Scheduling della CPU

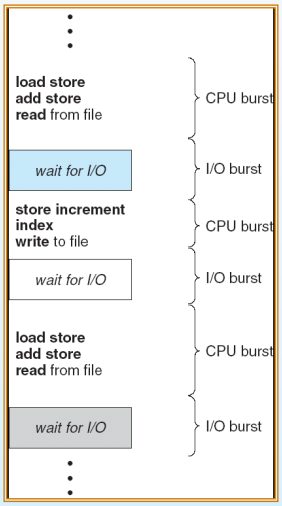
Capitolo 5: CPU Scheduling

- Criteri di Scheduling
- Algoritmi di Scheduling
- Multiple-Processor Scheduling
  - Asymmetric/Symmetric multiprocessing
  - Processori Multicore

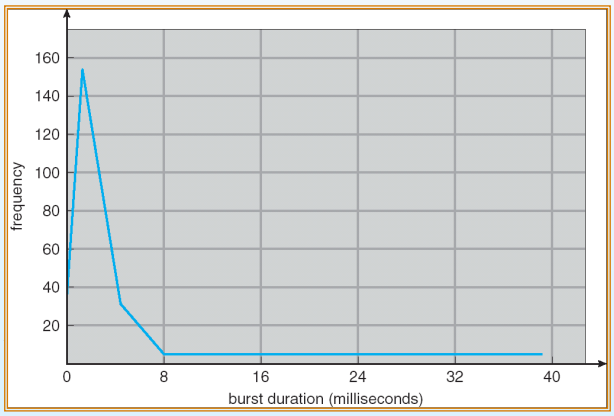
Concetti di Base

- Massimizzare l'utilizzo della CPU mediante multiprogrammazione
- CPU–I/O Burst Cycle
  - Processo alterna *cicli* di CPU ad attese di I/O
- Distribuzione dei cicli di CPU (CPU-burst)

Alternating Sequence of CPU And I/O Bursts



Istogramma dei CPU-burst



Schedulatore della CPU

- Sceglie tra i processi *pronti* quello da eseguire e vi alloca la CPU
- La Schedulazione della CPU avviene quando un processo:
  1. Passa da attivo in stato di attesa
  2. Passa da attivo a stato di pronto
  3. Passa da stato di attesa a stato di pronto
  4. Termina
- Scheduling secondo 1 e 4 e' *non-preemptive (senza prelazione)*
- Se si schedula secondo 2 e 3, *preemptive (con prelazione)*

Dispatcher

- Il Dispatcher passa il controllo della CPU al processo selezionato dallo schedulatore a breve termine:
  - Cambia il contesto (context switch)
  - Passa in modalita' utente
  - Salta alla locazione giusta da cui il processo schedulato deve ripartire
  - *Latenza di Dispatch latency* – tempo necessario al dispatcher per fermare un processo e farne ripartire un altro

Criteri di Scheduling

- Utilizzo della CPU utilization
  - CPU quasi sempre occupata
- Throughput – # di processi completati per unita' di tempo
- Tempo di Turnaround – tempo necessario per completare un particolare processo
- Tempo di attesa – tempo trascorso dal processo nella coda di dei processi in stato di pronto
- Tempo di risposta – tempo trascorso dal momento in cui una richiesta viene sottoposta al sistema fino al momento in cui una prima risposta appare in output (importante in ambienti time-sharing)

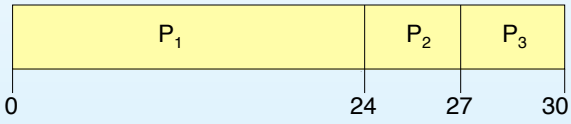
Criteri di Ottimizzazione

- Massimizza uso di CPU
- Massimizza throughput
- Minimizza turnaround
- Minimizza tempo d'attesa
- Minimizza tempo di risposta

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- I processi arrivano nell'ordine: P_1 , P_2 , P_3
  Il diagramma di Gantt per la schedulazione:



