



Corso di Laurea Triennale in Informatica
Università Degli Studi della Basilicata

Sistemi Operativi

Docente:
ugo.erra@unibas.it

5° Lezione

Scheduling della CPU

Sommario della lezione



- Introduzione allo scheduling
- Scheduler della CPU
 - ▣ Dispatcher
- Criteri di scheduling
- Algoritmi di scheduling
- Scheduling per multiprocessori

Introduzione allo scheduling

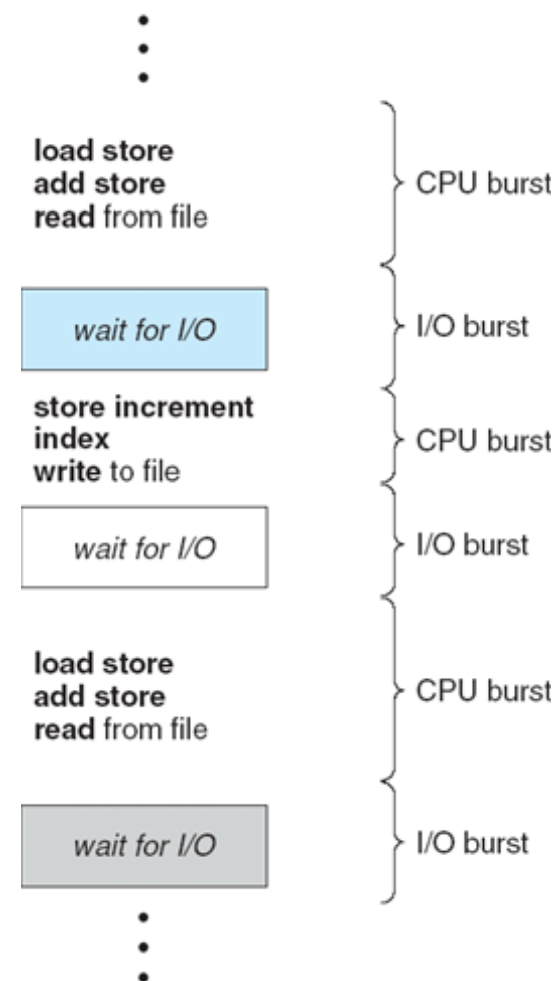
- Lo scheduling è una delle operazioni fondamentali dei moderni sistemi operativi multi-programmati
- L'operazione di scheduling permette di sfruttare al meglio le risorse di un sistema di calcolo rendendolo di fatto più produttivo
- Sebbene si parli di scheduling dei processi gli oggetti su cui opera lo scheduling sono i thread
- I termini scheduling dei processi e scheduling dei thread sono dunque equivalenti

Idea dello scheduling

- In un sistema monoprocesso un solo processo viene eseguito per volta
- Un processo nell'arco della sua esecuzione alterna due fasi:
 - ▣ **CPU burst** (utilizzo della CPU)
 - ▣ **I/O burst** (attesa di input/output da un dispositivo)
- Poiché durante una fase di I/O burst la CPU sarebbe inutilizzata l'idea di base è di utilizzare questi tempi di attesa assegnando la CPU ad un altro processo
- In questo modo più processi possono essere mantenuti in memoria in attesa che venga loro assegnato il controllo della CPU

Fasi CPU burst – I/O burst

- Un processo consiste in un **ciclo** di elaborazione svolto dalla CPU e da un tempo di attesa da un dispositivo di I/O
- In fase di lancio un processo esegue un lungo ciclo di utilizzo della CPU
- Successivamente le fase si alternano tra CPU burst e I/O burst
- Processi con prevalenza di I/O (*I/O bound*) producono frequenti accessi alla CPU ma di breve durata
- Processi con prevalenza di CPU (*CPU bound*) producono pochi accessi alla CPU ma molto lunghi



