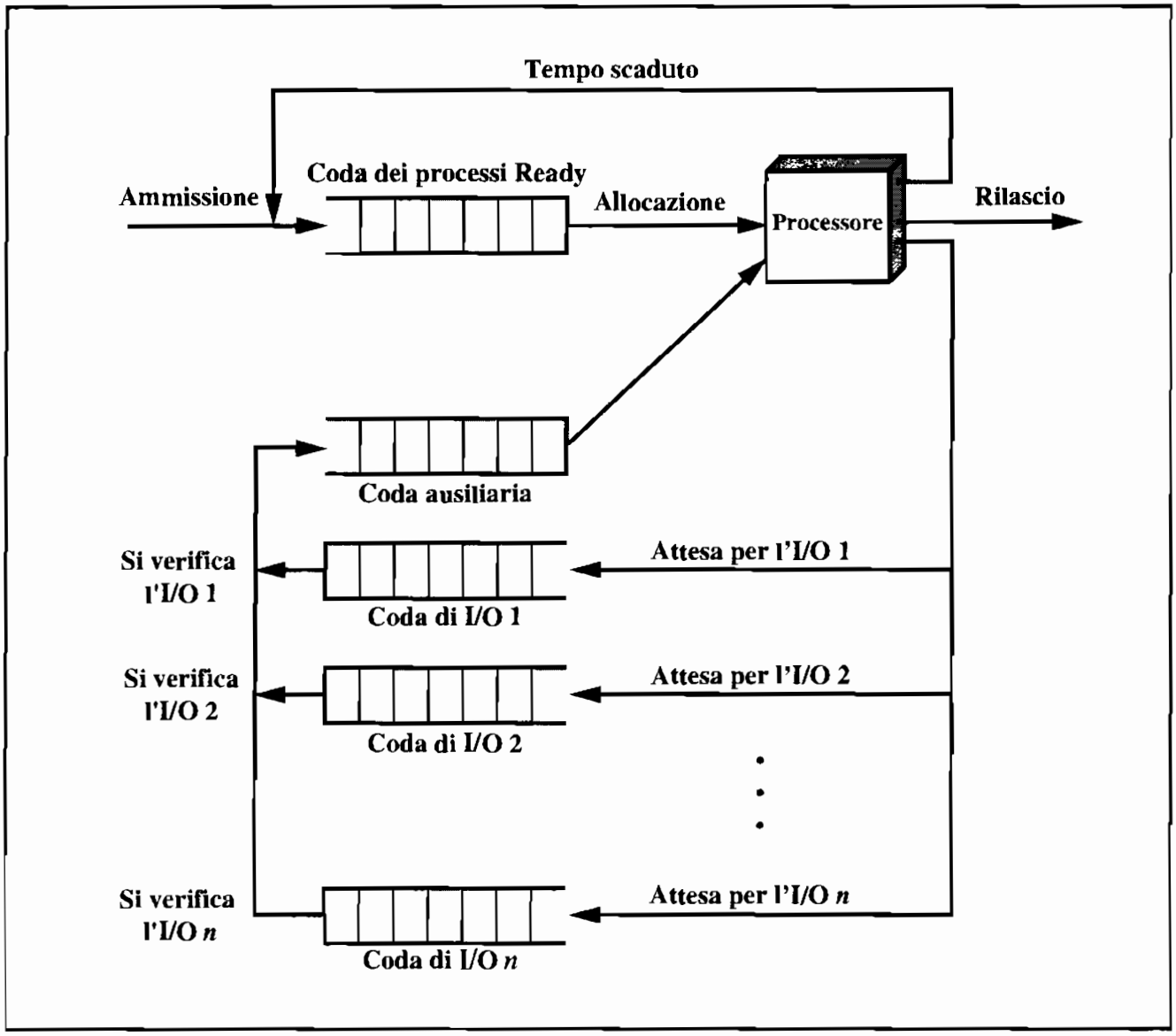

Figura 9.7 *Diagramma a code per lo scheduler round robin virtuale*

so con il minor tempo esecuzione previsto. Perciò un processo breve balzerà in testa alla coda sorpassando i job più lunghi.

La Figura 9.5 e la Tabella 9.4 mostrano i risultati per l'esempio considerato. Bisogna notare che il processo 5 è servito molto prima di quando si è usato FCFS, e le prestazioni complessive sono significativamente migliorate in termini di tempo di risposta. Comunque, la variabilità dei tempi di risposta aumenta, specialmente per i processi più lunghi, e perciò la prevedibilità è ridotta.

Una difficoltà che sorge nell'uso della strategia SPN è la necessità di conoscere, o almeno stimare, il tempo di esecuzione previsto per ogni processo. Per i job in batch, il sistema può richiedere al programmatore di stimare il valore e fornirlo al sistema operativo; se la stima del programmatore è decisamente inferiore al tempo di esecuzione effettivo, il sistema può "abortire" il processo. In un ambiente di produzione, gli stessi job girano frequentemente e si possono fare statistiche. Per i processi interattivi, il sistema operativo può calcolare una media corrente ad ogni burst per ogni processo. La seguente formula permette di fare il calcolo più semplice:

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i \tag{9.1}$$

dove

T_i = tempo di esecuzione del processore per la i-esima istanza di questo processo (tempo totale di esecuzione per job in batch; tempo di un burst di processo per job interattivo)

S_i = valore previsto per la i-esima istanza

S_1 = valore previsto per la prima istanza; non calcolato

Per evitare di ricalcolare l'intera somma ogni volta, si può riscrivere la formula precedente come:

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n \tag{9.2}$$

Bisogna notare che questa formula dà uguale peso ad ogni istanza; ma tipicamente, si vorrà dare più peso alle istanze più recenti, perché prevedono più verosimilmente il comportamento futuro. Una tecnica comune per predire un valore futuro sulle basi di una serie temporale dei valori passati è la *media esponenziale*:

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n \tag{9.3}$$

dove α è un peso costante ($0 < \alpha < 1$) che determina il peso relativo dato alle osservazioni più e meno recenti. Confrontando questa equazione con la (9.2), con un valore costante di α, indipendentemente dal numero delle osservazioni passate, tutti i valori passati sono considerati, ma i più distanti hanno meno peso. Per vedere questo più chiaramente, si consideri la seguente espansione dell'Equazione (9.3):

$$S_{n+1} = \alpha T_n + (1 - \alpha)\, \alpha T_{n-1} + \ldots + (1 - \alpha)^i\, \alpha T_{n-i} + (1 - \alpha)^n S_1 \tag{9.4}$$

Poiché sia α che $(1 - \alpha)$ sono minori di uno, ogni termine successivo nell'equazione (9.4) è sempre più piccolo. Per esempio, per $\alpha = 0.8$, l'Equazione (9.4) diventa

$$S_{n+1} = 0.8T_n + 0.16T_{n-1} + 0.032T_{n-2} + 0.0064T_{n-3} + \ldots$$

Più vecchia è l'osservazione, meno conta nella media.

La dimensione del coefficiente, come funzione della sua posizione nell'espansione, è mostrata in Figura 9.8. Più grande è il valore di α, maggiore è il peso dato alle osservazioni più recenti: per $\alpha = 0.8$, virtualmente tutto il peso è dato alle quattro osservazioni più recenti, mentre per $\alpha = 0.2$, la media considera le otto osservazioni più recenti. Il vantaggio di usare un valore di α vicino ad uno è che la media rifletterà un rapido cambio nella quantità osservata; lo svantaggio è che se c'è un breve sbalzo nel valore della quantità osservata, che poi si fissa su qualche valore medio, l'uso di un grande valore di a risulterà in cambiamenti a balzi della media.

