Capitolo 6: CPU scheduling

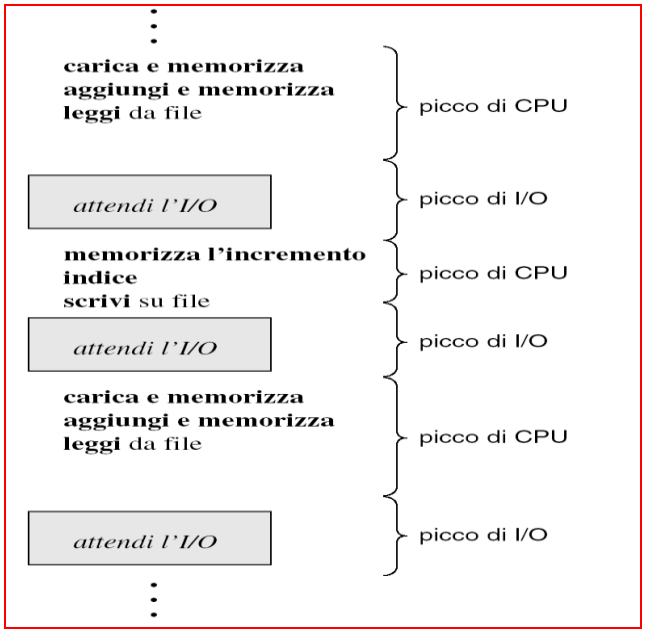
- Concetti di base.
- Criteri di schedulazione.
- Gli algoritmi di schedulazione.
- Schedulazione per sistemi multiprocessore.
- Schedulazione per sistemi in tempo reale.
- Schedulazione dei thread.

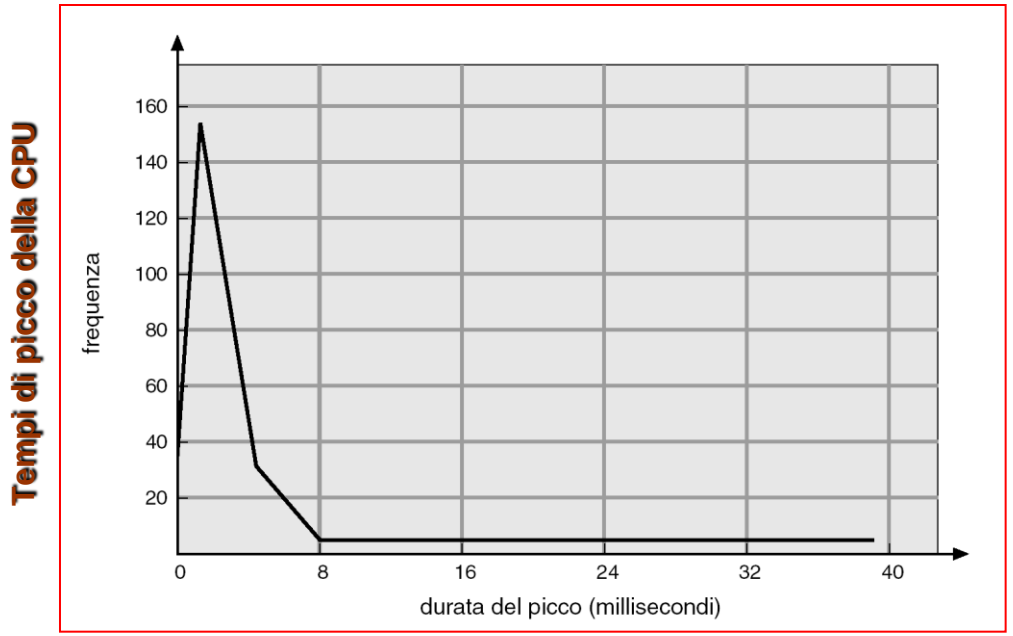
Concetti di base

- La multiprogrammazione cerca di massimizzare l'utilizzo della CPU.
- Vengono mantenuti in memoria molti processi in contemporanea così che quando uno di essi deve attendere (tipicamente per una operazione di I/O), la CPU viene riasegnata dall'O.S. ad un altro processo.
- Quasi tutte le risorse di una macchina vengono schedulate ed in primis questo accade alla CPU.
- Si può affermare che sostanzialmente l'esecuzione di un processo consiste in un **ciclo** di esecuzione nella CPU ed in un'attesa di I/O.
  - Sostanzialmente i processi si alternano tra cicli di picco di CPU e cicli di picco di I/O.
  - Un ciclo di picco della CPU si chiude con una richiesta di terminazione al sistema.

