Scheduling della CPU

Scheduling della CPU

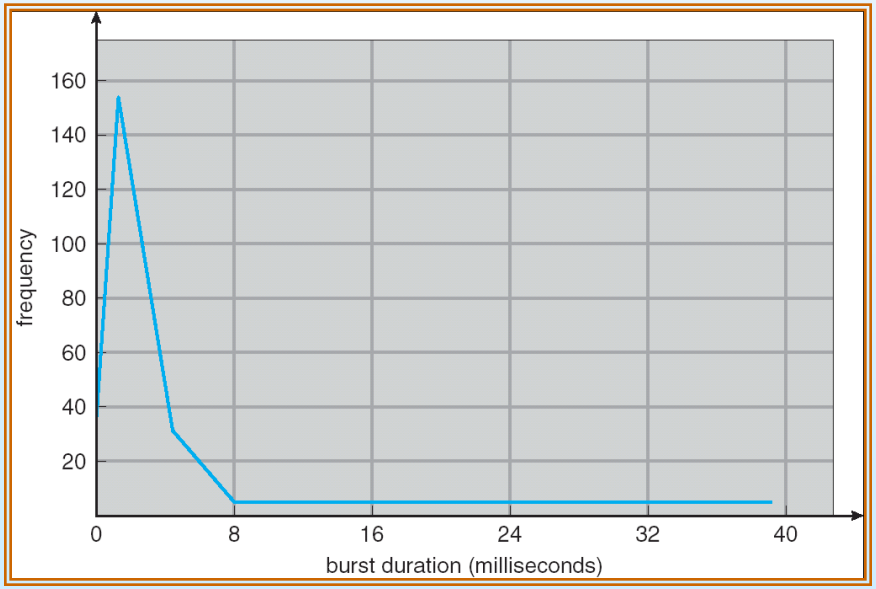
- Concetti fondamentali
- Criteri di scheduling
- Algoritmi di scheduling
- Scheduling dei thread
- Scheduling multiprocessore
- Scheduling real–time
- Scheduling in Linux
- Valutazione degli algoritmi

Lo scheduling della CPU

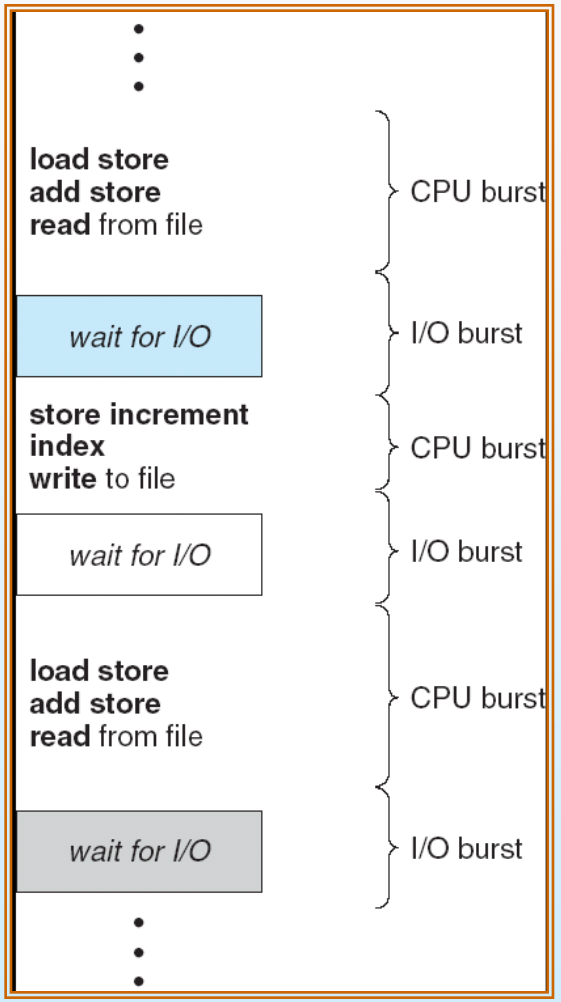
- Lo scheduling è una funzione fondamentale dei sistemi operativi
  - Si sottopongono a scheduling quasi tutte le risorse di un calcolatore
- Lo scheduling della CPU è alla base dei sistemi operativi multiprogrammati
  - Attraverso la commutazione del controllo della CPU tra i vari processi, il SO rende più "produttivo" il calcolatore

Concetti fondamentali

- Il massimo impiego della CPU è pertanto ottenuto con la multiprogrammazione
- Ciclo di CPU–I/O burst – L'esecuzione di un processo consiste di *cicli* di esecuzione nella CPU ed attese ai dispositivi di I/O



Distribuzione dei burst di CPU



Lo scheduler della CPU – 1

- Lo scheduler della CPU gestisce la coda dei processi pronti, selezionando il prossimo processo cui verrà allocata la CPU
  - Gli elementi nella ready queue sono i PCB dei processi pronti
  - La ready queue può essere realizzata come una coda FIFO (*first–in–first–out*), una coda con priorità, una lista concatenata o un albero

Lo scheduler della CPU – 2

- Lo scheduler della CPU deve prendere una decisione quando un processo…
  1. …passa da stato *running* a stato *wait* (richiesta di I/O o attesa terminazione di un processo figlio)
  2. …passa da stato *running* a stato *ready* (interrupt)
  3. …passa da stato *wait* a stato *ready* (completamento di un I/O)
  4. …termina
- Se lo scheduling viene effettuato solo nei casi 1 e 4, si dice che lo schema di scheduling è *nonpreemptive* (senza diritto di prelazione) o *cooperativo*
- Altrimenti si ha uno schema *preemptive*

Il dispatcher

- Il modulo *dispatcher* passa il controllo della CPU al processo selezionato dallo scheduler a breve termine; il dispatcher effettua:
  - Context switch
  - 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

- Utilizzo della CPU — la CPU deve essere più attiva possibile
- Throughput (produttività) — numero dei processi che completano la loro esecuzione nell'unità di tempo
- Tempo di turnaround (tempo di completamento) — tempo impiegato per l'esecuzione di un determinato processo
  - Tempo di attesa in memoria, nella ready queue, durante l'esecuzione e nelle operazioni di I/O
- Tempo di attesa — tempo passato dal processo in attesa nella ready queue
- Tempo di risposta — tempo che intercorre tra la sotto-missione di una richiesta e la prima risposta prodotta
  - Nei sistemi time–sharing, il tempo di turnaround può essere influenzato dalla velocità del dispositivo di output

Criteri di ottimizzazione – 1

- Massimo utilizzo della CPU
- Massimo throughput
- Minimo tempo di turnaround
- Minimo tempo di attesa
- Minimo tempo di risposta

Criteri di ottimizzazione – 2

- Normalmente si ottimizzano i valori medi, ma talvolta è più opportuno ottimizzare i valori minimi/massimi
  - Per esempio… per garantire che tutti gli utenti ottengano un buon servizio ⇨ ridurre il tempo massimo di risposta
- Per i sistemi time–sharing è invece più significativo ridurre la varianza del tempo medio di risposta: un sistema con tempo di risposta prevedibile è meglio di un sistema mediamente più rapido, ma molto variabile

Scheduling First–Come–First–Served (FCFS)

- La CPU viene assegnata al processo che la richiede per primo
- La realizzazione del criterio FCFS si basa sull'implementazione della ready queue per mezzo di 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 assegnata al processo che si trova alla testa della ready queue, rimuovendolo da essa

Scheduling FCFS (Cont.)

