Scheduling della CPU

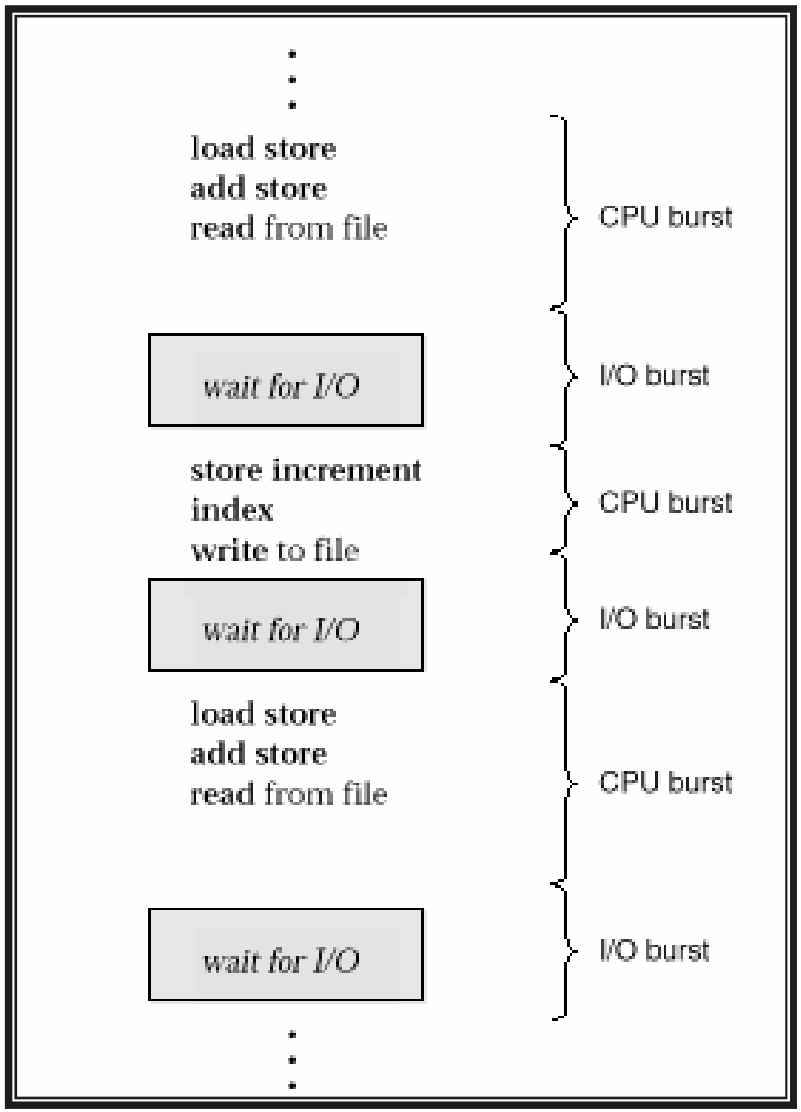
- Concetti fondamentali
- Criteri di scheduling
- Algoritmi di scheduling

Concetti fondamentali

- L'obiettivo della multiprogrammazione è di avere processi sempre in esecuzione al fine di massimizzare l'utilizzo della CPU.
- L'idea è semplice: più processi sono tenuti in memoria e se uno è in attesa di un evento (es. richiesta di I/O), il sistema operativo gli sottrae la CPU in favore di un altro processo.
- Lo scheduling è una funzione fondamentale dei sistemi operativi e viene applicato a quasi tutte le risorse di un calcolatore.

Concetti fondamentali

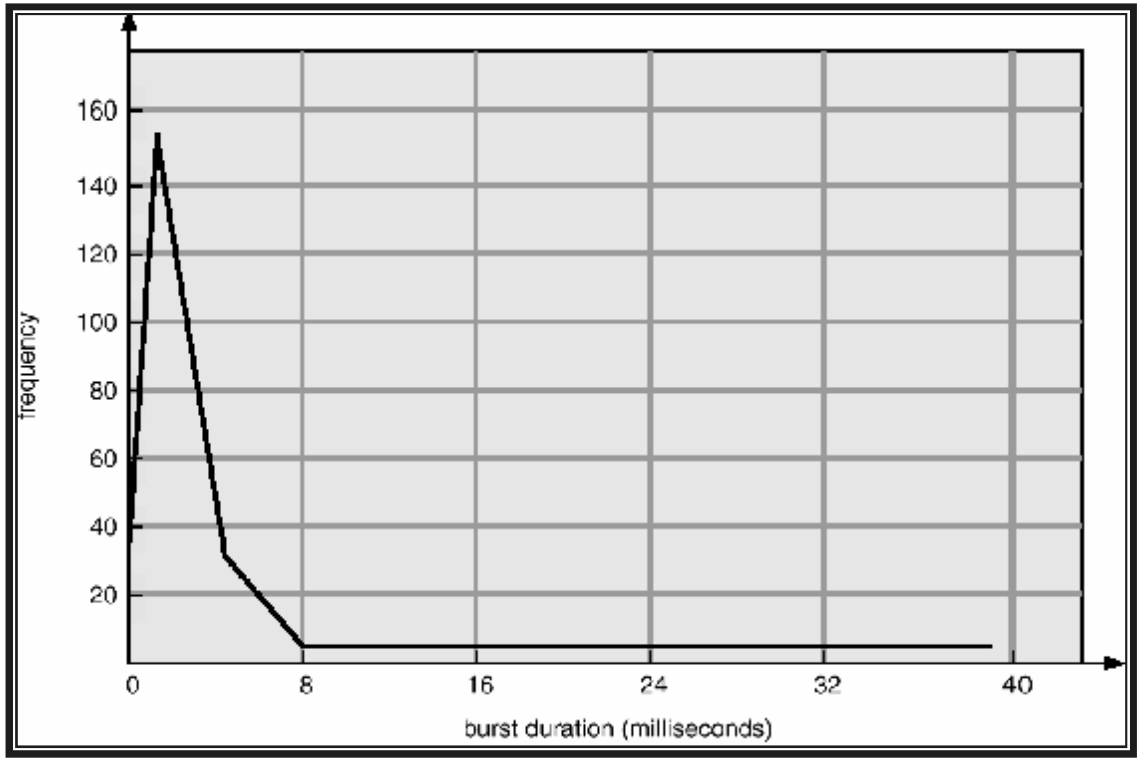
- Il massimo impiego della CPU è ottenuto con la multiprogrammazione.
- *Ciclo di CPU–I/O burst* — L'esecuzione di un processo consiste di *cicli* di esecuzione da parte della CPU ed attese di I/O.
- In generale l'esecuzione inizia con una sequenza di operazioni di elaborazioni (*CPU burst*), seguita da una sequenza di operazioni di I/O (*I/O burst*), in maniera ciclica.



