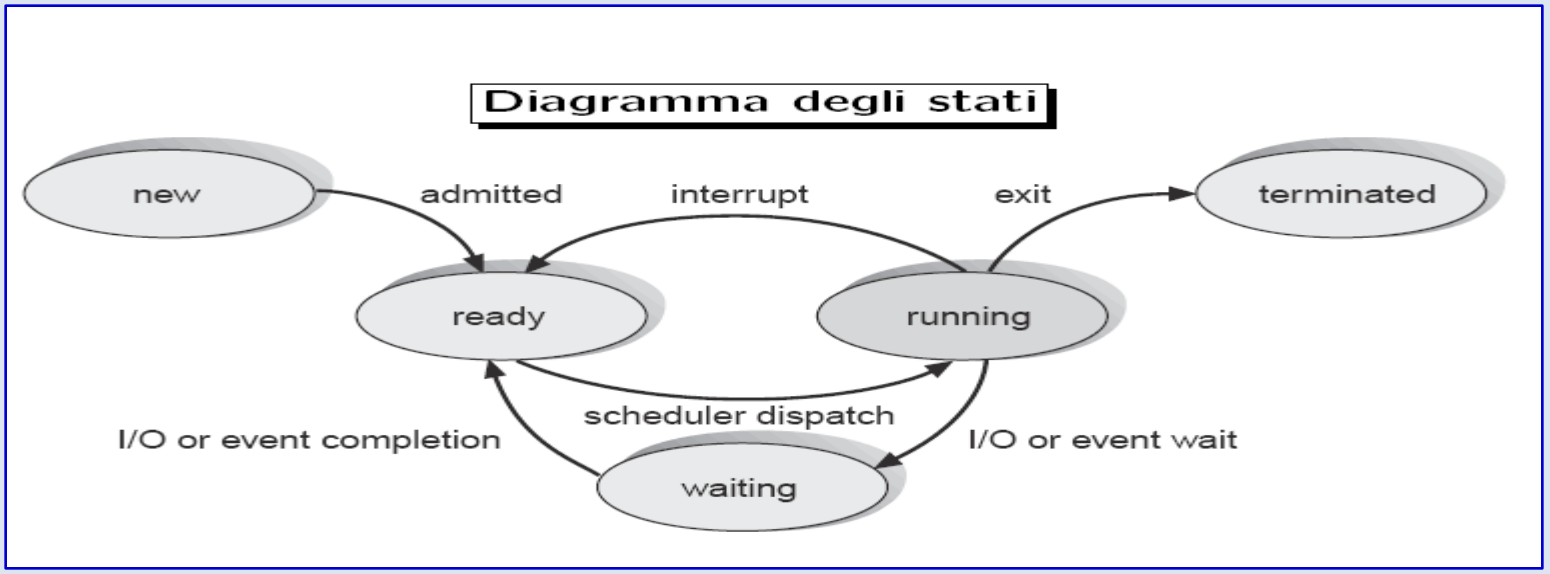
Politiche di schedulazione del processore

Gli stati di un processo

Gli **stati possibili** nei quali si può trovare un processo sono:

- **Hold (parcheggio)**: il programma (chiamato job) è stato proposto al sistema e **attende di essere caricato in memoria centrale**. Esso è *in attesa di esecuzione* e *si trova in memoria di massa* .
- **Ready (pronto)**: il programma è diventato **processo** e *si trova in memoria centrale*, pronto per essere eseguito. Esso **attende che gli venga assegnata la CPU**.
- **Run (esecuzione)**: il **processo è in esecuzione** (in ogni istante un solo processo si trova in questo stato) perché gli è stata assegnata la CPU.
- **Wait (attesa)**: il **processo è in attesa** (ad esempio deve attendere la fine di un'operazione di I/O).
- **Terminate (terminazione)**: il **processo è terminato** e può essere recuperato lo spazio della memoria centrale in cui era caricato.

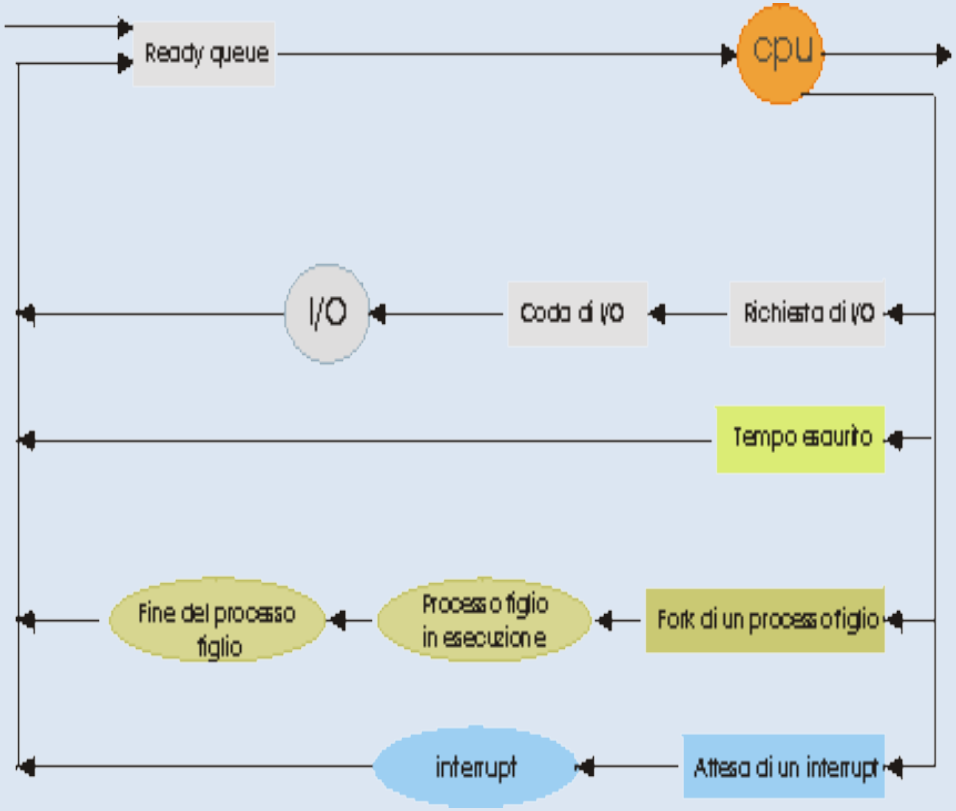


Un processo passa la propria vita a fare le code.

Quando un nuovo processo nasce, è in coda nello stato di **HOLD** e richiede uno spazio nella memoria RAM dove essere caricato; nel momento in cui gli viene assegnato questo spazio viene messo nella **READY queue** nella quale attenderà la disponibilità della risorsa processore; appena lo scheduler lo seleziona esso otterrà il possesso del processore con cui procedere alla sua elaborazione (stato di **RUN**).