- Esempio 1

Processo	Tempo di burst (millisecondi)
P_1	24
P_2	3
P_3	3

- I processi arrivano al sistema nell'ordine: P_1 , P_2 , P_3
- Per descrivere come si realizza lo scheduling si usa un diagramma di Gantt
  - Un istogramma che illustra una data pianificazione includendo i tempi di inizio e di fine di ogni processo

Scheduling FCFS (Cont.)

- Il diagramma di Gantt per lo scheduling FCFS è:

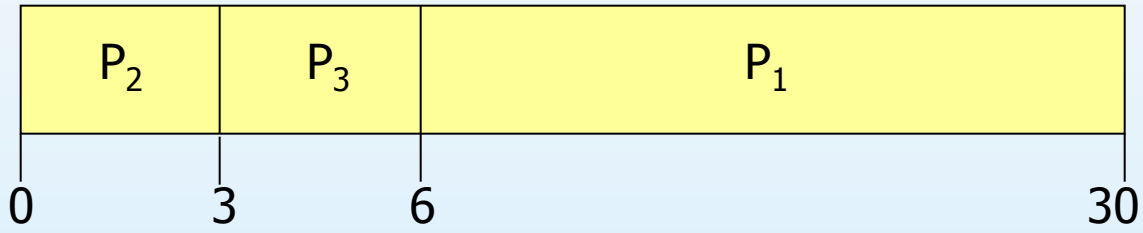
- Tempi di attesa: $P_1 \rightarrow 0$, $P_2 \rightarrow 24$, $P_3 \rightarrow 27$
- Tempo medio di attesa $T_a = (0+24+27)/3 = 17$

Scheduling FCFS (Cont.)

- Supponiamo che l'ordine di arrivo dei processi sia

$$P_2 , P_3 , P_1$$

- Il diagramma di Gantt diventa:



- Tempi si attesa: $P_1 \rightarrow 6$, $P_2 \rightarrow 0$, $P_3 \rightarrow 3$
- Tempo medio di attesa $T_a = (6+0+3)/3 = 3$
- Molto inferiore al caso precedente: in questo caso, non si verifica l'*effetto convoglio*, per cui processi di breve durata devono attendere che un processo lungo liberi la CPU

Scheduling Shortest–Job–First (SJF)

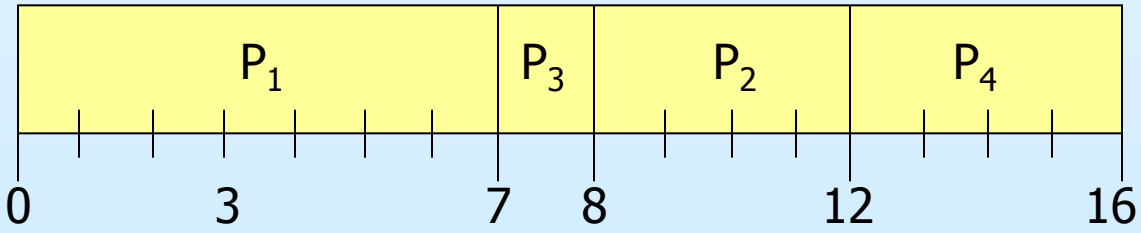
- Si associa a ciascun processo la lunghezza del suo burst di CPU successivo; si opera lo scheduling in base alla brevità dei CPU burst
- Due schemi:
  - non–preemptive — dopo che la CPU è stata allocata al processo, non gli può essere prelazionata fino al termine del CPU burst corrente
  - preemptive — se arriva un nuovo processo con burst di CPU minore del tempo rimasto per il processo corrente, il nuovo processo prelaziona la CPU: *Shortest–Remaining–Time–First* (SRTF)
- SJF è *ottimo* — minimizza il tempo medio di attesa

SJF non–preemptive

- Esempio 2

Processo	Tempo di arrivo	Tempo di burst
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

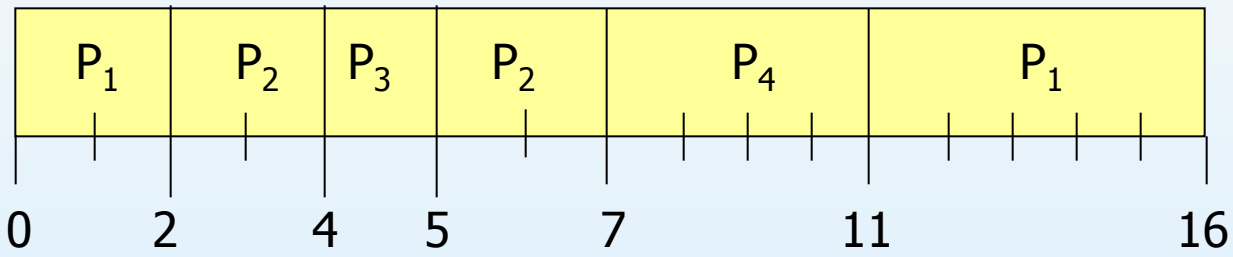
- SJF non–preemptive



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

SJF preemptive

- Esempio 3
  - SJF preemptive



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

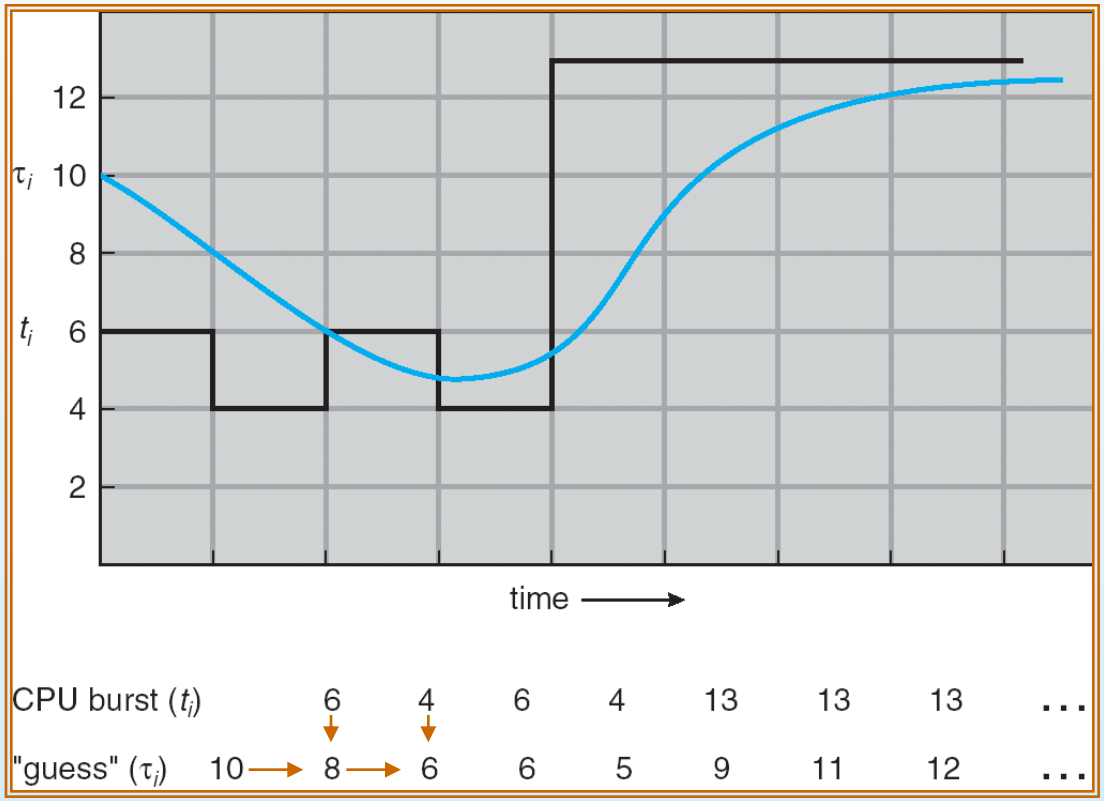
Lunghezza del CPU burst successivo

- Può essere stimato come una media esponenziale delle lunghezze dei burst di CPU precedenti (impiegando una combinazione convessa)
  1. t_n = lunghezza dell'n–esimo CPU burst
  2. τ_{n+1} = valore stimato del prossimo CPU burst
  3. $0 \leq \alpha \leq 1$