Distribuzione dei burst di CPU

Concetti fondamentali

- Di norma, la curva di frequenza della durata delle sequenze di operazioni di CPU è di tipo esponenziale, con molte sequenze brevi di operazioni della CPU e poche sequenze lunghe.
- Un programma con **prevalenza di I/O** (*I/O bound*) produce molte sequenze di operazioni della CPU brevi.
- Uno con **prevalenza d'elaborazione** (*CPU bound*) produce poche sequenze di operazioni della CPU molto lunghe.



Durata dei burst di CPU

Scheduler della CPU

- Se la CPU passa nello stato d'inattività, il sistema operativo sceglie, fra i processi in memoria pronti ad essere eseguiti, un processo cui allocare la CPU.
- Lo scheduler della CPU deve prendere una decisione quando un processo…
  1. …passa da stato **running** a stato **waiting** (richiesta di I/O o attesa terminazione di un processo figlio);
  2. …passa da stato **running** a stato **ready** (interrupt);
  3. …passa da stato **waiting** a stato **ready** (completamento di un I/O);
  4. …termina.
- Se lo scheduling interviene solo nei casi 1 e 4, si dice che lo schema di scheduling è **senza prelazione** (*non–preemptive)*.
- Altrimenti si ha uno schema **con prelazione** (*preemptive).*

Dispatcher

- Il modulo allocatore (*dispatcher*) passa il controllo della CPU al processo selezionato dallo scheduler a breve termine; il dispatcher effettua:
  - Cambio di contesto
  - Passaggio a modo utente
  - Salto alla posizione corretta del programma utente per riavviarne l'esecuzione
- *Latenza di dispatch*: è il tempo impiegato dal dispatcher per sospendere un processo e avviare una nuova esecuzione.

Criteri di scheduling

- Esistono differenti algoritmi di scheduling, che possono essere valutati secondo i seguenti parametri:
  - **Utilizzo di CPU**: la CPU deve essere più attiva possibile.
  - **Produttività (***throughput):* numero di processi che completano la loro esecuzione nell'unità di tempo.
  - **Tempo di completamento** (*turnaround time)*: tempo impiegato per l'esecuzione di un determinato processo.
  - **Tempo di attesa**: tempo speso dal processo in attesa nella coda dei processi pronti.
  - **Tempo di risposta**: tempo tra la sottomissione di una richiesta e la prima risposta prodotta. In un sistema interattivo tale tempo può essere influenzato dalla velocità del dispositivo di output.

Criteri di ottimizzazione

- In generale si vuole:
  - Massimo utilizzo di CPU
  - Massima produttività
  - Minimo tempo di completamento
  - Minimo tempo di attesa
  - Minimo tempo di risposta
- In generale si ottimizzano i valori medi, sebbene in alcuni casi sia utile ottimizzare valori minimi e massimi (tempo di risposta in sistemi interattivi)
- In alcuni casi è importante ridurre la varianza di tali parametri *(prevedibilità)*.

Scheduling in ordine di arrivo (FCFS)

- Il più semplice degli algoritmi di scheduling è quello di **scheduling in ordine di arrivo** (*first come first served-FCFS): l*a CPU viene assegnata al processo che arriva per primo.
- L'implementazione avviene mediante una coda FIFO:
  - quando un processo entra nella coda dei processi pronti, si collega il suo PCB all'ultimo elemento della coda.
  - quando la CPU è libera, viene assegna al processo in testa alla coda dei processi pronti.
- I tempi di attesa sono spesso abbastanza lunghi.