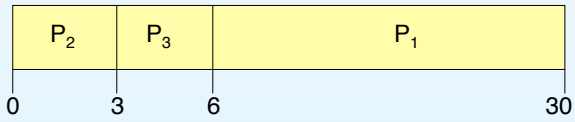
- Tempo di attesa: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Tempo medio di attesa: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Consideriamo ora l'ordine di arrivo

P_2 , P_3 , P_1

- Il diagramma di Gantt diventa:



- Tempi di attesa: $P_1 = 6; P_2 = 0, P_3 = 3$
- Tempo medio di attesa: (6 + 0 + 3)/3 = 3
  - Miglioramento significativo
- *Evitato l'effetto convoglio:* processi brevi dopo processi lunghi

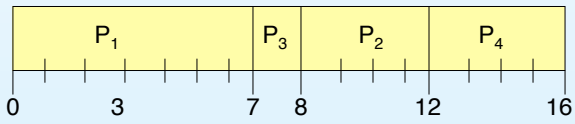
Shortest-Job-First (SJF) Scheduling

- Scegli il processo con il piu' piccolo CPU burst
- Schemi possibili:
  - Non-preemptive – una volta assegnata la CPU ad un processo attendi fino a completamento del CPU burst
  - preemptive – se un nuovo processo arriva con CPU burst piu' breve del tempo rimanente al CPU burst del processo in esecuzione, prelaziona
    - Schema noto come: Shortest-Remaining-Time-First (SRTF)
- SJF ottimizza (minimizza) il tempo medio di attesa

Esempio di SJF Non-Preemptive

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

- SJF (non-preemptive)

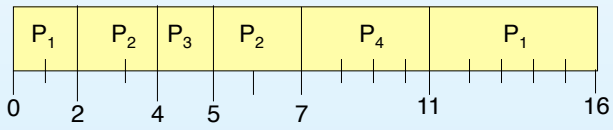


- Tempo media di attesa = (0 + 6 + 3 + 7)/4 = 4

Esempio di SJF Preemptive

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

- SJF (preemptive)



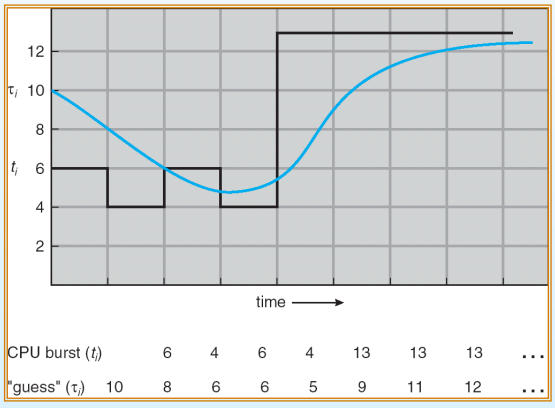
- Tempo medio di attesa = (9 + 1 + 0 +2)/4 = 3

La Lunghezza dei CPU Burst

- Possiamo solo stimare tale lunghezza
- Possibilita': usa le lunghezza dei precedenti CPU bursts
  - Es.: media esponenziale

1. t_n = lunghezza effettiva dell'n_{mo} CPU burst
2. τ_{n+1} = valore atteso del prossimo CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Definisci: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$.

Previsione della lunghezza di CPU Burst



Es.: Media Esponenziale

- α =0
  - $\tau_{n+1} = \tau_n$
  - La storia recente non ha valore
- α =1
  - $\tau_{n+1} = \alpha\ t_n$
  - Solo il precedente CPU burst ha valore
- Se espandiamo la formula otteniamo:

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

- Nota che α e $(1 - \alpha)$ sono ≤ 1, quindi ogni termine nella formula pesa meno del precedente

Scheduling a Priorita'

- Ogni processo ha associato un valore di priorita' (intero)
- La CPU e' allocata al processo con priorita' piu' alta (spesso nueri piccoli indicano priorita' alta)
  - Preemptive
  - Non-preemptive
- SJF e' uno sceduling a priorita' definita dal tempo previsto di CPU burst.
- Problemi: Starvation – processi con priorita' bassa potrebbero essere "dimenticati"
- Soluzione: Aging – incrementare la priorita' dei processi in funzione del tempo trascorso in stato di pronto