Picco di CPU

- Le durate dei picchi della CPU variano in base all'architettura ed al processo, ma hanno un andamento comune.
- La curva ha un andamento (iper)esponenziale con un elevato numero di picchi brevi e pochi picchi lunghi.
  - Un programma I/O-bound ha molti picchi brevi di CPU.
  - Un programma CPU-bound ha pochi picchi di lunga durata.





Schedulatore della CPU

- Il sistema operativo deve scegliere fra i processi in memoria che sono pronti per l'esecuzione ed assegnare la CPU ad uno di essi.
- Le decisioni sulla schedulazione della CPU possono avvenire nelle seguenti quattro circostanze:
  1. Quando un processo passa dallo stato di esecuzione alla coda di wait (I/O o attesa per la terminazione di un processo figlio).
  2. Quando un processo passa dallo stato di esecuzione alla coda di ready (interrupt).
  3. Quando un processo passa dalla coda di wait alla coda di ready (termine di una operazione di I/O).
  4. Quando un processo termina.
- La schedulazione ai punti 1 e 4 è detta *nonpreemptive* (senza sospensione dell'esecuzione).
  - Deve essere selezionato per l'esecuzione un nuovo processo dalla coda di ready.
- Tutte le altre schedulazioni sono dette *preemptive* (con sospensione dell'esecuzione).

Schedulazione preemptive

- È più sofisticata e richiede meccanismi di gestione più complessi.
- Comporta un costo legato all'accesso a dati condivisi. Ipotizziamo che:
  - un processo venga terminato a seguito di preemption mentre sta modificando dei dati a beneficio di un secondo processo che attende la sua terminazione.
    - Occorre impostare meccanismi di sincronizzazione degli accessi a dati se si vuole evitare che essi risultino inconsistenti.
- Influenza anche il progetto del kernel.
  - Durante l'esecuzione di una chiamata di sistema il kernel potrebbe essere occupato da una attività per conto di un processo che può comportare la modifica di dati rilevanti.
  - Se il processo viene interrotto e il kernel deve leggere i dati, ne potrebbero derivare delle gravi inconsistenze.
  - Alcuni O.S. affrontano il problema attendendo il completamento della chiamata di sistema.
    - Ne deriva un peggioramento del supporto all'interattività in tempo reale ed al multiprocessing.
  - Gli interrupt sono asincroni per definizione e non possono essere sistematicamente ignorati dal kernel.
    - Le sezioni di codice affette interrupt devono essere protette da accessi concorrenti.

Process loading

- Il **dispatcher** è il modulo del sistema operativo che dà il controllo della CPU ad un processo selezionato dal micro-schedulatore. 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.

Criteri di schedulazione

- Gli algoritmi di schedulazione della CPU hanno diverse proprietà e possono favorire una categoria di processi piuttosto che un'altra.
- I criteri generali di comparazione posono essere riassunti in:
  - **Utilizzo della CPU** – mantenere la CPU il più impegnata possibile.
    - Variazioni nei sistemi reali dal 40% al 90%.
  - Frequenza di completamento (**throughput**) – numero di processi completati per unità di tempo.
    - Variazioni nei sistemi reali da 1 a 36000 processi/h.
  - Tempo di completamento (**turnaround time**) – intervallo che va dal momento dell'immissione del processo nel sistema al momento del completamento.
    - $T_{loading}+T_{ready}+T_{CPU}+T_{I/O}$
  - **Tempo di attesa** – somma dei tempi spesi in attesa nella coda dei processi pronti.
    - Si tratta della variabile influenzata dall'algoritmo di schedulazione.
  - **Tempo di risposta** – tempo che intercorre dalla formulazione della richiesta fino alla produzione della prima risposta.
    - Tempo necessario al processo per cominciare a rispondere , non per fornire l'output definitivo. Criterio di riferimento per sistemi interattivi.

Criteri di ottimizzazione

- Massimizzare l'utilizzo della CPU.
- Massimizzare il throughput.
- Minimizzare il turnaround time.
- Minimizzare il tempo di attesa.
- Minimizzare il tempo di risposta.
- Nella maggioranza dei casi si ottimizzano i valori medi anche se in taluni casi è opportuno ottimizzare i valori minimi e massimi.
- Minimizzare il response time massimo per accresce la QoS in modo indistinto su sistemi interattivi.
  - Nei sistemi interattivi time-sharing è più opportuno minimizzare la varianza del response time piuttosto che il suo valor medio.
    - Incremento della predicibilità.
    - È desiderabile un sistema meno variabile piuttosto che più reattivo in media (response time medi inferiori).