La Figura 9.9 confronta la media semplice con la media esponenziale (per due diversi valori di α). Nella Figura 9.9a il valore osservato parte da 1, cresce gradualmente fino a 10 e poi si ferma, mentre nella Figura 9.9b, parte da 20, scende gradualmente a 10 e poi si ferma; in entram-

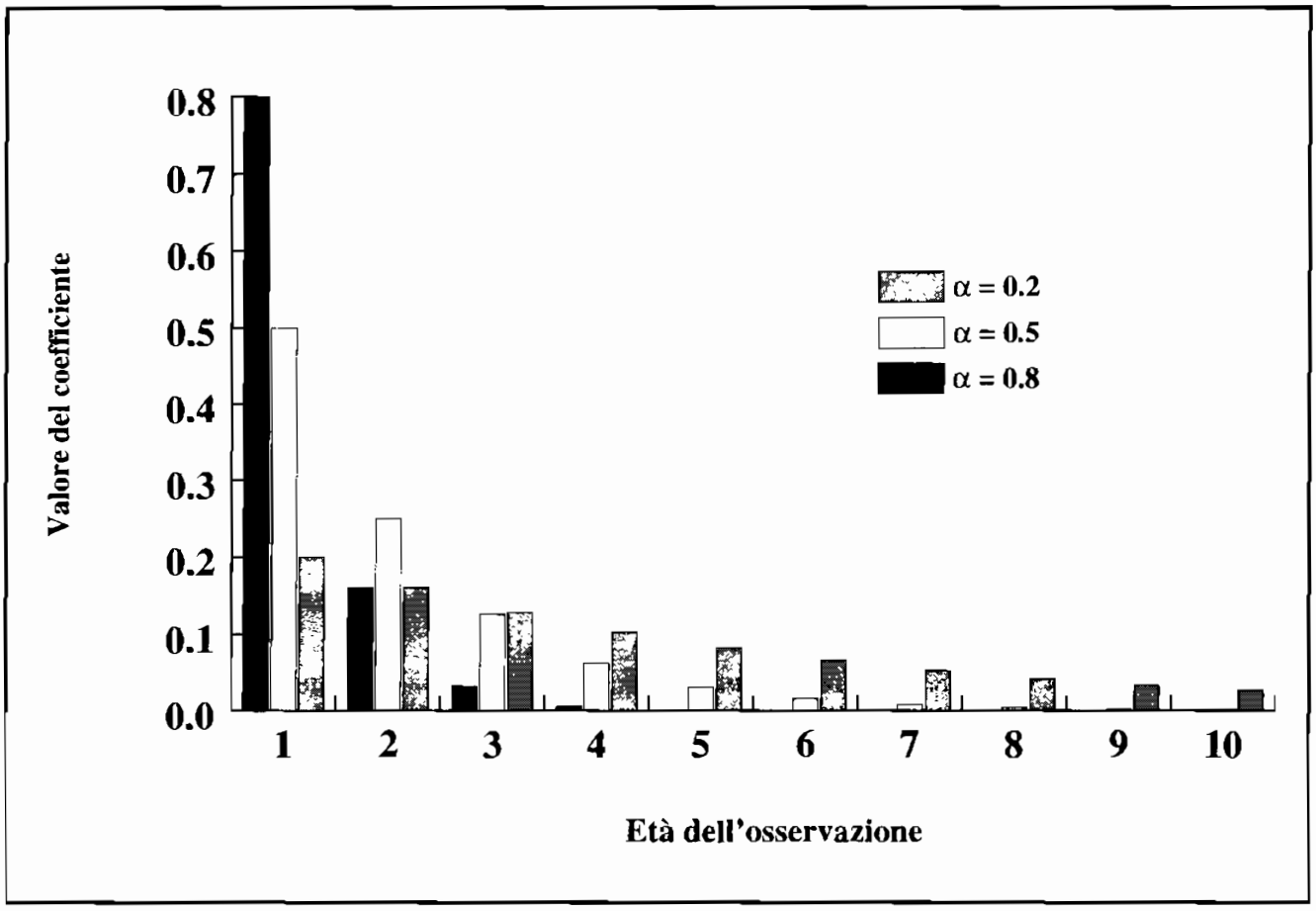

Figura 9.8 *Coefficienti per smorzare la media esponenziale*

bi i casi, si comincia con una stima di $S_1 = 0$, che dà priorità maggiore al nuovo processo. Bisogna notare che la media esponenziale segue i cambiamenti di comportamento del processo più velocemente della media semplice, e che il valore più grande di α provoca una reazione più rapida ai cambiamenti nel valore osservato.

Con SPN i processi più lunghi rischiano la starvation, se c'è un arrivo stabile di processi più brevi; d'altra parte, sebbene SPN riduca il vantaggio a favore dei processi più lunghi, non è ancora accettabile per un ambiente time sharing o orientato alle transazioni, a causa della mancanza del prerilascio. Guardando indietro all'analisi del caso peggiore descritto per FCFS, i processi A, B, C e D saranno ancora eseguiti nello stesso ordine, penalizzando pesantemente il processo C.

Shortest Remaining Time

La strategia dello Shortest Remaining Time (SRT: tempo rimanente più breve) è una versione di SPN con prerilascio: in questo caso, lo scheduler sceglie sempre il processo che ha il minor tempo di esecuzione rimanente previsto. Quando un nuovo processo è inserito nella coda dei processi Ready, può avere un tempo rimanente più breve di quello del processo in esecuzione al momento; di conseguenza, lo scheduler può dover prerilasciare ogni volta che un nuovo processo diventa Ready. Come per SPN, lo scheduler deve avere una stima del tempo di elaborazione per eseguire la funzione di selezione, e c'è il rischio di starvation per i processi più lunghi.

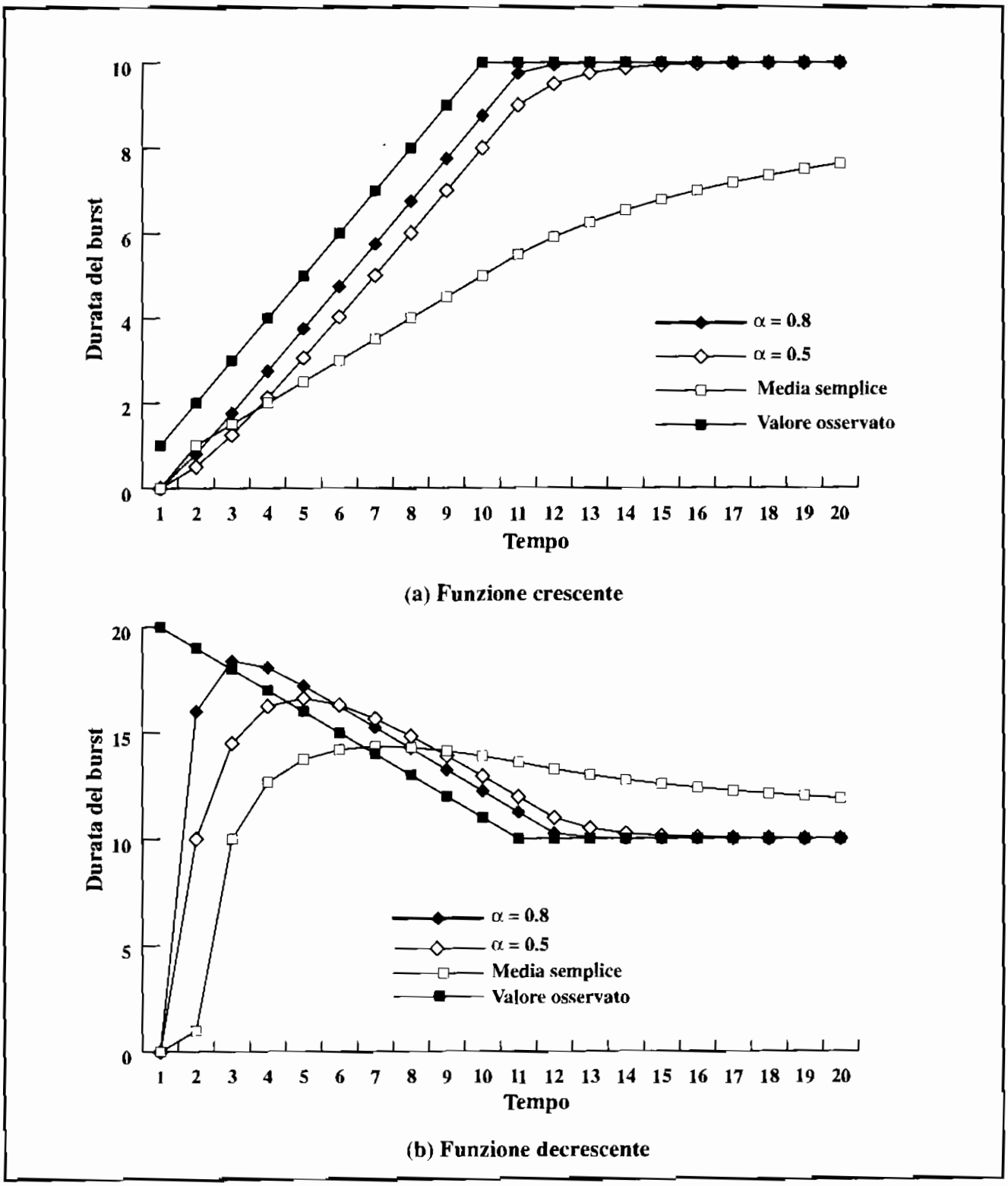

Figura 9.9 *Uso della media esponenziale*

SRT non avvantaggia i processi più lunghi, come fa FCFS, e a differenza del round robin, non si generano interrupt aggiuntivi, riducendo l'overhead. D'altra parte, i tempi di servizio trascorsi devono essere registrati, e questo contribuisce all'overhead. SRT potrebbe anche portare ad un tempo di turnaround superiore alle prestazioni di SPN, poiché si dà una preferenza immediata ai lavori brevi, a discapito di quelli lunghi in esecuzione.