Scheduling in ordine di arrivo (FCFS)

ESEMPIO 1

Processo	Tempo di burst
P_1	24
P_2	3
P_3	3

- I processi arrivano al sistema nell'ordine: P_1 , P_2 , P_3.
  Il **diagramma di Gantt** per lo scheduling **FCFS** è:

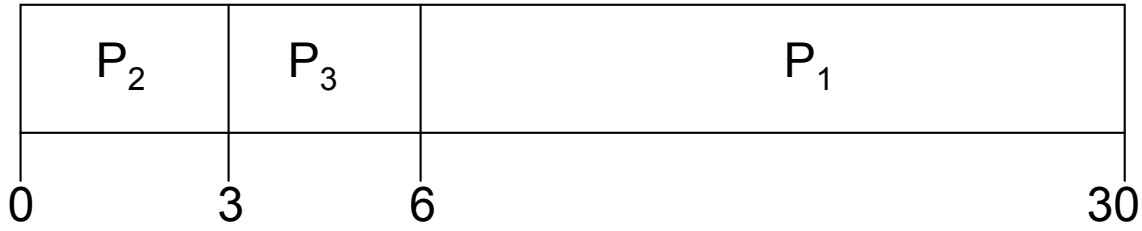
- Tempi di attesa: P_1 = 0; P_2 = 24; P_3 = 27.
- Tempo medio di attesa = (0 + 24 + 27)/3 = 17.

Scheduling FCFS

- Se l'ordine di arrivo è

$$P_2 , P_3 , P_1,$$

il diagramma di Gantt risulta…



- Tempi di attesa: $P_1 = 6$; $P_2 = 0$; $P_3 = 3$.
- Tempo medio di attesa = (6 + 0 + 3)/3 = 3.
- In questo caso, non si verifica l'*effetto convoglio*, per cui processi di breve durata devono attendere che un processo molto lungo liberi la CPU.

Scheduling per brevità (SJF)

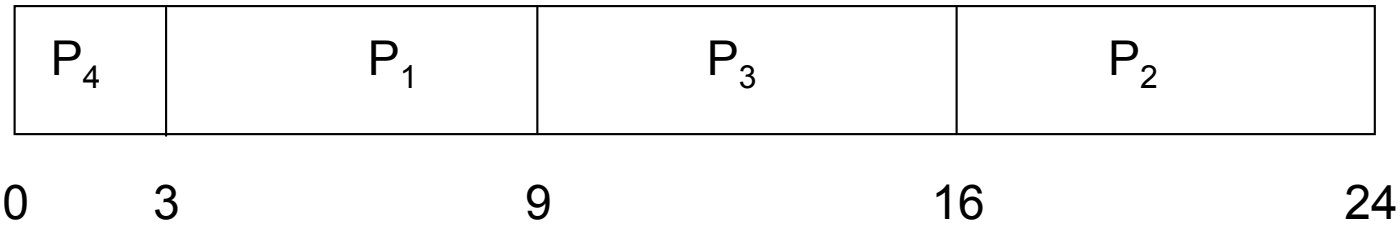
- Si associa a ciascun processo la lunghezza della successiva sequenza di operazioni della CPU. Si opera lo scheduling in base alla brevità di tale sequenza.
- Se due processi hanno le successive sequenze di burst della stessa lunghezza, si applica a questi l'algoritmo FCFS.
- Tale algoritmo prende in considerazione le sequenze di burst e non la lunghezza totale del processo.
- **SJF** è *ottimale* — rende minimo il tempo medio di attesa per un dato insieme di processi.

Scheduling SJF preemptive

ESEMPIO 2

Processo	Tempo di burst
P_1	6
P_2	8
P_3	7
P_4	3

- SJF :



- Tempo medio di attesa = (3 + 16 + 9 +0)/4 = 7.
- Con FCFS il tempo medio è 10.25

Predizione del CPU burst successivo

- Spostando un processo breve prima di un processo lungo, il tempo d'attesa del processo breve diminuisce più di quanto aumenti quello del processo lungo ➔ **Tempo medio ottimale**.
- L'algoritmo SJF non può essere utilizzato per lo scheduling a breve termine, poiché non esiste alcun modo per conoscere la lunghezza del successivo CPU burst.
- Se non è possibile conoscerlo, si può provare a *predirlo*, a partire da informazioni sui valori precedenti.
- Calcolando un valore approssimato della lunghezza, si può scegliere il processo con la più breve fra le lunghezze previste.

Predizione del CPU burst successivo

- Può essere stimato utilizzando la lunghezza dei burst di CPU precedenti, impiegando la **media esponenziale**.