Frequenza dei CPU burst

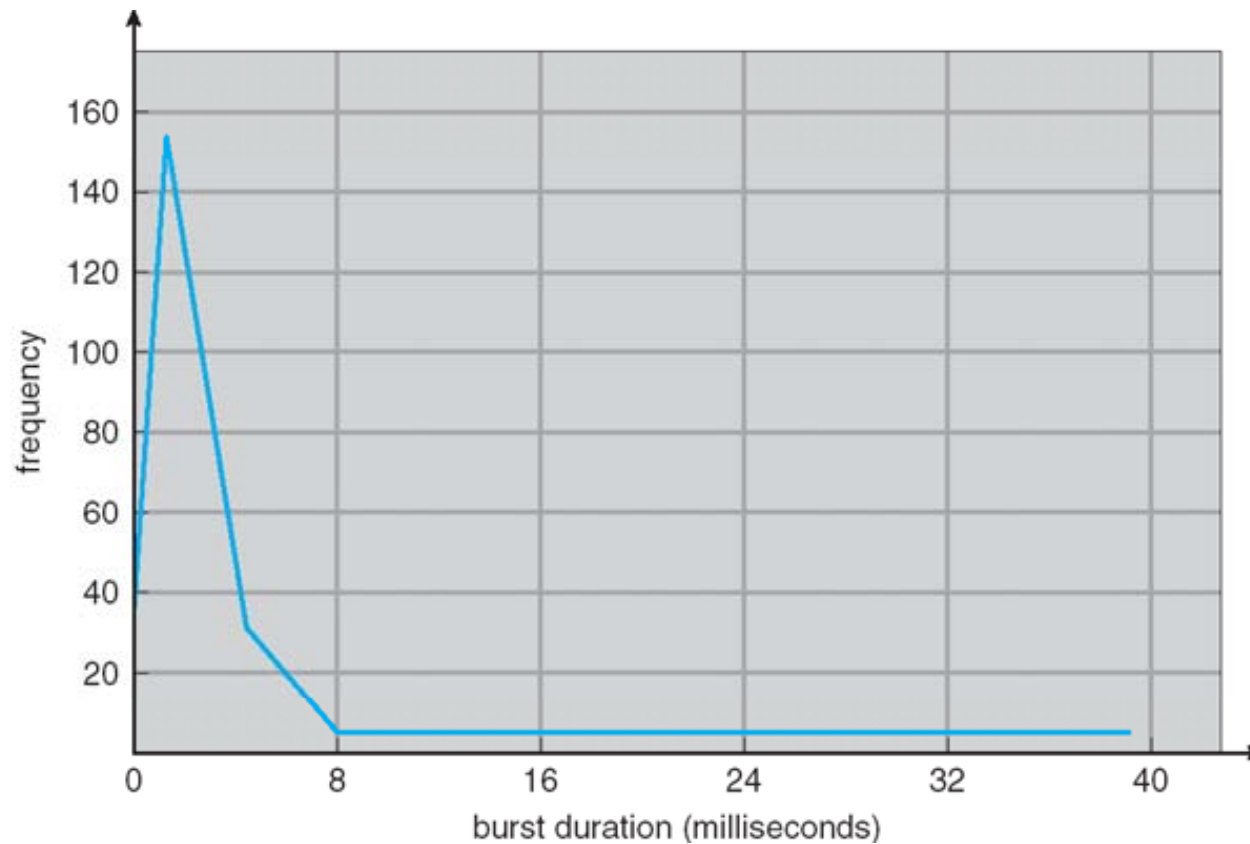
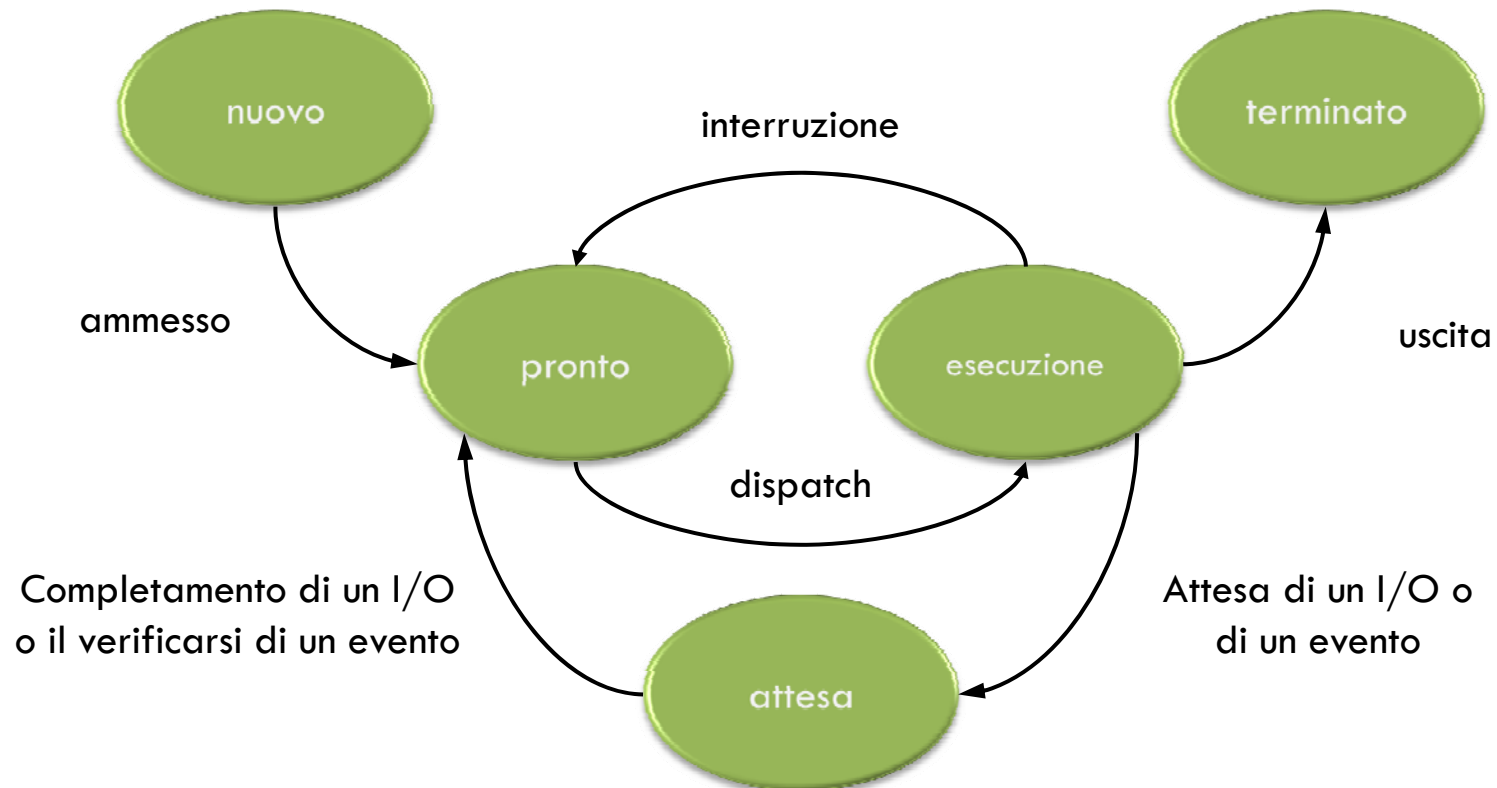
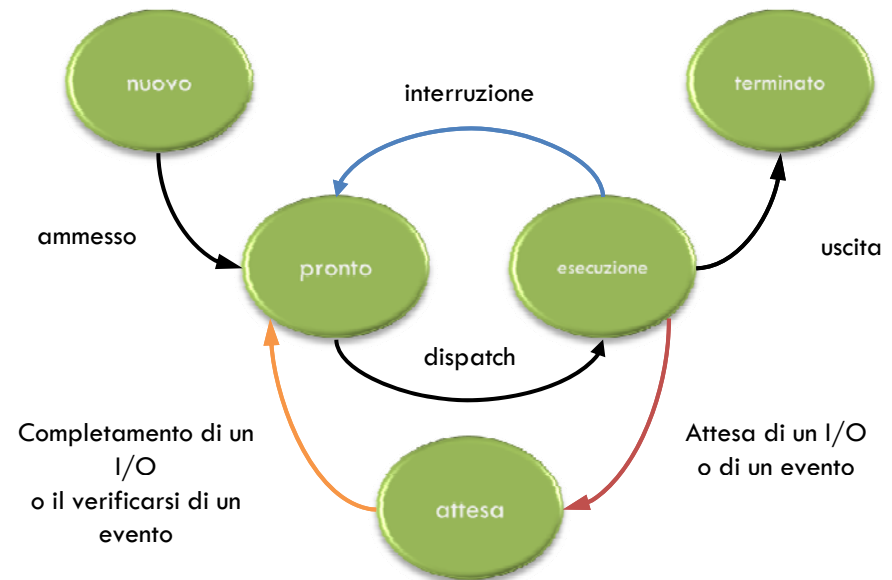