Schedulazione FCFS (1/2)

Processo	Durata del picco
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 di attesa medio: (0 + 24 + 27)/3 = 17

Schedulazione FCFS (2/2)

Se i processi arrivano nell'ordine:

$$P_2 , P_3 , P_1$$

- Il diagramma di Gantt per la schedulazione è:

- Tempo di attesa per $P_1 = 6; P_2 = 0, P_3 = 3$
- Tempo di attesa medio: (6 + 0 + 3)/3 = 3
- Forte variabilità del tempo di attesa medio.
- Algoritmo nonpreemptive. La CPU viene rilasciata solo quando il processo termina o fa I/O.
  - Risulta essere problematico per i sistemi a condivisione di tempo.
- Effetto di ritardo a catena (*convoy effect):* molti processi I/O bound attendono sistematicamente che un processo CPU-bound rilasci la CPU (aumento dei delay nell'utilizzo della CPU e delle periferiche).

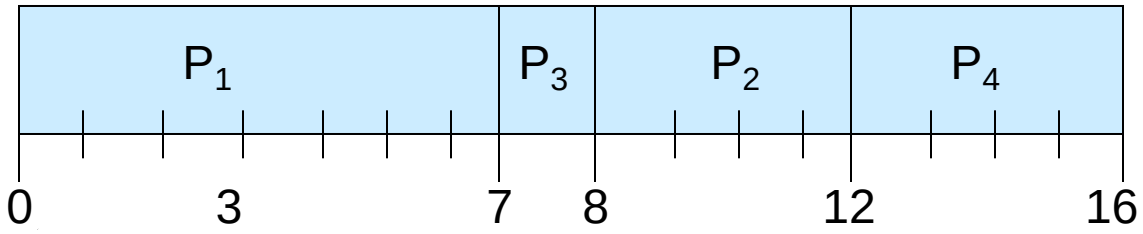
Schedulazione SJF

- Associa a ciascun processo la lunghezza del successivo picco di CPU del processo medesimo (shortest next-CPU-burst first).
- Usa questa lunghezza per schedulare il processo con il minor tempo di CPU richiesto.
- L'algoritmo SJF può essere:
  - **nonpreemptive** – quando un processo arriva nella coda dei processi pronti mentre il processo precedente è ancora in esecuzione l'algoritmo permette al processo di finire il suo uso della CPU.
  - **preemptive** – quando un processo arriva nella coda dei processi pronti mentre il processo precedente è ancora in esecuzione l'algoritmo ferma il processo correntemente in esecuzione. Questa schedulazione è anche detta *shortest-remaining-time-first*.
- SJF è ottimale – fornisce il minor tempo di attesa medio per un dato gruppo di processi.

SJF Non-Preemptive

Processo	Tempo di arrivo	Durata del picco
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

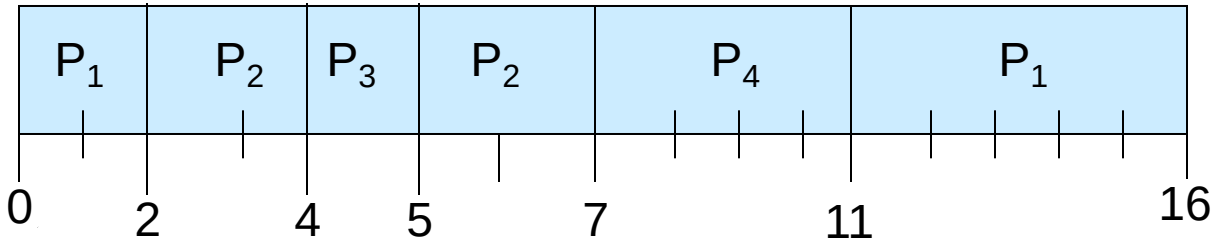


- Tempo di attesa medio = (0 + 3+ 6 + 7)/4 = 4

SJF Preemptive

Processo	Tempo di arrivo	Durata del picco
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



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

Stima del successivo picco di CPU

- SJF è notoriamente ottimale, ma è possibile fare solo una stima della lunghezza della successiva richiesta di CPU.
- Nel caso di sistemi batch si può adoperare la stima del tempo limite di ciascun processo specificato dall'utente quando un job viene mandato in esecuzione.
  - Necessità di una stima accurata del tempo limite di un processo.
- In sistemi più evoluti si adopera una predizione in media esponenziale utilizzando la lunghezza dei precedenti picchi di CPU:

