Scheduling della CPU

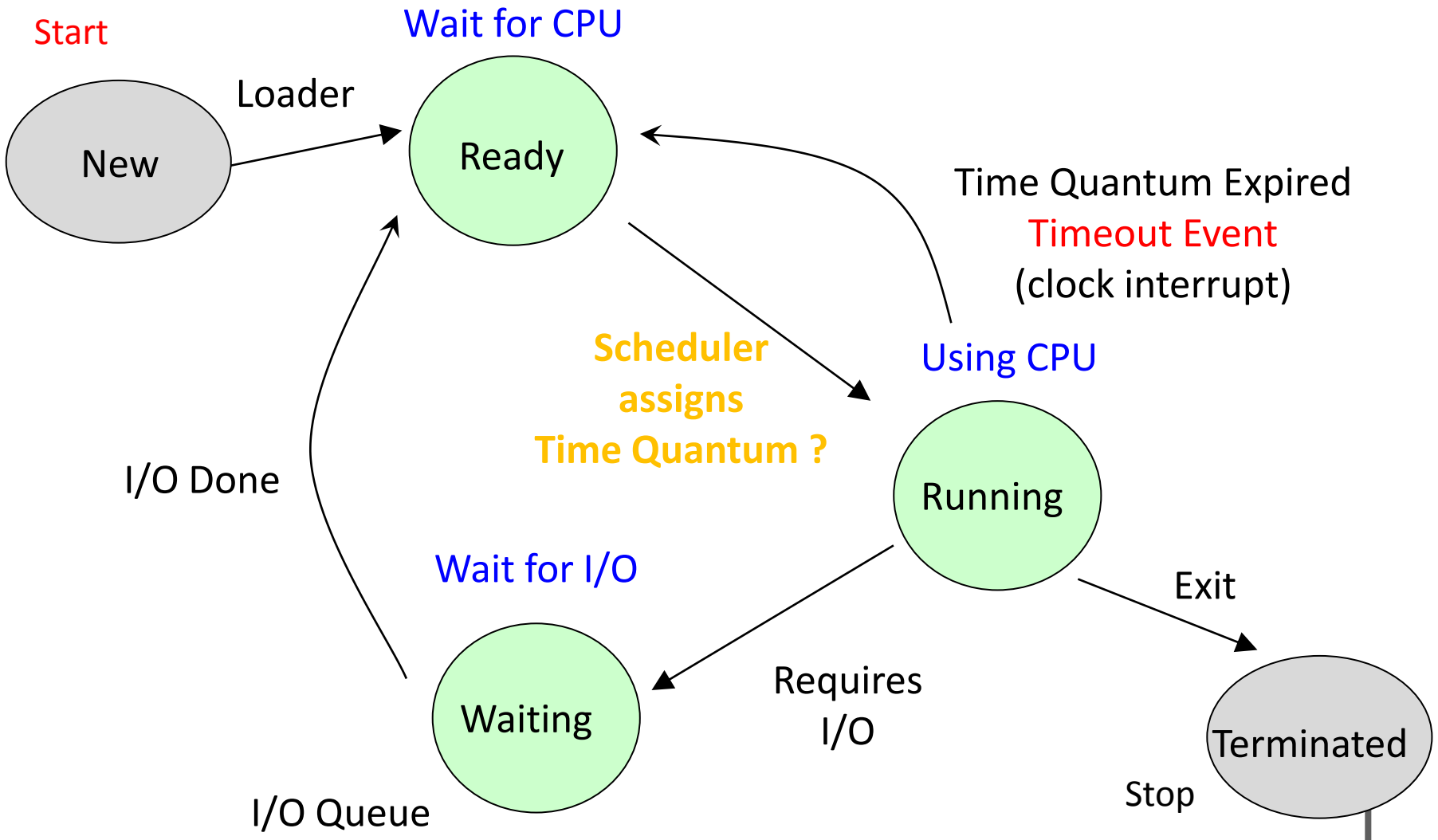
Sommario

- Scheduling
  - Concetti di base
  - Criteri di Scheduling
  - Algoritmi di Scheduling
  - ~~Scheduling per i thread~~
  - ~~Esempi di scheduler (solaris, windows, linux).~~
  - Valutazione degli algoritmi di scheduling.

Scheduling

- Lo scheduling della CPU è il meccanismo sul quale sono basati i sistemi che supportano la multiprogrammazione (sistema **multitasking**):
  - Più processi sono in memoria
  - Quando un processo entra in attesa, il SO concede la CPU ad un altro processo
- In questo tipo di sistema, l'obiettivo è quello di massimizzare il tempo d'uso della CPU
- In alcuni contesti è detto **scheduler a breve termine** per contrasto allo scheduler di lungo termine (che decideva quali processi caricare in memoria).