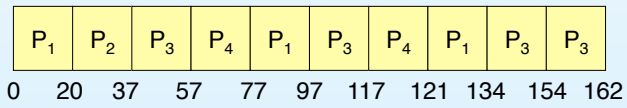
Round Robin (RR)

- Ogni processo riceve la CPU per un breve intervallo (*time quantum*), tipicamente 10-100 millisecondi. Passato tale quanto di tempo, il processo ritorno all fine della coda dei processi in stato di pronto.
- Se *n* processi in coda di pronto e quanto di tempo = *q*, ogni processo riceve 1/*n* di CPU time in blocchi di *q* unita' per volta. Nessun processo attende piu' di (*n*-1)*q* unita'.
- Performance
  - *q* grande ⇒ FIFO
  - *q* piccolo ⇒ *q* deve esser maggiore del tempo di context switch, per evitare eccessivo

Es.: RR con quanto di tempo = 20

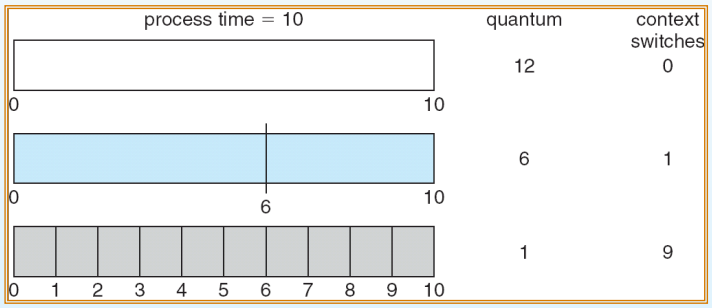
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Il diagramma di Gantt:

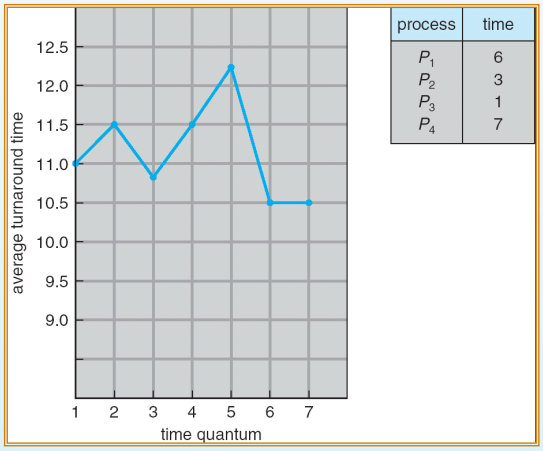


- Tipicamente, rispetto a SJF, tempo di turnround piu' alto, ma migliore tempo di risposta

Tempo di Context Switch



Turnaround Time dipende da q

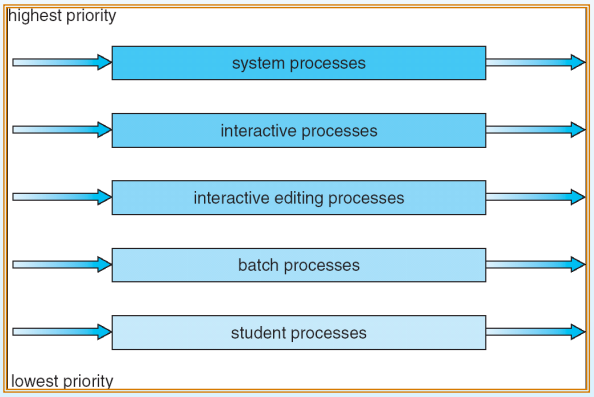


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Code Multilivello

- La coda dei processi pronti divisa in code separate:
  foreground (interactive)
  background (batch)
- Ogni coda e' gestita al proprio algoritmo di scheduling
  - foreground – RR
  - background – FCFS
- Scheduling tra le code
  - Scheduling a priorita' fissa; (prima tutti I processi in foreground quindi quelli in background). Rischio di starvation.
  - Time slice – ogni coda ha assegnata una percentuale di tempo di CPUin cui puo' schedulare I propri processi;
    - es.: 80% to foreground in RR
      - 20% to background in FCFS