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

  τ_1 = valore costante o stimato come media dei burst del sistema

Predizione della lunghezza del CPU burst successivo



α=1/2, τ_1=10

Esempi di calcolo

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - La storia recente non viene presa in considerazione: le condizioni attuali sono transitorie
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Si considera solo l'ultimo CPU burst: la storia passata è irrilevante
- Espandendo la formula si ottiene:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^n \tau_1$$

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

Scheduling a priorità

- Un valore di priorità (intero) viene associato a ciascun processo
- La CPU viene allocata al processo con la priorità più alta (intero più basso≡priorità più alta)
  - Preemptive
  - Non–preemptive
- SJF è uno scheduling a priorità in cui la priorità è rappresentata dal successivo tempo di burst
- Problema: *Starvation* ("inedia", 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 a priorità (Cont.)

- Esempio 4

Processo	Durata	Priorità
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Il diagramma di Gantt è:

- Tempi di attesa: $P_1 \rightarrow 6$, $P_2 \rightarrow 0$, $P_3 \rightarrow 16$, $P_4 \rightarrow 18$, $P_5 \rightarrow 1$
- Tempo medio di attesa $T_a=(6+0+16+18+1)/5=8.2$

Scheduling Round Robin (RR) – 1

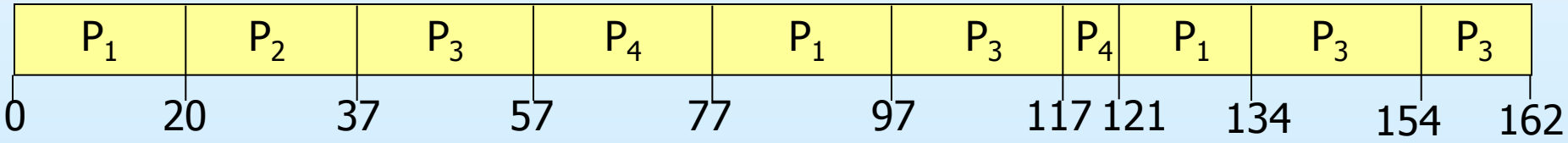
- A ciascun processo viene allocata una piccola quantità di tempo di CPU, un *quanto di tempo*, generalmente 10–100 millisecondi
  - Dopo un quanto di tempo, il processo è forzato a rilasciare la CPU e accodato alla ready queue
- Se vi sono n processi nella ready queue ed il quanto di tempo è q, ciascun 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
- Prestazioni:
  - q grande ⇨ FCFS
  - q piccolo ⇨ q deve essere grande rispetto al tempo di context switch, altrimenti l'overhead è troppo alto

Scheduling RR – 2

- Esempio 5

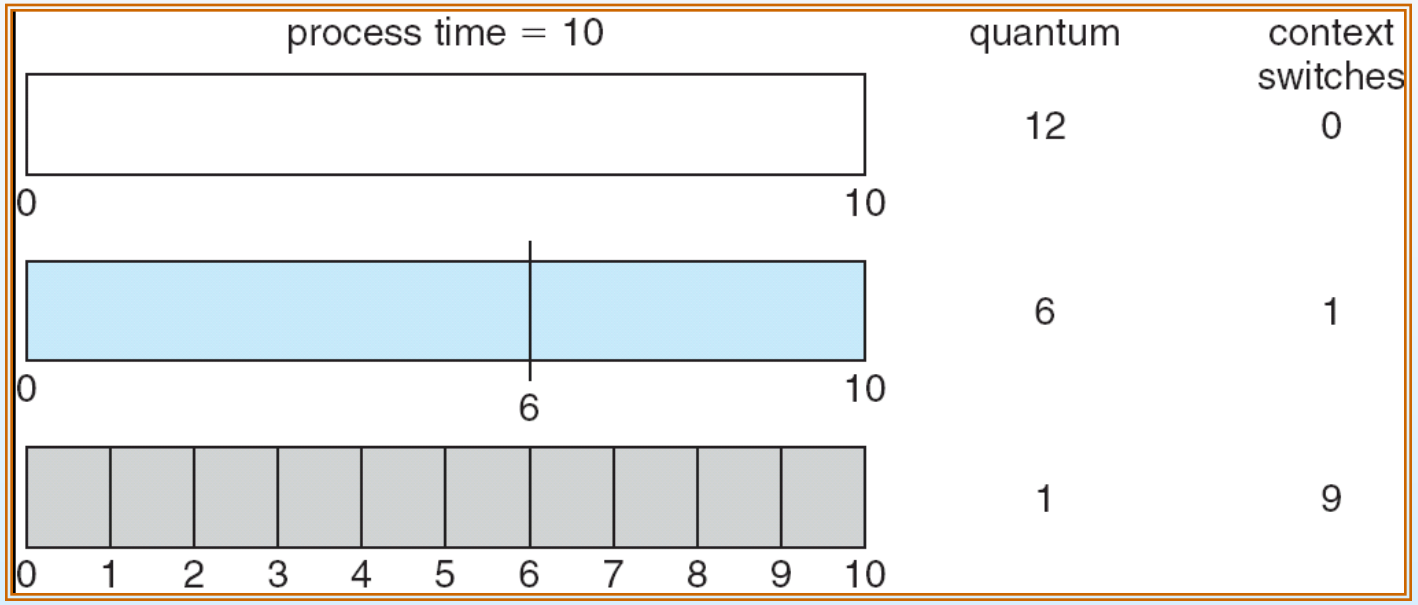
Processo	Tempo di burst
P_1	53
P_2	17
P_3	68
P_4	24

- Il diagramma di Gantt con q=20 è



- In genere si ha un tempo medio di attesa e di turnaround maggiore rispetto a SJF, tuttavia si ottiene un miglior tempo medio di risposta