Bisogna notare che nell'esempio precedente (Tabella 9.4) i tre processi più brevi sono subito serviti, ottenendo per ciascuno un tempo di turnaround normalizzato di 1.0.

Highest Response Ratio Next

Nella Tabella 9.4 si è usato il tempo di turnaround normalizzato che è il rapporto tra il tempo di turnaround e il tempo di servizio attuale, come valore discriminante. Si voglia minimizzare questo rapporto per ogni processo e il suo valor medio per tutti i processi. Sebbene questa sia una misura a posteriori, si può approssimarla con una misura a priori come criterio di selezione in uno scheduler senza prerilascio. Specificatamente, si consideri il seguente rapporto, in cui RR è il rapporto di risposta:

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

dove

w = tempo speso ad aspettare il processore

s = tempo di servizio previsto

Se il processo con questo valore è allocato immediatamente, RR è uguale al tempo di turnaround normalizzato. Si noti che il valore minimo di RR è 1.0, che si ottiene quando un processo entra nel sistema per la prima volta.

Perciò, la regola di scheduling da seguire diventa: quando il processo in esecuzione termina o si blocca, si sceglie il processo Ready con il più grande valore di RR. Questo approccio si chiama Highest Response Ratio Next (HRRN: il più alto rapporto è il prossimo), ed è attraente perché tiene in considerazione l'età del processo: mentre sono favoriti i processi più brevi (un denominatore più piccolo conduce ad un maggior rapporto), invecchiare senza essere serviti aumenta il rapporto, così un processo più lungo supererà prima o poi i processi più brevi.

Come per SRT e SPN, per applicare HRRN si deve stimare il tempo di servizio previsto.

Feedback

Se non si hanno indicazioni sulla lunghezza relativa dei vari processi, allora non si può usare nessuno tra SPN, SRT e HRRN. Un altro modo per stabilire una preferenza per i job più brevi è penalizzare quei job che sono stati più a lungo in esecuzione. In altre parole, se non è possibile concentrarsi sul tempo d'esecuzione rimanente, ci si preoccupa del tempo speso in esecuzione fino ad ora.

Il modo per fare ciò è il seguente: lo scheduling è fatto su basi di prerilascio e si usa un meccanismo di priorità dinamico. Quando un processo entra per la prima volta, è posto nella coda Ready RQ0 (vedere Figura 9.4). Dopo la sua prima esecuzione, quando torna in stato Ready, è posto in RQ1. Ogni volta successiva, è spostato nella coda successiva con minore priorità: un processo più breve completerà prima la sua esecuzione, senza spostarsi molto in basso nella gerarchia delle code dei processi Ready, mentre un processo più lungo scenderà sempre più in basso. Perciò, i processi più brevi, arrivati da poco, sono favoriti rispetto a quelli

più vecchi e più lunghi. All'interno di ogni coda, eccetto nella coda a più bassa priorità, si usa un semplice meccanismo di FCFS. Una volta che un processo è nella coda con più bassa priorità, non può scendere più in basso, e si reinserisce ripetutamente in quest'ultima coda finché non ha completato la sua esecuzione; per questo motivo, questa coda è trattata con la strategia del round robin.

La Figura 9.10 illustra il meccanismo di scheduling con feedback, mostrando il cammino che un processo seguirà attraverso le varie code.[2] Questo approccio è conosciuto anche come feedback multilivello, nel senso che il sistema operativo alloca il processore ad un processo e, quando il processo si blocca od è prerilasciato, lo reinserisce in una delle diverse code di priorità.

Ci sono diverse variazioni a questo schema: una semplice versione suggerisce di eseguire il prerilascio nello stesso modo usato per il round robin: ad intervalli periodici. L'esempio riportato lo dimostra (Figura 9.5 e Tabella 9.4) per un quanto di tempo unitario: in questo caso il comportamento è simile a quello del round robin con un quanto di tempo unitario.

Il semplice schema appena esposto presenta però un problema: il tempo di turnaround dei processi più lunghi può aumentare in modo allarmante, ed è quindi possibile che si verifichi starvation se nuovi lavori entrano regolarmente nel sistema. Per compensare questo inconveniente, è possibile variare il tempo di prerilascio a seconda della coda: un processo schedulato

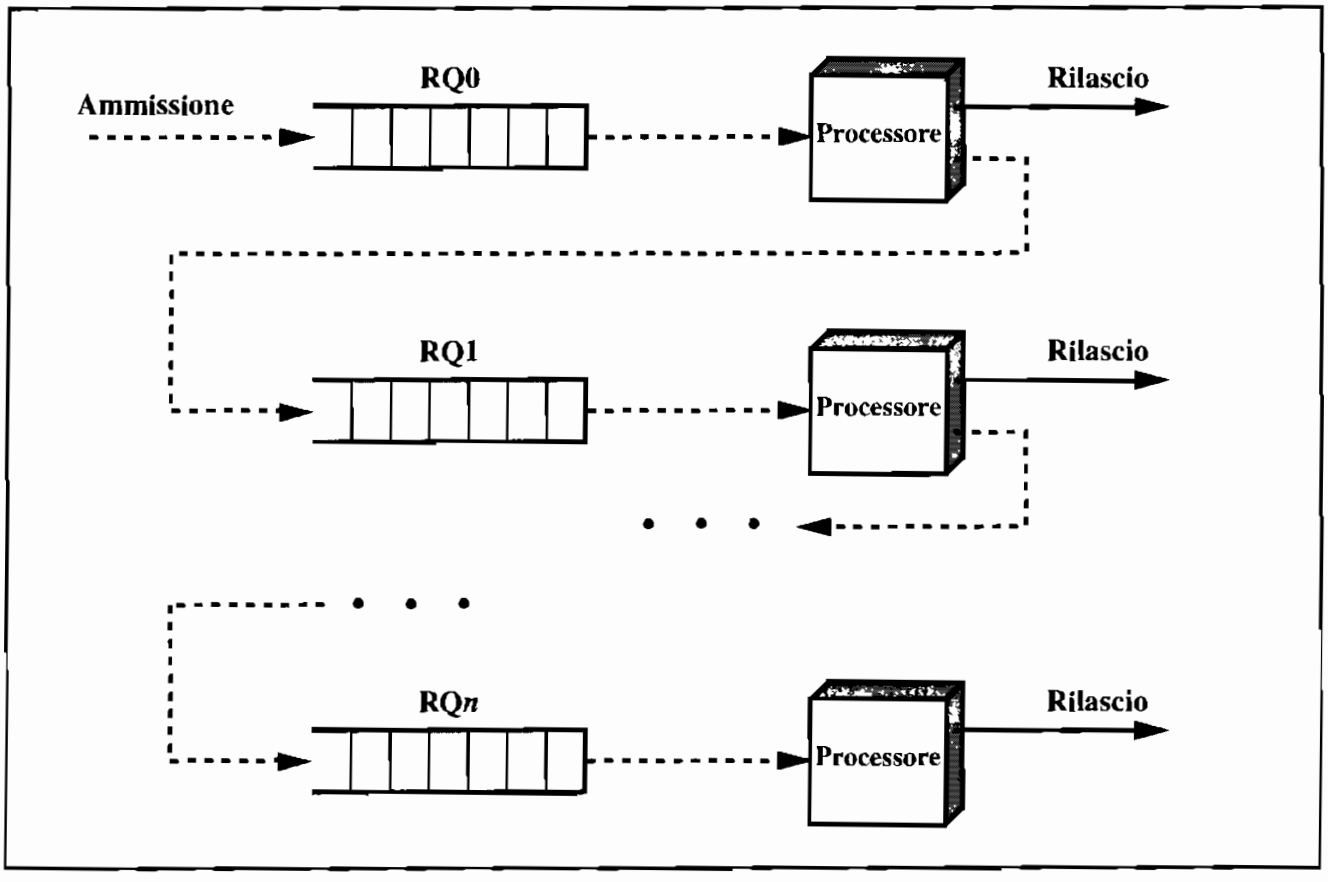

Figura 9.10 *Scheduling con feedback*

[2] Le linee punteggiate sono usate per evidenziare che questo è un diagramma di una sequenza di tempo piuttosto che una descrizione statica delle possibili transizioni, come nella Figura 9.4.

da RQ0 è autorizzato all'esecuzione per un'unità di tempo e poi è prerilasciato; un processo schedulato da RQ1 è autorizzato all'esecuzione per due unità di tempo, e così via. In generale, un processo schedulato da RQ_i è autorizzato all'esecuzione per 2^i unità di tempo prima del prerilascio: l'applicazione di questo schema al nostro esempio è illustrata in Figura 9.5 e nella Tabella 9.4.