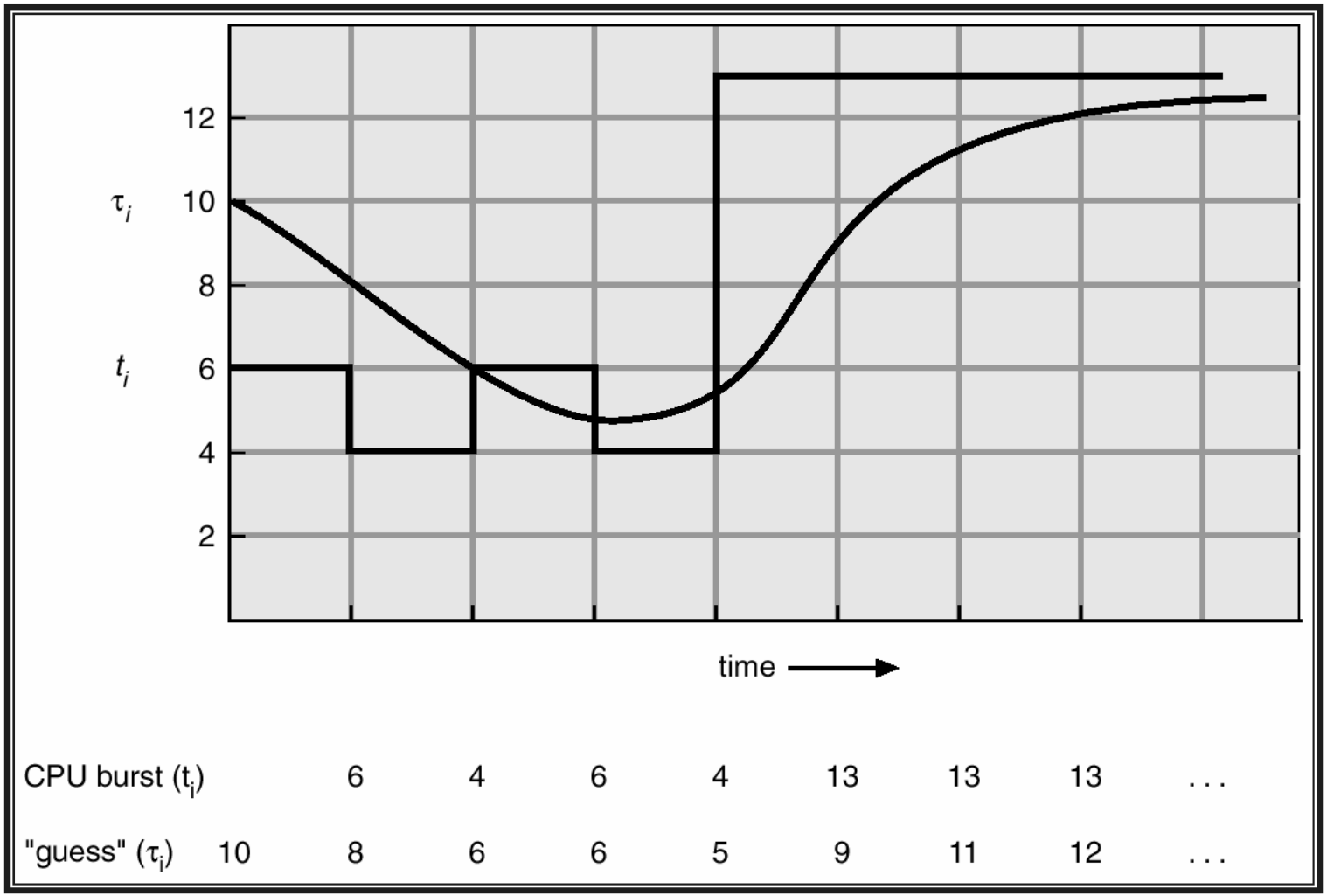
1. $t_n = \text{lunghezza dell'}n-\text{esimo CPU burst}$
2. $\tau_{n+1} = \text{valore stimato del prossimo CPU burst}$
3. $\alpha, 0 \le \alpha \le 1$
4. Si definisca :

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

- α =0
  - $\tau_{n+1} = \tau_n$ ➔ La storia recente non è presa in considerazione.
- α =1
  - $\tau_{n+1} = t_n$ ➔Viene considerato soltanto l'ultimo CPU burst.

Esempi di media esponenziale

- Esempio con α =1/2, τ_0 arbitrario



Esempi di media esponenziale

- Espandendo la formula si ottiene:

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

- Poiché α e $(1-\alpha)$ sono entrambi minori o uguali ad 1, ciascun termine ha minor peso del suo predecessore.

Scheduling SJF con prelazione

- Quando nella coda dei processi pronti arriva un nuovo processo mentre un altro processo è in esecuzione, si possono avere due possibilità:
  - **non–preemptive** — una volta che la CPU è stata allocata al processo, non gli può essere prelazionata fino al termine del CPU burst corrente;
  - **preemptive** — se arriva un nuovo processo con burst di CPU minore del tempo rimasto per il processo corrente, il nuovo processo prelaziona la CPU. Questo schema è noto come ***Shortest–Remaining–Time–First*** (**SRTF**).

Scheduling SJF con prelazione

ESEMPIO 3

Processo	Tempo di arrivo	Tempo di burst
P_1	0.0	8
P_2	1.0	4
P_3	2.0	9
P_4	3.0	5

- SJF con prelazione:

- Tempo medio di attesa = ((10-1) + (1-1) + (17-2) + (5-3))/4 = 26/4 = 6,5.
- Senza prelazione il tempo medio è 8,75

Scheduling per priorità

- Un valore di priorità (intero) è associato a ciascun processo e la CPU viene allocata al processo con la priorità più alta.
- **SJF** è uno scheduling a priorità dove la priorità è rappresentata dal successivo tempo di burst.
- Talvolta a numeri alti sono associate priorità alte, talvolta l'opposto: nel seguito numeri bassi rappresentano priorità alte.

