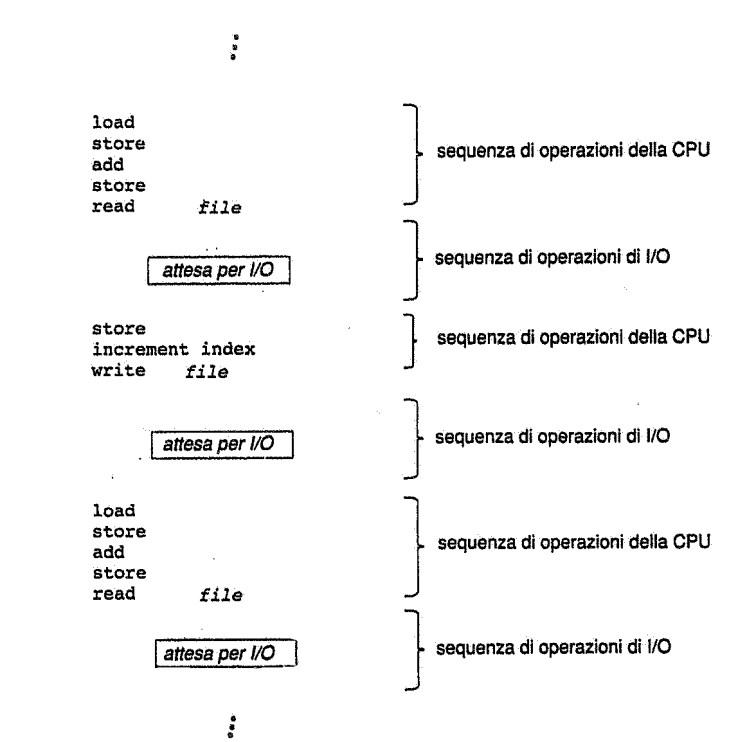
# CAPITOLO 5: SCHEDULING DELLA CPU

## Con la multi-programmazione si cerca di impiegare i tempi di attesa in maniera produttiva: si tengono più processi in memoria contemporaneamente e quando un processo deve attendere un evento, il SO gli sottrae il controllo della CPU per cederla a un altro processo.

Lo scheduling è una funzione fondamentale dei SO, si sottopongono a scheduling quasi tutte le risorse di un calcolatore.

## 5.1.1 Ciclicità delle fasi d’elaborazione e di IO



L’esecuzione di un processo consiste in un ciclo di elaborazione ed un ciclo di completamento. I processi si alternano in questi due cicli fino al loro completamento. Comincia con una sequenza effettuata dalla CPU (CPU burst), seguita da una sequenza di operazioni di IO (IO burst) e cosi via fino a quando non viene inviata una richiesta di terminazione.

Un programma con prevalenza di IO(IO bound) produce molta sequenza di operazioni della CPU a breve durata. Un programma con prevalenza di operazioni di elaborazione(CPU bound), produce poche sequenze di operazioni della CPU ma molto lunghe. Quindi deve sussistere un buon algoritmo per la scelta di queste operazioni.

## 5.1.2 Scheduler della CPU

Ogniqualvolta la CPU passa nello stato d’inattività, il sistema operativo sceglie per l’esecuzione uno dei processi presenti nella coda dei processi pronti. In particolare lo scheduler a breve termine che sceglie tra i processi presenti in memoria ad assegnarli alla CPU. Tutti i processi della coda dei processi pronti sono posti nella lista d’attesa per accedere alla CPU, generalmente tutti gli elementi delle code sono i *process control block* (PCB) dei processi.

**5.1.3 Scheduling con diritto di prelazione**

Le decisioni riguardanti lo scheduling della CPU si possono prendere nelle seguenti circostanze:

* Un processo passa dallo stato di esecuzione in uno stato di attesa (x es.:richiesta di I/O o richiesta di attesa(WAIT)per la terminazione di uno dei processi figli) NB : senza diritto di prelazione
* Un processo passa dallo stato di esecuzione allo stato di pronto (x es.:quando si verifica un segnale di interruzione) NB : con diritto di prelazione
* Un processo passa dallo stato di attesa allo stato di pronto (x es.:completamento di un'operazione di I/O) NB : con diritto di prelazione
* Un processo termina NB : senza diritto di prelazione

I casi 1 e 4 non comportano alcuna scelta di scheduling; a essi segue la scelta di un nuovo processo da eseguire , sempre che ce ne sia uno pronto.

Una scelta di scheduling si deve fare nei casi 2 e 3.

Nel caso dello scheduling **senza diritto di prelazione** , quando si assegna la CPU a un processo , questo rimane in possesso della CPU fino al momento del suo rilascio , dovuto al termine dell'esecuzione o al passaggio nello stato di attesa.

Lo scheduling con **diritto di prelazione** presenta un inconveniente: si consideri il caso in cui 2 processi condividono dati ; mentre uno di questi aggiorna i dati , si ha la sua prelazione in favore dell'esecuzione dell'altro. Il secondo processo in questo caso può tentare di leggere i dati che sono stati lasciati in uno stato incoerente dal primo processo.

## 5.1.4 Dispatcher

Il dispatcher è il modulo che passa effettivamente il controllo della CPU ai processi scelti dallo scheduler a breve termine.