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

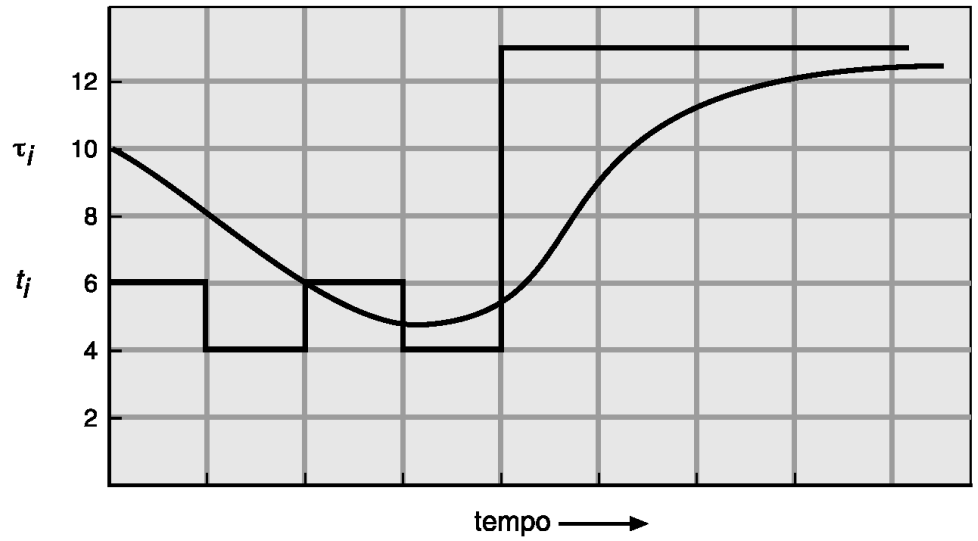
1. t_n = actual duration of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$ (weigth parameter)

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

1. $\alpha, (1-\alpha) < 1 \Rightarrow$ il termine i^{th} ha un peso superiore rispetto a quello $(i+1)^{th}$
2. τ_0 = è un valore costante

Media esponenziale

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La storia recente non ha nessun effetto.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Conta solo il picco più recente di CPU.



picco di CPU (t_i)	6	4	6	4	13	13	13	...
previsione (τ_i) 10	8	6	6	5	9	11	12	...

Schedulazione a priorità

- Si associa una priorità numerica a ciascun processo.
- La CPU viene allocata al processo con la priorità più alta (più piccolo è il valore numerico più alta è la priorità).
  - Preemptive (requisisce la CPU se la priorità del nuovo processo è maggiore)
  - Nonpreemptive (mette il nuovo processo in cima alla coda di ready)
- SJF è un algoritmo con priorità dove la priorità è l'inverso del successivo picco (previsto) di CPU.
- Le priorità possono essere definite:
  - internamente al sistema (si utilizzano grandezze misurabili)
    - uso di memoria, file aperti, rapporto tra picchi medi di di I/O e di CPU
  - esternamente (si usano criteri estranei all'O.S.)
    - rilevanza del processo, criticità
- Problema: blocco indefinito (*starvation*) – i processi a bassa priorità non vengono mai eseguiti.
- Soluzione: invecchiamento (*aging*) – accresce gradualmente la priorità di un processo che attende nel sistema per un lungo periodo.

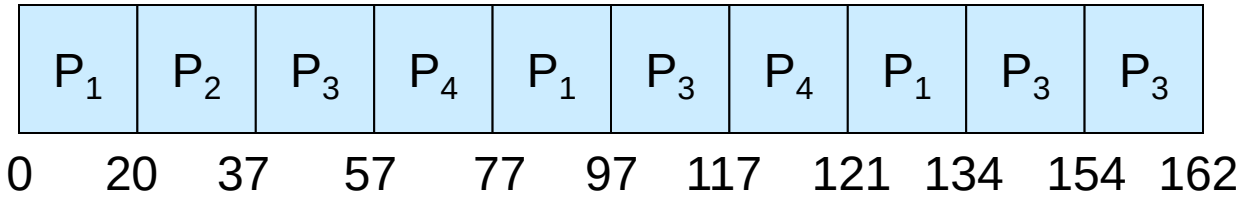
Schedulazione Round Robin (RR)

- Si tratta di un algoritmo di tipo FCFS con l'aggiunta della preemption per migliorare l'alternanza dei processi.
- A ogni processo viene assegnata un quanto del tempo di CPU (*time slice*), generalmente 10-100 millisecondi. Se entro questo arco di tempo il processo non lascia la CPU, viene interrotto (mediante un interrupt) e rimesso nella coda dei processi pronti.
- Se ci sono *n* processi nella coda dei processi pronti e il quanto di tempo è *q*, allora ciascun processo ottiene 1/*n* del tempo di CPU in parti lunghe al più *q* unità di tempo. Ciascun processo non deve attendere più di (*n* – 1) x *q* unità di tempo.
- Le prestazioni dipendono dalla dimensione del time slice q:
  - *q* grande $\Rightarrow$ FIFO
  - *q* piccolo $\Rightarrow$ maggior effetto di "parallelismo virtuale" tra i processi però aumenta il numero di context switch, e quindi l'overhead.
    - Processor sharing (su n processi, virtualizza un processore per processo a 1/n della velocità del processore reale)