Diagramma di transizione degli stati di un processo



Scheduler della CPU

- Lo **scheduler a breve termine** della CPU seleziona tra tutti i processi presenti in memoria e pronti per essere eseguiti quello a cui assegnare la CPU
- Lo scheduler della CPU può prendere una decisione nelle seguenti condizioni:
 1. Un processo passa dallo stato di **esecuzione allo stato di attesa**
 2. Un processo passa dallo stato di **esecuzione allo stato di pronto**
 3. Un processo passa dallo stato di **attesa allo stato di pronto**
 4. Un processo termina



Tipi di scheduler della CPU

- Se lo scheduler interviene solo nelle fasi 1 e 4 allora è chiamato **senza diritto di prelazione** (*nonpreemptive*) o cooperativo (*cooperative*)
 - ▣ Windows 3.x
- Altrimenti è chiamato **con diritto di prelazione** (*preemptive*)
 - ▣ Windows 95, XP, Mac OS X, Linux

Problemi dello scheduler con prelazione

- Alcuni problemi dello scheduler con prelazione:
 - ▣ Due processi che lavorano su dati condivisi potrebbero avere problemi di incoerenza nel momento in cui uno dei due processi viene “prelazonato” mentre sta modificando i dati
 - ▣ Durante una chiamata di sistema il processo potrebbe essere prelazonato mentre il sistema operativo è in modalità kernel
 - ▣ Durante una interruzione diversi processi potrebbero accedere in modo concorrente al codice di una interruzione

Dispatcher

- Il **dispatcher** è il modulo del sistema operativo che si occupa di passare alla CPU il processo scelto dallo scheduler a breve termine
- Le operazioni del dispatcher sono:
 - ▣ Il cambio di contesto
 - ▣ Il passaggio alla modalità utente
 - ▣ Il salto alla giusta posizione del programma utente per riavviare l'esecuzione
- La **latenza di dispatch** è definito come il tempo necessario per fermare un processo e avviare l'esecuzione di un altro processo

Criteri di scheduling

- Alla base della scelta di un algoritmo di scheduling ci sono diversi criteri:
 - ▣ **Utilizzo della CPU**
 - Indica la percentuale di occupazione della CPU
 - ▣ **Produttività**
 - Numero di processi completati nell'unità di tempo (*throughput*)
 - ▣ **Tempo di completamento**
 - Il tempo impiegato al processo per completare l'esecuzione comprensivi dei tempi di attesa per entrare in memoria, nella coda dei processi pronti, tempo di esecuzione sulla CPU e nella operazioni di I/O (*turnaround time*)
 - ▣ **Tempo d'attesa**
 - Somma degli intervalli di tempo passati nella coda dei processi pronti
 - ▣ **Tempo di risposta**
 - Tempo trascorso dalla sottomissione di un processo fino alla sua prima risposta (escluso il tempo di output del dispositivo)

Cosa ottimizzare?

- Massimizzare l'utilizzo della CPU
 - ▣ Tenere occupata quanto più è possibile la CPU
- Massimizzare la produttività
 - ▣ Svolgere quanti più lavori nell'unità di tempo
- Minimizzare il tempo di completamento
 - ▣ Ridurre le fasi di attesa nella coda e l'esecuzione
- Minimizzare il tempo di attesa
 - ▣ Ridurre gli intervalli di tempo che un processo resta nella coda dei processi pronti (in genere si considera la media)
- Minimizzare il tempo di risposta
 - ▣ Produrre quanto prima un risultato per garantire un buon servizio agli utenti

Algoritmi di scheduling



- ❑ **First-come, first-served (FCFS)**
- ❑ Shortest-job-first (SJF)
- ❑ Scheduling per priorità
- ❑ Round Robin (RR)
- ❑ Scheduling a code multiple
- ❑ Scheduling a code multiple con retroazione