Il **processo può perdere l'uso del processore** fondamentalmente per tre motivi:

- Il **processo emette una richiesta di I/O** e quindi viene posto nella coda del dispositivo di cui deve far uso (dischi, nastri, stampante ...);
- Il **processo crea un processo figlio** e ne deve attendere la terminazione;
- Il **processo subisce una interruzione** e forzatamente deve abbandonare la cpu per ripassare nella ready queue.



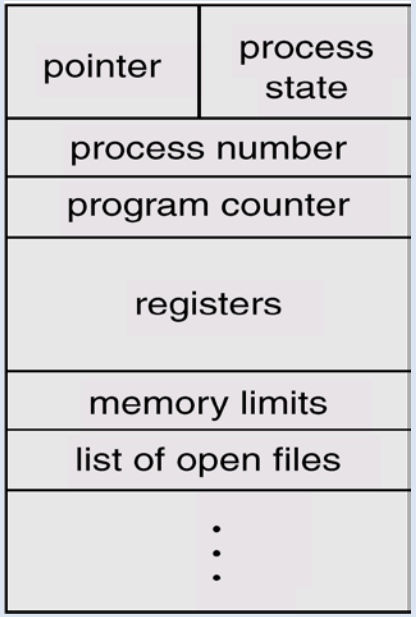
PCB e Tabella dei processi

Il **PCB** (Process Control Blocks) o **descrittore di processo** è una struttura dati che mantiene in memoria tutte le informazioni su un certo processo.
All'**insieme dei PCB dei processi attivi** nel Sistema Operativo si accede utilizzando la ***tabella dei processi.***

Ciascun PCB contiene le seguenti informazioni:

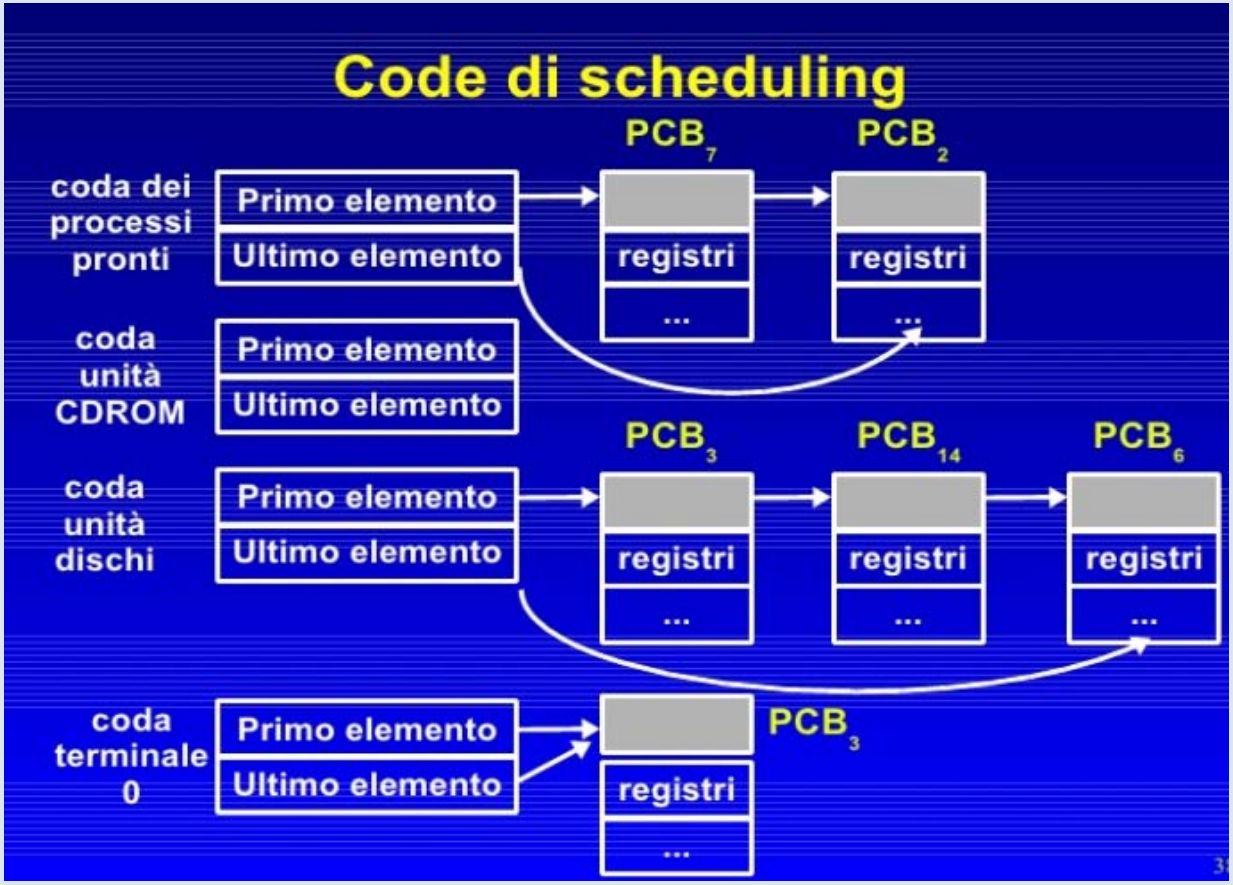
- PID (identificatore del processo)
- lo stato del processo
- il program counter (PC)
- il contenuto dei registri della CPU
- Le informazioni sulla memoria occupata e puntatore allo stack
- Limiti di tempo
- Privilegi per l'accesso a servizi e risorse
- Puntatore al padre
- Puntatore alla lista dei processi figli
- Lista dei file aperti dal processo
- Informazioni per lo schedulatore (priorità, tempo stimato di uso CPU, tempo di CPU residuo,…
- ….

PCB di un processo



Le code

I **descrittori dei processi** sono organizzati in code, una per ciascuno stato (READY, WAITING, …) e per ogni dispositivo del quale può essere fatta richiesta di utilizzo al SO.



