# La schedulazione dei processi

# cap. A6 del libro

# PROCESSI

# Processi

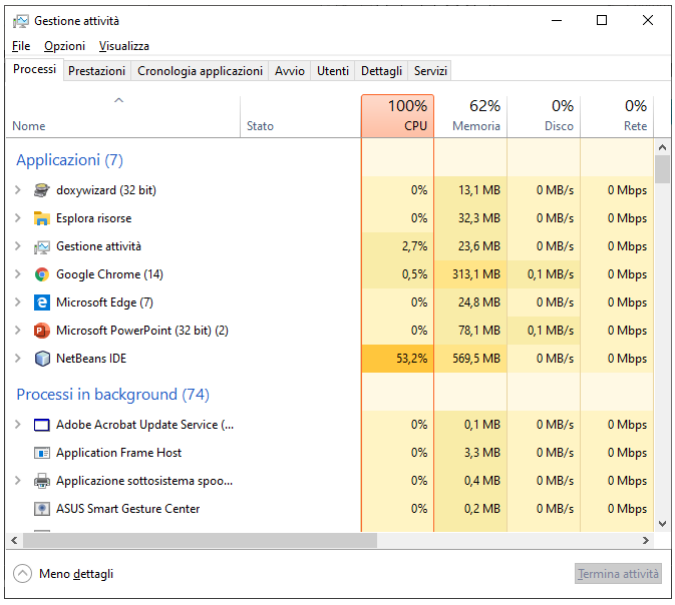
Gli attuali calcolatori, diversamente da quanto avveniva una volta, consentono la multiprogrammazione, ossia l'esecuzione concorrente di più programmi.

Una tale caratteristica necessita di un efficiente sistema di gestione e di ripartizione delle risorse nonché di efficaci servizi di sistema e di controllo dei programmi.

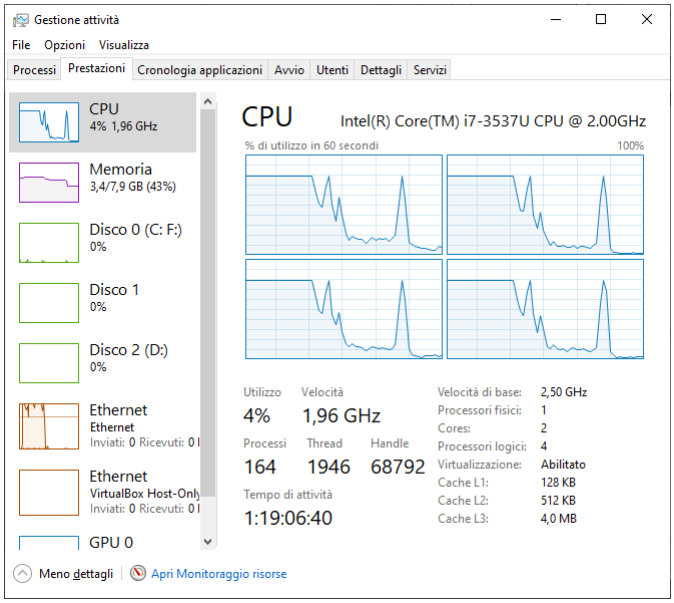
Alla base di questa tecnica vi è la suddivisione dei programmi in **processi**.

# Processi

è possibile verificare in ogni momento con lo strumento «Gestione attività» di Windows:



Lo stesso strumento permette di monitorare il carico di lavoro dei singoli core del processore:




# Multi-processing

*L'utente ha la sensazione che le proprie applicazioni siano eseguite contemporaneamente dal computer:*
può, per esempio, ascoltare un brano musicale mentre scrive un documento.