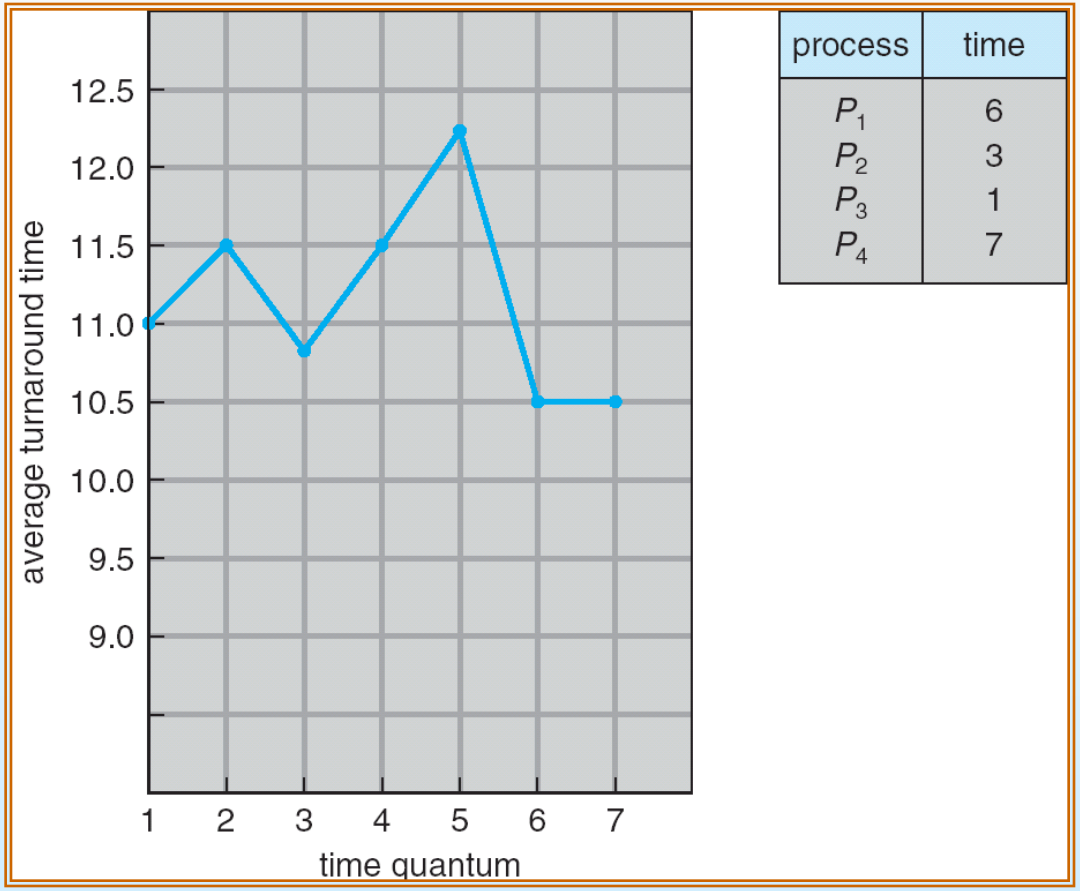
Quanto di tempo e tempo di context switch



Un quanto di tempo minore incrementa il numero di context switch

Quanto di tempo e tempo di turnround

Empiricamente: il quanto di tempo deve essere più lungo dell'80% dei CPU burst



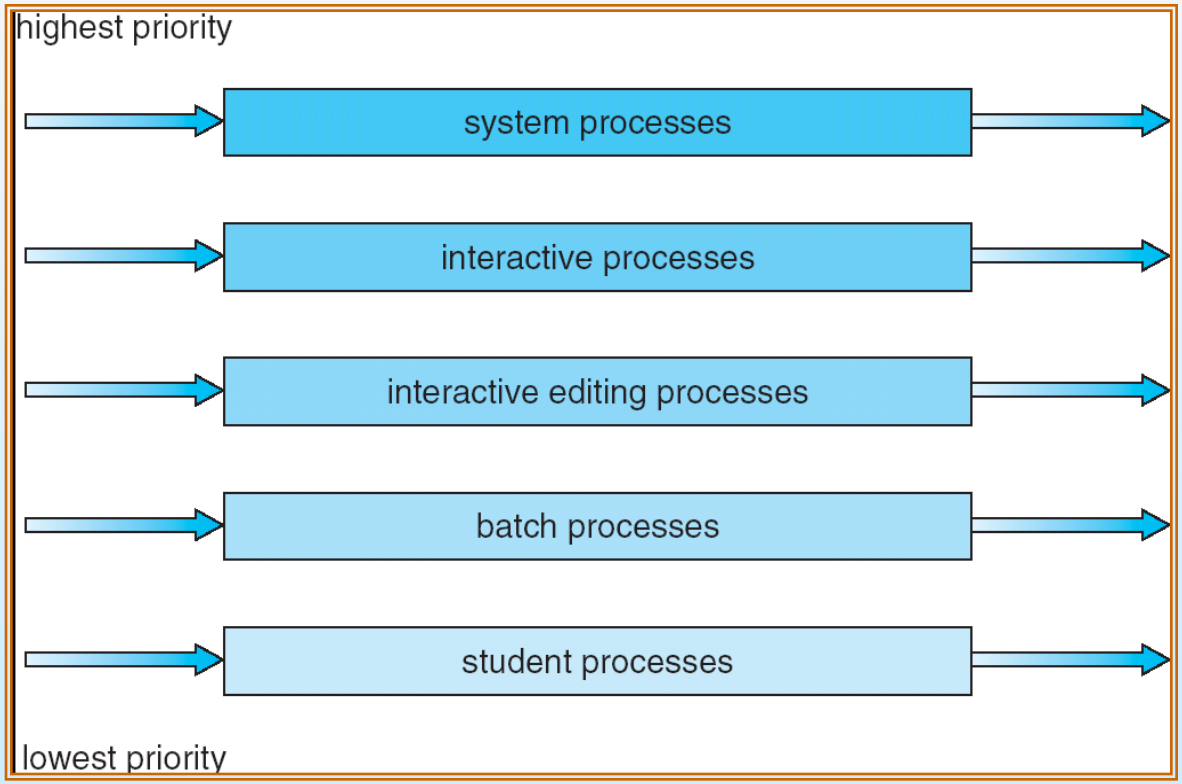
Variazione del tempo medio di turnaround in funzione della lunghezza del quanto di tempo

Scheduling con code multiple – 1

- La ready queue è suddivisa in più code separate:
  - foreground (interattiva)
  - background (batch)
- Ciascuna coda ha il suo proprio algoritmo di scheduling:
  - foreground: RR
  - background: FCFS
- È necessario effettuare lo scheduling tra le code
  - *Scheduling a priorità fissa*: si servono tutti i processi foreground poi quelli 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 con code multiple – 2

- I processi si assegnano in modo permanente ad una coda, generalmente secondo qualche caratteristica (invariante) del processo