Ciclo di Vita dei Processi



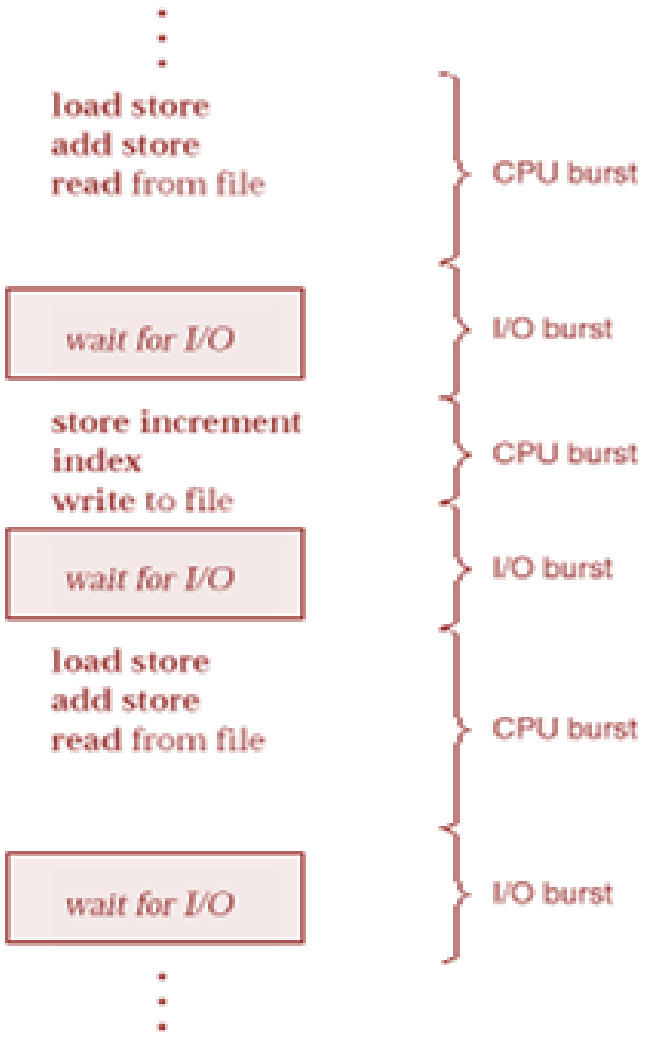
Scheduling

- Lo scheduler a breve termine, opera sulla ready queue per individuare il prossimo processo che passerà dallo stato di ready allo stato di running
- Usiamo una prospettiva storica:
  - Multitasking NON time sharing: l'esecuzione si interrompe quando il processo esegue una operazione di I/O bloccante.
  - Time sharing: l'esecuzione è scandita dal timer
  - Moderni: combinano soluzioni diverse

CPU e I/O burst

- In un sistema multiprogrammato le prestazioni sono influenzate dal tipo di attività dei processi (più orientata al calcolo o più orientata all'I/O)
- I processi sono costituiti da cicli di esecuzione (**CPU burst**), durante i quali necessitano della CPU, e attese per l'I/O (**I/O burst**).
- Il ciclo di vita (normale) di un processo:
  - inizia con uno CPU burst
  - alterna I/O burst e CPU burst
  - durante un ciclo di CPU burst chiede al SO di terminare.

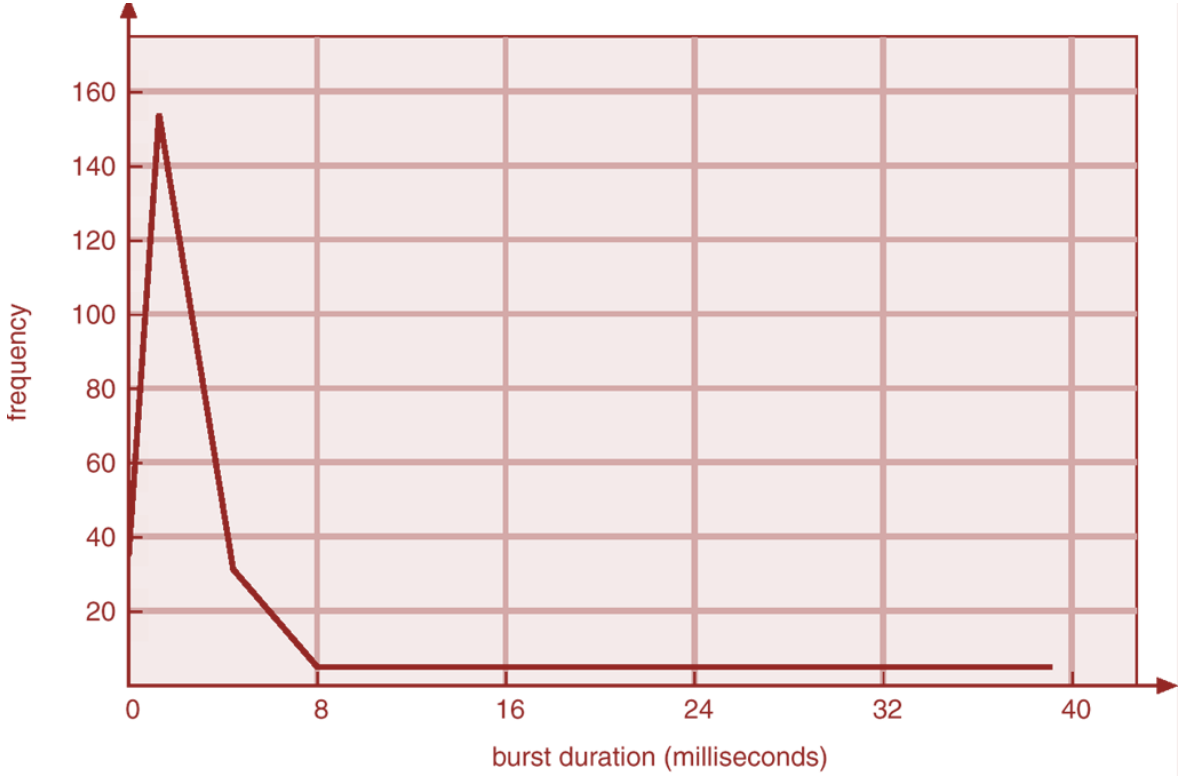
Sequenza CPU-I/O burst



Frequenza

- In un processo
  - **I/O bound** i cicli di CPU burst sono molti ma molto brevi perché frequentemente interrotti da attività di I/O
  - **CPU bound** i cicli di CPU burst sono meno ma più duraturi
- Mediamente i cicli CPU burst sono solitamente molto brevi.
- Il sistema di scheduling deve tenere conto di questa distribuzione.

CPU burst


160
140
120
100
80
60
40
20
frequency
0
8
16
24
32
40
burst duration (milliseconds)

Process Control Block

- Il SO deve rappresentare il processo come struttura dati e infatti ogni processo è descritto da una struttura dati chiamata **Process Control Block** (**PCB**).
- Il PCB dipende da SO a SO ma contiene tipicamente tutte quelle informazioni che servono al sistema per gestire il processo (per esempio cambiandone lo stato) e per eseguirlo (quindi farlo avanzare nella computazione).
- Le informazioni associate a ogni processo sono:
  - **Stato del processo**
  - **Program counter = (segmento:offset)**: prossima istruzione da eseguire per questo processo.
  - **Registri della CPU**: nella condizione da ripristinare per il processo in esecuzione.
  - **Informazioni sullo scheduling** della CPU: tra cui la priorità ma anche i puntatori alle code di scheduling e altre informazioni "interne".
  - ….