Code Multilivello



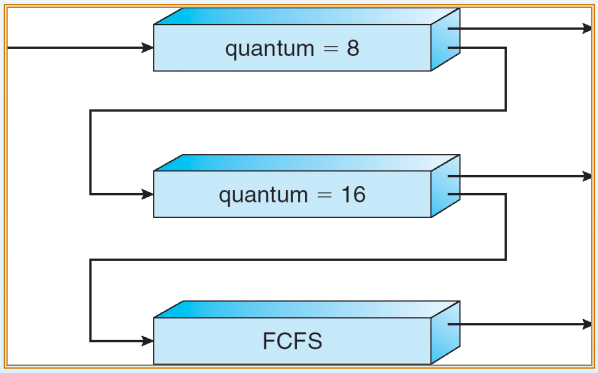
Code Multilivello con retroazione

- Processi possono cambiare coda;
  - Possibilita' di implementare tecniche di aging
- Parametri critici:
  - numero di code
  - Algoritmo di scheduling usato in ciascuna coda
  - Criterio di promozione dei processi tra le code
  - Criterio di declassamento dei processi tra le code
  - Criterio di assegnazione di un processo ad una coda in funzione del servizio richiesto

Es.: Code Multilivello con retroazione

- Tre code:
  - Q_0 – RR con q=8 milliseconds
  - Q_1 – RR con q=16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - Ogni nuovo job e' ammesso alla coda Q_0 servita con FCFS. Quando attivo, il processo ha 8 milliseconds per completare. Se non completa passa alla coda Q_1.
  - In Q_1 i job sono schedulati con criterio FCFS con q=16. Se tale quanto non risulta sufficiente, il processo passa in Q_2.

Multilevel Feedback Queues



Highest Response Ratio Next

- w : tempo speso dal processo in attesa in stato di pronto
- s : tempo totale necessario al processo per arrivare a terminazione
- R = (w+s)/s

- Lo scheduling HRRN sceglie il processo con max R
  - Processi brevi hanno “priorita’”
  - Implementa una sorta di “aging” implicito

Esempi: Unix classico

- Code Multilivello con retroazione
- CPU(i) = CPU(i-1)/2 stima del tempo di CPU
- P(i) = Base + CPU(i)/2 + nice
- Un valore alto di P(i) indica priorita' bassa
- Base : serve per definire le code o classi di processi
  - Processi che usano molta CPU vengono penalizzati

Sistemi Multi-Processore

- CPU scheduling deve bilanciare il carico e le condivisioni tra I vari processori
- *Multiprogrammazione asimmetrica*: un solo processore accede alle strutture dati di sistema e gestisce la suddivisione per tutti
- *Multiprogrammazione Simmetrica*: ogni processore fa scheduling autonomamente;

Multiple-Processor Scheduling

Multiprocessing Simmetrico: ogni processore si schedula autonomamente;

- Alternative:
  - Coda comune
  - Code separate
- Affinita' di processo: meglio evitare che un processo passi da un processore all'altro
  - Affinita' forte: no switching
  - Affinita' debole: no switching preferito ma non garantito
- Bilanciamento del carico (in contrasto con l'affinita' di processo)
  - Migrazione mediante push
  - Migrazione mediante pull

Multicore Processors

- Processori multipli sullo stesso chip
  - Ogni core ha il suo insieme di registri
  - Risparmio energetico e piu' veloce
- Multi-threading
  - I Core possono schedulare diversi thread
    - Multi-Thread gestito in hardware
  - Dual core, dual thread = (virtualmente) 4 processori
    - Scheduling parallelo di processi e (hw thread)