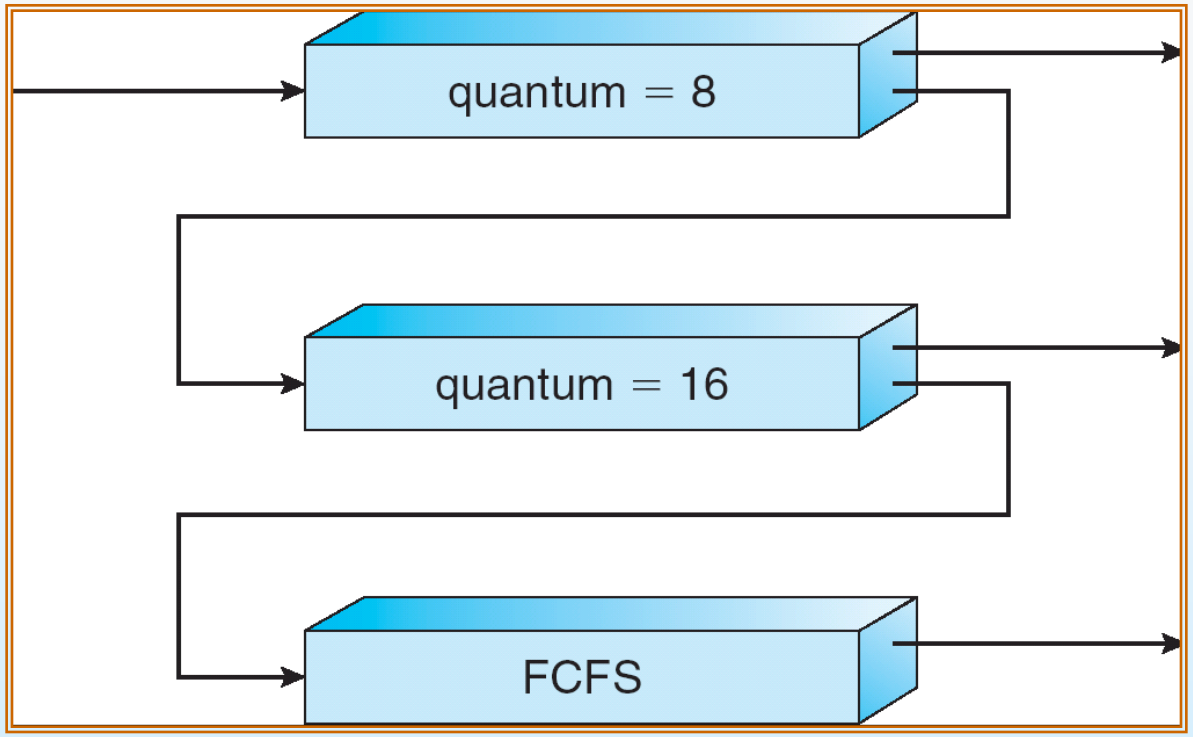
Code multiple con feedback – 1

- Un processo può spostarsi fra le varie code; si può implementare l'aging
- Lo scheduler a code multiple con feedback è definito dai seguenti parametri:
  - Numero di code
  - Algoritmo 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 entra nel sistema

Code multiple con feedback – 2

- Esempio 6
  - Tre code:
    - Q_0 – RR con quanto di tempo di 8 millisecondi
    - Q_1 – RR con quanto di tempo di 16 millisecondi
    - Q_2 – FCFS
- Scheduling
  - Un nuovo job viene immesso nella coda Q_0 che è servita con RR; quando prende possesso della CPU il job riceve 8 millisecondi; se non termina, viene spostato nella coda Q_1
  - Nella coda Q_1 il job è ancora servito RR e riceve ulteriori 16 millisecondi; se ancora non ha terminato, viene spostato nella coda Q_2, dove verrà servito con criterio FCFS all'interno dei cicli di CPU lasciati liberi dai processi delle code Q_0 e Q_1

Code multiple con feedback – 3



- L'idea sottesa all'algoritmo di scheduling è quella di separare i processi in base alle loro caratteristiche d'uso della CPU ⇨ Massima priorità con CPU–burst brevi

Scheduling dei thread – 1

- Scheduling locale: nei modelli molti–a–uno e molti–a–molti, è l'algoritmo con cui la libreria dei thread decide quale thread (a livello utente) allocare ad un LWP disponibile (*ambito di competizione ristretto al processo*) – normalmente un algoritmo a priorità con prelazione
  - È la libreria dei thread che pianifica l'esecuzione su LWP, ma…
  - …il thread prescelto non è effettivamente in esecuzione su una CPU fisica finché il SO non pianifica l'esecuzione del thread a livello kernel corrispondente

Scheduling dei thread – 2

- Scheduling globale: l'algoritmo con cui il kernel decide quale thread a livello kernel dovrà venire eseguito successivamente (*ambito di competizione allargato al sistema*)
- I sistemi operativi caratterizzati dall'impiego del modello uno–a–uno (quali Windows XP e Linux) pianificano i thread unicamente con scheduling globale

Scheduling multiprocessore – 1

- Lo scheduling diviene più complesso quando nel sistema di calcolo sono presenti più CPU
- Ipotesi: le unità di elaborazione sono, in relazione alle loro funzioni, identiche – sistemi omogenei
- Ripartizione del carico di lavoro
- Multielaborazione asimmetrica – lo scheduling, l'elaborazione dell'I/O e le altre attività del sistema sono affidate ad un solo processore, detto *master server*
  - Si riduce la necessità di condividere dati, grazie all'accesso di un solo processore alle strutture dati del sistema

Scheduling multiprocessore – 2

- Multielaborazione simmetrica (SMP, *Symmetric Multi-Processing*) – i processi pronti vanno a formare una coda comune oppure vi è un'apposita coda per ogni processore; ciascun processore ha un proprio scheduler che esamina la coda opportuna per prelevare il prossimo processo da eseguire
  - L'accesso concorrente di più processori ad una struttura dati comune rende delicata la programmazione degli scheduler che…
    - …devono evitare di scegliere contemporaneamente lo stesso processo da eseguire
    - …e devono evitare che qualche processo pronto "vada perso"
- Adottata da molti SO attuali, per es. Windows XP e Linux

Multielaborazione simmetrica – 1

- Predilezione per il processore – Si mira a mantenere un processo in esecuzione sempre sullo stesso processore, per riutilizzare il contenuto della cache per burst successivi
  - Linux implementa la *predilezione forte* (*hard affinity*): dispone di chiamate di sistema con cui specificare che un processo non può abbandonare un dato processore
  - Solaris supporta la *predilezione debole* (*soft affinity*): non garantisce che, per particolari condizioni di carico, i processi non subiscano spostamenti

Multielaborazione simmetrica – 2

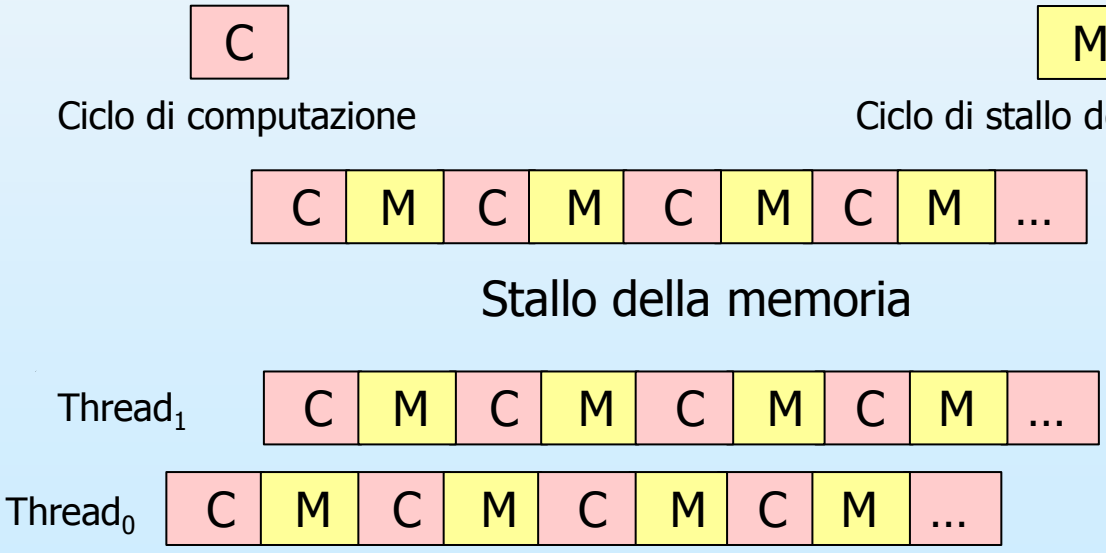
- Bilanciamento del carico – Si mira ad una ripartizione uniforme del carico di lavoro sui diversi processori
  - Nei sistemi con ready queue comune è automatico
  - Migrazione guidata (*push migration*): un processo dedicato controlla periodicamente il carico dei processori per effettuare eventuali riequilibri
  - Migrazione spontanea (*pull migration*): un processore inattivo sottrae ad uno sovraccarico un processo in attesa
  - Linux le implementa entrambe: esegue il proprio algoritmo di bilanciamento ad intervalli regolari (200 msec) e ogniqualvolta si svuota la coda di esecuzione di un processore
  - Il bilanciamento del carico è antitetico rispetto alla filosofia di predilezione del processore