Ready Queue

- La **ready queue** è la struttura dati in cui lo scheduler mantiene i PCB (Process Control Block) dei processi pronti per eseguire.
- Ovviamente, l'organizzazione della struttura dati (ready queue) dipende dall' algoritmo che la utilizza (scheduler).
- La ready queue può essere organizzata in diversi modi:
  - Lista non ordinata
  - FIFO (coda)
  - Lista ordinata
  - Albero
  - Multicoda

Decisioni

- Le decisioni dello scheduler intervengono quando il processo cambia di stato:
  1. Passa da new a ready (appena creato).
  2. Passa da running a waiting (richiesta di I/O).
  3. Passa da running a ready (interrupt).
  4. Passa da waiting a ready (concluso I/O)
  5. Termina.
- Nei casi 2 e 5 il processo non può procedere e dunque occorre prelevare un altro processo dalla ready queue per fargli guadagnare la CPU.
- Nei casi 1, 3 e 4 il processo torna in condizione di ready e occorre decidere come gestirlo (dove inserirlo).

Preemptive

- Quando un nuovo processo entra nella ready queue si possono adottare due differenti politiche:
  - **Preemptive scheduling (con prelazione)**: se il processo rientrato ha priorità maggiore di quello in esecuzione prima del rientro, lo spodesta. Lo scheduler entra in funzione ogni rientro nella ready queue.
  - **Non preemptive (senza prelazione)**: anche se un nuovo processo è rientrato, il controllo della CPU resta al processo in esecuzione prima del rientro, finché questo non la rilascia. Lo scheduler entra in funzione solo in quel momento.

Dispatcher

- Il **dispatcher** è quella parte del SO che passa effettivamente il controllo della CPU al processo selezionato dallo scheduler.
- Le operazioni di cui deve occuparsi sono le seguenti:
  - Lo switching del contesto ovvero il ripristino del contesto del processo che sta per essere eseguito
  - Il passaggio alla modalità utente.
  - Il riavvio dell'esecuzione del programma utente dal punto in cui era stata interrotta (jump)

Dispatcher

- Il dispatcher viene invocato ogni volta che avviene un context switch.
- Il tempo di latenza introdotto dall'esecuzione del dispatch (**dispatch latency**) deve essere minimizzato per assicurare efficienza al SO.
- Nota: in alcune fonti trovate il termine dispatcher usato per indicare lo scheduling a breve termine (per esempio Windows o Solaris). Qui lo usiamo per individuare la sola componente che si occupa di caricare e lanciare il prossimo processo.

Criteri di Scheduling

- I diversi algoritmi di scheduling hanno proprietà differenti per cui favoriscono alcune classi di processi rispetto ad altri.
- Il confronto tra gli algoritmi di scheduling avviene tipicamente sulla base dei seguenti parametri:
  - Utilizzo di CPU
  - Throughput.
  - Tempo di turnaround.
  - Tempo di attesa.
  - Tempo di risposta.

Tempo di CPU

- Principio (un po' superato): La CPU costa e deve essere sfruttata il più possibile cioè deve essere attiva il più possibile.
- L'**utilizzo della CPU** (**CPU utilization**) è un parametro percentuale che varia quindi da 0 a 100.
- Nei SO attuali l'utilizzo della CPU è in genere scarso: la maggior parte dei processi sono I/O bound e interattivi, consumano poca CPU.
  - Tranne quando fanno busy waiting ! ☹☹☹☹

Throughput

- Il **throughput (produttività)** è il numero di processi completati nell'unità di tempo.
- E' una misura del lavoro effettivamente svolto dalla CPU, ma dipende fortemente dal tipo (e dalla durata) dei processi che girano sulla macchina.
- Può essere espressa in processi/s se i processi sono brevi o in processi/h per processi di media durata.

Tempo di turnaround

- Per fare misure su un processo specifico si utilizza il **tempo di completamento** (**turnaround time**) che è il tempo complessivamente necessario per eseguire il processo (comprese le attese nella ready queue o il tempo di esecuzione dell'I/O).
- Corrisponde all'intervallo di tempo tra l'istante in cui il processo è stato lanciato e quello in cui il processo termina.

Tempi di attesa

- Il criterio di scheduling della CPU:
  - NON influisce sul tempo che il processo passerà in attesa di I/O.
  - Influisce sul tempo che il processo passerà nella ready queue in attesa della CPU.
- Un parametro di valutazione può dunque essere il **tempo di attesa** (**waiting time**) trascorso dal processo nella coda dei ready.

Tempo di risposta

- Se il processo è altamente interattivo il tempo di turnaround può non essere una misura efficace.
- Si misura in questi casi il **tempo di risposta** (**response time**) che è il tempo che intercorre tra la prima richiesta fatta al processo e la prima risposta prodotta da questo.
- Dipende dal/dai dispositivo/i di immissione dei dati.

Criteri di ottimizzazione

- Il SO deve cercare di
  - Massimizzare l'uso della CPU.
  - Massimizzare il throughput.
  - Minimizzare il tempo di turnaround.
  - Minimizzare il tempo d'attesa.
  - Minimizzare il tempo di risposta.

Quali valutazioni

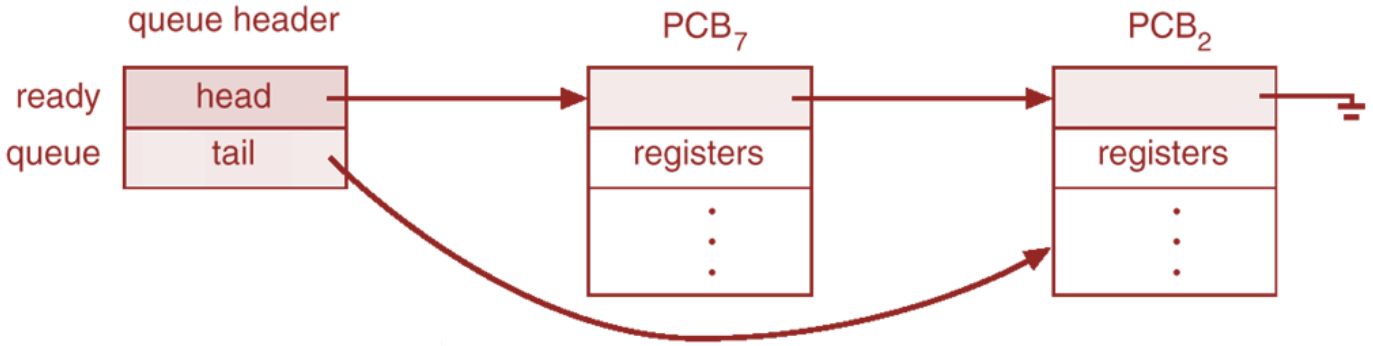
- Il SO può:
  - ottimizzare **mediamente** le misure.
  - Tentare di offrire servizi che stiano tra un **minimo** e **massimo**.
- Per i sistemi altamente interattivi si tenta di minimizzare la varianza del tempo di risposta ovvero di dare al SO un comportamento **prevedibile**.