Scheduling per priorità

ESEMPIO 4

Processo	Tempo di burst	Priorità
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Scheduling con priorità:

- Tempo medio di attesa = 8,2.

Scheduling per priorità

- Lo scheduling per priorità può essere:
  - **preemptive**: se il processo in esecuzione ha una priorità minore rispetto ad uno arrivato nella ready queue, la CPU gli viene sottratta.
  - **non–preemptive**: se arriva un processo con priorità più alta dii quelli presenti nella ready queue, viene messo in testa.
- Problema ≡ **Starvation** (*blocco indefinito*) — i processi a bassa priorità potrebbero non venir mai eseguiti.
- Soluzione ≡ **Aging** (*invecchiamento*) — aumento graduale della priorità dei processi che si trovano in attesa nel sistema da lungo tempo.

Scheduling Round Robin (RR)

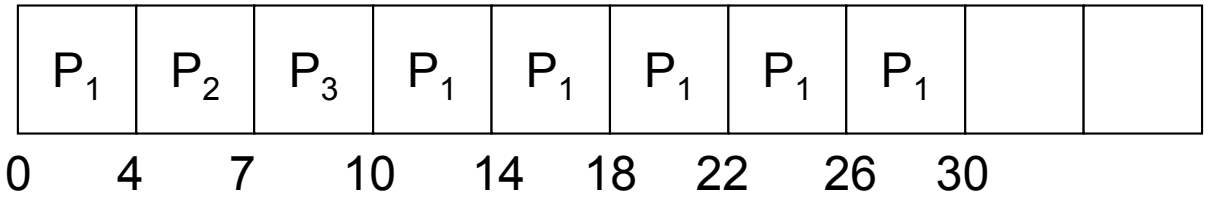
- L'**algoritmo di scheduling circolare** (*Round Robin-RR)* è stato progettato appositamente per i sistemi a partizione del tempo: è simile a FCFS, ma ha in più la capacità di prelazione.
- A ciascun processo viene allocata una piccola unità di tempo di CPU (*quanto di tempo*), generalmente 10–100ms. Dopo un quanto di tempo, il processo è forzato a rilasciare la CPU e accodato alla ready queue.
- Per realizzare lo scheduling RR si gestisce la coda dei processi pronti come una una coda FIFO; il primo processo è scelto per essere mandato in esecuzione.
- Se il tempo di burst è maggiore del quanto di tempo, il temporizzatore invia un'interruzione al sistema operativo che esegue il cambio di contesto e accoda nuovamente il processo.

Scheduling RR con quanto = 4

ESEMPIO 5

Processo	Tempo di burst
P_1	24
P_2	3
P_3	3

- Il diagramma di Gantt è:

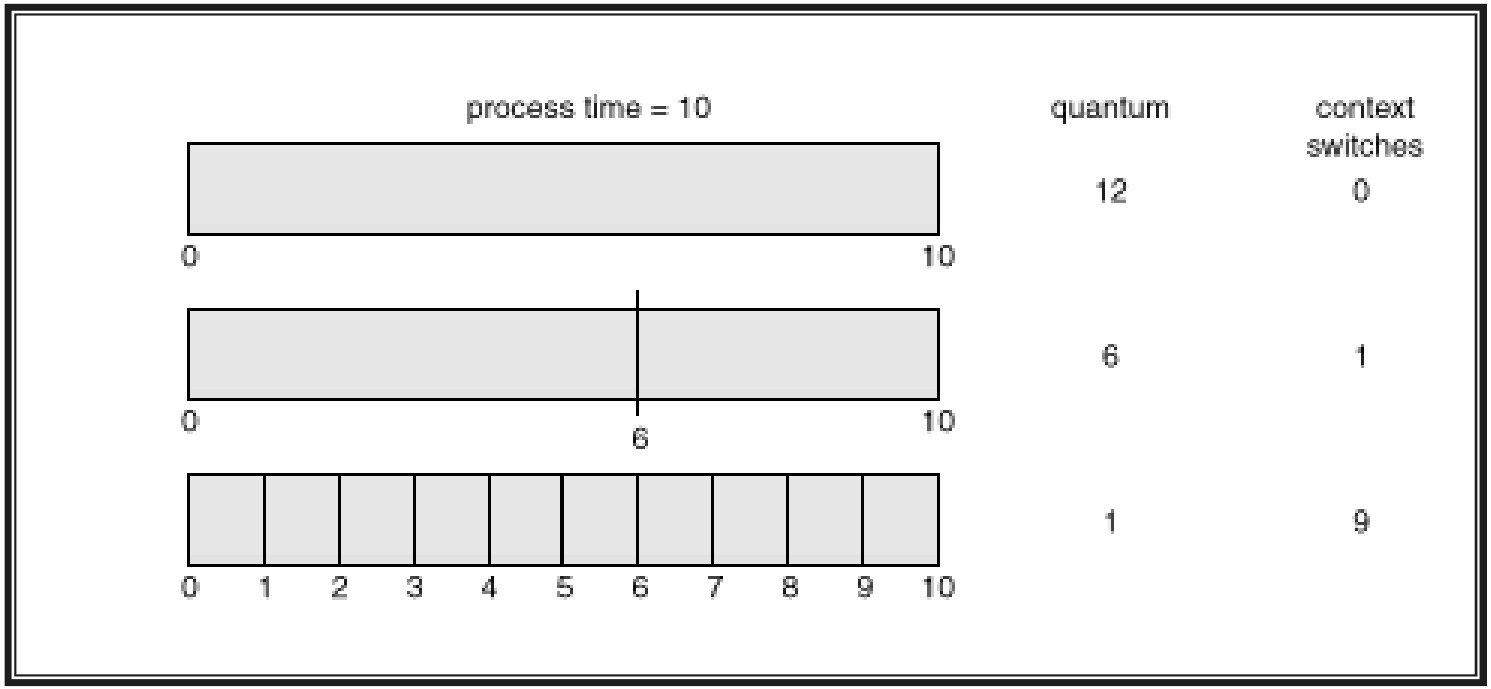


- Tempo medio di attesa: 17/3= 5,66
- In genere si ha un tempo medio di completamento maggiore rispetto a **SJF**, tuttavia si ha una miglior tempo di risposta.

Scheduling Round Robin (RR)

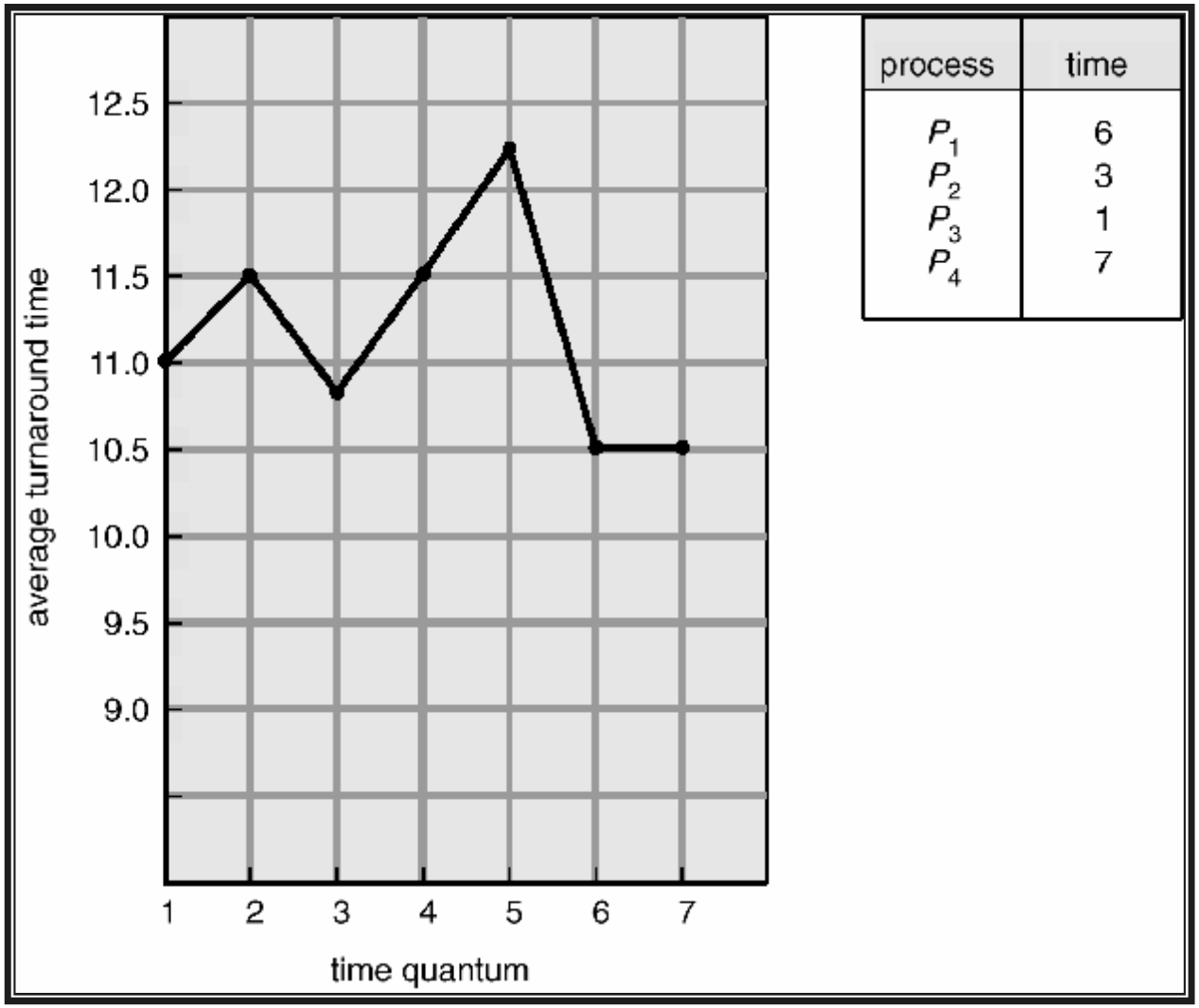
- Se ci sono n processi nella ready queue ed il quanto di tempo è q, ciascun processo occupa $1/n$ del tempo di CPU in frazioni di, al più, q unità di tempo.
- Nessun processo attende per più di $(n-1)\times q$ unità di tempo: ad esempio, se ci sono 5 processi e il quanto di tempo è 20ms, un processo usa 20ms ogni 100ms.
- Se q è molto piccolo ciascun utente ha l'impressione che ciascuno degli n processi in esecuzione venga eseguito da una CPU di velocità $1/n$ della velocità della CPU reale.
- Prestazioni:
  - q grande $\Rightarrow$ FCFS
  - q piccolo $\Rightarrow$ q deve essere grande rispetto al tempo di context switch, altrimenti l'overhead è troppo alto.