RR con quanto di tempo 20

Processo	Tempo del picco
P_1	53
P_2	17
P_3	68
P_4	24

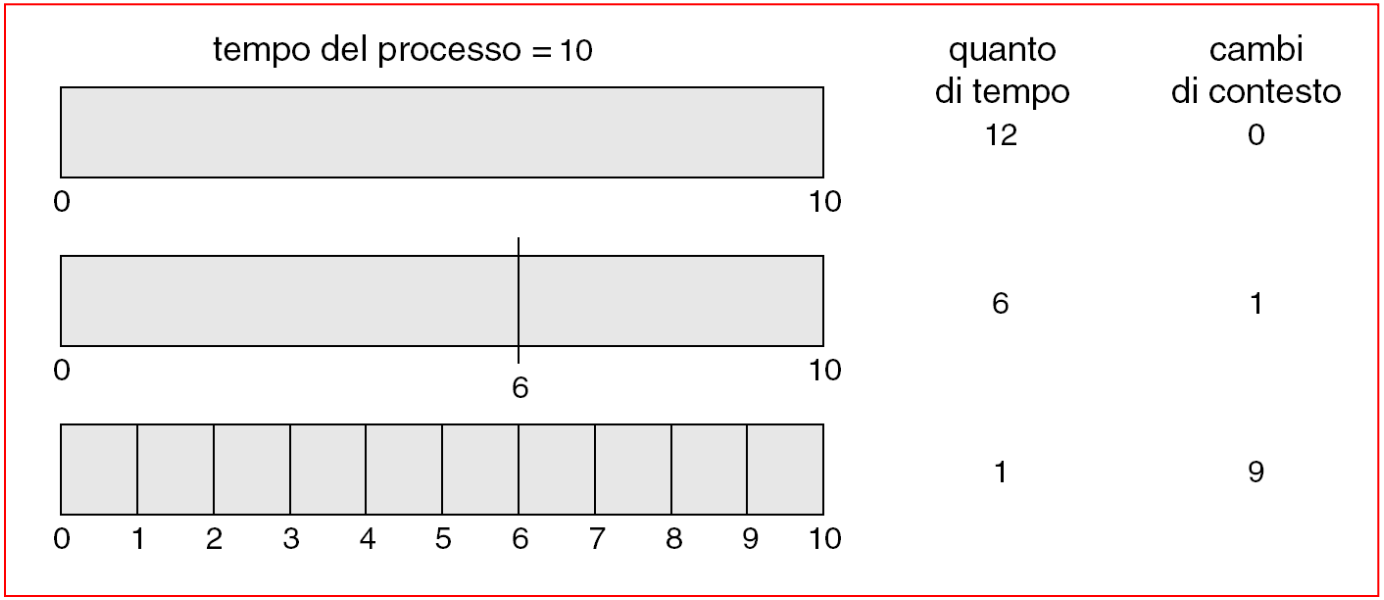
- Il diagramma di Gantt è:



- Di solito produce una media di turnaround più alta rispetto a SJF, ma una migliore risposta.

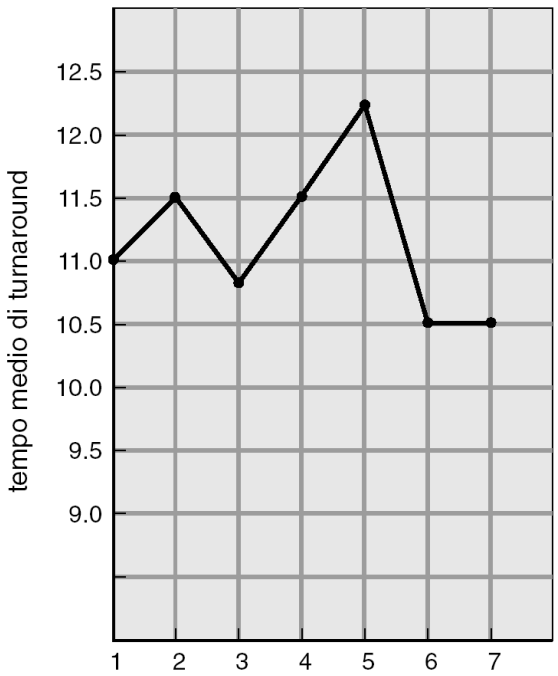
RR e context switching

- Un time slice deve possedere una durata sufficientemente superiore rispetto a quella per il context switching.
  - $T_{context\_switching} \approx 10\% T_{time\_slice} \rightarrow 10\% T_{CPU}$ speso per cambi di contesto
  - Nei sistemi moderni:
    - T_{time_slice} ≈ 10-100 ms, $T_{context\_switching}$ ≈ 10 µs.