Anche con la concessione di una maggiore quantità di tempo alla priorità più bassa, un processo più lungo può ancora soffrire di starvation. Un possibile rimedio consiste nella promozione di un processo ad una coda con priorità più alta, dopo che ha aspettato per un certo periodo il servizio nella coda in cui si trova.

da RQ0 è autorizzato all'esecuzione per un'unità di tempo e poi è prerilasciato; un processo schedulato da RQ1 è autorizzato all'esecuzione per due unità di tempo, e così via. In generale, un processo schedulato da RQ_i è autorizzato all'esecuzione per 2^i unità di tempo prima del prerilascio: l'applicazione di questo schema al nostro esempio è illustrata in Figura 9.5 e nella Tabella 9.4.

Anche con la concessione di una maggiore quantità di tempo alla priorità più bassa, un processo più lungo può ancora soffrire di starvation. Un possibile rimedio consiste nella promozione di un processo ad una coda con priorità più alta, dopo che ha aspettato per un certo periodo il servizio nella coda in cui si trova.

Confronto sulle prestazioni

Chiaramente, le prestazioni delle varie strategie di scheduling rappresentano un fattore critico nella scelta della strategia di scheduling. È anche impossibile fare un confronto completo perché le prestazioni relative dipenderanno da svariati fattori come la distribuzione di probabilità dei tempi di servizio dei processi, l'efficienza dello scheduling e del meccanismo di cambio di processo, la natura delle richieste di I/O e le prestazioni del sottosistema di I/O. In ogni caso, si tenterà di trarre di seguito alcune conclusioni generali.

Analisi delle code

In questa sezione, si farà uso delle formule basilari relative alle code, con le comuni ipotesi di arrivo secondo la distribuzione Poissoniana e i tempi di servizio esponenziali. Un riassunto di questi concetti si può trovare nell'Appendice A.

Innanzitutto, bisogna osservare che qualunque strategia, che scelga il prossimo elemento da servire indipendentemente dal tempo di servizio, obbedisce alla seguente relazione:

$$\frac{T_q}{T_s} = \frac{1}{1-\rho}$$

dove

T_q = tempo di turnaround; tempo totale trascorso nel sistema, aspettando ed in esecuzione
T_s = tempo di servizio medio; tempo medio trascorso in stato Running
ρ = utilizzazione del processore

In particolare, uno scheduler basato sulla priorità, nel quale la priorità di ogni processo è assegnata indipendentemente dal tempo di servizio previsto, fornisce lo stesso tempo di turnaround medio e lo stesso tempo di turnaround normalizzato medio della semplice strategia FCFS. Inoltre la presenza o l'assenza di prerilascio non fa differenza su queste medie.

Ad eccezione del round robin e di FCFS, le varie strategie di scheduling considerate finora selezionano i processi in base al tempo di servizio previsto. Sfortunatamente, risulta abbastanza difficile sviluppare modelli analitici chiusi per queste strategie, comunque, ci si può fare un'idea delle prestazioni relative di tali algoritmi di scheduling, a confronto con FCFS, considerando uno scheduling a priorità in cui la priorità è basata sul tempo di servizio.

In figura, il processo A è schedulato per primo; dopo un secondo, è prerilasciato, ed a questo punto i processi B e C hanno la priorità più alta, e si schedula il processo B. Alla fine della seconda unità di tempo, il processo A ha la più alta priorità: la sequenza di schedulazione si ripete, perché il kernel schedula i processi nell'ordine: A, B, A, C, A, B, ecc. Perciò il 50% del processore è allocato per il processo A, che costituisce un gruppo, e l'altro 50% è allocato per B e C, che costituiscono un altro gruppo.

9.3 Lo scheduling tradizionale di UNIX

In questa sezione si esaminerà lo scheduler tradizionale di UNIX, che è usato sia in SVR3 che in 4.3 BSD UNIX. Questi sistemi sono principalmente mirati ad ambienti interattivi time-sharing. L'algoritmo di scheduling è progettato per fornire un buon tempo di risposta agli utenti interattivi, assicurando che i job eseguiti in background a bassa priorità non soffrano di starvation. Sebbene questo algoritmo sia stato rimpiazzato nei moderni sistemi UNIX, vale la pena di esaminarlo perché è rappresentativo degli algoritmi di scheduling per time sharing. Lo schema di scheduling per SVR4 comprende facilitazioni per i requisiti in tempo reale, e quindi la sua trattazione è rimandata al capitolo 10.

Lo scheduler tradizionale di UNIX utilizza il feedback multilivello, sfruttando la strategia del round robin all'interno di ciascuna coda di priorità. Il sistema fa uso del prerilascio ogni secondo, cioè, se un processo in esecuzione non si blocca o non termina in un secondo, è prerilasciato. La priorità è basata sul tipo di processo e la storia dell'esecuzione. Si applicano le seguenti formule:

$$P_j(i) = \text{Base}_j + \frac{\text{CPU}j(i-1)}{2} + \text{nice}_j$$

$$\text{CPU}_j(i) = \frac{U_j(\text{i})}{2} + \frac{\text{CPU}_j(i-1)}{2}$$

dove

$P_j(i)$ = priorità del processo j all'inizio dell'intervallo i: valori più bassi equivalgono a più alte priorità

Base_j = priorità base del processo j

$U_j(i)$ = utilizzazione del processore da parte del processo j nell'intervallo i

$\text{CPU}_j(i)$ = utilizzazione media del processore pesata esponenzialmente, da parte del processo j nell'intervallo i

nice_j = fattore di assestamento, controllabile dall'utente

La priorità di ogni processo è ricalcolata una volta al secondo, nel momento in cui si prende una nuova decisione di scheduling. Lo scopo della priorità di base è dividere tutti i processi in bande fissate di livelli di priorità. I componenti CPU e nice si limitano ad evitare che un processo esca dalla banda cui è stato assegnato (assegnato dal livello di priorità di base). Queste bande

sono utilizzate per ottimizzare l'accesso ai dispositivi a blocchi (come i dischi) e permettere al sistema operativo di rispondere rapidamente alle chiamate di sistema. In ordine decrescente di priorità, le bande sono le seguenti:

- Swapper
- Controllo dei dispositivi di I/O a blocchi
- Manipolazione di file
- Controllo dei dispositivi di I/O a caratteri
- Processi utente.

Questa gerarchia dovrebbe dare il più efficiente uso dei dispositivi di I/O. All'interno della banda dei processi utente, l'uso della storia di esecuzione tende a penalizzare i processi processor