Quanto di tempo VS context switch



Un quanto di tempo minore incrementa il numero di cambi di contesto

Quanto di tempo e tempo di turnaround



Variazione del tempo di turnaround in funzione della lunghezza del quanto di tempo

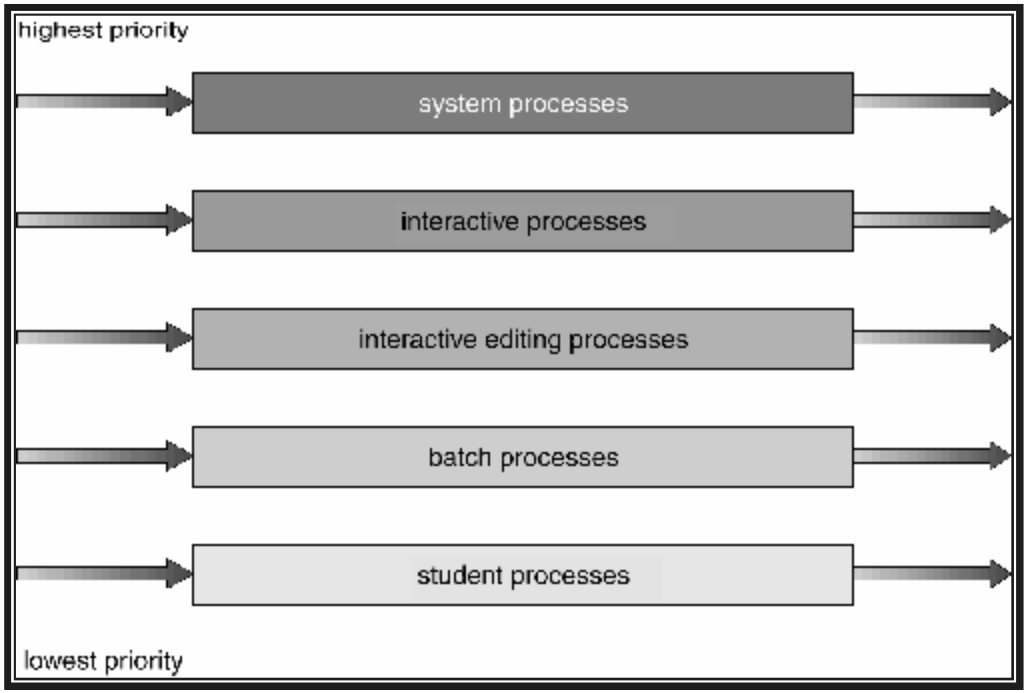
Empiricamente: il quanto di tempo deve essere ≥ 80% dei CPU burst.

Scheduling con code multiple

- Supponiamo di poter classificare i processi in due gruppi diversi e dividiamo la ready queue in code separate:
  - **foreground** (interattiva)
  - **background** (batch)
- Ciascuna coda ha il suo proprio algoritmo di scheduling:
  - foreground – **RR**
  - background – **FCFS**
- È necessario uno scheduling tra code. Ad esempio:
  - *Scheduling a priorità fissa*: serve tutti i processi in foreground poi quelli in background. Rischio di starvation.
  - *Time slice*: ciascuna coda occupa un certo tempo di CPU che suddivide fra i propri processi. Ad esempio:
    - ✓ 80% per foreground in **RR**
    - ✓ 20% per background in **FCFS**