Questa funzione riguarda:

* Il cambio di contesto.
* Il passaggio alla modalità utente.
* Il salto alla giusta posizione del prog. Utente per riavviarne l’esecuzione.

Siccome il dispatcher si attiva ad ogni cambio di contesto, esso deve essere il più rapido possibile.

Il tempo richiesto dal dispatcher per fermare un processo e avviare l'esecuzione di un altro è nota come latenza **di dispatch.**

## 5.2 CRITERI DI SCHEDULING

Esistono diversi algoritmi di scheduling della CPU e hanno proprietà differenti e possono favorire una determinata classe di processi. Nella scelta dell’algoritmo di scheduling bisogna tener conto:

* **Utilizzo della CPU:** la CPU deve restare il più a lungo possibile in attività.
* **Produttività:** la CPU è attiva quando svolge del lavoro. Una misura del lavoro svolto è data dal numero dei processi completati nell’unita di tempo (THROUGHPUT)
* **Tempo di completamento:** ossia considerare il tempo che intercorre tra la sottomissione del processo e il completamento della sua esecuzione
* **Tempo di attesa:** l’algoritmo di scheduling della CPU non influisce sul tempo impiegato per l’esecuzione di un processo o di un’operazione di IO. Il tempo di attesa è la somma degli intervalli d’attesa passati nella coda dei processi pronti.
* **Tempo di risposta:** il tempo di risposta è il tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta.

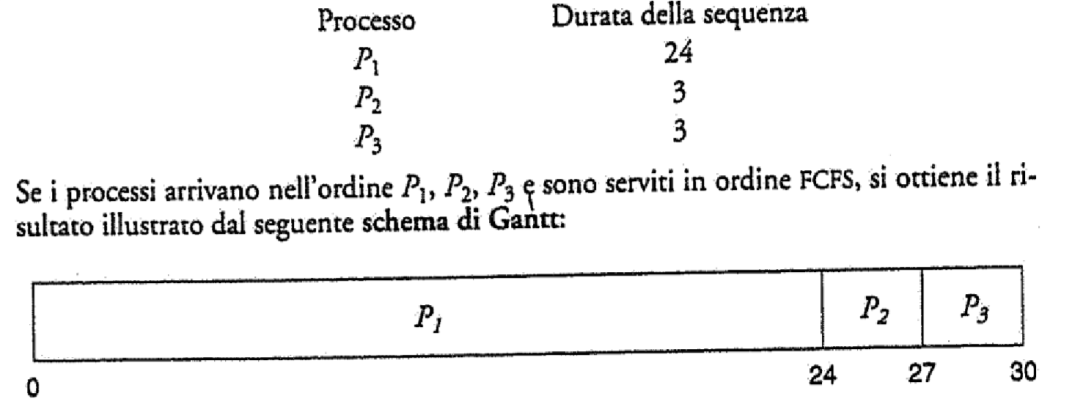
## 5.3 ALGORITMI DI SCHEDULING

Lo scheduling della CPU riguarda la scelta dei processi pronti cui assegnare la CPU

## 5.3.1 Scheduling in ordine di arrivo FCFS

Il più semplice algoritmo di scheduling della CPU è quello della scelta dei processi in base all’ordine di arrivo. Tale algoritmo noto anche come FCFS sceglie il processo in testa alla 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 si assegna al processo che si trova in testa alla coda. È spesso molto lungo.

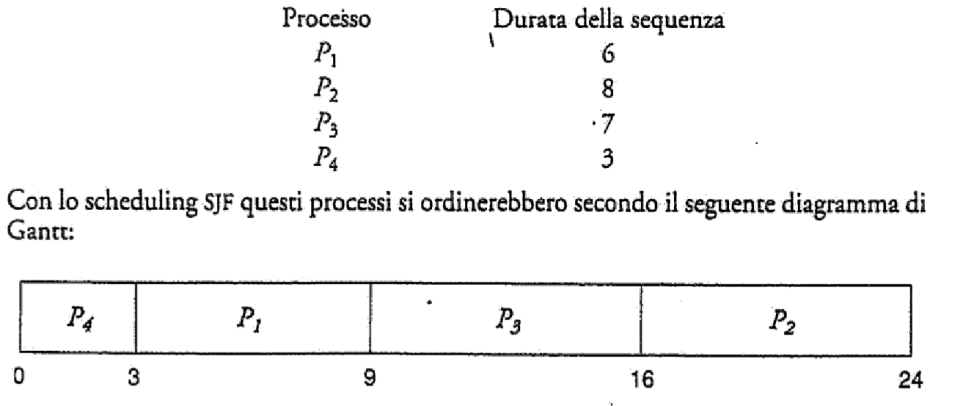
Un problema che si può verificare con la scelta di tale algoritmo è l’effetto del convoglio: tutti i processi attendono che un lungo processo liberi la CPU mentre nella coda ci sono processi a breve termine che attendono un lungo periodo. È senza prelazione, una volta che la CPU viene assegnata ad un processo, questo lo trattiene fino al suo rilascio.