Processori multicore – 1

- Tradizionalmente i sistemi SMP hanno reso possibile la concorrenza fra thread con l'utilizzo di diversi processori fisici
- I sistemi SMP su processori multicore sono più veloci e consumano meno energia
- Quando un processore accede alla memoria, una quantità significativa di tempo (fino al 5%) trascorre in attesa della disponibilità dei dati: stallo della memoria
  - Progetti hardware recenti (*hyperthreading*) implementano unità di calcolo multithread in cui due o più thread hardware sono assegnati ad una singola CPU

Processori multicore – 2

- Dal punto di vista del SO, ogni thread hardware appare come un processore logico in grado di eseguire un thread software
  - Esempio: sistema dual core, dual threading, il SO vede quattro processori logici



Sistema multicore e multithread

Processori multicore – 3

- Un processore multithread e multicore richiede due livelli di scheduling
  - Scheduling effettuato dal SO per stabilire quale thread software mandare in esecuzione su ciascun thread hardware → algoritmi standard
  - Scheduling relativo a ciascun core per decidere quale thread hardware mandare in esecuzione
    - Round Robin (UltraSPARC T1, 8 core a 4 thread)
    - Priorità o *urgency* (Intel Itanium, 2 core a 2 thread)

Virtualizzazione e scheduling

- Il SO ospitante crea e gestisce le macchine virtuali, ciascuna dotata di un proprio SO ospite e proprie applicazioni
- Ogni SO ospite può essere messo a punto per usi specifici
  - Ogni algoritmo di scheduling che fa assunzioni sulla quantità di lavoro effettuabile in tempo prefissato verrà influenzato negativamente dalla virtualizzazione, perché i singoli SO virtualizzati sfruttano a loro insaputa solo una parte dei cicli di CPU disponibili
  - La virtualizzazione può vanificare i benefici di un buon algoritmo di scheduling implementato dal SO ospite sulla macchina virtuale

Scheduling real–time

- Sistemi hard real–time: richiedono il completamento dei processi critici entro un intervallo di tempo predeterminato e garantito
- Sistemi soft real–time: richiedono solo che ai processi critici sia assegnata una maggiore priorità rispetto ai processi di routine

Scheduling in Linux – 1

- Prima della versione 2.5 del kernel, era una variante dell'algoritmo tradizionale di UNIX (code multiple con feedback gestite perlopiù con RR)
  - Basato su priorità interne: dipendente dall'uso della CPU che fanno i processi (più è grande, minore è la priorità)
  - Periodicamente il kernel calcola quanta CPU un processo ha consumato dall'ultimo controllo: questa quantità è "proporzionale" alla priorità del processo fino al prossimo controllo (l'aging si realizza "naturalmente")
  - Poco scalabile e non adeguato ai sistemi SMP
- Ora: scheduling in tempo costante, $\mathcal{O}(1)$, a prescindere dal numero di task nel sistema
  - Adatto a SMP: bilanciamento del carico e predilezione per il processore
- Due algoritmi: time–sharing e real–time

Scheduling in Linux – 2

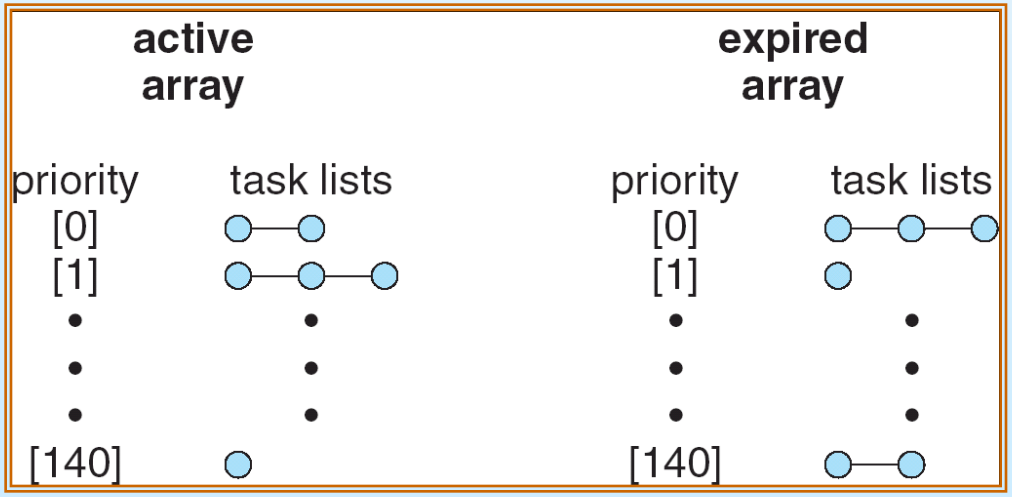
- Scheduling con prelazione
  - Quanti di tempo più lunghi per i task a priorità più alta

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

Relazione fra priorità e lunghezza del quanto di tempo

Scheduling in Linux – 3

- Time–sharing (task *nice*, priorità 100–140)
  - Un task pronto per l'esecuzione è considerato "attivo" dalla CPU se non ha ancora esaurito il suo quanto di tempo
  - Una volta esaurito il quanto di tempo, il task viene considerato "scaduto" e non verrà più eseguito finché tutti gli altri task del sistema siano scaduti
    - La *runqueue* contiene due array di priorità *attivo* e *scaduto*



Scheduling in Linux – 4

- Quando l'array *attivo* è vuoto, gli array delle priorità si scambiano i ruoli
- Priorità dinamica (+5,–5) basata sui punteggi assegnati al task in base a tempo di attesa di I/O: i task maggiormente interattivi hanno tempi di attesa più lunghi
  - ⇨I task fortemente interattivi sono candidati ad ottenere bonus e quindi ad aumentare la loro priorità
- La priorità dinamica di un task viene ricalcolata al suo passaggio nell'array *scaduto*
  - Quando gli array *attivo* e *scaduto* si scambiano i ruoli, tutti i task del nuovo array *attivo* avranno ricevuto nuove priorità basate sulla "storia" dei task stessi