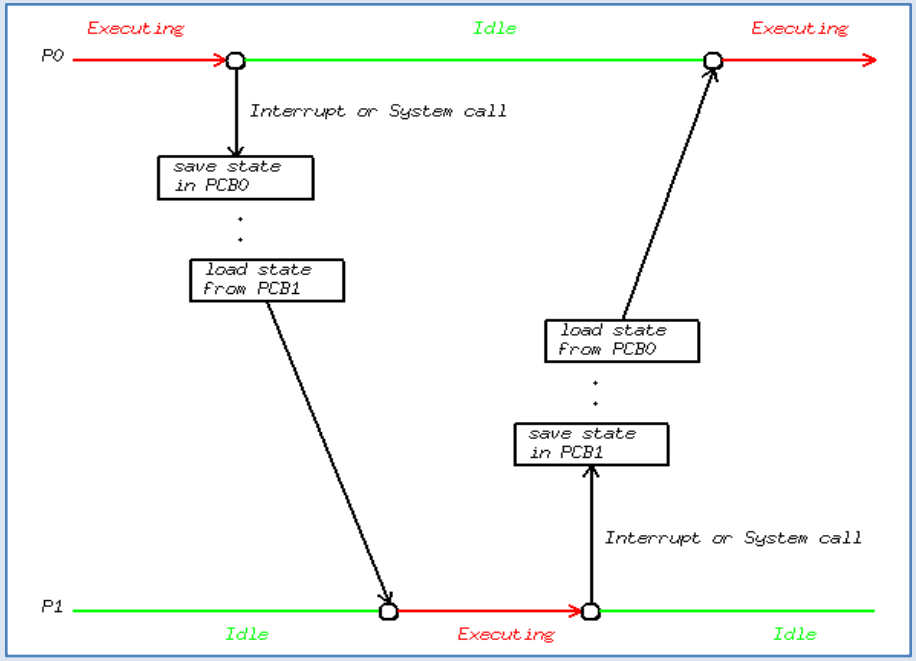
Schedulatore a breve termine e context switch

La componente del Sistema Operativo che si occupa di scegliere di volta in volta il processo che deve ottenere l'uso della CPU è lo **scheduler a breve termine** che sostanzialmente dice qual è il prossimo processo che deve passare dallo stato di READY allo stato RUNNING, ovvero andare in esecuzione.

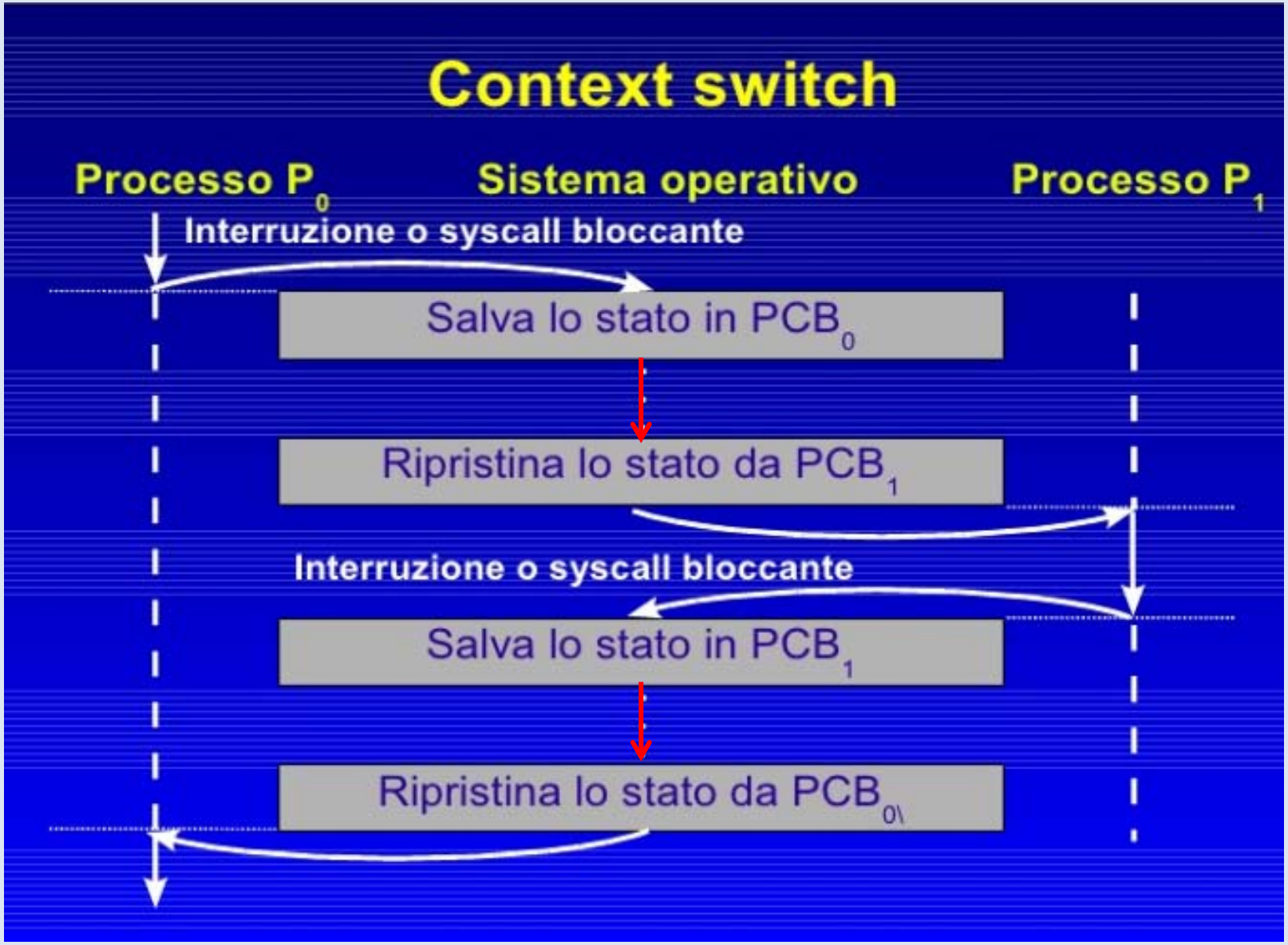
Lo **scheduling** avviene seguendo un certo **algoritmo** (FCFS, RoundRobin, ShortestJobFirst, a priorità statica o dinamica...) e il passaggio da un processo all'altro è detto **cambio di contesto** (**context switch**).

Il sistema operativo deve:

- **salvare lo stato del processo che abbandona la CPU**, ovvero andare ad aggiornare il **PCB** (Process Control Blocks detto anche **descrittore**) del processo
- **caricare il nuovo processo ripristinando le aree di memoria da lui utilizzate**, ripristinandone lo stato puntuale, ovvero il suo contesto.



Schedulatore a breve termine e context switch



Un **processo** in esecuzione può **richiedere al S.O.**

operazioni di I/O

su file, sul monitor,

da tastiera , da mouse,

ricezione e trasmissione dati in rete,

visualizzazione di testi e immagini .....