Scheduling First-Come First-Server

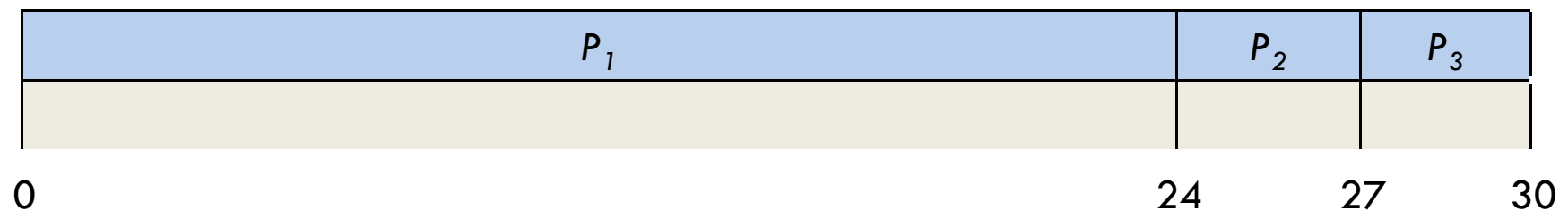
- Lo **scheduling in ordine di arrivo** (*scheduling first-come, first-served, FCFS*) assegna la CPU al primo processo che la richiede
- La realizzazione è affidata da una coda FIFO (First In First Out)
 - ▣ Il PCB viene collegato all'ultimo elemento della coda
- Quando la CPU è libera si assegna al primo processo in testa alla coda dei processi pronti

Esempio di Scheduling FCFS

- Supponiamo di avere i seguenti processi

Processo	Durata della sequenza
P_1	24
P_2	3
P_3	3

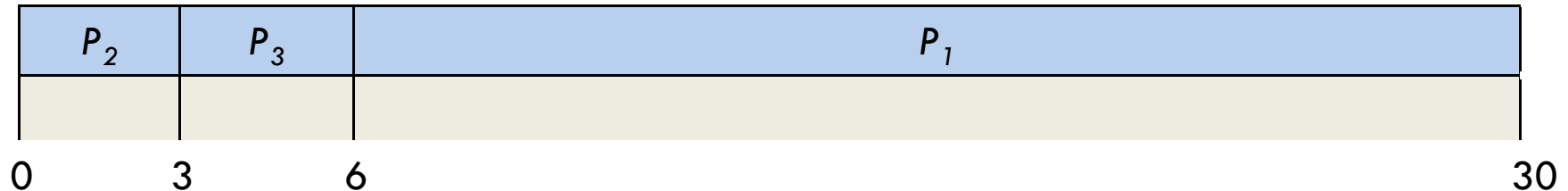
- Il **diagramma di Gantt** esprime l'ordine di arrivo dei processi con i tempi di attesa
- Supponendo che i processi arrivano nell'ordine P_1 , P_2 e P_3 avremo:



- Il tempo medio di attesa sarà
$$(0+24+27)/3 = 17 \text{ millisecondi}$$

Esempio Scheduling First-Come First-Server

- Supponendo che i processi arrivano invece nell'ordine P_2, P_3 e P_1 avremo:



- Il tempo medio di attesa sarà
$$(0+3+6)/3 = 3 \text{ millisecondi}$$
- Quindi il tempo medio di attesa varia in base alla durata delle sequenze di operazioni della CPU dei processi e di conseguenza anche in base all'ordine di arrivo

FCFS senza prelazione

- Nello scheduling FCFS si verifica l'**effetto convoglio** quando un processo occupa per molto tempo la CPU e numerosi processi al termine di un I/O si accodano
- FCFS è senza prelazione, la CPU viene rilasciata dal processo dopo che
 - ▣ Termina l'esecuzione
 - ▣ Richiede una operazione di I/O
- Non è un buon scheduler negli ambienti multi-utente in cui ad ogni utente deve essere garantito un tempo di utilizzo della CPU

Algoritmi di scheduling



- First-come, first-served (FCFS)
- **Shortest-job-first (SJF)**
- Scheduling per priorità
- Round Robin (RR)
- Scheduling a code multiple
- Scheduling a code multiple con retroazione

Shortest-Job-First

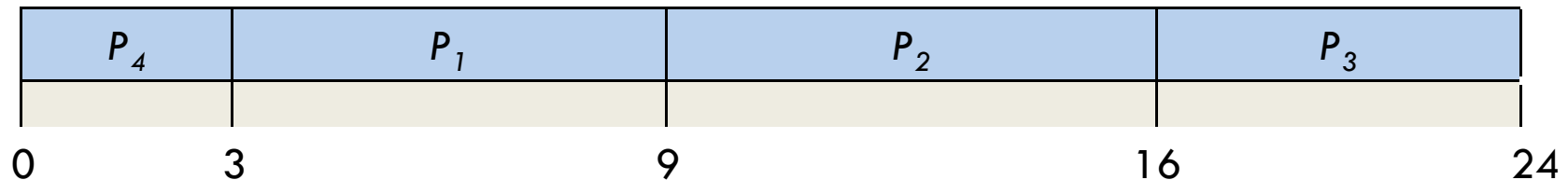
- Lo **scheduling per brevità** (*shortest-job-first*, SJF) assegna la CPU al processo con il CPU burst minore
 - ▣ Se due processi hanno lo stesso tempo di occupazione della CPU si utilizza uno scheduler FCFS
- L'algoritmo SJF può essere:
 - ▣ *Senza prelazione* – una volta che la CPU è stata assegnata non può essere prelazionata finché il processo non completa il suo CPU burst
 - ▣ *Con prelazione* – se arriva un nuovo processo con CPU burst inferiore a quello attualmente in esecuzione il processo viene prelazonato. Questo schema è chiamato *Shortest-Remaining-Time-First* (SRTF)
- SJF è ottimale
 - ▣ Il tempo media di attesa è minimo per un dato insieme di processi

Esempio Shortest-Job-First

- Supponiamo di avere i seguenti processi

Processo	Durata della sequenza
P ₁	6
P ₂	8
P ₃	7
P ₄	3

- Con lo scheduler SJF i processi sarebbero ordinati nel seguente modo:



- Il tempo medio di attesa è

$$(0 + 3 + 9 + 16) / 4 = 7 \text{ millisecondi}$$

Media esponenziale - 1

- La difficoltà nello SJF è determinare la durata del successivo CPU burst
- Non è possibile conoscere la lunghezza del prossimo CPU burst ma è possibile predire il suo valore attraverso la seguente **media esponenziale**:

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

dove:

- ▣ t_n è la lunghezza dell' n -esima sequenza di operazioni della CPU
- ▣ τ_{n+1} il valore previsto per la successiva sequenza di operazioni della CPU
- ▣ Il valore di α è compreso tra $0 \leq \alpha \leq 1$

Media esponenziale - 2

- Media esponenziale

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

- Al variare del parametro α otteniamo:

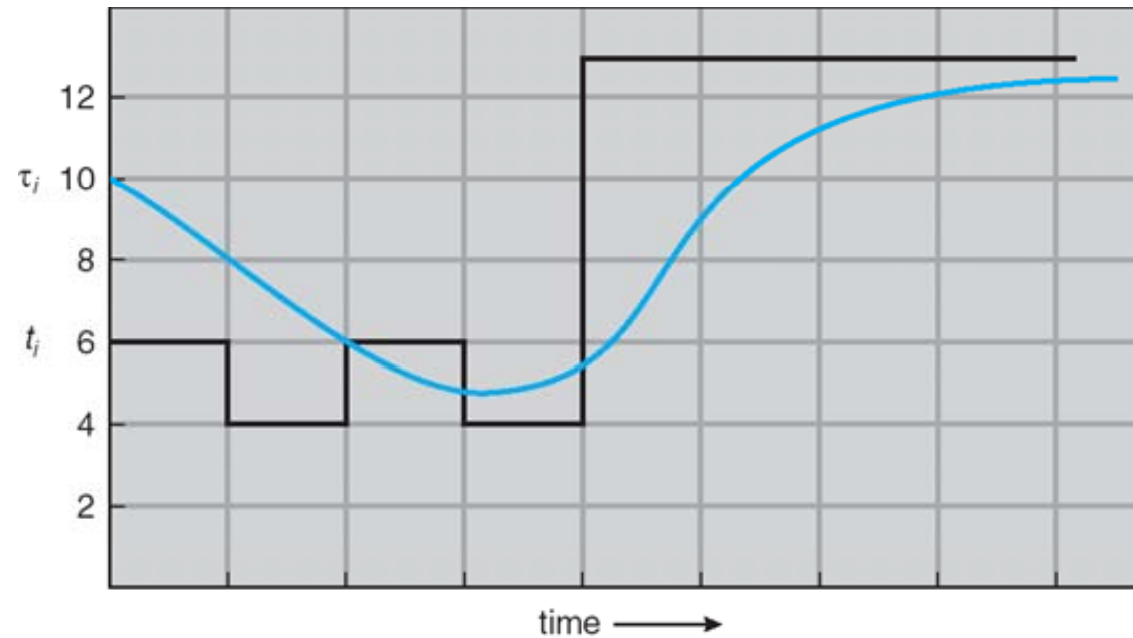
- ▣ Se $\alpha = 0$ allora la storia recente non ha effetto ($\tau_{n+1} = \tau_n$)
- ▣ Se $\alpha = 1$ allora diamo importanza solo alla storia recente ($\tau_{n+1} = t_n$)
- ▣ Se $\alpha = 1/2$ allora la storia recente e la storia passata hanno lo stesso peso

- Sviluppando la formula per τ_{n+1} otteniamo

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

poiché i valori di α ed $(\alpha - 1)$ sono minori o uguali ad 1 i termini passati hanno meno peso

Predizione della lunghezza del successivo CPU burst



CPU burst (t_i)	6	4	6	4	13	13	13	...	
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

$$\tau_0 = 10$$

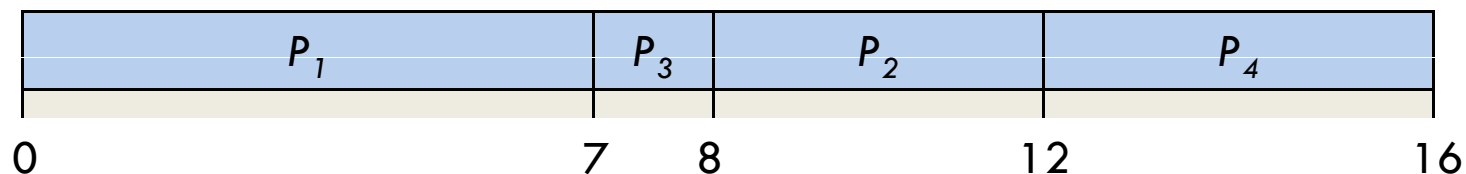
$$\alpha = 1/2$$

Esempio Shortest-Job-First (senza prelazione)

- Supponiamo di avere i seguenti processi

Processo	Istante di arrivo	Durata della sequenza
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4

- Con lo scheduler SJF *senza prelazione* i processi sarebbero ordinati nel seguente modo:



- Il tempo medio di attesa è

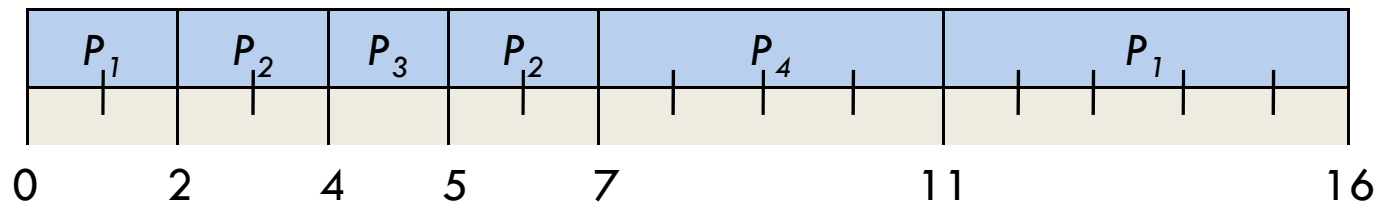
$$(0 + 3 + 6 + 7) / 4 = 4 \text{ millisecondi}$$