Algoritmi di scheduling

- Vedremo i seguenti algoritmi di scheduling:
  - FCFS (ovvero su struttura FIFO)
  - Shortest Job First (SJF)
  - Priority Scheduling
  - Round Robin
  - Scheduling Multilivello

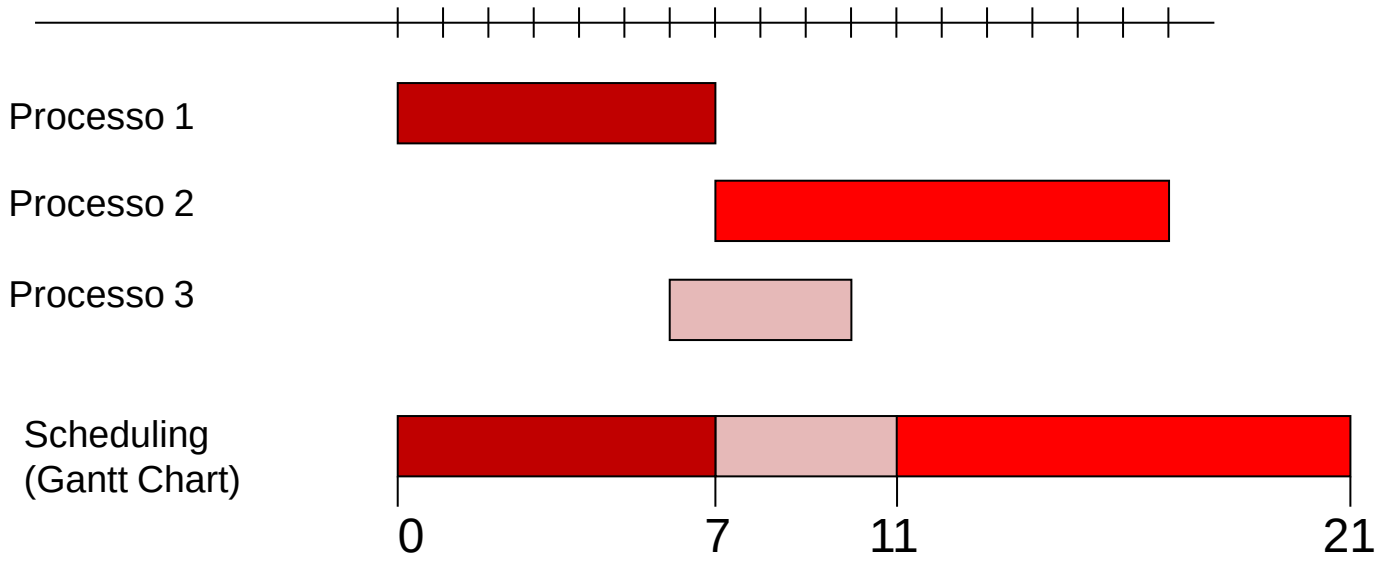
FCFS/FIFO

- L'algoritmo di scheduling più semplice è quello che gestisce la ready queue come una coda **FIFO** (**First In- First Out**).
- L'algoritmo utilizza dunque una politica nota come FCFS (**First Come – First Served**):
  - Quando un processo entra nella ready queue, il suo PCB viene aggiunto in fondo alla coda.
  - Quando la CPU è libera, viene allocata al processo il cui PCB è in testa alla coda.



FCFS

- In questo primo esempio vediamo come ordinare i processi in uno schema di Gantt:

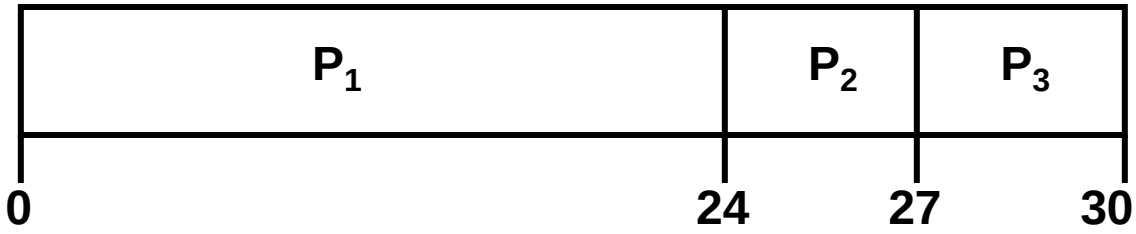


FCFS

- ESEMPIO:

Processo	Burst Time
P_1	24
P_2	3
P_3	3

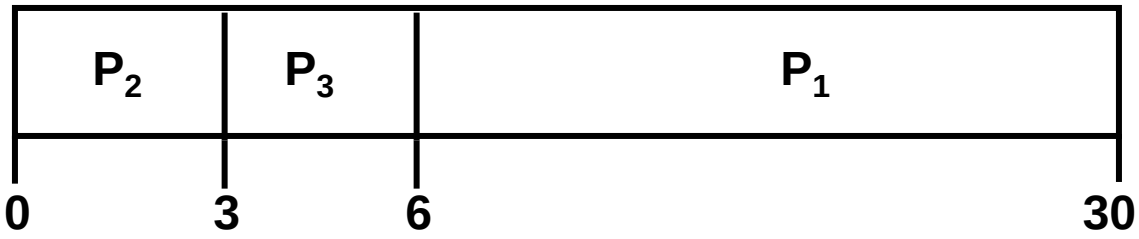
- Supponiamo che l'ordine di arrivo sia P_1 , P_2 , P_3



- Il tempo di attesa è $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Il tempo di attesa medio è $(0 + 24 + 27)/3 = 17$

FCFS

- Se invece l'ordine di arrivo è P_2 , P_3 , P_1



- Il tempo di attesa è $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Il tempo di attesa medio è $(6 + 0 + 3)/3 = 3$

FCFS

- L'algoritmo non minimizza il tempo di attesa, che dipende da fattori casuali (ordine di arrivo dei processi).
- Un solo processo molto lungo può bloccare molti processi che avrebbero eseguito velocemente (**effetto convoglio**, **convoy effect**) aumentando notevolmente i tempi medi di attesa.
- FCFS è senza prelazione (ovvero **non preemptive**) e dunque particolarmente inadatto a sistemi real-time.