Scheduling in Linux – 5

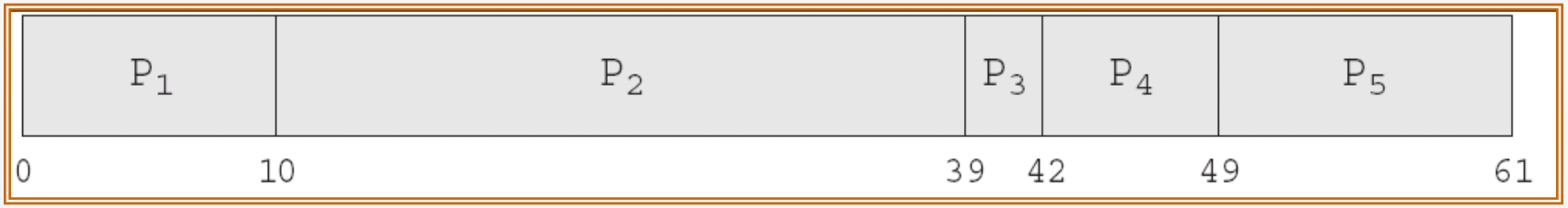
- Real–time (priorità 0–99)
  - Implementa lo standard POSIX.1b
    - Priorità (esterne) statiche
  - Soft real–time
    - Le priorità relative dei processi di elaborazione in tempo reale sono assicurate, ma il nucleo non fornisce garanzia sui tempi di attesa nella coda dei processi pronti
    - Se un segnale d'interruzione dovesse rendere eseguibile un processo real–time mentre il nucleo è impegnato nell'esecuzione di una chiamata di sistema, il processo attende

Valutazione degli algoritmi – 1

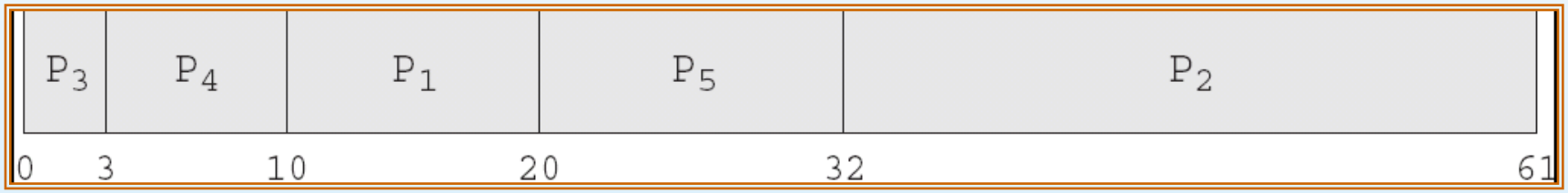
- Modelli deterministici – valutazione analitica, che considera un carico di lavoro predeterminato e definisce le prestazioni di ciascun algoritmo per quel carico di lavoro
  - Esempio 7

Processo	Durata
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

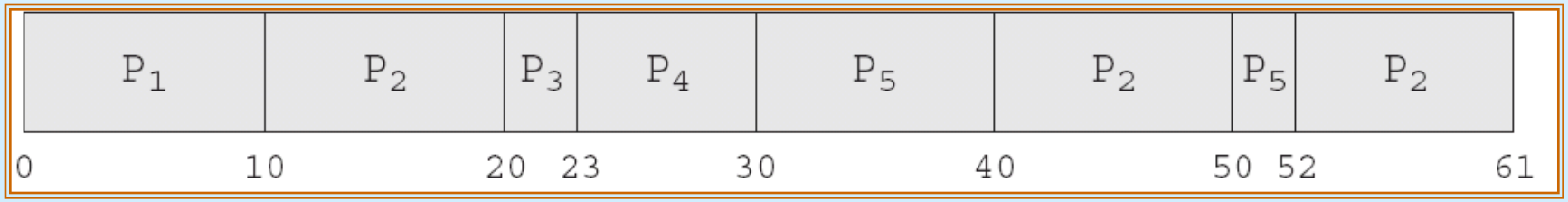
Valutazione degli algoritmi – 2



- FCFS: $T_a=(0+10+39+42+49)/5=28$



- SJF (non preemptive): $T_a=(10+32+0+3+20)/5=13$



- RR con quanto di tempo =10: $T_a=(0+32+20+23+40)/5=23$

Valutazione degli algoritmi – 3

- Reti di code – il sistema di calcolo viene descritto come una rete di server, ciascuno con una coda d'attesa
  - La CPU è un server con la propria coda dei processi pronti, ed il sistema di I/O ha le sue code dei dispositivi
  - Se sono noti l'andamento degli arrivi e dei servizi (sotto forma di distribuzioni di probabilità), si può calcolare l'utilizzo di CPU e dispositivi, la lunghezza media delle code, il tempo medio d'attesa, etc.
  - Problema: difficoltà nell'utilizzo di distribuzioni complicate/metodi matematici appositi ⇨ modelli non realistici

Valutazione degli algoritmi – 4

Esempio 8

- Sia n la lunghezza media di una coda e siano W il tempo medio di attesa nella coda e λ l'andamento medio di arrivo dei nuovi processi
- Se il sistema è stabile, il numero di processi che lasciano la coda deve essere uguale al numero di processi che vi arrivano

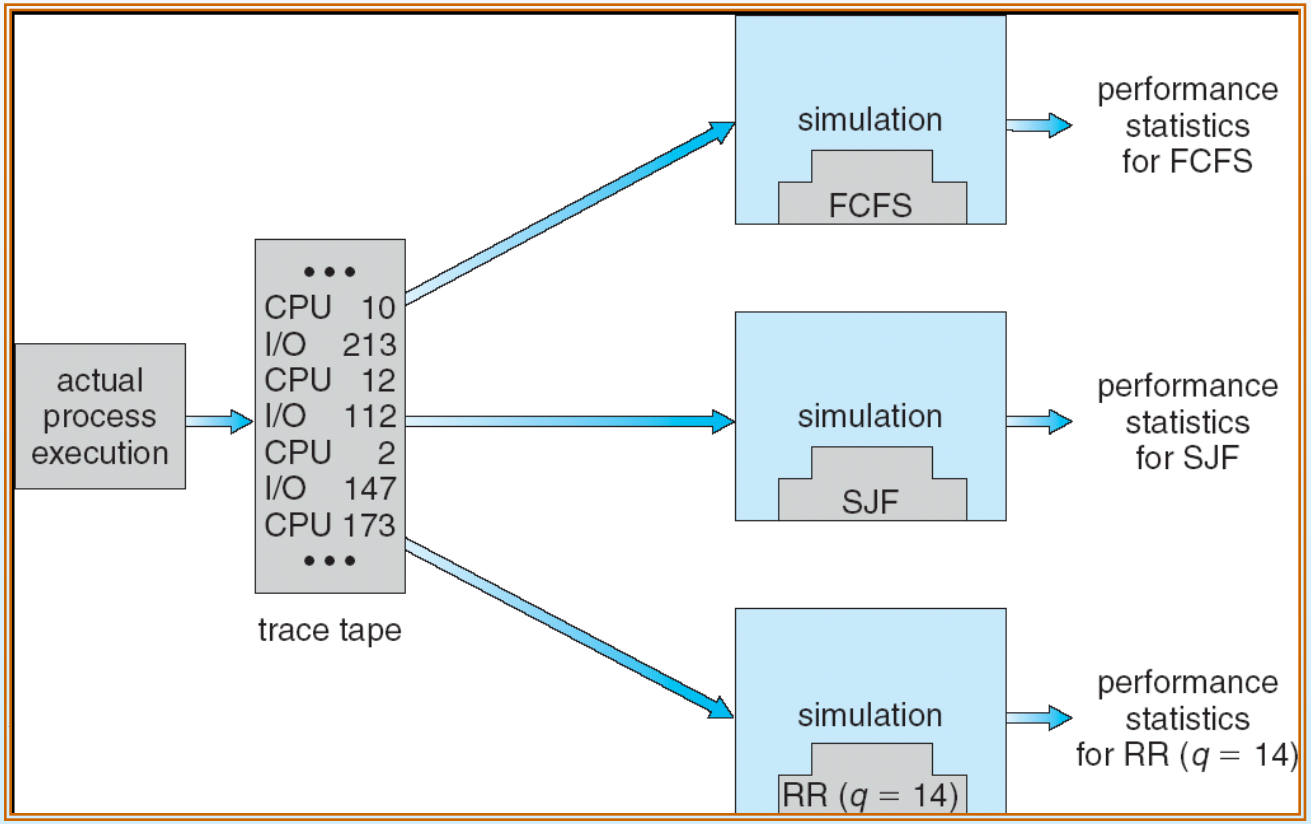
$$n = \lambda \times W$$

  è la formula di Little, valida per qualsiasi algoritmo di scheduling e distribuzione degli arrivi
- È utilizzabile per il calcolo di una delle tre variabili, quando sono note le altre due: per esempio, se ogni secondo arrivano 7 processi e la coda ne contiene mediamente 14, il tempo medio d'attesa per ogni processo è pari a 2 secondi