Esempio Shortest-Job-First (con prelazione)

- Supponiamo di avere i seguenti processi

Processo	Istante di arrivo	Durata della sequenza
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4

- Con lo scheduler SJF con *prelazione* i processi sarebbero ordinati nel seguente modo:



- Il tempo medio di attesa è

$$(9 + 1 + 0 + 2) / 4 = 3 \text{ millisecondi}$$

Algoritmi di scheduling



- First-come, first-served (FCFS)
- Shortest-job-first (SJF)
- **Scheduling per priorità**
- Round Robin (RR)
- Scheduling a code multiple
- Scheduling a code multiple con retroazione

Scheduling per priorità

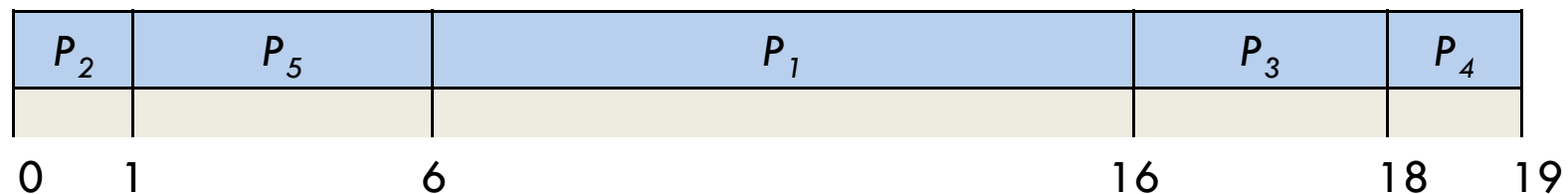
- Nello **scheduling per priorità** associamo un valore di priorità ad ogni processo
 - ▣ Lo SFJ è un caso particolare dello scheduling per priorità
- La CPU è allocata al processo con la priorità più alta
 - ▣ Ad esempio il valore 0 potrebbe indicare una priorità bassa mentre il valore 4095 una priorità alta
 - ▣ In generale non esiste un modo univoco per indicare una priorità alta e la priorità bassa

Esempio Scheduling per priorità

- Supponiamo di avere i seguenti processi

Processo	Durata della sequenza	Priorità
P ₁	10	3
P ₂	1	1
P ₃	2	4
P ₄	1	5
P ₅	5	2

- Con lo scheduler per priorità i processi sarebbero ordinati nel seguente modo:



- Il tempo medio di attesa è

Sistemi Operativi 2008/09 $(0 + 1 + 6 + 16 + 18) / 5 = 8,2$ millisecondi

Assegnazione delle priorità

- Le priorità possono essere assegnate
 - ▣ Internamente: in base al numero di operazioni di I/O e alla lunghezza delle sequenze di operazioni della CPU
 - ▣ Esternamente: importanza del processo, tipo di processo, fondi pagati, politica di assegnazione delle priorità
- Lo scheduling per priorità può essere:
 - ▣ Con prelazione: nel momento in cui arriva un processo con priorità maggiore viene prelazionato il processo attuale in esecuzione
 - ▣ Senza prelazione: l'ultimo processo arrivato viene inserito in testa alla coda dei processi pronti

Starvation

- L'**attesa indefinita** o **starvation** è un problema che si verifica con lo scheduling per priorità
- Un processo con bassa priorità ed in attesa della CPU potrebbe rimanere in attesa per un tempo indefinito a causa di processi con priorità maggiore
 - ▣ Il processo solitamente è eseguito di notte quando la maggior parte dei processi con alta priorità non è in esecuzione
- Una soluzione consiste nell'applicare una sorta di **invecchiamento** (*aging*) in cui la priorità dei processi viene aumentata man mano che passa il tempo

Algoritmi di scheduling



- First-come, first-served (FCFS)
- Shortest-job-first (SJF)
- Scheduling per priorità
- **Round Robin (RR)**
- Scheduling a code multiple
- Scheduling a code multiple con retroazione

Round Robin - 1

- Nell'algoritmo di **scheduling circolare** (*round robin*, RR) ad ogni processo è concesso un **quanto di tempo** o **porzione di tempo** (*time slice*)
 - ▣ Solitamente da 10 a 100 millisecondi
- Al termine del quanto di tempo il processo in esecuzione è prelazionato ed inserito alla fine della coda circolare
- Lo scheduler della CPU scorre la coda dei processi pronti per individuare il primo processo della coda a cui assegnare il prossimo quanto di tempo
- Una volta determinato il processo lo scheduler imposta un timer pari al quanto di tempo ed attiva il dispatcher per l'esecuzione del processo

Round Robin - 2

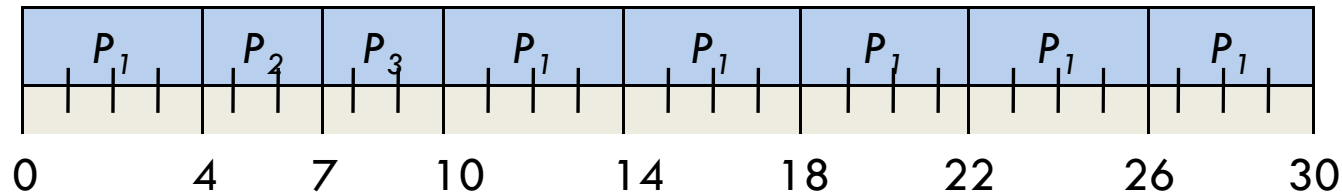
- Durante l'esecuzione del processo possono accadere due cose:
 - ▣ Il processo ha un sequenza di operazioni della CPU minore del quanto di tempo e quindi rilascerà spontaneamente la CPU
 - ▣ Il processo termina il suo quanto di tempo e lo scheduler passa all'esecuzione del prossimo processo
- Se ci sono n processi nella coda dei processi pronti e il quanto di tempo è q allora ogni processo otterrà $1/n$ del tempo della CPU e nessun processo attenderà più di $(n-1)q$ quanti di tempo
- Performance
 - ▣ Se q è molto grande si ottiene lo scheduling FCFS
 - ▣ Se q è piccolo il context switch potrebbe essere prevalente sul tempo totale creando un eccessivo overhead