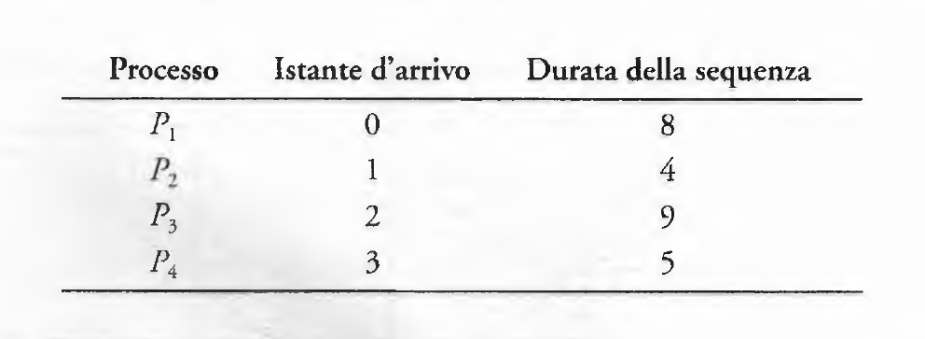
Il tempo di attesa medio è di 17 millisecondi (0+24+27) /3=17

## 5.3.2 Scheduling per brevità SJF

Un criterio diverso è quello dello scheduling della CPU per brevità noto anche come SJF. Questo algoritmo associa per ogni processo la lunghezza della successiva sequenza di operazioni. Ogni volta si assegna la CPU al processo che ha la più breve sequenza. Se 2 processi hanno la stessa sequenza allora in questo caso si applica l’algoritmo FCFS.



In questo caso il tempo di attesa è (3+16+9+0) /4=7.

Si può dimostrare che è ottimale ma la difficoltà reale implica nel conoscere la durata della successiva richiesta di CPU. Tale algoritmo può essere sia con diritto di prelazione che senza. Nel secondo caso l’algoritmo continua la sua esecuzione così come è stato presentato, mentre nel primo caso (con diritto di prelazione) se durante l’esecuzione di un processo arriva un nuovo processo in cui la sua sequenza è minore della sequenza restante del processo in esecuzione, allora tale processo viene posto nella coda dei processi pronti e quello appena arrivato viene assegnato alla CPU.



All’istante 0 si avvia il processo *P1,* poiché e l’unico che si trova nella coda. All’istante 1 arriva

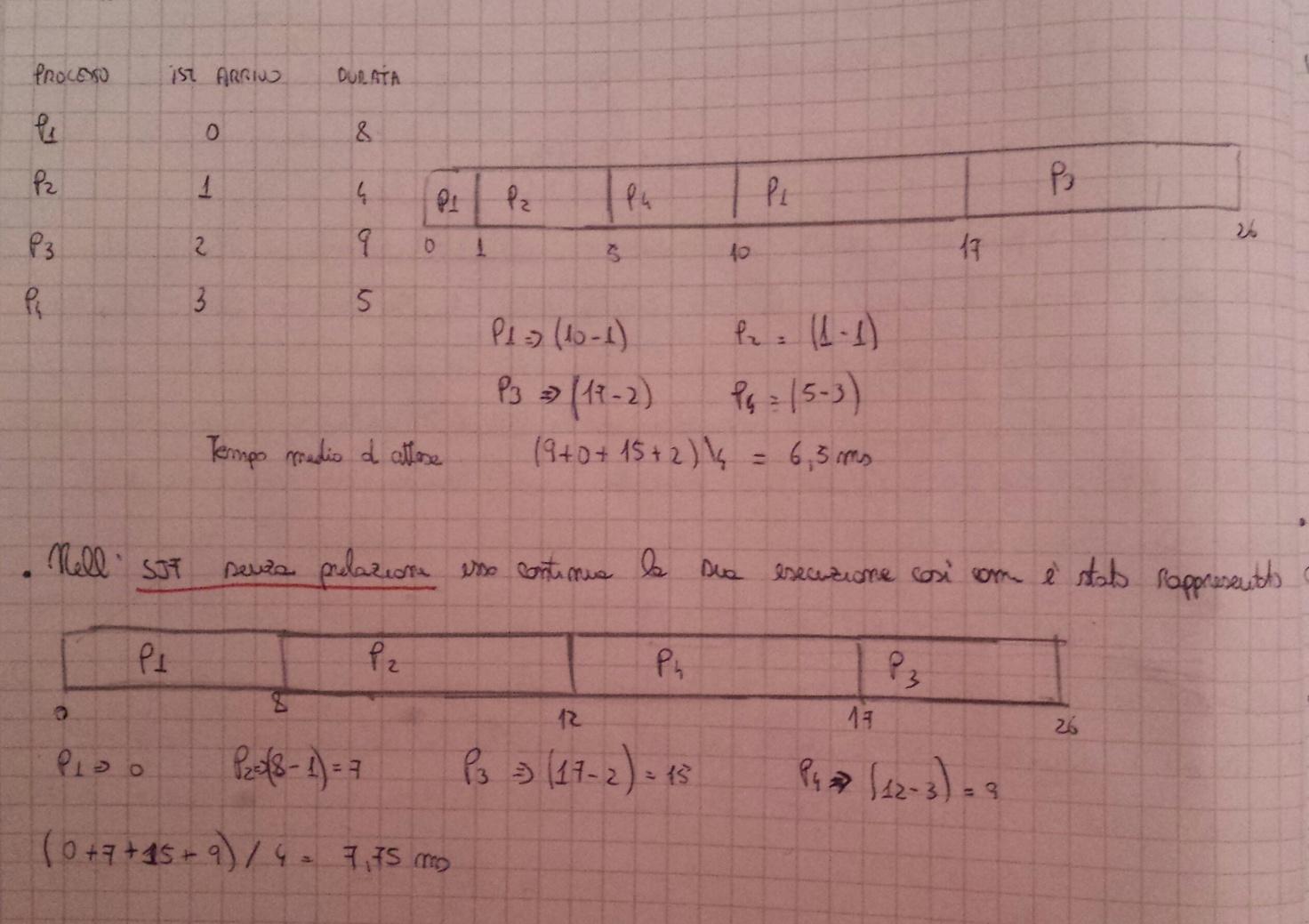
il processo *P2.* Il tempo necessario per completare il processo *P1* (7 millisecondi) è maggiore