SJF

- Per ridurre i tempi di attesa medi di FCFS, si possono usare algoritmi che privilegiano i processi che termineranno prima.
- **Shortest Job First** (**SJF**) è un algoritmo di scheduling che seleziona il prossimo processo da eseguire prelevando dalla ready queue il processo che ha il prossimo CPU burst più breve.
- Se ci sono più processi con lo stesso CPU burst viene usata una tecnica FIFO.

SJF

- Due modalità operative (**entrambe** con **starvation**):
  - **Non preemptive**: quando un processo ottiene la CPU non viene interrotto finche non completa il CPU burst
  - **Preemptive**: quando arriva un nuovo processo con CPU burst minore del CPU burst rimanente per il processo correntemente in esecuzione, il nuovo processo prende la CPU (il vecchio processo in esecuzione viene scavalcato). Questo Algoritmo è chiamato **Shortest-Remaining-Time-First** (**SRTF**).

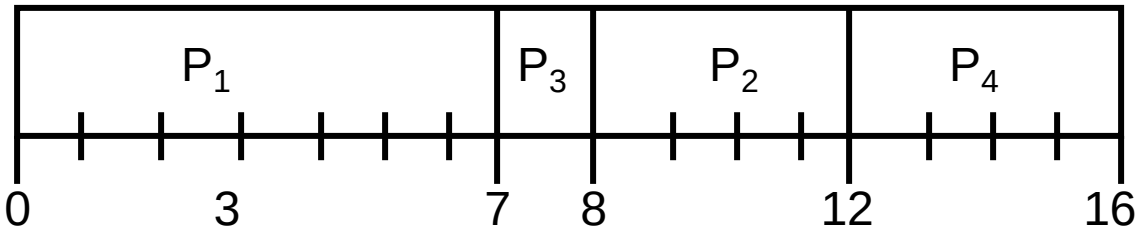
Starvation

- Con **starvation** (termine inglese che tradotto letteralmente significa **inedia**) si intende l'impossibilità, da parte di un processo pronto all'esecuzione, di ottenere le risorse di cui necessita.
- A volte è indicata anche come **indefinite blocking**
- Gli algoritmi con **priorità (di qualunque tipo)** generano starvation:
  - Processi con priorità più alta possono passare davanti a processi con priorità più bassa all'infinito
  - I processi a bassa priorità potrebbero non ottenere mai ciò che serve alla loro esecuzione

SJF (non-preemptive)

- Esempio:

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

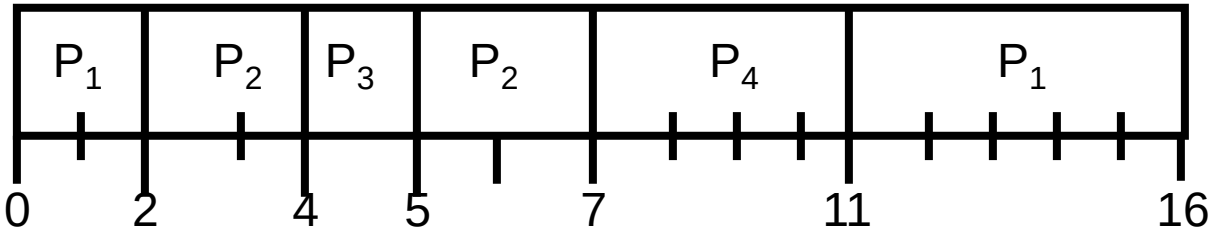


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

SJF (preemptive)

- Esempio:

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



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

SJF

- Si può dimostrare che SJF è un algoritmo **ottimale** in termini di waiting time medio.
- Spostando un processo con esecuzione breve davanti a uno con esecuzione superiore il tempo medio di attesa diminuisce.
- Il problema è come stabilire la lunghezza del prossimo CPU burst.

Misura del CPU burst

- La difficoltà di stabilire la lunghezza del CPU burst del processo può essere risolta:
  - A livello di **scheduling di lungo termine** chiedendo agli utenti di indicare un tempo limite di elaborazione (devono essere onesti, altrimenti rischiano l'interruzione della computazione).
  - A livello di **scheduling della CPU**, non potendo conoscere la lunghezza del prossimo CPU burst, si tenta di **predirla in modo approssimato**.

Misura del CPU burst

- Le predizioni sulla lunghezza del prossimo CPU burst vengono fatte sulla base della seguente considerazione: è probabile che un CPU burst successivo abbia lunghezze analoghe a quelli precedenti.
- La lunghezza del CPU burst successivo è quindi ottenuta effettuando la **media esponenziale** delle misure dei CPU burst precedenti.

Media esponenziale

- La formula che viene usata è la seguente:

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

dove:

- τ_{n+1} è il valore **previsto** per il prossimo CPU burst
- t_n è il valore **rilevato** dell'ultimo CPU burst
- α è un valore compreso tra **0 e 1** ($0 \leq \alpha \leq 1$)

- Il valore previsto iniziale τ_0 sarà una costante definita empiricamente o statisticamente dal sistema

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 sia α che $(1-\alpha)$ sono compresi tra 0 e 1, ogni termine successivo (più remoto nel tempo) ha un valore inferiore a quello del suo predecessore.