Esempio Round Robin

- Supponiamo di avere i seguenti processi

Processo	Durata della sequenza
P ₁	24
P ₂	3
P ₃	3

- Con uno scheduler Round Robin ed una durata del quanto di 4 millisecondi abbiamo:



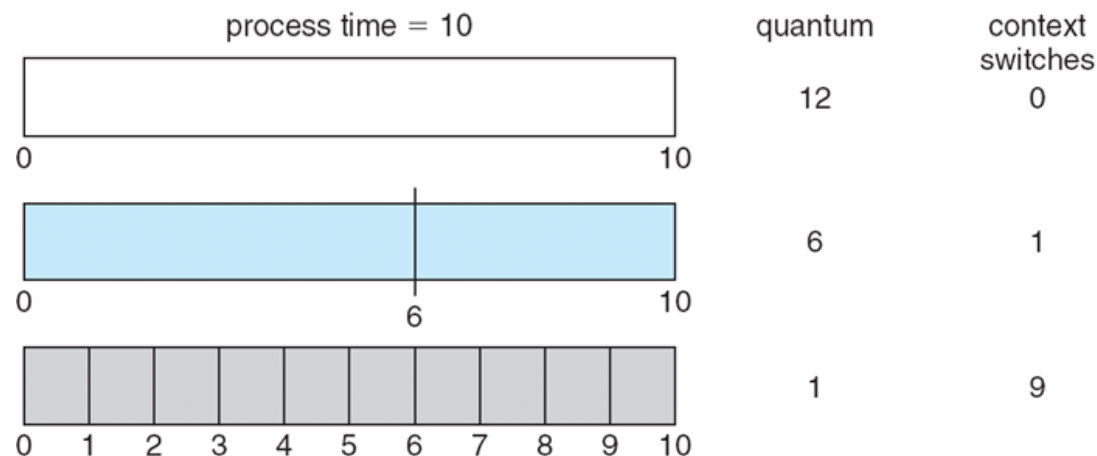
- Il tempo medio di attesa è
 $(6 + 4 + 7) / 3 = 5,66$ millisecondi

Nota:

- P₁ attende in coda 0 + (10 - 4) = 6 millisecondi
- P₂ attende in coda 4 millisecondi
- P₃ attende in coda 7 millisecondi

Dimensione quanto di tempo

- La dimensione del quanto di tempo è fondamentale per evitare overhead dei cambi di contesto
- Ad esempio se un processo dura 10 unità di tempo in base alla durata del quanto di tempo q avremo:
 - ▣ Se $q=12$, nessun cambio di contesto
 - ▣ Se $q=6$, un solo cambio di contesto
 - ▣ Se $q=1$, 9 cambi di contesto



Algoritmi di scheduling



- First-come, first-served (FCFS)
- Shortest-job-first (SJF)
- Scheduling per priorità
- Round Robin (RR)
- **Scheduling a code multiple**
- Scheduling a code multiple con retroazione

Scheduling a code multiple

- Nei sistemi in cui i processi sono classificabili in gruppi diversi è possibile creare code distinte per ogni gruppo
- Una distinzione comune è
 - ▣ Processi eseguiti in **primo piano** (*foreground*) o **interattivi**
 - ▣ Processi eseguiti in **sottofondo** (*background*)
- L'algoritmo di scheduling a code multiple utilizza per ogni coda un algoritmo di scheduling differente, ad esempio:
 - ▣ Per i processi in foreground si può utilizzare un algoritmo RR
 - ▣ Per i processi in background si può utilizzare un algoritmo FCFS
- Lo scheduling deve essere applicato anche tra le diverse code
 - ▣ Tutti i processi in foreground devono essere eseguiti prima dei processi in background
 - ▣ Oppure assegnare ad ogni coda un certa quantità di tempo della CPU che può utilizzare per i diversi processi della coda
 - 80% per i processi foreground usando RR
 - 20% per i processi in background usando FCFS

Esempio Scheduling a code multiple

Priorità più alta



Priorità più bassa

Algoritmi di scheduling



- First-come, first-served (FCFS)
- Shortest-job-first (SJF)
- Scheduling per priorità
- Round Robin (RR)
- Scheduling a code multiple
- **Scheduling a code multiple con retroazione**

Scheduling a code multiple con retroazione

- Nello **scheduling a code multiple con retroazione** i processi possono essere spostati da una coda ad un'altra
 - ▣ In generale non sarebbe possibile perché un processo non può cambiare le proprie caratteristiche
- Si potrebbe ipotizzare un comportamento del genere:
 - ▣ Un processo che utilizza troppi cicli di CPU è spostato su di una coda con priorità più bassa
 - ▣ Processi con prevalenza di I/O sono spostati in code a priorità più alta
 - ▣ Processi che attendono da troppo tempo l'uso della CPU possono essere spostati in code a priorità più alta