del tempo richiesto dal processo *P2* (4 millisecondi), perciò si ha la prelazione sul processo

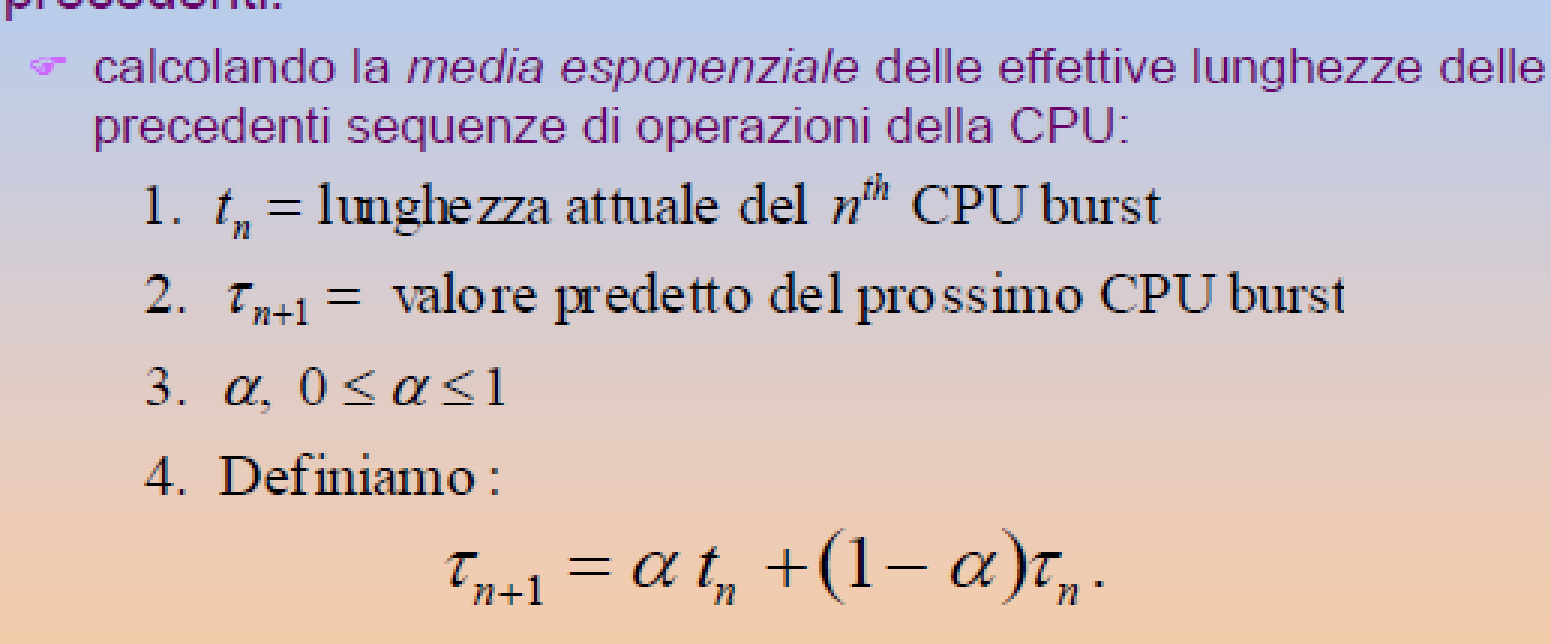
*P1* sostituendolo col processo *P2.* Il tempo d’attesa medio per questo esempio e

((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3)) /4= 26/4 = 6,5 millisecondi.

Con uno scheduling SJF senza prelazione si otterrebbe un tempo d’attesa medio di 7,75 millisecondi.



Sebbene sia ottimale, l'algoritmo SJF non si può realizzare a livello di scheduling della CP a breve termine, poiché non esiste alcun modo per conoscere la lunghezza della successiva sequenza di operazione della CPU. Un possibile metodo consiste nel tentare di approssimare lo scheduling SJF, predire il suo valore, quindi calcolando un valore approssimativo si può scegliere il processo con la più breve lunghezza. La lunghezza della successiva sequenza di operazione della CPU si ottiene calcolando la media esponenziale delle effettive precedenti sequenze di operazioni della CPU