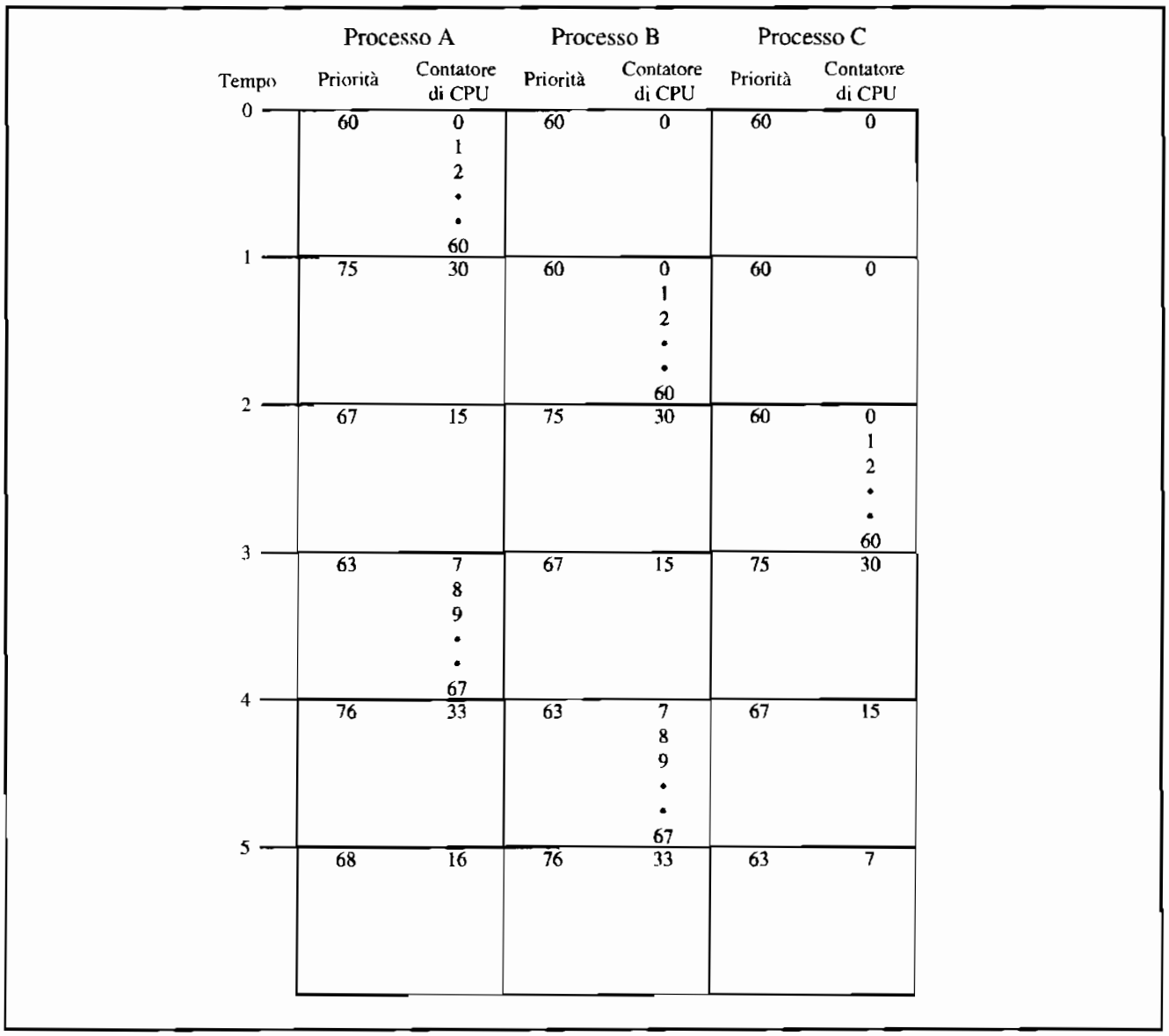

Figura 9.17 *Esempio dello scheduling tradizionale dei processi in UNIX [BACH86]*

bound, alle spese dei processi I/O bound: anche questo migliorerebbe l'efficienza. Accoppiato con lo schema del round robin con prerilascio, la strategia di scheduling è ben organizzata per soddisfare i requisiti del time sharing generale.

Un esempio di scheduling di processi è mostrato in Figura 9.17. I processi A, B e C sono creati nello stesso momento con priorità base di 60 (si ignorerà il valore nice). Il clock interrompe il sistema 60 volte al secondo ed aumenta il contatore del processo in esecuzione. L'esempio suppone che nessuno dei processi si blocchi e che nessun altro processo sia pronto per l'esecuzione. Confrontare questa figura con la 9.16.

9.4 Sommario

Il sistema operativo deve prendere tre tipi di decisioni di scheduling rispetto all'esecuzione dei processi. Lo scheduling a lungo termine determina quando sono ammessi nuovi processi nel sistema, lo scheduling a medio termine è una parte della funzione di trasferimento su disco dei processi, e determina quando un programma va caricato totalmente o parzialmente in memoria in modo da eseguirlo; infine lo scheduling a breve termine determina quale processo Ready sarà eseguito successivamente dal processore. Questo capitolo si concentra sugli argomenti relativi allo scheduling a breve termine.

Una varietà di criteri è usata nella progettazione dello scheduling a breve termine: alcuni di questi criteri sono in relazione al comportamento del sistema, come è percepito dall'utente individuale (orientati all'utente), mentre gli altri vedono l'efficacia totale del sistema andando incontro alle necessità di tutti gli utenti (orientati al sistema). Alcuni criteri sono in relazione specificatamente con misure quantitative delle prestazioni, mentre altri sono più qualitativi. Dal punto di vista dell'utente, il tempo di risposta è generalmente la più importante caratteristica di un sistema, mentre dal punto di vista del sistema, è importante l'utilizzazione del processore, o il throughput.

Per prendere decisioni di scheduling a breve termine, fra tutti i processi pronti, sono stati sviluppati diversi algoritmi:

- **First come first served**: Seleziona il processo che ha il tempo di attesa più lungo.
- **Round robin**: Usa il time-slicing per limitare tutti i processi in fase di esecuzione ad un piccolo periodo d'uso del processore e ruota su tutti i processi Ready.
- **Shortest process next**: Seleziona il processo con il minore tempo di uso del processore previsto e non prerilascia il processo.
- **Shortest remaining time**: Seleziona il processo con il minore tempo di uso del processore previsto; un processo può essere prerilasciato quando un altro processo diventa Ready.
- **Highest response ratio next**: Basa la decisione di scheduling su una stima del tempo di turnaround normalizzato.
- **Feedback**: Crea un insieme di code per lo scheduling, ed alloca i processi in tali code a seconda della loro storia di esecuzione, o su altri criteri.

La scelta di un algoritmo per lo scheduling dipenderà dalle prestazioni previste e dalla complessità dell'implementazione.

9.5 Letture raccomandate

Generalmente tutti i testi sui sistemi operativi trattano l'argomento dello scheduling. Rigorose analisi delle code e varie politiche per lo scheduling sono presentate su [STUC85], [KLEI76] e [CONW67].

CONW67 Conway, R.; Maxwell, W. e Miller, L. *Theory of Scheduling*. Reading, MA: Addison-Wesley, 1967.

KLEI76 Kleinrock, L. *Queuing Systems, Volume II: Computer Applications*. New York: Wiley, 1976.

STUC85 Stuck, B. e Arthurs, E. *A Computer and Communication Network Performance Analysis Primer* Englewood Cliffs, NJ: Prentice Hall, 1985.

9.6 Problemi

9.1 Considerare il seguente insieme di processi

Nome processo	Tempo di arrivo	Tempo di esecuzione
1	0	3
2	1	5
3	3	2
4	9	5
5	12	5

Si faccia la stessa analisi di questo insieme come descritto nella Tabella 9.4 e in Figura 9.5

9.2 Ripetere il Problema 9.1 con il seguente insieme:

Nome processo	Tempo di arrivo	Tempo di esecuzione
A	0	1
B	1	9
C	2	1
D	3	9

9.3 Provare che, fra gli algoritmi di scheduling senza prerilascio, SPN permette di ottenere il minimo tempo di attesa media, supponendo che tutti i job arrivino contemporaneamente.

9.4 Si consideri la seguente sequenza di tempi di burst per un processo: 6, 4, 6, 4, 13, 13, 13, e si supponga che il valore iniziale stimato sia 10. Produrre un grafico simile a quelli di Figura 9.9

9.5 Si consideri la seguente coppia di equazioni come un alternativa all'equazione (9.3):

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

$$X = \min[\text{Ubound}, \max[\text{Lbound}, (\beta S_{n+1})]]$$

Dove Ubound e Lbound sono il limite superiore e quello inferiore sul valore stimato di T precedentemente scelti. Il valore X è usato nell'algoritmo shortest process next invece del valore S_{n+1}. Quali funzioni sviluppano α e β e quali sono gli effetti di loro valori maggiori o minori?

9.6 In un sistema uniprocessore senza prerilascio la coda dei processi Ready contiene 3 job al tempo t immediatamente dopo il completamento di un job. Questi job hanno tempi arrivo t_1, t_2, t_3, e tempi stimati di esecuzione r_1, r_2, r_3, rispettivamente.
La Figura 9.18 mostra la crescita lineare della loro risposta media nel tempo. Si usi quest'esempio per trovare una variante a HRRN, conosciuta come **minimax response ratio** scheduling (minimo dei rapporti massimi di risposta), che minimizza la risposta media massima per un dato gruppo di job, ignorando arrivi successivi (*Suggerimento*: si decida subito quale job schedulare per ultimo).