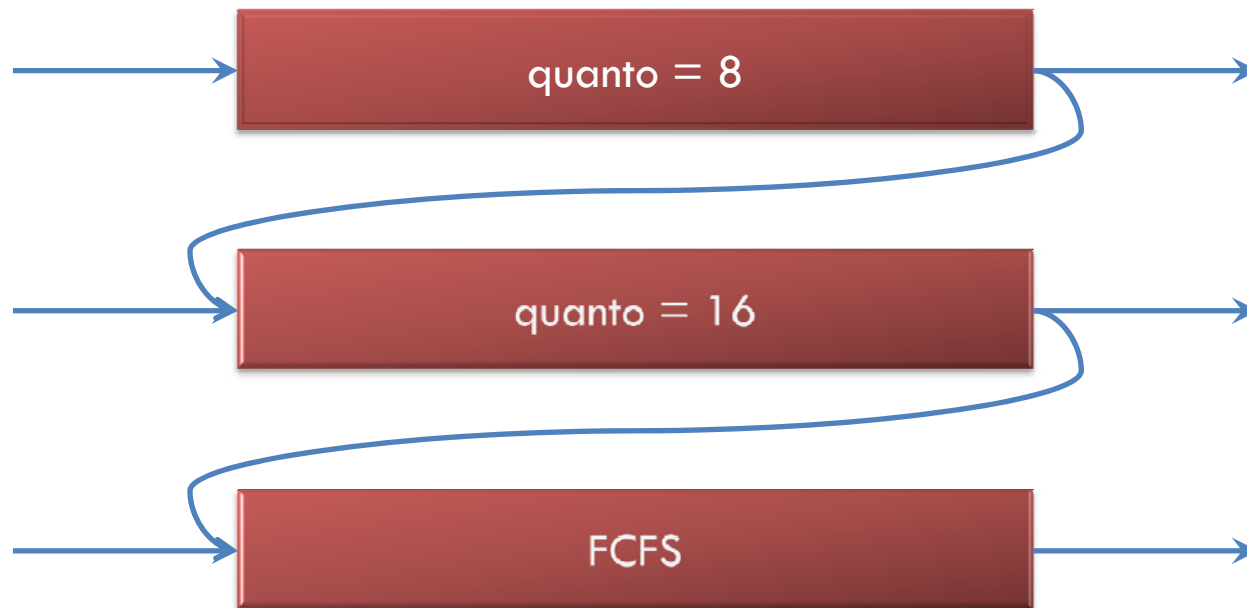
Parametri

- Uno scheduler a code multiple con retroazione è caratterizzato dai seguenti parametri:
 - ▣ Numero di code
 - ▣ Algoritmo di scheduling per ciascuna coda
 - ▣ Metodo utilizzato per spostare un processo su una coda con priorità maggiore
 - ▣ Metodo utilizzato per spostare un processo su una coda con priorità minore
 - ▣ Metodo da utilizzare per decidere in quale coda spostare un processo quando viene lanciato

Esempio Scheduling a code multiple

- Consideriamo uno scheduler con tre code:
 - ▣ Q_0 – servita con RR con un quanto di tempo di 8 millisecondi
 - ▣ Q_1 – servita con RR con un quanto di tempo di 16 millisecondi
 - ▣ Q_2 – servita con FCFS
- Lo scheduler opera nel seguente modo:
 - ▣ Un nuovo processo entra nella coda Q_0 . Quando riceve la CPU se non termina nel quanto di 8 millisecondi è spostato nella coda Q_1
 - ▣ Un processo nella coda Q_1 ricevi 16 millisecondi se non termina entro il quanto assegnato viene spostato nella coda Q_2
 - ▣ I processi nella coda Q_2 sono eseguiti solo se non ci sono processi nella coda Q_1
 - ▣ I processi nella coda Q_1 sono eseguiti solo se non ci sono processi nella coda Q_0
 - ▣ I processi a priorità maggiore hanno diritto di prelazione sui processi a priorità minore
- L'intero scheduler favorisce i processi che non richiedono un uso della CPU per non più di 8 millisecondi

Esempio Scheduling a code multiple



Scheduling per sistemi multiprocessori

- Al crescere del numero di processori è possibile gestire le CPU come risorse da assegnare ai processi ma...
- ...lo scheduling diventa più complesso
- Nei **sistemi omogenei** i processori sono identici tra di loro e quindi qualunque CPU può essere utilizzata
- In alcuni sistemi omogenei può capitare che un processore sia l'unico responsabile dell'I/O quindi i processi devono essere eseguiti su questo processore

Tipi di multielaborazione

- Esistono due tipi di multielaborazione
 - ▣ **Multielaborazione asimmetrica** – un solo processore si occupa di gestire il sistema e gli altri processori eseguono solo il codice
 - ▣ **Multielaborazione simmetrica** (*symmetric multiprocessing, SMP*)- i processi sono mantenuti in una coda comune oppure ogni processore ha un coda propria
 - I dati condivisi devono essere gestiti adeguatamente

Predilezione per il processore

- I processori moderni dispongono di enormi quantità di cache per aumentare le prestazioni
- Se un processo è eseguito su di un processore i dati della cache saranno invalidati nel momento in cui il processo è spostato su di un altro processore
- Per evitare situazioni del genere si utilizza la **predilezione per il processore** (*processor affinity*) in cui un processo “preferisce” essere eseguito su di una particolare CPU
- Può essere di due tipi:
 - ▣ **Predilezione debole** – un processo può cambiare processore
 - ▣ **Predilezione forte** – un processo non può cambiare processore

Bilanciamento del carico



- Nei sistemi SMP è importante che ogni processore sia sempre occupato al pari degli altri
- Il bilanciamento del carico cerca di distribuire uniformemente il lavoro su tutti i processori
 - ▣ Se la coda è comune lavoreranno tutti i processori poiché al termine di un processo sarà immediatamente selezionato dalla coda un nuovo processo

Migrazione di un processo

- Se un processore è sbilanciato sono possibili due soluzioni:
 - ▣ **Migrazione guidata** – un processo si occupa di bilanciare periodicamente il carico di lavoro spostando i processi sui diversi processori
 - ▣ **Migrazione spontanea** – un processore poco carico sottrae processi ad un processore troppo carico
- Linux utilizza un approccio ibrido:
 - ▣ Utilizza la migrazione guidata ogni 200 millisecondi
 - ▣ Utilizza la migrazione spontanea quando un processore ha svuotato la propria coda