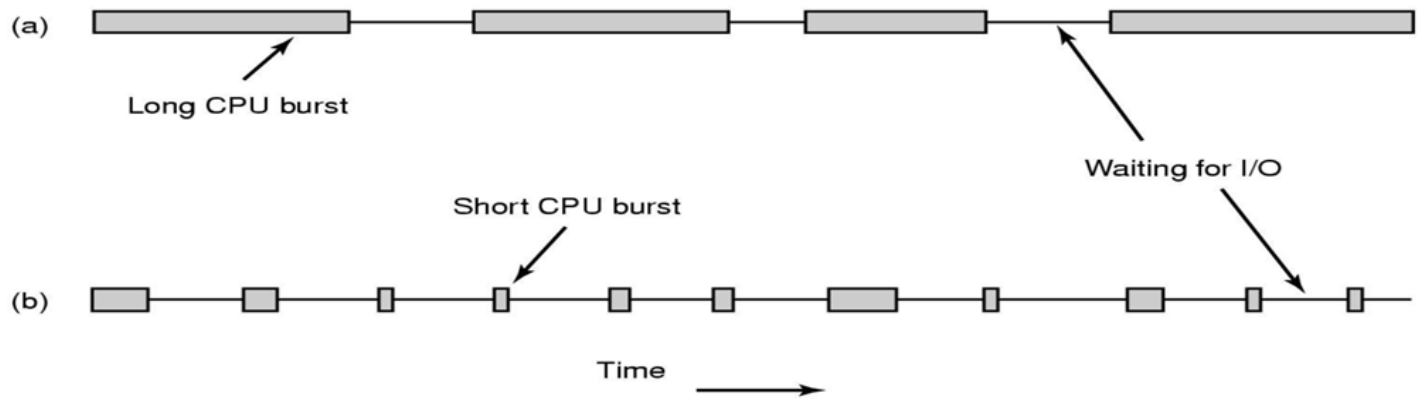
I processi che prevedono molte interazioni con gli utenti sono spesso in stato di WAITING e si dicono **I/O-bound.**

P1: RUN WAIT RUN WAIT RUN WAIT

I processi che eseguono molte elaborazioni di dati presenti in memoria non rilasciano quasi mai il processore restando quindi in stato di RUN e si dicono **CPU-bound.**

P2: RUN WAIT RUN WAIT RUN

Scheduling



- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process

Il **S.O.** ha il compito di **ottimizzare l'uso del processore**.

L'insieme dei **processi attivi** dovrebbe essere composto in modo equilibrato da CPU-bound e I/O-bound.

Considerazioni:

- Se i processi **CPU-bound** sono molto numerosi

  i pochi processi **I/O-bound vengono rallentati peggiorando il tempo di risposta dei processi interattivi** e lasciando inattivi i dispositivi periferici

- Se i processi **I/O-bound** sono molto numerosi

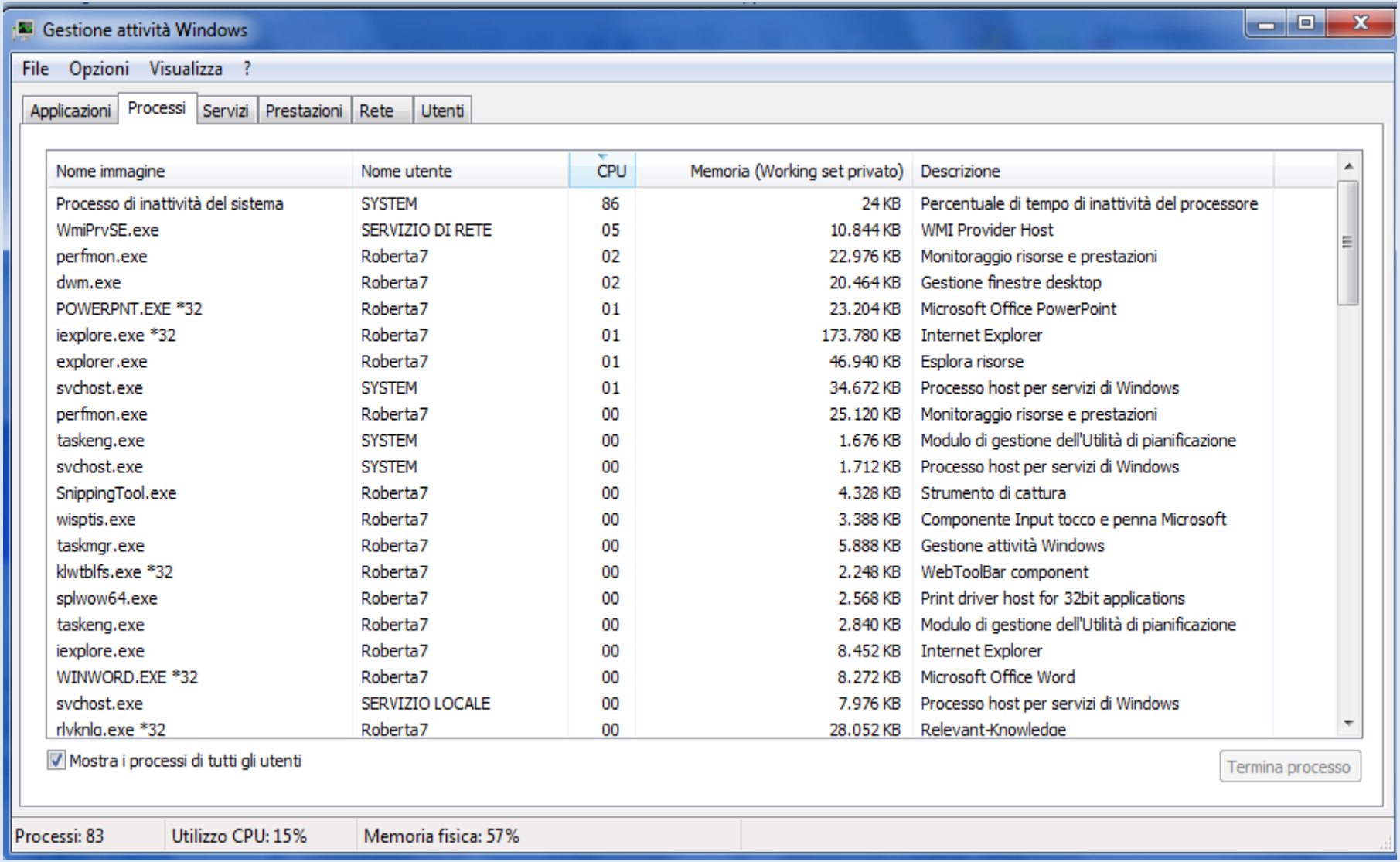
  Questo può portare ad **inattività del processore**, perché aumenta la probabilità che tutti i processi attivi siano in stato di attesa (WAIT)