9.7 Si provi che l'algoritmo di minimax response ratio trattato nel precedente problema minimizza la risposta media massima per un dato gruppo di job (*Suggerimento*: focalizzare l'attenzione sul job che totalizzerà il più grande rapporto di risposta e su tutti i job

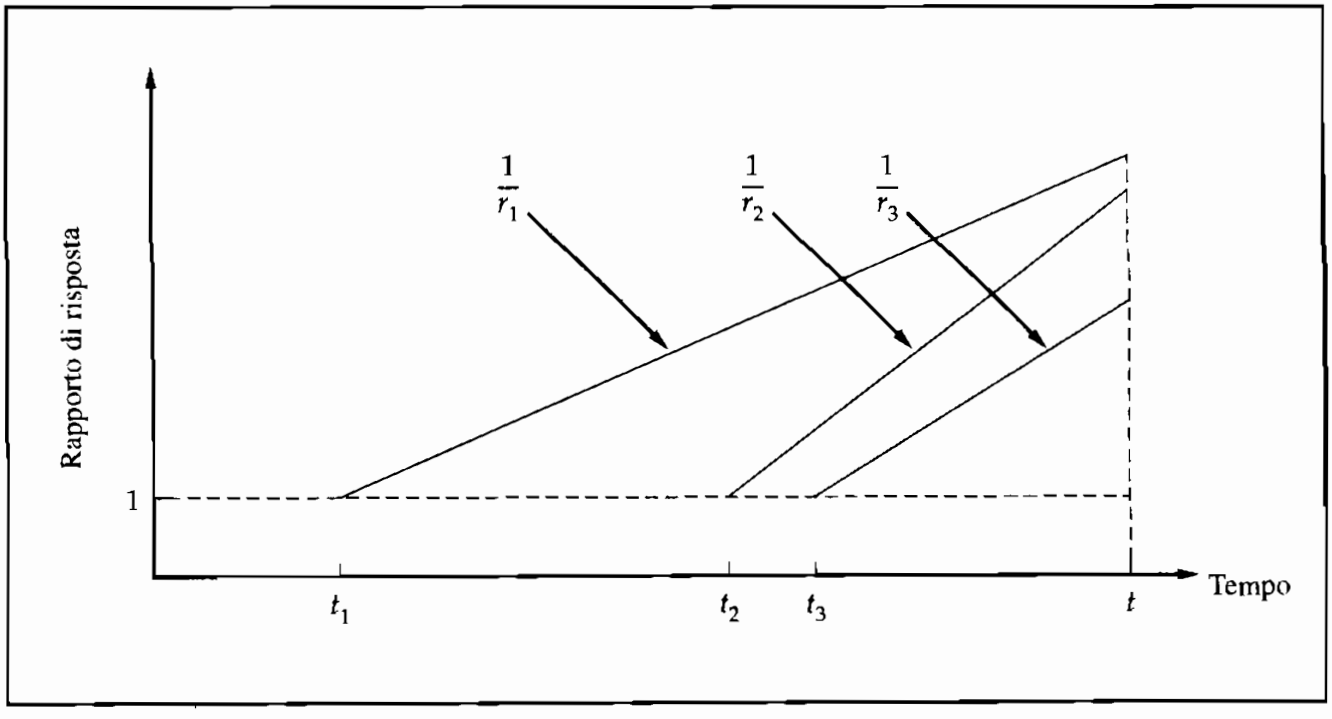

Figura 9.18 *Tempo di risposta in funzione del tempo*

eseguiti prima di esso. Si consideri lo stesso sottoinsieme di job schedulati in un qualsiasi altro ordine, e si osservi il rapporto di risposta del job eseguito per ultimo tra di essi. Si noti che questo sottoinsieme può essere adesso mischiato con altri job presi dall'insieme totale)

9.8 Si definisca il tempo di risposta R come il tempo medio totale che un processo passa aspettando un servizio. Si mostri che per la politica FIFO, $R = S/(1-\rho)$, dove S è il tempo di servizio medio.

9.9 Un processore è multiprogrammato ad una velocità infinita fra tutti i processi presenti nella coda Ready senza overhead. (Questo è un modello ideale di uno scheduling round-robin sui processi Ready usando dei quanti di tempo molto piccoli rispetto al tempo di esecuzione). Dimostrare che con un input di Poisson da una sorgente infinita con tempi di servizio esponenziali, la media del tempo di risposta Rx di un processo con tempo di servizio x è data da $Rx = x/(1-\rho)$. (*Suggerimento*: si riguardino le equazioni delle code nell'Appendice A, quindi si consideri q, la dimensione media delle code nel sistema, all'arrivo del processo preso in considerazione)

9.10 La maggior parte degli scheduler round-robin usano una dimensione fissata del quanto. Si cerchino argomentazioni a favore di un piccolo quanto e viceversa, a favore un grande quanto. Si paragonino e si contrappongano i tipi di sistema e di job sui quali le argomentazioni sono state applicate. Ce ne sono certi per cui *entrambe* sono ragionevoli?

9.11 In un sistema a code i nuovi job devono attendere un po' prima di essere serviti. Mentre un job attende, la sua priorità cresce linearmente nel tempo da 0 a velocità α. Un processo attende fintanto che la sua priorità non arriva alla priorità del processo in servizio; in quel momento, quindi, incomincia a condividere il processore in modo equo con gli altri processi in esecuzione usando lo scheduling round-robin, mentre la sua priorità continua a crescere, ma con velocità minore β. L'algoritmo è detto selfish round robin perché i job in esecuzione cercano (invano) di monopolizzare il processore aumentando la loro priorità in continuazione. Si usi la Figura 9.19 per mostrare che il tempo di risposta medio R_x per un processo al tempo di servizio x è dato da:

$$R_x = \frac{s}{1-\rho} + \frac{x-s}{1-\rho'}$$

Dove

$$\rho = \lambda s \qquad \rho' = \rho\,(1 - \frac{\beta}{\alpha}) \qquad 0 \le \beta \le \alpha$$

Supponendo che il tempo di arrivo e quello di servizio siano distribuiti esponenzialmente con media $1/\lambda$ e s rispettivamente. (*Suggerimento*: Si consideri il sistema totale e i 2 sottosistemi separatamente).

9.12 Un sistema interattivo che usa lo scheduling round-robin e lo swapping cerca di dare una risposta garantita alle richieste irrisorie come segue: dopo aver completato un ciclo round-robin su tutti i processi Ready, il sistema determina quanto tempo concedere ad ogni

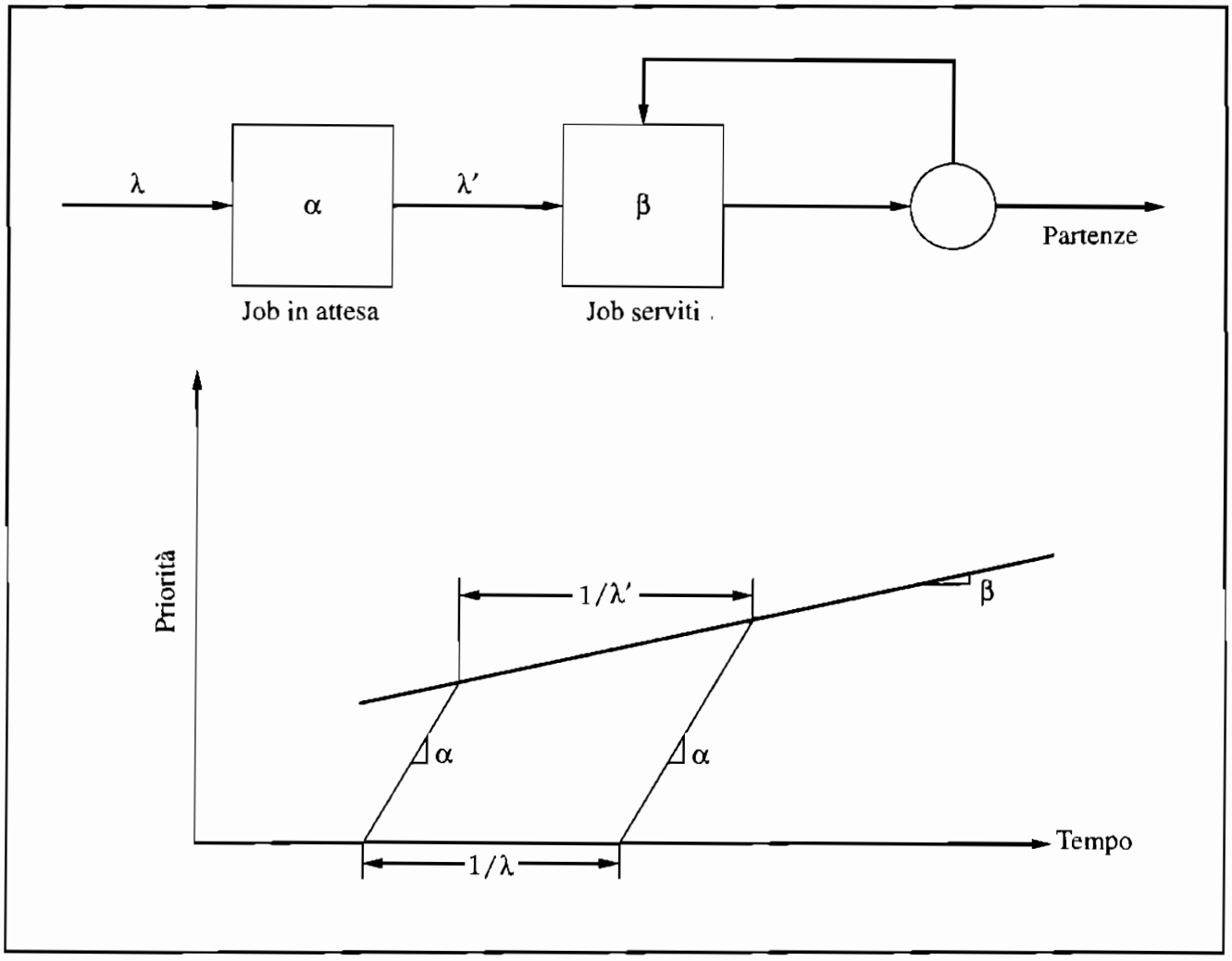

Figura 9.19 *Selfish Round Robin*

processo Ready durante il ciclo successivo, dividendo il tempo di risposta massimo per il numero dei processi che richiedono il servizio. Questa strategia è praticabile?