* per a=0 🡪  *(tao)tn*+1=tn

la storia recente non ha valore

* per a=1 🡪 (*tao*)tn+1=atn

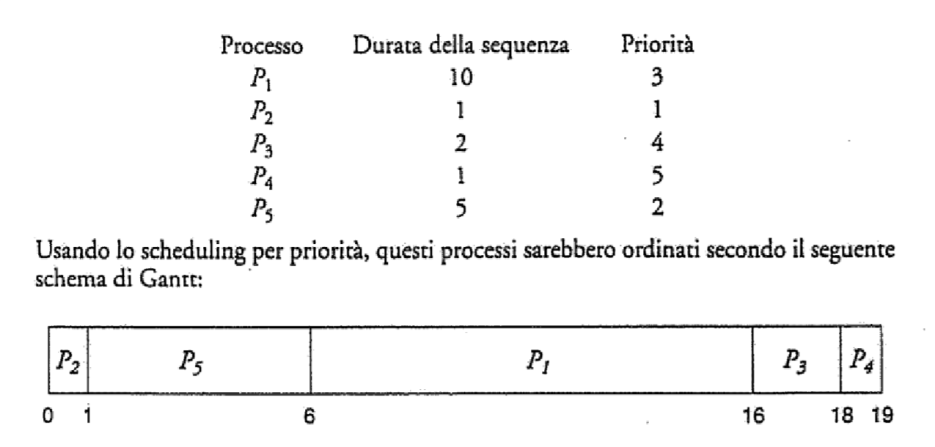
solo il precedente burst ha valore

## (a rappresenta il peso della storia recente e passata.)

## 5.3.3 Scheduling per priorità

L’algoritmo SJF è un caso particolare del più generale algoritmo di scheduling per priorità. Tale algoritmo oltre a presentare la durata della sequenza, si associa una priorità a ogni processo e si assegna la CPU al processo con priorità più alta.

Un algoritmo SJF non è altro che un algoritmo con priorità dove tale priorità è l’inverso della lunghezza.



Il tempo medio di attesa è di 8.2 millisecondi.

L’algoritmo per priorità è sia con diritto che senza diritto di prelazione.

L'algoritmo con **diritto di prelazione** sottrae la CPU al processo attualmente in esecuzione se la priorità dell'ultimo processo arrivato è superiore.

L'algoritmo **senza diritto di prelazione** si limita a porre l'ultimo processo arrivatoalla testa della coda dei processi pronti.

Un problema importante è il problema della **starvation** (attesa indefinita). Un processo pronto per l’esecuzione che non dispone della CPU si può considerare bloccato nell’attesa nel caso questo abbia una priorità bassa e non venga mai eseguito. ("Un processo con priorità maggiore può impedire a un processo con priorità minore di accedere alla CPU").

Una soluzione a questo problema è costituita dall’invecchiamento. Ogni qualvolta passa un periodo di tempo un processo che sta nell’attesa della CPU ottiene un aumento graduale della priorità facendo in modo che venga prima o poi eseguito.

## 

## 5.3.4 Scheduling circolare Round Robin

L’algoritmo circolare noto anche come Round Robin è stato progettato appositamente per i sistemi a partizione del tempo.

È simile allo scheduling FCFS ma in più ha la capacità di prelazione per la commutazione tra processi.

Ciascun processo riceve una piccola quantità fissata del tempo della CPU, chiamata quanto di tempo (Time Slice); la coda dei processi pronti è trattata come una coda circolare.

Lo scheduler della CPU scorre la coda dei processi pronti, assegnando la CPU a ciascun processo per una durata massima pari al quanto di tempo (varia di solito tra 10-100 ms).

La coda dei processi pronti viene gestita come una coda FIFO. I nuovi processi si aggiungono alla fine della coda dei processi pronti.

Lo scheduler della CPU individua il primo processo della coda dei processi pronti, imposta un timer in modo che invia un segnale di interruzione alla scadenza di un intervallo e attiva il dispatcher per l'effettiva esecuzione del processo. A questo punto si possono verificare due situazioni: il processo a una sequenza di operazioni della CPU più di durata minore del quanto di tempo, quindi il processo stesso rilascia volontariamente la CPU e lo scheduler passa il processo successivo della coda dei processi pronti; oppure la durata della sequenza di operazioni della CPU del processo attualmente in esecuzione è più lunga di un quanto di tempo, in questo caso si raggiunge la scadenza e il timer invia un segnale di interruzione a sistema operativo che esegue un cambio di contesto e aggiunge il processo alla coda dei processi pronti. Da qui si può denotare che algoritmo round Robin è pertanto con prelazione. Le prestazioni di questo algoritmo dipendono molto dalla dimensione del quanto di tempo, impostando un quanto di tempo lungo il criterio di scheduling round Robin si riduce all'algoritmo FCFS. Se il quanto di tempo è molto breve il criterio RR si chiama condivisione della CPU e teoricamente gli utenti hanno l'impressione che ciascuno degli n processi disponga di una propria CPU. Riguardo alle prestazioni dello scheduling round Robin, occorre tuttavia considerare l'effetto dei cambi di contesto, segue che il quanto di tempo deve essere ampio rispetto alla durata del cambio di contesto. Anche il tempo di completamento (turnaround Time dipende) dalla dimensione del quando ti tempo.