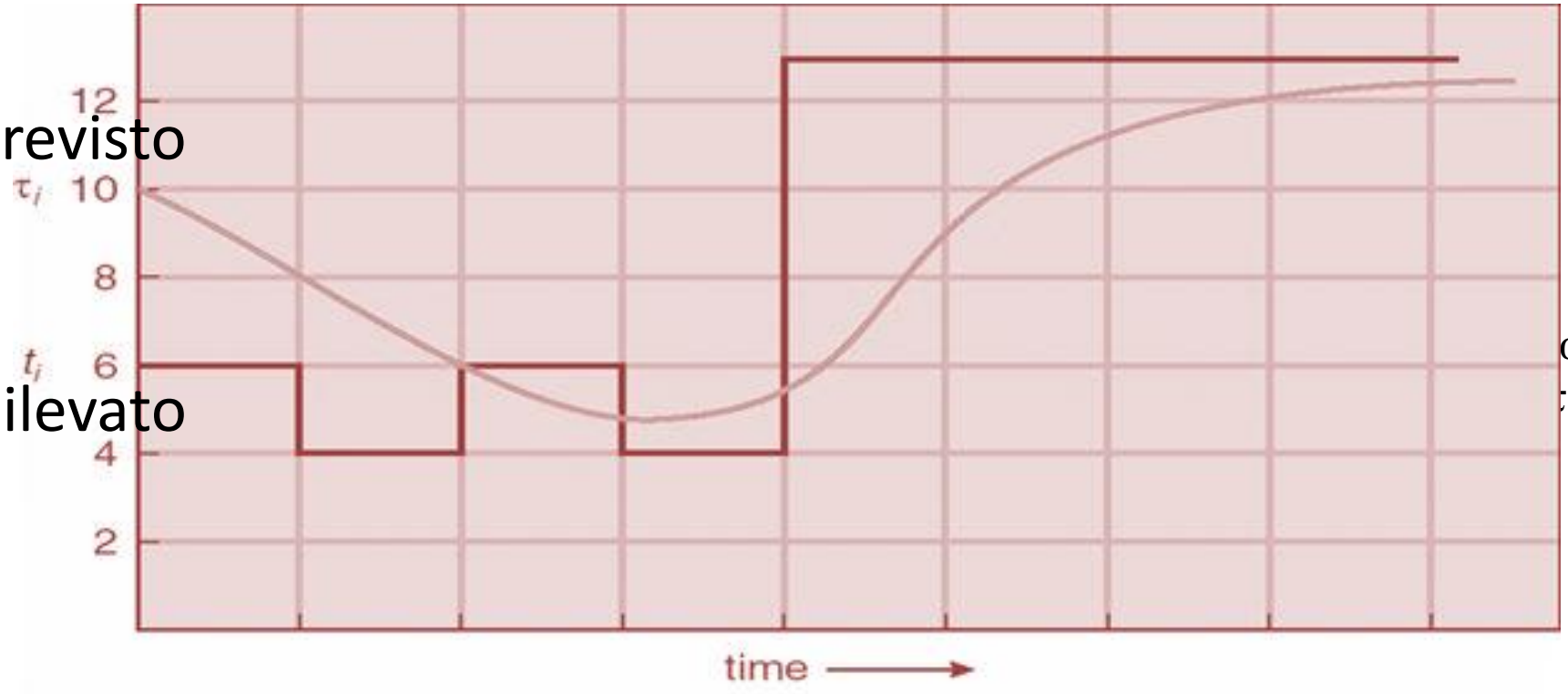
Media esponenziale

- Se si sceglie α =0
  - $\tau_{n+1} = \tau_n$
  - La storia recente non viene considerata.
- Se si sceglie α =1
  - $\tau_{n+1} = t_n$
  - Viene considerato solo l'ultimo CPU burst.

CPU burst: come varia il CPU burst previsto in funzione di quello rilevato

Previsto

Rilevato



$\alpha=½$

$\tau_0=10$

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

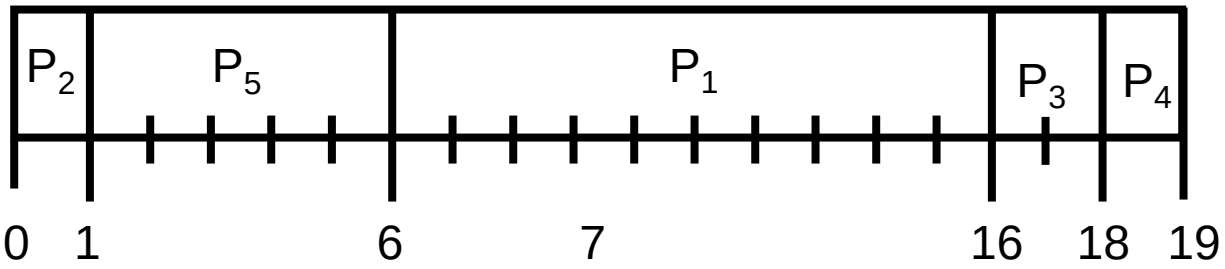
Priority Scheduling

- L'algoritmo SJF è un caso speciale di scheduling con priorità in cui la priorità è data dal valore atteso per la lunghezza dello CPU burst.
- In generale in un algoritmo di **scheduling con priorità** ad ogni processo è associato un numero intero che rappresenta la sua priorità.
- La CPU è allocata al processo con la priorità più alta ovvero con il numero intero più piccolo.

Priority Scheduling

- Esempio (processi arrivati tutti al tempo 0):

Processo	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2



- Average waiting time = (6 + 0 + 16 + 18 + 1) = 8,2

Priority Scheduling

- Come nel caso dello SJF, anche lo scheduling con priorità può essere:
  - Preemptive.
  - Non preemptive.
- Il principale problema degli scheduler con priorità è che possono lasciare i processi a bassa priorità ad attendere indefinitamente per la CPU perché finché saranno presenti processi ad alta priorità, non arriverà mai il loro turno (**starvation**).
- Una soluzione è quella di aumentare progressivamente la priorità dei processi con un meccanismo di **invecchiamento** (**aging**) del processo.

Fino ad ora, Morale per Scheduling multitasking

- Si deve tenere conto di CPU burst e I/O burst e gestire processi CPU bound e altri I/O bound
- Conta il tempo medio di attesa, che andrebbe minimizzato: consigliati
  - Shortest job first
  - FIFO
- Se ci sono priorità (basate sul tempo o no) ci può essere starvation.

- LIMITI
  - scheduler invocato solo al termine del CPU burst o all'ingresso in ready queue di un altro processo

Round Robin

- Gli scheduler visti fino ad ora non sono molto adatti ai sistemi in time sharing in cui lo scheduler viene invocato (oltre che nelle “solite” occasioni) anche ogni **quanto di tempo** (**time slice**).
- Ogni quanto dura tipicamente da 1 a 100 ms
- Round Robin (o scheduling circolare) è un algoritmo di scheduling progettato per operare efficacemente in sistemi time sharing

Round Robin

- Lo scheduler progettato appositamente per i sistemi in **time sharing** è detto **Round Robin** (**RR**).
- E' una variante dello scheduling FCFS in cui la coda è gestita come una coda circolare.
- La politica FCFS ritorna ad essere efficace perché siamo in condizioni di time sharing.
- Ogni volta che un processo viene messo in esecuzione, lo scheduler imposta un timer che ne limita il tempo di esecuzione.