9.13 Che tipo di processo è generalmente favorito da uno scheduler con code con feedback multilivello, un processo processor-bound o uno I/O bound? Si dia una piccola spiegazione.

9.14 In uno scheduling basato sulla priorità, lo scheduler passa il controllo ad un particolare processo solo se non ci sono processi con una priorità più alta in stato Ready. Si supponga che non vi siano altre informazioni per fare lo scheduling dei processi, e che le priorità dei processi, stabilite al tempo di creazione dei processi, non cambino mai. In un sistema che lavora in tale modo perché è pericoloso utilizzare la soluzione di Dekker per il problema della mutua esclusione? Lo si spieghi dicendo *quale* evento indesiderato potrebbe presentarsi e *come*.

9.15 Cinque job batch, da A fino a E, arrivano praticamente allo stesso tempo all'unità di elaborazione. Essi hanno un tempo di esecuzione stimato di 15, 9, 3, 6 e 12 minuti rispettivamente; le loro priorità (definite esternamente) sono rispettivamente 4, 7, 3, 1 e 6, dove

10 è la priorità massima. Per ognuno dei seguenti algoritmi di scheduling si determini il tempo di turnaround di ciascun processo e il turnaround medio di tutti i job. Non si tenga conto dell'overhead dovuto al cambio di processo e si spieghi come si è arrivati alla risposte. Negli ultimi tre casi, si supponga di eseguire un solo processo per volta e di finire solo quando tutti i job sono completamente processor-bound.

a. Round robin con un quanto che vale 1

b. Scheduling con priorità

c. FCFS (si esegue in ordine 15, 9, 3, 6 e 12)

d. Shortest job first.

Appendice 9A Tempo di risposta

Il tempo di risposta è il tempo che un sistema richiede per reagire ad un certo input. In una transazione interattiva, può essere definito come il tempo che intercorre da quando l'ultimo tasto della tastiera è stato premuto dall'utente, a quando il risultato è visualizzato dal computer sullo schermo. Per diversi tipi di applicazione, serve una definizione leggermente differente; ma in generale è il tempo che il sistema si prende per rispondere alla richiesta di eseguire un compito particolare.

Idealmente, piacerebbe a tutti che il tempo di risposta fosse il più breve possibile per ogni applicazione. Tuttavia spesso un tempo di risposta più breve richiede un costo maggiore: il costo, infatti, dipende da due fattori:

- **Potenza di elaborazione**: Più veloce è il computer, più breve sarà il tempo di risposta; ovviamente, aumentare la potenza del processore significa aumentare i costi.
- **Competizione tra richieste**: Dare tempi di risposta rapidi a certi processi può penalizzarne altri.

Per questi motivi si deve valutare il livello del tempo di risposta rispetto al costo di ottenere quel tempo di risposta.

La Tabella 9.6, basata su [MART88], elenca sei intervalli generali di tempi di risposta. Si incontrano difficoltà di progettazione quando è richiesto un tempo di risposta inferiore ad un secondo: una tale richiesta è tipica di un sistema che controlla o interagisce con un'attività esterna, come in una catena di montaggio: questa richiesta è immediata. Quando consideriamo un'interazione uomo-macchina, come in un'applicazione di inserimento dati, siamo in un intervallo di risposta di tipo conversazionale. In questo caso c'è ancora una richiesta per un tempo di risposta breve, ma una durata accettabile potrebbe essere difficile da valutare.

Il tempo di risposta rapido è la base per avere produttività nelle applicazioni interattive, cosa confermata da più studi [SHNE84; THAD81; GUYN88]. Questi studi dimostrano che se un computer ed un utente interagiscono di pari passo, in modo che nessuno dei due debba attendere l'altro, la produttività cresce significativamente, il costo del lavoro fatto al computer perciò si

Tabella 9.6 *Livelli del tempo di risposta*

Maggiore di 15 secondi Impossibile avere interazioni con domanda e risposta. Per certi tipi di applicazioni, gli utenti potrebbero essere contenti di sedersi davanti ad un terminale ed aspettare più di 15 secondi per avere una risposta ad una loro semplice domanda. Tuttavia per persone molto impegnate un'attesa di 15 secondi non è tollerabile. Se vi è un tale ritardo, il sistema dovrebbe essere progettato in modo da permettere all'utente di passare ad altre attività e rimandare la risposta ad un altro momento.
Maggiore di 4 secondi Questo tempo è troppo lungo per una conversazione che richiede all'operatore di mantenere delle informazioni in memoria a breve termine (quella dell'operatore, non del computer!). Tali ritardi possono limitare fortemente attività di soluzione di problemi, e sono frustranti nell'inserimento dei dati. Tuttavia dopo una importante chiusura, ritardi da 4 a 15 secondi possono essere tollerati.
Da 2 a 4 secondi Un ritardo sopra i 2 secondi può essere proibitivo per operazioni al terminale che richiedono un alto livello di concentrazione. Un'attesa da 2 a 4 secondi al terminale può sembrare estremamente lunga quando un utente è personalmente teso a completare il più presto possibile quello che sta facendo. Di nuovo un ritardo in questo livello può essere accettabile dopo che interviene una più piccola chiusura.
Meno di 2 secondi Quando l'utente al terminale deve ricordare molte risposte, il tempo di risposta deve essere breve. Più dettagli si devono ricordare, maggiore è il bisogno di un tempo di risposta inferiore ai 2 secondi. Per attività al terminale complicate, 2 secondi rappresentano un importante tempo di risposta limite.
Tempi di risposta sotto il secondo Certi lavori che richiedono un'attenzione intensa, specialmente con applicazioni grafiche, richiedono un tempo di risposta molto breve per mantenere l'interesse dell'utente e l'attenzione per un lungo periodo.
Tempi di risposta in decimi di secondo In risposta alla pressione di un tasto della tastiera o del mouse, vedere il carattere sullo schermo o selezionare un oggetto sullo schermo deve essere quasi istantaneo - meno di 0,1 secondi dopo l'azione. Un'interazione con il mouse richiede una risposta estremamente veloce, se il progettista vuole evitare di usare una sintassi aliena (con comandi, punteggiatura mnemonica).

abbassa e la qualità tende ad aumentare. Era un dato di fatto largamente accettato che una risposta relativamente lenta, fino a 2 secondi, fosse accettabile per molte applicazioni interattive poiché una persona doveva pensare al task successivo. Adesso, tuttavia, sembra che la produttività aumenti se si ottengono veloci tempi di risposta.

I risultati riportati sul tempo di risposta sono basati su un'analisi delle transazioni in linea. Una transazione si compone di un comando utente dal terminale e della risposta del sistema, e rappresenta l'unita di lavoro fondamentale per gli utenti in linea. Può essere divisa in due sequenze temporali:

- **Tempo di risposta dell'utente**: È il tempo che intercorre da quando l'utente riceve una risposta ad un comando a quando invia un altro comando. Si dice anche *think time* (tempo per pensare).
- **Tempo di risposta del sistema**: È il tempo che intercorre da quando l'utente invia un comando a quando la risposta è visualizzata sul terminale.

Come esempio sull'effetto di una riduzione del tempo di risposta, la Figura 9.20 mostra i risultati di uno studio effettuato da ingegneri usando un programma di grafica CAD per il progetto di chip di circuiti integrati e schede [SMIT83]: ogni transazione consiste di un comando inviato da un ingegnere, che modifica l'immagine a schermo. Il risultato mostra che la frequenza delle transazioni aumenta se il tempo di risposta scende, ed aumenta drasticamente se il tempo di risposta scende sotto 1 secondo. Quello che sta succedendo è che, quando il tempo di risposta del sistema scende, così fa anche il tempo di risposta dell'utente: tutto questo è in relazione con gli effetti della memoria a breve termine e con l'attenzione.