In generale il tempo medio di completamento può migliorare se la maggior parte dei processi termina la sequenza in un solo quanto di tempo.



p1 -> 10-4= 6 ms, p2-> 4 ms , p3-> 7 ms ; 6+4+7=17 -> 17/3=5,66 ms

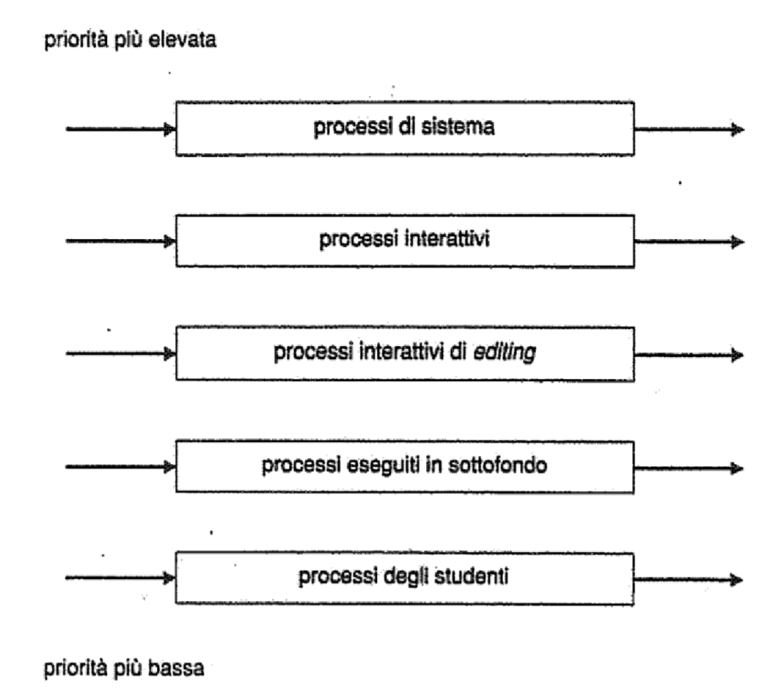
## 

## 5.3.5 Scheduling a code multiple

È stata creata una classe di algoritmi di scheduling adatta a situazioni in cui i processi si possono classificare facilmente in gruppi diversi. Una distinzione diffusa è per esempio quella che si fa tra i processi che si eseguono in primo piano (foreground) e quelli che si eseguono in sottofondo (background).

Questi 2 tipi di processi hanno tempi di risposta differenti e pertanto possono aver bisogno di diversi scheduling.

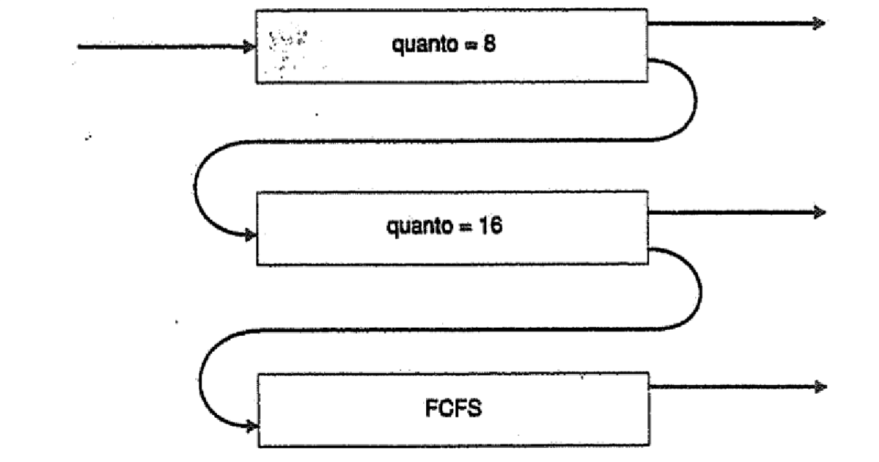
L'algoritmo di scheduling a code multiple suddivide la coda dei processi pronti in code distinte. I processi si assegnano in modo permanente a una coda, generalmente secondo qualche caratteristica del processo. Ogni coda ha il proprio algoritmo di scheduling, per esempio la coda dei processi in primo piano si può gestire con un algoritmo round Robin mentre quella dei processi in sottofondo si può gestire con un algoritmo FCFS. Ogni coda alla priorità assoluta sulle code di priorità più bassa; nessun processo della coda dei processi in sottofondo puoi iniziare l'esecuzione finché le code di un livello più alto non siano tutte vuote, se un processo di una coda superiore entrasse nella coda dei processi pronti durante l'esecuzione di un processo in sottofondo, se avrebbe la prelazione su quest'ultimo, questo causerebbe il rischio di attesa indefinita (starvation). Esiste anche la possibilità di impostare quanti di tempo per le code punto per ogni coda si stabilisce una parte del tempo dell’elaborazione della CPU, suddivisibile a sua volta tra i processi che la costituiscono.



## 5.3.6 Scheduling a code multiple con retroazione

Di solito in un algoritmo di scheduling a code multiple i processi vengono assegnati ad una coda e non possono quindi cambiarla.

Lo scheduling a code multiple con retroazione permette ai processi di spostarsi tra le code. L’idea consiste nel separare i processi che hanno caratteristiche diverse nelle sequenze delle operazioni della CPU, se un processo usa troppo tempo di elaborazione della CPU, viene spostato in una coda con priorità più bassa. Analogamente, si può spostare in una coda con priorità più elevata un processo che attende troppo a lungo (si attua una forma di invecchiamento che impedisce il verificarsi dell’attesa indefinita).