Round Robin

- Quando il tempo di esecuzione scade, il timer genera un interrupt e il SO viene invocato.
- Lo scheduler riaggancia il PCB del processo che deteneva la CPU in fondo alla coda e mette in esecuzione il processo in testa alla coda.
- Tipicamente ha un turnaround medio più alto che SJF ma migliore responsività.

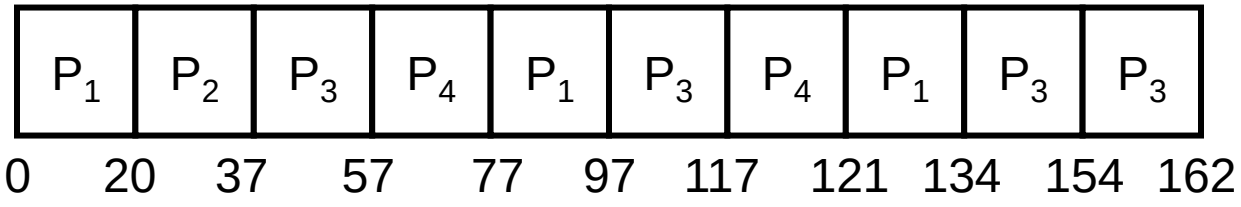
Round Robin

- In realtà possono verificarsi due casi:
  - Il processo in esecuzione ha uno CPU burst inferiore al time slice. In questo caso il processo rilascia volontariamente la CPU.
  - Il processo in esecuzione ha uno CPU burst superiore al time slice e quindi viene rimosso dalla CPU nel momento in cui scatta l'interrupt del clock.

Round Robin: Esempio

- Esempio (quanto = 20 ms):

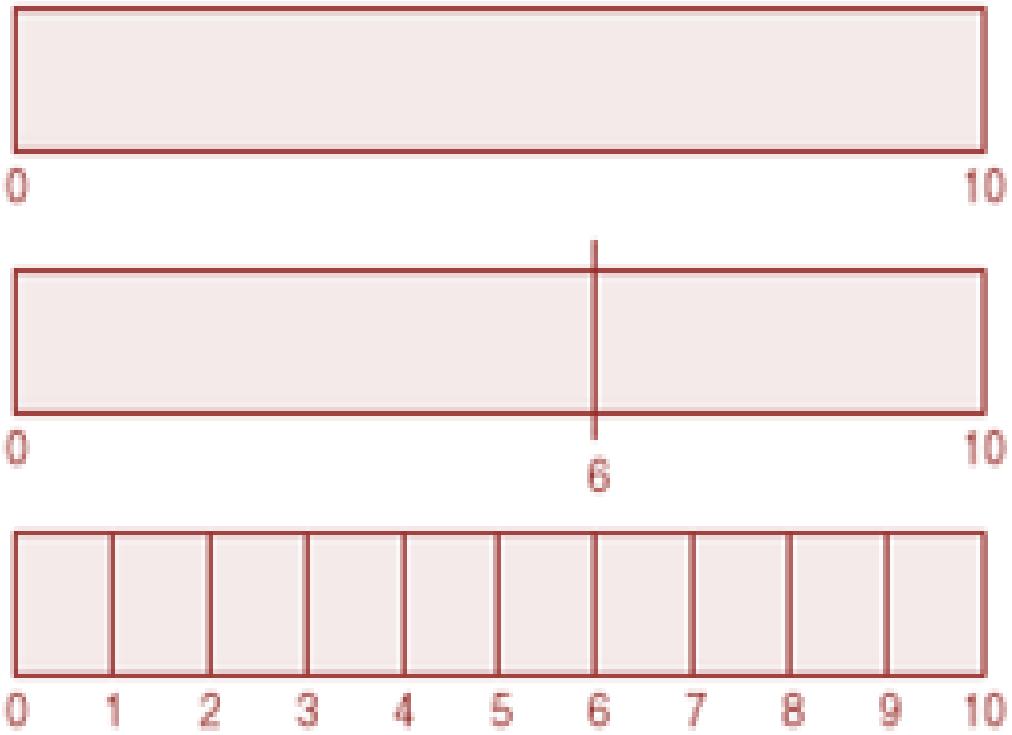
Processo	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24



Round Robin (RR)

- Se ci sono **n** processi nella ready queue e il quanto di tempo è **q**, allora ogni processo utilizzerà **1/n** del tempo totale di CPU in parti di lunghezza **q**.
- Performance:
  - **q** molto grande $\Rightarrow$ FIFO
  - **q** piccolo $\Rightarrow$ attenzione al contex switch e al relativo overhead.