  Durante i periodi di inattività del processore, se non vi sono processi in stato di READY da avviare nello stato di RUN, **il gestore dei processi**

  attiva l'esecuzione di un paricolare **processo di sistema** detto **IDLE**, che pone il computer in uno stato di *basso consumo energetico* ad esempio diminuendo la velocità del processore

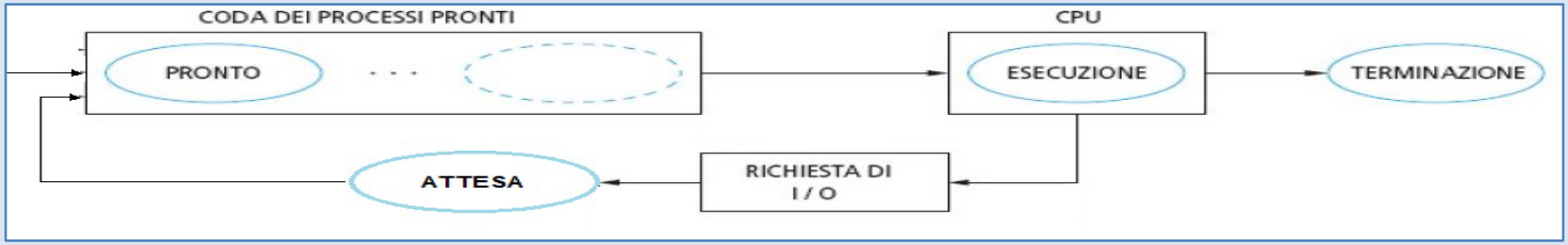
  oppure *vengono eseguiti processi di utilità* come ad esempio la scansione antivirus.

Task manager di Windows 7



Algoritmi di schedulazione del processore

Gli algoritmi di **rotazione tra i processi in esecuzione** utilizzati dallo schedulatore dei processi operano con precisi criteri per scegliere quale processo far avanzare *dallo stato di "pronto"* (READY) *allo stato di "esecuzione"* (RUN).



Questi algoritmi si possono dividere in due categorie:

- **NON PREEMPTIVE (senza prerilascio) :** una volta che **la CPU** è stata assegnata ad un processo, essa **non può essergli tolta** a meno che il processo non sia terminato oppure non abbia richiesto un'operazione di I/O.
- **PREEMPTIVE (con prerilascio) : viene usato nei sistemi operativi multitasking,** con il significato che **il S.O. che può interrompere un processo in esecuzione** contro la volontà di questo in favore di un altro processo che ha una maggiore priorità oppure se il processo ha utilizzato la CPU per il massimo tempo consentito (fine del "quanto" di tempo).

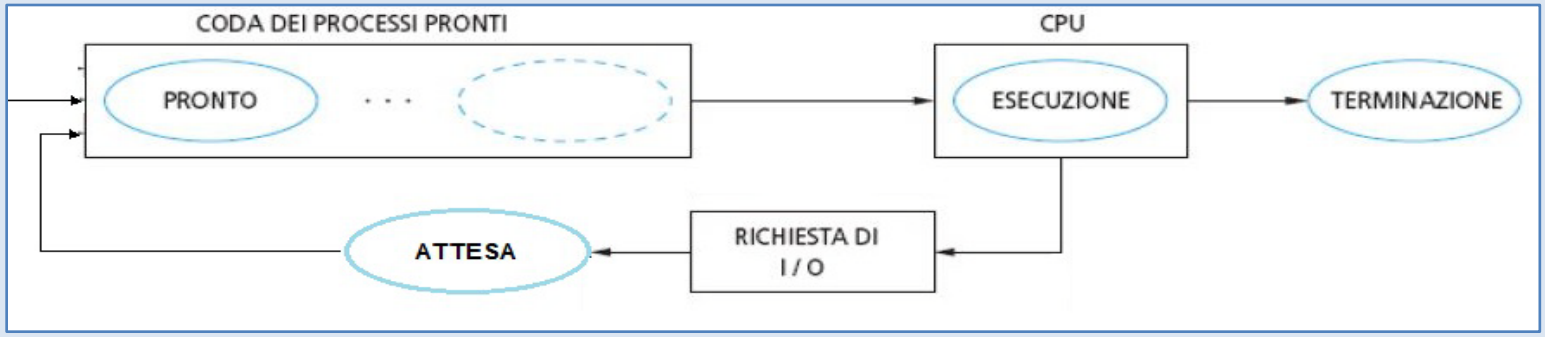
Mentre **Unix** è stato sviluppato quasi da subito come sistema di tipo **preemptive**, nei sistemi **Microsoft** la preemption è stata introdotta solo con **Windows 95**.
Questo è giustificato dal fatto che i sistemi operativi di questa famiglia, a partire dall'MS-Dos, sono nati come sistemi mono utente.

Politiche di scheduling

SENZA PRERILASCIO

- **FCFS** (***First Come First Served***): il processore viene assegnato ai processi in stato di pronto a seconda dell'ordine di arrivo.

  Difetti: tempo medio di attesa abbastanza lungo e **vengono penalizzati i processi brevi** a causa di quelli che richiedono molto tempo di CPU.



Politiche di scheduling

SENZA PRERILASCIO

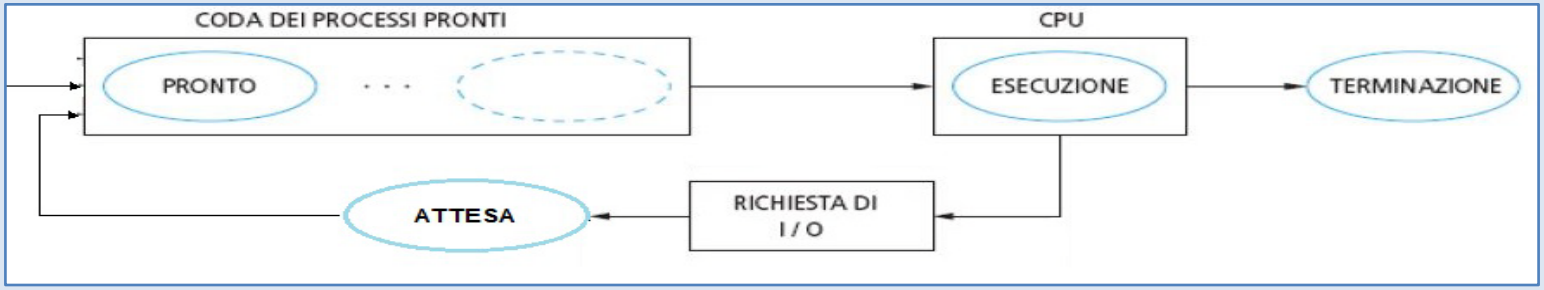
- **SJF (Shortest Job First):** il processore viene assegnato al processo, tra tutti quelli in stato di pronto, che prevede l'intervallo di tempo più breve di utilizzo complessivo della CPU o prima di un I/O.
  In questo caso la **coda** viene **ordinata** per valori crescenti di **durata**.
  Questa politica consente di avere **tempi medi di attesa inferiori** rispetto a FCFS.