**//coda 0**

**//coda 1**

**//coda 2**

Lo scheduler fa eseguire tutti i processi della coda 0 attuando l’algoritmo RR con quanto=8, non appena si svuota esegue quelli in coda 1 con RR a quanto=16, infine quelli nella coda 2 con algoritmo FCFS. All’inizio tutti vengono collocati nella coda 0, i processi che non terminano vengono collocati nella coda 1 che a loro volta se non terminano vengono collocati nella coda 2.

Generalmente uno scheduler a code multiple con retrazione e caratterizzato seguenti parametri:

* numero di code;
* algoritmo di scheduling per ciascuna coda;
* metodo usato per determinare quando spostare un processo in una coda con priorità maggiore;
* metodo usato per determinare quando spostare un processo in una coda con priorità minore;
* metodo usato per determinare in quale coda si deve mettere un processo quando richiede un servizio.

**La definizione di scheduling a code multiple con retroazione costituisce il più grande criterio di scheduling della CPU, che nella fase di progettazione si può adeguare a un sistema specifico, sfortunatamente corrisponde anche all’algoritmo più complesso.**

## 4.SCHEDULING PER SISTEMI MULTIPROCESSORE //non presente negli argomenti dell’indice fino a esempio LINUX

## Soluzioni di scheduling per multiprocessore

Se sono disponibili più unita d’elaborazione, anche il problema dello scheduling è proporzionalmente più complesso.

Una prima strategia di scheduling della CPU per i sistemi multiprocessore affida tutte la decisione ad un CPU detta master e tutti i processi utenti alle altre cpu(MULTI-ELABORAZIONE ASIMMETRICA). Quando invece ciascun processore provvede al proprio scheduling, si parla di multielaborazione simmetrica SMP.

## Predilezione per il processore

Si consideri cosa accade alla memoria cache non appena che un processo abbia terminato la sua esecuzione nella CPU: i dati che ha trattato per ultimo permango nella cache, se il processo viene continuato da un altro processore, i contenuti della memoria cache devono essere invalidati sul processore di partenza e convalidati sull’altro. A causa di alti costi di svuotamento di memoria cache, si applicano delle regole impedendo ad un processo di passare da un processore ad un altro.

## Bilanciamento del carico

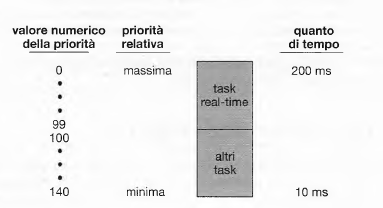
Sui sistemi SMP è importante che il carico sia distribuito uniformemente tra i vari processori in modo da non permettere che un processore sia saturo un altro sia inattivo. Il bilanciamento del carico quindi tenta di ripartire il carico di lavoro uniformante tra i processi. Ci sono 2 tecniche di bilanciamento: migrazione guidata e migrazione spontanea. La prima prevede che un processo controlli costantemente il carico di ogni processore e quando verifica che uno sia saturo, sposta i processi su altri processori. La 2 invece fa si che un processore inattivo sottragga spontaneamente processi a processori saturi.

## Multithread simmetrico

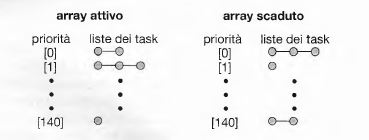
I sistemi SPM consentono l’esecuzione contemporanea, su processori fisici multipli di numerosi thread. Una possibile alternativa è quello che ha fatto l’intel noto come multithread simmetrico SMT che fornisce diversi processori logici. L’idea che caratterizza la SMT è la creazione di più processori logici basati sullo stesso processore fisico cosi da presentare al sistema operativo una serie di processori anche in presenza di uno solo fisico. Bisogna sapere che la SMT è a livello fisico, no software.

## 5.6.3 Esempio: scheduling di Linux

Prima della versione 2.5, kernel di Linux impiegava, per lo scheduling, una variante dell'algoritmo tradizionale di UNIX. Quest'ultimo, però, comporta due problemi: non fornisce strumenti adeguati per i sistemi SMP, e risponde male al crescere dell'ordine di grandezza del numero di task nel sistema. A partire dalla versione 2.5, il kernel offre un algoritmo di scheduling che gira in tempo costante a prescindere dal numero di tasca nel sistema. Lo scheduler di Linux ricorre un algoritmo di scheduling con prelazione, basato sulle priorità, con due gamme di priorità separate: un intervallo real-time che va da 0 a 99 è un intervallo detto **nice** compreso tra 100 e 140. queste due gambe sono poi tradotte all'interno di una scala globale di priorità, in cui valori più bassi numericamente rappresentano le priorità più alte. Linux assegna e task con priorità più alta porzioni di tempo più cospicue e a quelle dalla priorità più bassa quanti di tempo più brevi.