Un'altra area dove il tempo di risposta è diventato critico è l'uso del World Wide Web, sia su Internet sia in un'intranet. Il tempo che impiega una normale pagina web ad essere visualizzata può cambiare notevolmente. Il tempo di risposta può essere stimato in base al livello di coinvolgimento dell'utente; in particolare alcuni sistemi con un tempo di risposta molto veloce

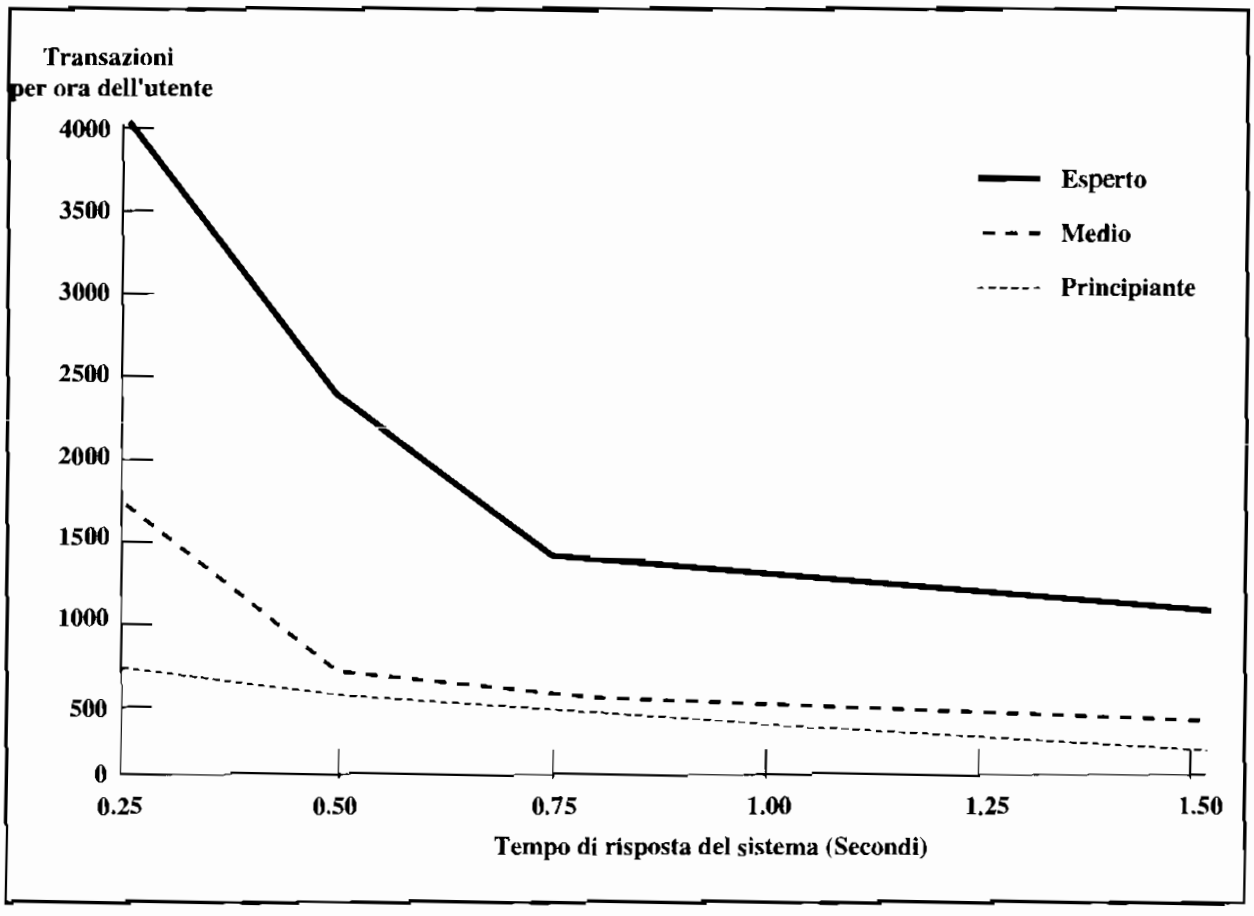

Figura 9.20 *Tempi di risposta per una funzione grafica*

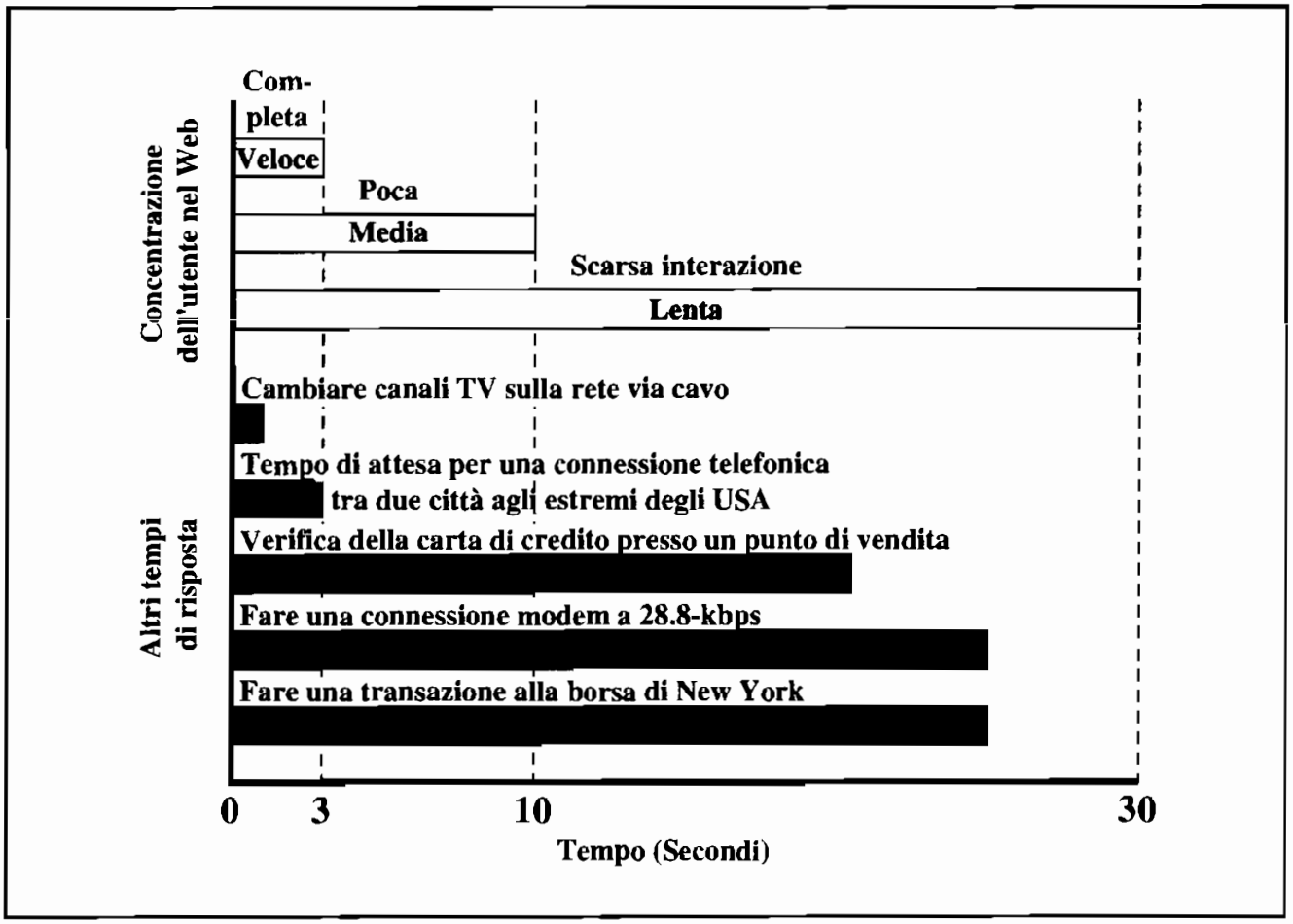

Figura 9.21 *Tempi di risposta richiesti [SEVC96]*

tendono a richiedere maggiore attenzione da parte dell'utente. Come ci mostra la Figura 9.21, sistemi Web con un tempo di risposta di 3 secondi o migliore mantengono un alto livello di attenzione dell'utente. Con un tempo di risposta tra i 3 e i 10 secondi si perde una parte della concentrazione dell'utente, e i tempi di risposta superiori ai 10 secondi scoraggiano l'utente, che può semplicemente terminare la sessione.