RR e turnaround

- Il turnaround time medio di un gruppo di processi non migliora necessariamente all'aumentare della durata del time slice.
- Migliora se la maggioranza dei processi termina il proprio picco successivo in un singolo time slice.
- Considerando il tempo per il cambio di contesto, il turnaround medio aumenta al diminuire della durata dei quanti.



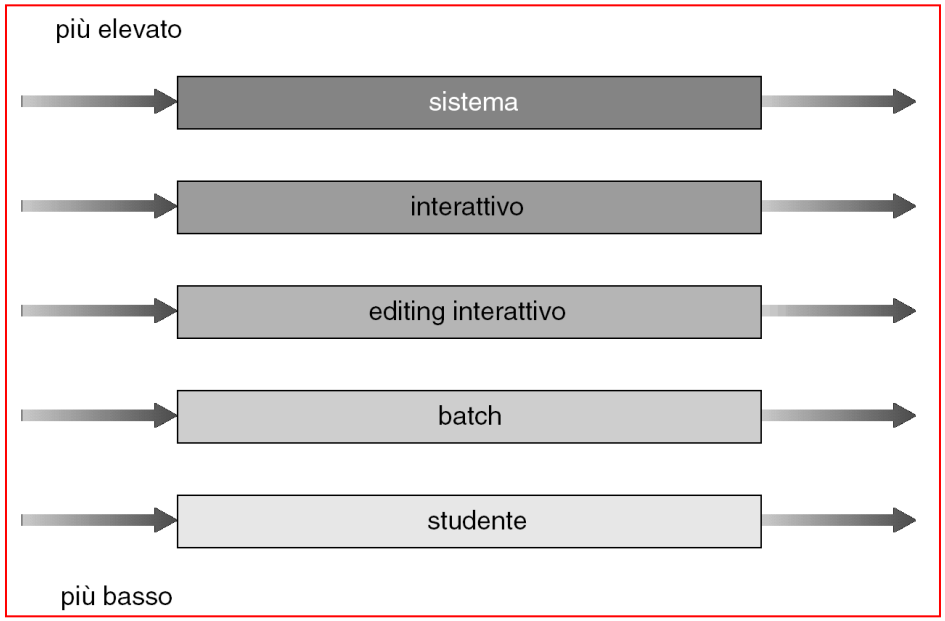
processo	tempo
P_1	6
P_2	3
P_3	1
P_4	7

Coda a più livelli (1/2)

- Algoritmo valido nei casi in cui i processi possano essere classificati in modo distinto. La coda di ready è ripartita in code separate (processi con tempi di riposta e richieste diverse).
- Un processo viene assegnato in modo permanente ad una sola coda.
- La classificazione tipica è:
  - *foreground* (interattivi)
  - *background* (sullo sfondo)
- Ciascuna coda ha il suo algoritmo di schedulazione:
  - foreground – RR
  - background – FCFS

Coda a più livelli (2/2)

- Ci deve essere una schedulazione tra le code.
  - Schedulazione preemptive a priorità fissa (ad esempio la coda dei processi in foreground può avere priorità assoluta su quella dei processi in background). Possibilità di starvation.
  - Partizionamento del tempo tra le code (*time slice*) – ciascuna coda ha una certa quantità di tempo di CPU che può schedulare fra i processi in essa contenuti. Ad esempio il foreground ha l'80% del tempo di CPU per la schedulazione RR, il background riceve il 20% della CPU da assegnare ai suoi processi secondo l'algoritmo FCFS.