Il kernel elenca i task pronti per l'esecuzione in una struttura dati detta **runqueue** (coda di esecuzione). Ogni coda di esecuzione contiene due array priorità: **attivo** e **scaduto**. Il primo contiene tutti i task che hanno ancora tempo da sfruttare mentre il secondo elenca tutti i task scaduti. Una volta che tutti i task nell'array attivo abbiano consumato il loro quanto di tempo e quindi, sono diventati scaduti, gli Array delle priorità si scambiano di ruoli: quello scaduto diventa attivo e viceversa. Linux implementa lo scheduling Real-Time, in cui i task ricevono priorità statiche, mentre gli altri task hanno priorità a dinamiche, basate sul loro valore nice cui si aggiunge o sottrae il valore 5.



## 5.7 Valutazione degli algoritmi

Il primo problema da affrontare riguardo la valutazione degli algoritmi, riguarda la definizione dei criteri da usare per la scelta dell'algoritmo. Per scegliere un algoritmo occorre innanzitutto stabilire l'importanza relativa di queste misure. Tra i criteri suggeriti si possono inserire diverse misure, per esempio le seguenti:

* Rendere massimo l'utilizzo della CPU col vincolo di un tempo di risposta minimo.
* rendere massima la produttività in modo Che tempo di completamento sia linearmente proporzionale al tempo di esecuzione totale.

## 5.7.1 Modelli deterministici

Fra i metodi di valutazione sono di grande importanza quelli che rientrano nella classe della valutazione analitica. La valutazione analitica fornisce una formula o un numero che valuta le prestazioni dell’algoritmo per quel carico di lavoro. La definizione e lo studio di un modello deterministico è un tipo di valutazione analitica, che considera un carico di lavoro predeterminato e definisce le prestazioni e ciascuno algoritmo per quel carico di lavoro. **La definizione e lo studio di un modello deterministico è semplice e rapida; i risultati sono numeri esatti che consentono il confronto tra gli algoritmi.**

## 5.7.2 Reti di code

In molti sistemi i processi seguiti variano di giorno in giorno, quindi non esiste un insieme statico di processi da usare nei modelli deterministici. Si possono però determinare le distribuzioni del sequenze di operazioni della CPU e delle sequenze di operazioni di I/O, poiché si possono misurare e quindi approssimare si ottiene una formula matematica che indica la probabilità di una determinata sequenza di operazioni della CPU. comunemente questa distribuzione è di tipo esponenziale ed è descritta dalla sua media. Il sistema di calcolo si descrive come una rete di server, ciascuno con una coda d'attesa. La CPU è un server con la propria coda dei processi pronti, e il sistema di I/O ha le sue code dei dispositivi. Se sono noti l'andamento degli arrivi e dei servizi, si possono calcolare l'utilizzo, la lunghezza media delle code, il tempo medio d'attesa e così via. Questo tipo di studio si chiama analisi delle reti di code.

***n****= lunghezza media di una coda*

***W****= tempo medio di attesa* 

***lambda****= andamento medio di arrivo dei nuovi processi nella coda*

Questa equazione e nota come formula di **little** ed è utile soprattutto perché è valida per qualsiasi algoritmo di scheduling e distribuzione di arrivi. Questa formula e utilizzabile per il calcolo di una delle tre variabili, quando sono note le altre due. L'analisi delle reti di code può essere utile per il confronto degli algoritmi di scheduling, ma presenta alcuni limiti. Attualmente le classi di algoritmi e distribuzione trattabili sono piuttosto limitate punto per poter calcolare una risposta, le reti di code spesso si limitano ad approssimare un sistema reale, rendendo discutibile la precisione dei risultati ottenuti.

## 5.7.3 Simulazioni

Per riuscire ad avere una valutazione più precisa degli algoritmi di scheduling ci si può servire di **simulazioni**. le simulazioni implicano 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 un clock; con aumentare del valore di questa variabile, il simulatore modifica lo stato del sistema in modo da descrivere l'attività dei dispositivi, dei processi e dello scheduler. Durante l'esecuzione della simulazione si raccolgono e si stampano statistiche che indicano le prestazioni degli algoritmi. I dati necessari per condurre la simulazione si possono ottenere in vari modi, il metodo più diffuso impiega un generatore di numeri casuali, programmato per la creazione di processi, durata delle sequenze di operazioni della CPU ecc. Tuttavia, una simulazione condotta secondo la distribuzione può non essere precisa, a causa delle relazioni esistenti tra eventi successivi nel sistema reale. La distribuzione relativa alle frequenze, si limita ad indicare quanti eventi di una data categoria si verificano, senza fornire informazioni su loro ordine. Per rimediare a questo problema si può sottoporre il sistema reale a un controllo continuo, con la registrazione della sequenza degli eventi effettivi- in questo modo si ottiene un cosiddetto **trace tape.**

## 5.7.4 Realizzazione

Persino una simulazione ha dei limiti per quel che riguarda la precisione. l'unico modo assolutamente sicuro per valutare un algoritmo di scheduling consiste nel codificarlo, inserirlo nel sistema operativo e osservarne il comportamento nelle reali condizioni di funzionamento del sistema. Il problema principale di questo metodo e il suo costo: le spese non sono dovute solo alla codifica dell'algoritmo e alle modifiche da fare il sistema operativo, ma anche alle reazioni degli utenti a fronte di costanti modifiche del sistema operativo. Un'altra difficoltà da affrontare per fare qualsiasi valutazione di un algoritmo e il cambiamento dell'ambiente in cui lo si usa infatti l'ambiente si modifica anche in seguito alle prestazioni dello scheduler.