Quantum



quantum	context switches
12	0
6	1
1	9

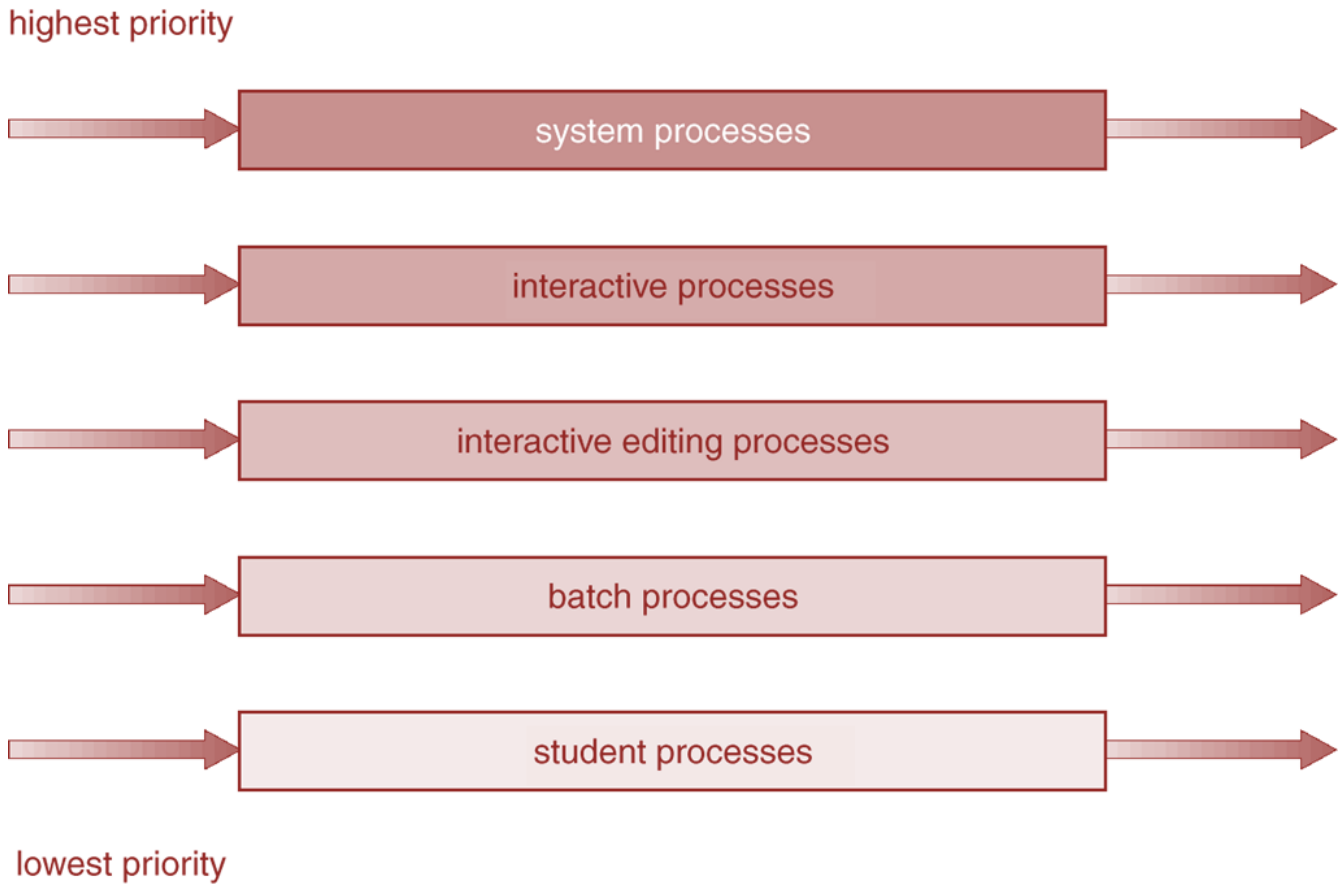
Code multiple

- Ci sono situazioni in cui è possibile classificare i processi ed adottare politiche di scheduling differenziate per le diverse classi
- Una suddivisione diffusa è quella tra processi **foreground** (**interattivi**) e processi **background** (**batch**)
- L'algoritmo di scheduling a **code multiple** è adatto a gestire queste situazioni

Code multiple

- La ready queue viene partizionata in più code separate e ciascun processo è associato in modo statico ad una coda.
- Vengono applicati **scheduler diversi su code diverse** (ad esempio RR sui processi foreground e FCFS per i processi in background).
- Occorre definire il comportamento dello **scheduling tra le code** (da che coda prelevo il prossimo processo da eseguire?)

Code multiple



Code multiple

- Possibili algoritmi di scheduling tra le code sono i seguente:
  - Non viene eseguito nessun processo della coda n se la coda n-1 non è vuota. Se un processo di livello inferiore ad n entra nella sua ready queue scatta la prelazione.
  - Definiamo dei quanti di tempo tra le code (per esempio 80% del tempo di CPU per i processi foreground e 20% per quelli background).

Code multiple con feedback

- Scheduling a code multiple con retroazione
- Nel meccanismo a code multiple ciascun processo è destinato a rimanere sempre nella stessa coda.
- Si può invece fare in modo che i processi mutino dinamicamente posizione, ad esempio in funzione del CPU burst che hanno.

Code multiple con feedback

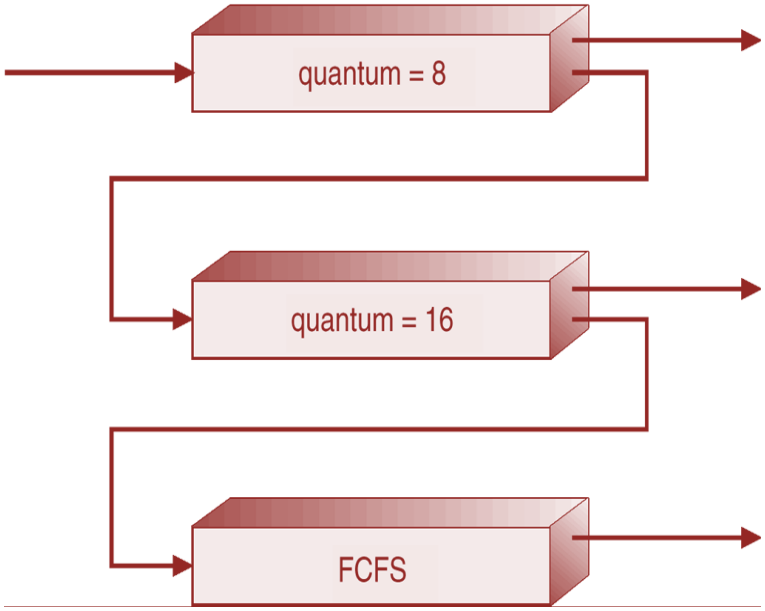
- Si crea quindi una struttura di code associate a tempi di CPU burst decrescenti, per cui la coda di livello più alto è quella che contiene i processi con CPU burst inferiore.
- Code più lente hanno time slice più grandi.
- Lo scheduling tra le code avviene dando massima priorità alle code per CPU burst brevi.

Code multiple con feedback

- Un processo viene:
  - Inserito inizialmente nella coda più veloce.
  - Retrocesso alla coda di livello inferiore quando non termina nel time slice di quella coda.
- Processi con molto I/O hanno CPU burst corti e restano in alto. Processi con poco I/O tendono a scendere su code lente.

Code multiple con feedback

- Tre code:
  - Q0: quanto di tempo di 8 millisecondi
  - Q1 : quanto di tempo di 16 millisecondi
  - Q2: FCFS



Code multiple con feedback

- Generalmente uno scheduler a code multiple con feedback è definito dai seguenti parametri:
  - Numero di code.
  - Algoritmi di scheduling per ciascuna coda.
  - Metodo per decidere quando promuovere un processo e/o metodo per decidere quando regredirlo.
  - Metodo per posizionare un nuovo processo