Valutazione degli algoritmi – 5

- Simulazione – implica la programmazione di un modello del sistema di calcolo; le strutture dati rappresentano gli elementi principali del sistema
  - Il simulatore dispone di una variabile che rappresenta il clock e modifica lo stato del sistema in modo da descrivere le attività dei dispositivi, dei processi e dello scheduler
  - Durante l'esecuzione della simulazione, si raccolgono e si stampano statistiche che descrivono le prestazioni degli algoritmi
  - I dati per la simulazione possono essere generati artificialmente o raccolti da un sistema reale mediante un *trace tape*

Valutazione degli algoritmi – 5



Valutazione di algoritmi di scheduling della CPU tramite simulazione e uso di trace tape

Valutazione degli algoritmi – 6

- Realizzazione – implica la codifica effettiva degli algoritmi di scheduling da valutare, ed il loro inserimento nel sistema operativo, per osservarne il comportamento nelle condizioni reali di funzionamento del sistema
  - Difficoltà nel fare accettare agli utenti un sistema in continua evoluzione

Considerazioni finali

- Gli algoritmi di scheduling più flessibili sono quelli che possono essere tarati dagli amministratori di sistema, che li adattano al particolare ambiente di calcolo, con la propria specifica gamma di applicazioni
- Molti degli attuali sistemi operativi UNIX–like forniscono all'amministratore la possibilità di calibrare i parametri di scheduling in vista di particolari configurazioni del sistema (es. Solaris)

Esercizi

Esercizio 1

Cinque lavori, indicati con le lettere da A a E, arrivano al calcolatore approssimativamente allo stesso istante. I processi hanno un tempo di esecuzione stimato di 8, 10, 2, 4 e 8 secondi, rispettivamente, mentre le loro priorità (determinate esternamente) sono 2, 4, 5, 1, 3 (con 5 priorità max). Per ognuno dei seguenti algoritmi di scheduling

- Round robin (2 sec)
- Scheduling a priorità (non preemptive)
- FCFS
- SJF (non preemptive)

si calcoli il tempo medio di turnaround. Si ignori l'overhead dovuto al cambio di contesto.

Esercizi (Cont.)

- ## Esercizio 2

Si consideri il seguente insieme di processi:

Processo	Tempo di arrivo	Tempo di burst (msec)
P_0	0	7
P_1	2	4
P_2	3	4
P_3	5	2
P_4	7	3
P_5	10	2

I processi arrivano nell'ordine indicato. Si calcoli il tempo medio di turnaround con scheduling SJF preemptive.

Esercizi (Cont.)

Esercizio 3

Si consideri il seguente insieme di processi:

Processo	Tempo di arrivo	Tempo di burst (msec)
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Si calcoli il tempo medio di attesa ed il tempo medio di turnaround, nel caso di scheduling FCFS, RR con quanto di tempo 1 e 4 e SJF non preemptive.

Esercizi (Cont.)

Esercizio 4

Si consideri un sistema con scheduling a priorità con tre code, A, B, C, di priorità decrescente e con prelazione tra code. Le code A e B sono servite con modalità RR con quanto di 15 e 20 ms, rispettivamente; la coda C è FCFS. Se un processo nella coda A o B consuma il suo quanto di tempo, viene spostato in fondo alla coda B o C, rispettivamente. Si supponga che i processi P_1, P_2, P_3, P_4 arrivino rispettivamente nelle code A, C, B, A, con CPU burst e tempi indicati nella tabella seguente:

Processo	Tempo di arrivo	Tempo di burst (msec)
P_1	0	20
P_2	10	25
P_3	16	20
P_4	25	20

Si tracci il diagramma di Gantt relativo all'esecuzione dei quattro processi e si calcoli il tempo medio di turnround.

Esercizi (Cont.)

Esercizio 5

Si considerino i seguenti processi, attivi in un sistema multiprogrammato:

Processo	Tempo di arrivo	Tempo di burst (msec)	Priorità
P_1	0	8	2
P_2	1	6	3
P_3	2	3	2
P_4	3	4	2
P_5	4	3	2

Descrivere la sequenza di esecuzione dei job, tramite diagramma di Gantt e calcolare l'istante in cui ogni job completa il suo task, ed il relativo tempo di turnaround, ottenuto mediante scheduling a code multiple a priorità e feedback, dove ogni coda è gestita con la strategia FCFS. Il feedback è definito come segue: la priorità diminuisce di 1 (fino al livello base 1) se si passano più di 6 unità di tempo (consecutive) in esecuzione ed aumenta di 1 per ogni 6 unità di tempo passate in attesa in una qualsiasi coda.

Esercizi (Cont.)

- **Esercizio 6**

  Si considerino N ($N \geq 2$) processi in contesa su una singola CPU gestita con scheduling Round Robin. Supponiamo che ciascun context switch abbia una durata di S msec e che il quanto di tempo sia pari a Q. Per semplicità, si assuma inoltre che i processi non siano mai bloccati a causa di un qualche evento e semplicemente passino dall'essere in esecuzione all'essere in attesa nella ready queue.

  Si calcoli il massimo quanto di tempo Q tale che, per ciascun processo, non trascorrano più di T msec tra l'inizio di due burst successivi.

  Si calcoli il massimo quanto di tempo Q tale che ciascun processo non debba attendere più di T msec nella ready queue.

  I valori di Q dovranno essere calcolati in funzione di N, S e T.

Esercizi (Cont.)

Esercizio 7

Si considerino i processi P_1, P_2 e P_3. Ciascun processo esegue un CPU–burst ed un I/O–burst, quindi nuovamente un CPU–burst ed un I/O–burst ed infine un ultimo CPU–burst. La lunghezza dei burst ed il tempo di arrivo dei processi (in millisecondi) è riportato in tabella:

Processo	Burst1	I/O_1	Burst2	I/O_2	Burst3	Arrivo
P_1	2	4	2	2	2	0
P_2	2	2	3	3	1	1
P_3	1	2	1	1	1	1

Si disegnino i diagrammi di Gantt che illustrano l'esecuzione dei tre processi utilizzando FCFS ed RR con quanto di tempo pari ad 1 e 2. Se il termine di un servizio di I/O ed un timeout della CPU si verificano nello stesso istante, si assegni la precedenza al processo che ha appena terminato il proprio I/O. Si calcoli il tempo medio di attesa ed il tempo medio di turnaround nei tre casi.