  Si può dimostrare che SJF è **ottimale.**

  Spostando un processo breve prima di uno lungo (anche se quest'ultimo è arrivato prima) si migliora l'attesa del processo breve più di quanto si peggiori l'attesa del processo lungo

  ⇒ **il tempo medio di attesa diminuisce!**

  Di conseguenza **diminuisce anche il turnaround**.



Politiche di scheduling

SJF

Nessun algoritmo può produrre un ***tempo di attesa medio*** e **un *turnaround medio* migliori.**

Infatti dati:

n processi con lo stesso tempo di arrivo

e tempo di esecuzione del *processo i-esimo* **t_exe** = $\mathbf{a_i}$,

il **processo j-esimo** dovrà ***aspettare*** **un tempo** $\mathbf{a_1 + a_2 + \ldots + a_{j-1}}$

Quindi:

T_medio di attesa = $[0 + a_1 + (a_1 + a_2) + (a_1 + a_2 + a_3) + \ldots + (a_1 \ldots + a_{n-1})] / n =$

$= [(n-1)a_1 + (n-2)a_2 + \ldots + a_{n-1}] / n$

che **risulta minimizzato** quando $\mathbf{a_1 < a_2 < \ldots < a_n}$

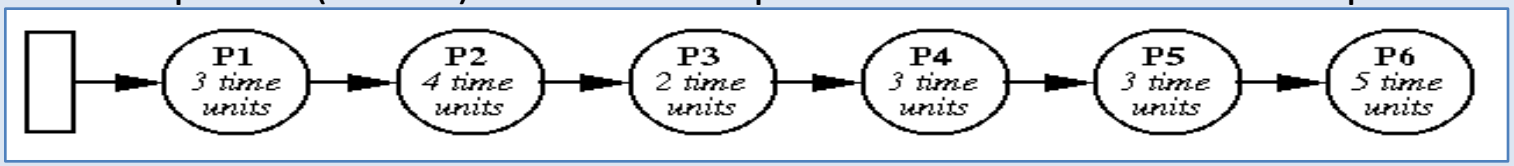
Ma c'è un problema che impedisce di usare questo algoritmo …

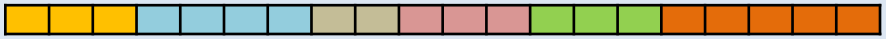
Difetti: spesso **non si hanno sufficienti informazioni** per stabilire quale processo richiederà per primo un'operazione di I/O o quanto tempo di utilizzo complessivo della CPU chiede ogni processo, perché normalmente questi dati sono disponibili solo a run time.

Inoltre i processi più lunghi rischiano la ***starvation*** (morire di fame…), cioè di non essere eseguiti.

- **FCFS**

  Coda dei pronti (READY) e unità di tempo di CPU richieste da ciascun processo



Tempo di CPU 

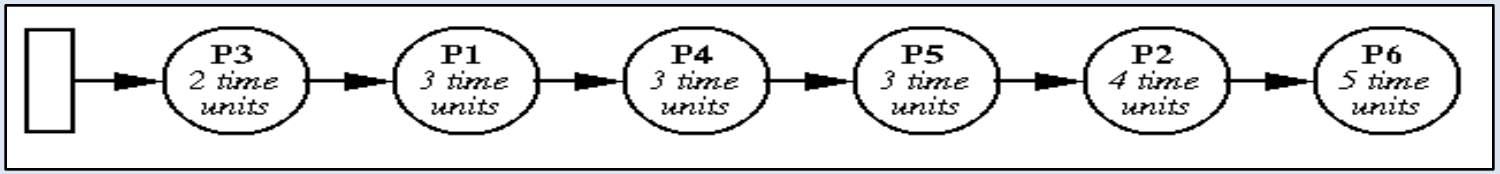
Processi P1 P2 P3 P4 P5 P6

Tempo medio di attesa: ?

Tempo medio di fine esecuzione: ?

- **SJF**

  Coda ORDINATA dei pronti (READY) e unità di tempo di CPU richieste da ciascun processo



Tempo di CPU 

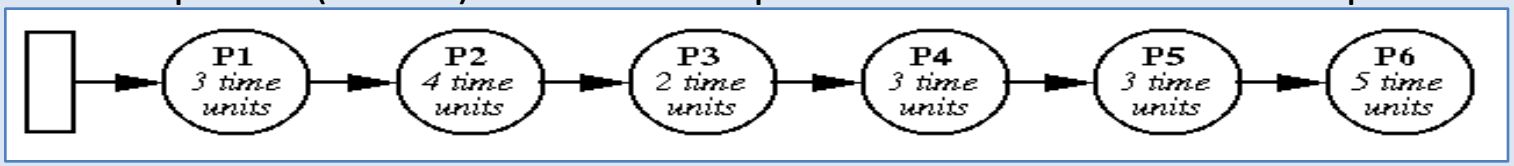
Processi P3 P1 P4 P5 P2 P6

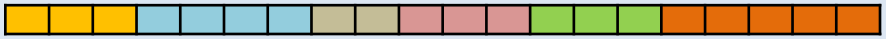
Tempo medio di attesa: ?

Tempo medio di fine esecuzione: ?

- **FCFS**

  Coda dei pronti (READY) e unità di tempo di CPU richieste da ciascun processo



Tempo di CPU 

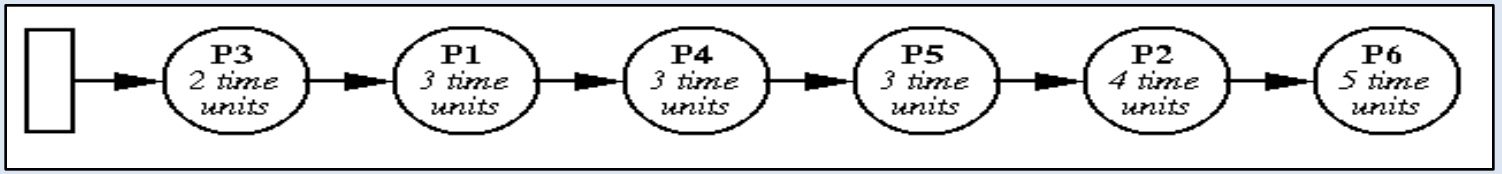
Processi P1 P2 P3 P4 P5 P6

Tempo medio di attesa: (0+3+7+9+12+15)/6 = 7.6 unità di tempo

Tempo medio di fine esecuzione: (3+7+9+12+15+20)/6 = 11 unità di tempo

- **SJF**

  Coda ORDINATA dei pronti (READY) e unità di tempo di CPU richieste da ciascun processo