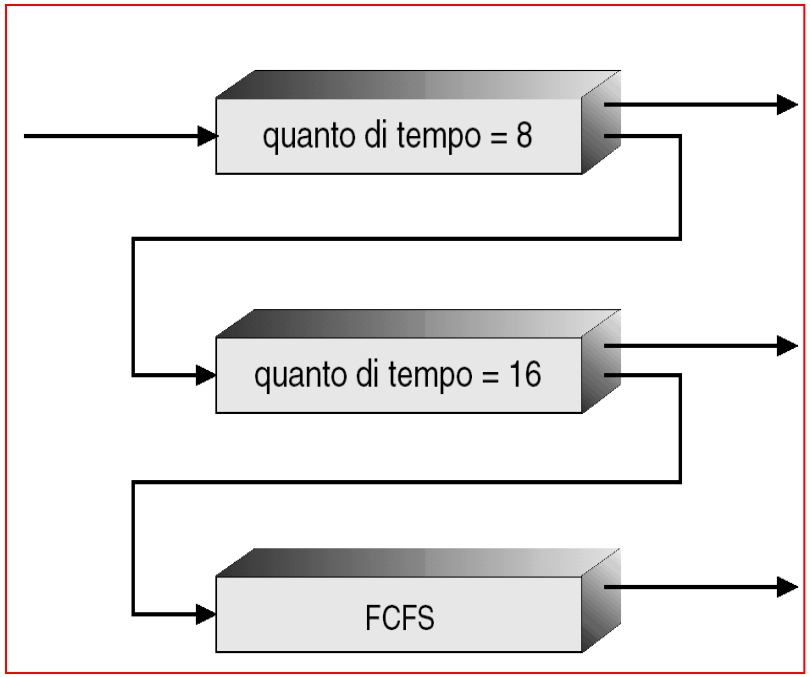


Coda a più livelli retroazionata

- Un processo può muoversi tra le varie code; questa forma di aging previene la starvation e separa i processi con differenti caratteristiche dei picchi di CPU.
- I processi interattivi e I/O bound sono lasciati in code ad alta priorità, quelli con elevato uso di CPU scendono in code a priorità progressivamente inferiore.
- Uno schedulatore con coda a più livelli con feedback è definito dai seguenti parametri:
  - numero di code;
  - algoritmo di schedulazione per ciascuna coda;
  - metodo utilizzato per determinare quando far salire un processo verso una coda a priorità più alta;
  - metodo utilizzato per determinare quando degradare 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

- Tre code:
  - Q_0 – quanto di tempo: 8 millisecondi
  - Q_1 – quanto di tempo: 16 millisecondi
  - Q_2 – FCFS
- Schedulazione
  - Un nuovo processo che è servito in ordine FCFS entra nella coda Q_0. Quando raggiunge la CPU, il processo riceve 8 millisecondi. Se non termina in 8 millisecondi, il processo viene spostato nella coda Q_1.
  - In Q_1 il job riceve 16 millisecondi aggiuntivi. Se ancora non ha completato, viene spostato nella coda Q_2.
  - In Q_2 i processi vengono schedulati secondo FCFS ed eseguiti solo quando Q_1 e Q_o sono vuote.



Sistemi multiprocessore

- Se sono disponibili più CPU il problema della schedulazione diviene più complesso.
- **Processori omogenei**.
  - Uno qualunque dei processori disponibili può essere adoperato per eseguire uno dei processi pronti.
  - **Suddivisione del carico** *(load sharing).*
    - Una coda di ready per processore (rischio di asimmetria nel carico).
    - Una unica coda di ready condivisa.
- *Asymmetric multiprocessing*: solo un processore accede alle strutture dati del sistema diminuendo la necessità della condivisione dei dati.
- *Symmetric multiprocessing*: ciascun processore schedula se stesso attingendo ad una coda di ready condivisa.
  - Si pongono problemi di controllo degli accessi concorrenti alle risorse condivise da parte di processori diversi.

Sistemi hard real-time (1/2)

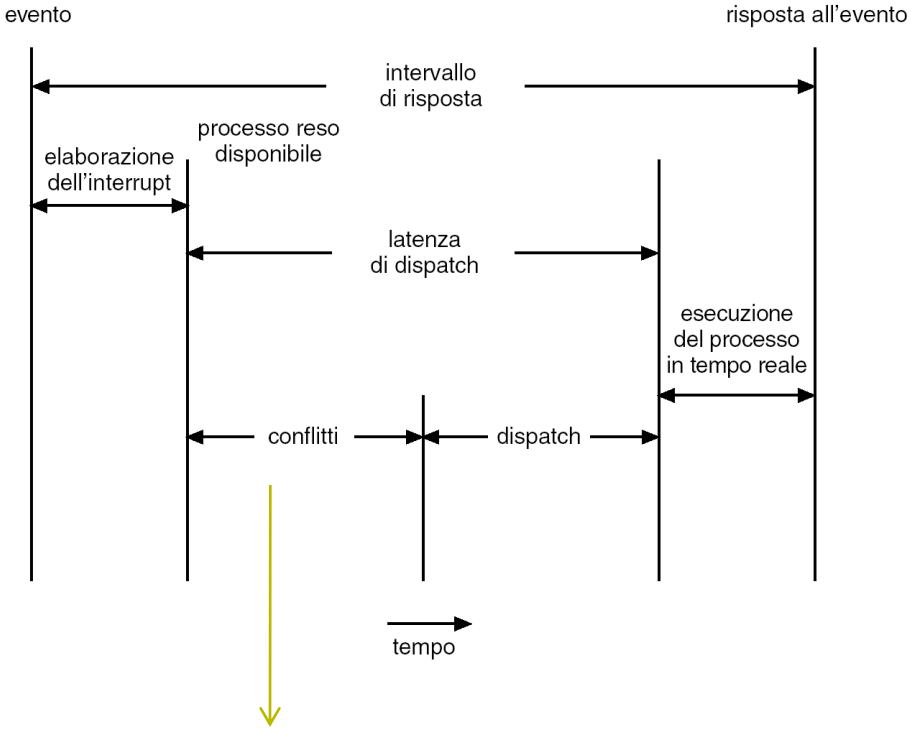
- Sistemi **hard real-time** (in tempo reale stretto) – devono completare un'operazione critica entro una quantità di tempo garantita.
  - Ogni processo viene avviato con un'asserzione sull'intervallo entro il quale deve essere completato e lo schedulatore effettua un *admission control* basata sul meccanismo del *resource reservation.*
  - Lo schedulatore deve garantire il rispetto delle tempistiche di ciascuna funzione dell'O.S..
    - Una tale garanzia è impossibile in presenza di storage di massa o memoria virtuale.
      - I sistemi real time utilizzano software specifico su hardware dedicato per ciascuna attività critica.