Scheduling a code multiple

ESEMPIO 6



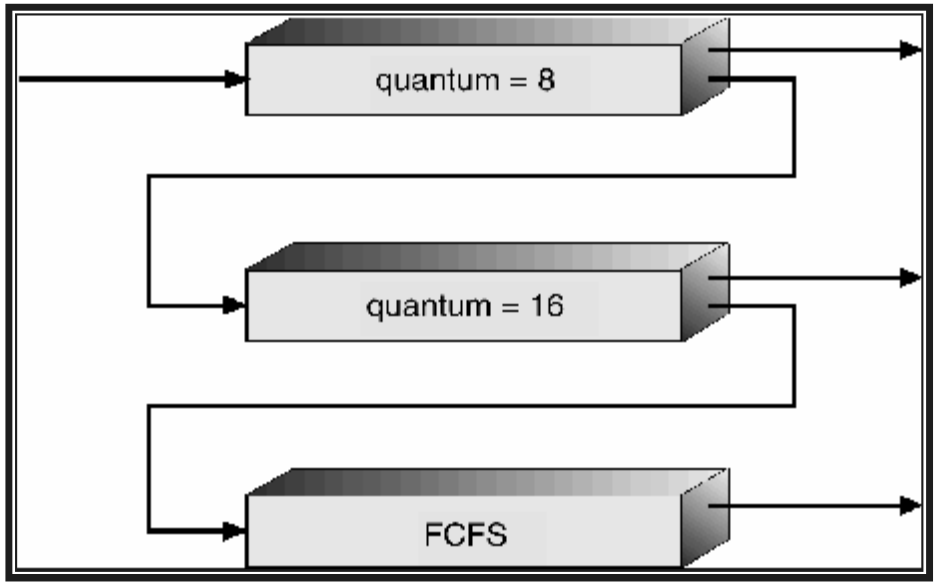
- Classificazione dei processi:
  - di sistema,
  - interattivi,
  - interattivi di *editing,*
  - in sottofondo,
  - degli studenti.
- Ogni coda ha priorità assoluta sulle code a priorità più bassa; nessun processo della coda dei processi in sottofondo può andare in esecuzione finché le code dei processi di sistema, interattivi e di editing non sono vuote.
- Se un processo di editing entrasse nella ready queue mentre è in esecuzione un processo in sottofondo, si avrebbe la prelazione su quest'ultimo (Solaris 2).

Code multiple con retroazione

- In un algoritmo di scheduling a code multiple, un processo che entra in una coda, non può spostarsi in un'altra coda.
- Lo **scheduling a code multiple con retroazione** (*multilevel feedback queue)* permette ai processi di spostarsi fra le varie code.
- Se un processo usa troppo tempo di CPU, viene spostato in una coda a priorità più bassa.
- Questo schema mantiene i processi interattivi e con prevalenza di I/O nelle code a priorità più alta; analogamente si può spostare in una coda a priorità più alta un processo che attende troppo (*aging*).

Esempio di code multiple con retroazione

- Tre code: Q_0 – quanto di tempo di 8ms; Q_1 – quanto di tempo di 16ms; Q_2 – **FCFS**.
- Scheduling:
  - Un nuovo job viene immesso nella coda Q_0 che è servita **RR**. Quando prende possesso della CPU il job riceve 8ms. Se non termina, viene spostato nella coda Q_1.
  - Nella coda Q_1 il job è ancora servito **RR** e riceve ulteriori 16ms. Se ancora non ha terminato, viene mosso nella coda Q_2.



Code multiple con retroazione

- Lo scheduler con code multiple con feedback è caratterizzato dai seguenti parametri:
  - Numero di code
  - Algoritmi di scheduling per ciascuna coda
  - Metodo impiegato per determinare quando spostare un processo in una coda a priorità maggiore
  - Metodo impiegato per determinare quando spostare un processo in una coda a priorità minore
  - Metodo impiegato per determinare in quale coda deve essere posto un processo quando richiede un servizio
- Lo scheduling a code multiple con retroazione è il più generale criterio di scheduling della CPU e la scelta di tutti i parametri può essere un compito complesso.

Sistemi multiprocessori

- Fin qui si sono trattati i problemi di scheduling su singola CPU; se sono disponibili più unità di elaborazione, il problema diventa proporzionalmente più complesso.
- Se sono disponibili più unità di elaborazione identiche, e non sono presenti unità di I/O collegate a un bus privato di una delle CPU, si usa un'unica ready queue.
  - Ciascuna unità di elaborazione esamina la ready queue
  - Una CPU ha in carico lo scheduling di tutte le altre.
- In alcuni sistemi una CPU ha il compito di effettuare non solo lo scheduling, ma anche l'elaborazione dell'I/O e le altre attività di sistema (*multielaborazione asimmetrica)*

Sistemi di elaborazione in tempo reale

- Con il termine **tempo reale stretto** (*hard real time*) si intendono quei sistemi in grado di completare un'operazione critica in un tempo definito.
- **Prenotazione delle risorse**: un processo è accompagnato da una dichiarazione di tempo entro cui completare l'operazione; se è possibile lo scheduler accetta, altrimenti rifiuta la richiesta.
- Lo scheduler deve sapere quanto dura ciascuna operazione; questa richiesta non può essere garantita nei sistemi con memoria virtuale o con memoria secondaria.
- E' chiaro che siffatti sistemi non possono offrire tutte le funzionalità dei moderni sistemi operativi.

Sistemi di elaborazione in tempo reale

- Nelle elaborazioni in **tempo reale debole** (*soft real time*) ci si limita a richiedere che i processi critici abbiano una priorità maggiore dei processi ordinari.
- Il criterio di assegnazione delle risorse risulta **iniquo**, che ritarda l'esecuzione di alcuni processi e può provocare situazioni di attesa indefinita.
- Tali sistemi sono d'uso generale e capaci di offrire, oltre alle funzioni tradizionali, un ambiente per l'esecuzione di applicazioni multimediali e per la grafica interattiva ad alte prestazioni.
- Il sistema deve disporre di uno scheduler per priorità, in cui la priorità dei processi in tempo reale non diminuisce col passare del tempo; l'allocatore deve avere una bassa latenza.
- Per mantenere bassa la latenza, le chiamate di sistema devono essere soggette a prelazione (punti di prelazione o nucleo con possibilità di prelazione).