Tempo di CPU 

Processi P3 P1 P4 P5 P2 P6

Tempo medio di attesa: (0+2+5+8+11+15)/6 = 6.8 unità di tempo

Tempo medio di fine esecuzione: (2+5+8+11+15+20)/6 = 10.16 unità di tempo

Politiche di scheduling

SENZA PRERILASCIO

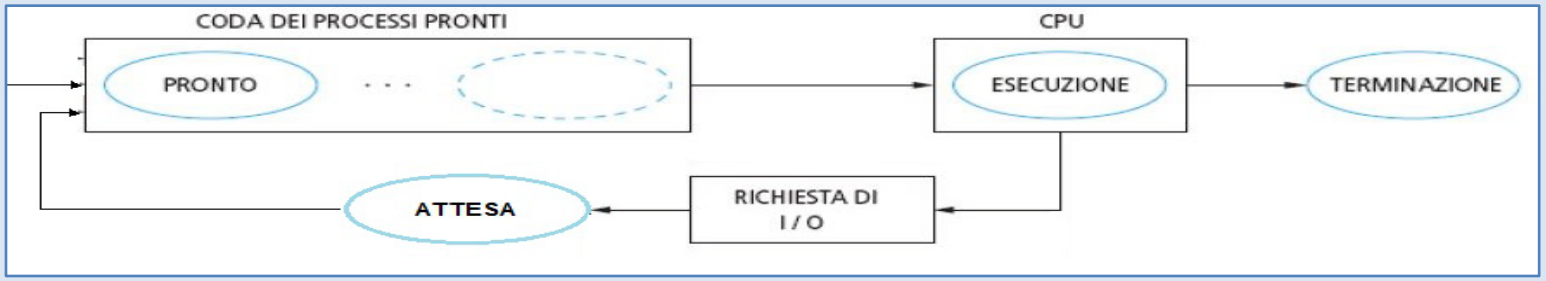
- **Con priorità:** *alcuni processi* vengono considerati *più importanti di altri*.

  La priorità viene assegnata dal S.O. in base a criteri legati al tipo di processo e all'utente che lo ha mandato in esecuzione.

  In questo caso la **coda** di READY viene **ordinata** per valori decrescenti di **priorità** dei processi e ogni volta viene scelto il processo con priorità più alta da mandare in esecuzione. Se due processi hanno la stessa priorità verrà servito il primo arrivato in coda.

  Se arriva in coda di READY un nuovo processo viene inserito rispettando l'ordinamento.

Difetti: se continuano ad arrivare processi con alta priorità quelli con priorità bassa rischiano la ***starvation.***



Politiche di scheduling

CON PRERILASCIO

- **Con priorità:** gli algoritmi che si basano sulla priorità possono essere anche di **tipo preemptive.**

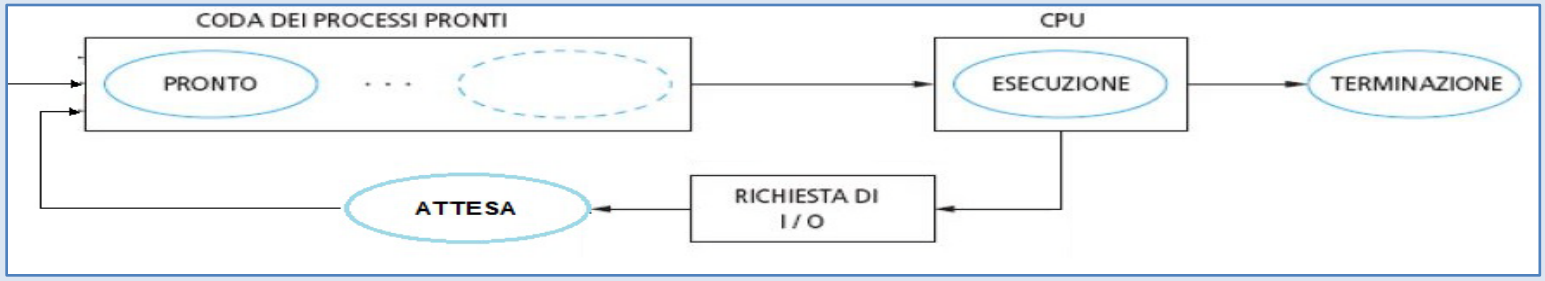
  La **coda** viene **ordinata** come nel caso precedente per **priorità** crescente.

  La *differenza* è la seguente:
  se arriva in coda un processo con priorità maggiore di quello che sta utilizzando la CPU, ***viene forzato il prerilascio***.

  Il **prerilascio** può avvenire:

  - Quando arriva un **nuovo processo** in stato di READY **che ha priorità maggiore** di quello in stato di RUN
  - Quando un **processo** passa dallo stato di WAIT allo stato di READY ed **ha priorità maggiore** di quello in stato di RUN

  Difetti: i processi con bassa priorità rischiano la ***starvation.*** Per questo spesso si usa una priorità dinamica che aumenta con l'anzianità del processo.



Politiche di scheduling

CON PRERILASCIO

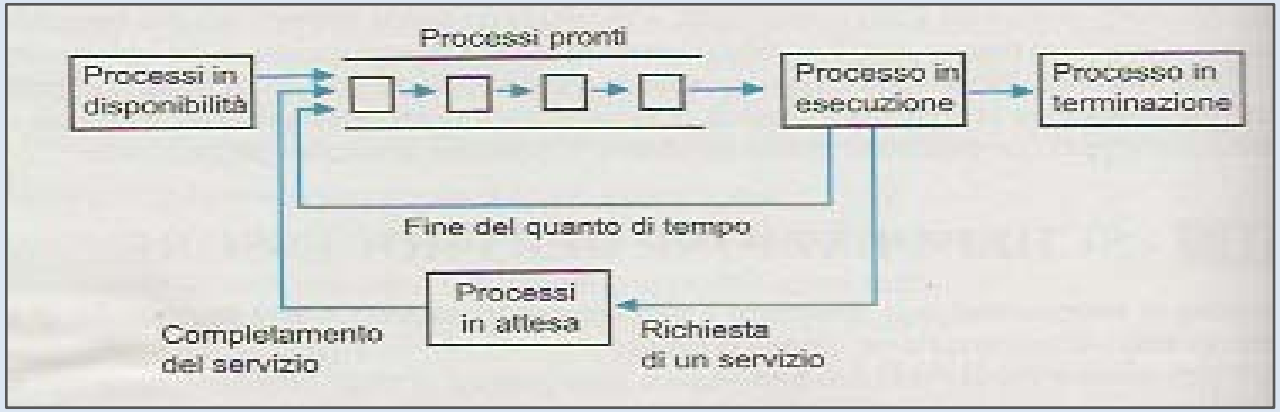
- **Round Robin:** *tutti i processi* vengono considerati *della stessa importanza*.