Sistemi hard real-time (2/2)

- Schedulazione:
  - Nel caso di processi periodici (attivati ad intervalli regolari) vale la relazione: $0 \le t \le d \le p$ ove t è il tempo di elaborazione, d la deadline e p il periodo.
    - La frequenza di un processo periodico è 1/p e gli schedulatori usano 1/p o d per la schedulazione.
  - La **schedulazione a frequenza monotona** adopera priorità statiche (proporzionali alla frequenza) e preemption. Si assume t sia costante per ogni picco di CPU.
  - La **schedulazione EDF (Earliest-Deadline First)** adopera priorità dinamiche (inversamente proporzionali alla terminazione prevista) aggiornate progressivamente per riflettere la scadenza di un nuovo processo rispetto a quelli già presenti nel sistema.
    - EDF è valido anche nel caso di processi non periodici e con tempo di elaborazione variabile.
    - Il limite teorico di EDF consente di rispettare le deadline di tutti i processi con un utilizzo al 100% della CPU.
      - La gestione del context switching e la risposta alle interruzioni rendono impraticabile un tale limite.

Sistemi soft real-time

- Computazione **soft real-time** (in tempo reale lasco) – richiede che i processi critici ricevano priorità su quelli meno importanti.
  - Schedulazione con priorità ove i processi rt devono essere maggiormente favoriti.
    - La priorità dei processi rt deve restare fissa nel tempo (eliminazione dell'aging).
    - Si deve puntare a ridurre la dispatch latency (un problema tecnologico dato che molte chiamate di sistema sono complesse e l'I/O potrebbe risultare lento).
      - Per permettere l'interruzione delle chiamate di sistema lunghe si inseriscono dei *preemption point* lungo l'elaborazione in punti "sicuri".
      - Essi controllano se deve essere eseguito un processo a più alta priorità e in caso affermativo, al suo termine si riprende la chiamata di sistema.



- Interruzione di ogni processo in esecuzione nel kernel
- Rilascio dei processi a bassa priorità delle risorse necessarie ai processi ad alta priorità
  - Inversione di priorità
  - Ereditarietà della priorità

Schedulazione dei thread

- *Schedulazione locale:* come la libreria dei thread decide quali thread utente allocare ad un LWP disponibile (il kernel non è a conoscenza dei thread user-space).
- *Schedulazione globale:* come il kernel decide quale sarà il prossimo thread da eseguire.
  - Per essere eseguiti su una CPU i thread user-level devono essere mappati su thread kernel-level generalmente mediante l'uso di LWP.
- Nei modelli molti-a-molti e molti-a-uno la libreria di thread schedula i thread lato utente secondo lo schema **PCS** (Process Contention Scope).
  - La contesa per un LWP avviene tra tutti i thread di un dato processo (contesa a livello di processo).
  - PCS è basato su priorità (definite dal programmatore e non alterate dalla thread lib). Esiste l'interrompibilità a beneficio di thread a maggiore priorità.
- I thread lato kernel invece vengono schedulati dal sistema operativo secondo lo schema **SCS** (System Contention Scope).
  - La contesa per la CPU avviene tra tutti i thread del sistema (contesa a livello di sistema).
- Nei modelli uno-a-uno si adopera solo lo schema SCS.