  A ciascun processo viene concesso l'uso del processore per un $\triangle T$ di tempo detto "**quanto**" di tempo (o **time-slice**).

  Il timer allo scadere del "quanto" di tempo invia un segnale di interruzione che provoca l'intervento del S.O. che ***forza il processo a prerilasciare il processore*** e a tornare in stato di READY.

  In questo modo **si evita la *starvation***, perché a turno tutti i processi usano la CPU per uno stesso intervallo di tempo.

  La politica RR è equivalente a FCFS con prerilascio.



Politiche di scheduling

Round Robin:

Se ci sono ***N processi in coda READY e il quanto di tempo è Q****, allora ogni processo riceve 1/N del tempo della CPU* in blocchi di almeno *Q unità di tempo per volta.*

Il **RR** è l'algoritmo di scheduling naturale per implementare il ***time sharing***, ed è quindi particolarmente **adatto per i sistemi interattivi.**

Nel caso peggiore un utente non aspetta mai più di

(N-1)*Q *unità di tempo*

prima che il suo processo venga servito.

Politiche di scheduling

Considerazioni su Round Robin

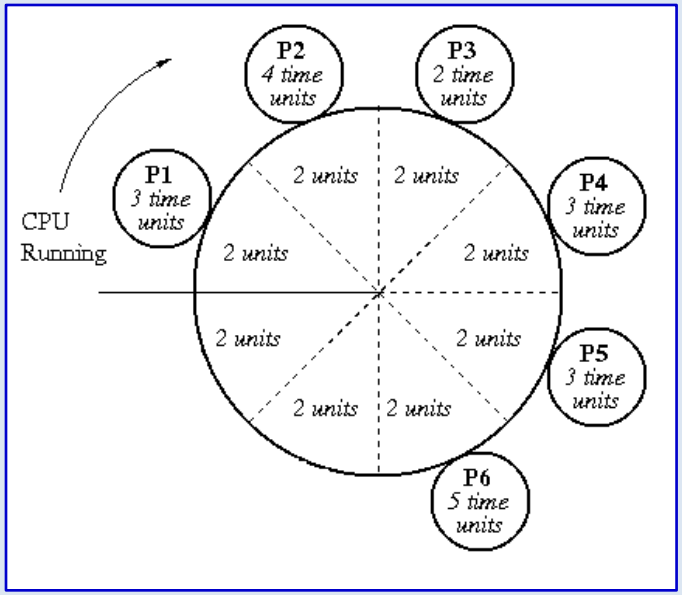
Il *comportamento* del RR dipende molto dal *valore del quanto* di tempo scelto.

Se il time-slice è molto breve → si verificano **interruzioni frequenti** per passare da un processo all'altro (context switch) e l'**efficienza del sistema diminuisce** (overhead).

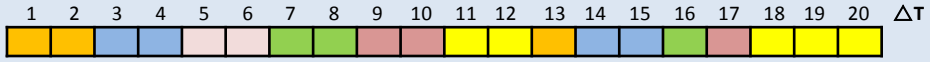
Se il time-slice è molto lungo → il sistema diventa come FCFS (time-slice infinito).
Si verificano **poche interruzioni**, ma si perdono i vantaggi del RR in quanto vengono nuovamente penalizzati i processi che richiedono poco tempo di CPU (aumenta il tempo di risposta dei processi interattivi).

- **Round Robin**

Processi in ordine di arrivo da P1 a P6 in READY e unità di tempo di CPU richieste da ciascun processo



Tempo di CPU



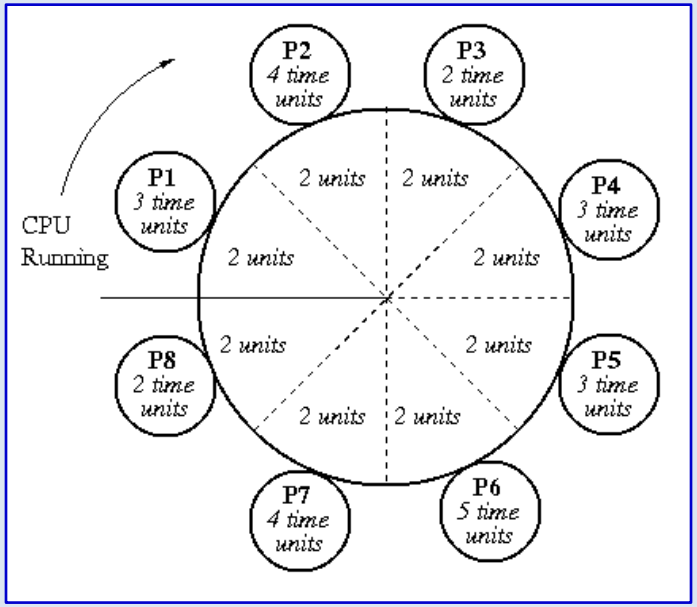
Processi P1 P2 P3 P4 P5 P6 P1 P2 P4 P5 P6 P6

Tempo medio di attesa: ((0+10)+(2+9)+(4)+(6+7)+(8+6)+(10+5))/6 = 11.6 ΔT

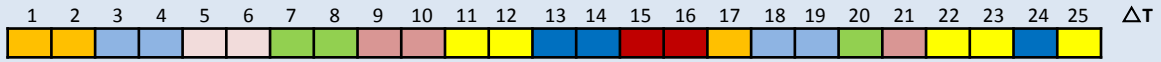
Tempo medio di fine esecuzione: (13+15+6+16+17+20)/6 = 14.5 ΔT

- **Round Robin**

Processi da P1 a P8 in coda dei pronti (READY) e unità di tempo di CPU richieste da ciascun processo.



Tempo di CPU



Processi P1 P2 P3 P4 P5 P6 P7 P8 P1 P2 P4 P5 P6 P7 P6

Tempo medio di attesa: ((0+14)+(2+13)+(4)+(6+11)+(8+10)+(10+9+1)+(12+9)+(14))/8 = 15.37 **△T**

Tempo medio di fine esecuzione : (17+19+6+20+21+25+24+16)/8 = 18.5 **△T**