In realtà *l'esperienza di contemporaneità* dell'utente è data dall'elevata velocità di esecuzione dei programmi da parte del computer: il sistema operativo, infatti, esegue le applicazioni singolarmente passando rapidamente il controllo dall'una all'altra e, di conseguenza interrompendone continuamente l'esecuzione

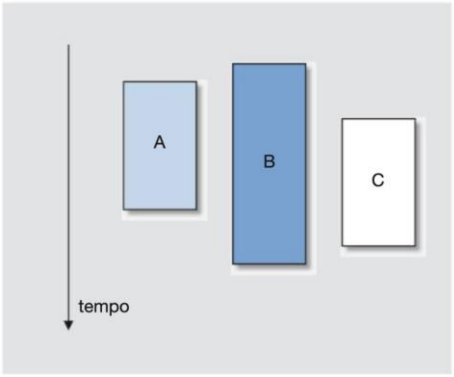



FIGURA **1**

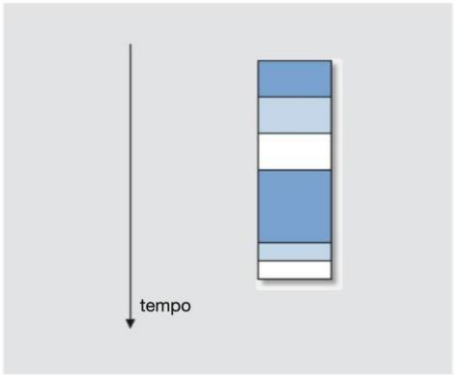
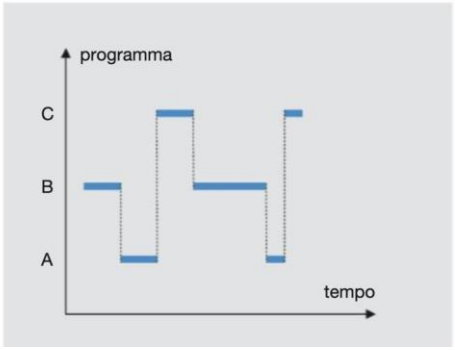



FIGURA **2**

Avendo 3 programmi contemporaneamente attivi A, B e C la visione dell'utente è quella di una esecuzione contemporanea (FIGURA **1**).
In realtà il sistema operativo gestisce in modo strettamente sequenziale l'esecuzione dei tre programmi interrompendone e riprendendone l'esecuzione più volte (FIGURA **2**).
Il grafico di FIGURA **3** illustra quello che avviene in realtà: l'esecuzione frammentata e alternata del codice dei singoli programmi da parte dell'unico processore disponibile.




# Processi

**Un processo è definito come un programma in esecuzione**.

Il codice del programma è normalmente contenuto in un file memorizzato su una unità di memoria persistente; al momento dell'attivazione il codice *viene caricato dal sistema operativo nella memoria del computer*, da dove può essere effettivamente eseguito: da questo momento il programma *diviene un processo*.

# Processi

**OSSERVAZIONE**
Un programma esiste indipendentemente dal computer che è in grado di eseguirlo: per esempio il pacchetto di installazione di un videogioco memorizzato su una memory-stick USB contiene i file con il codice eseguibile del programma che, al momento dell'installazione, sono copiati in una unità di memoria persistente del computer.
Ma è solo quando si esegue il programma – il videogioco in questo esempio- che il sistema operativo crea il processo corrispondente.

# DESCRITTORE DI PROCESSO

# Processi

Nel corso della propria esecuzione normalmente un processo impiega, oltre al processore, altre **risorse hardware o software** rese disponibili dal sistema operativo: memoria allocata, file aperti, connessioni di rete attivate, dispositivi di input e di output utilizzati, ecc.

I descrittori di tutte le risorse usate da un processo costituiscono lo **stato del processo** a un certo istante della propria esecuzione.

Inoltre un processo usa e/o produce **dati** (per esempio un'applicazione multimediale legge il contenuto di un file che contiene un filmato e produce i dati necessari alla scheda video e alla scheda audio per la sua riproduzione).
*I dati elaborati da un processo sono registrati nelle aree di memoria riservate dal sistema operativo al processo stesso.*

# Descrittore di processo

Il sistema operativo associa a ogni processo fin dal momento della sua creazione un identificativo numerico univoco denominato **PID (Process ID)**

Ad ogni processo il S. O. associa una struttura dati, detta **descrittore di processo** o Process Control Block (**PCB**), nella quale vengono registrate una serie di informazioni utili per la gestione dello stesso processo.

# Descrittore di processo

Un **PCB** contiene:

- **Stato del processo**.
- **Contatore di programma** (program counter) contiene l'indirizzo della successiva istruzione da eseguire per tale processo.
- **Registri di CPU**: I registri variano in numero e tipo in base all'architettura, essi comprendono accumulatori, registri d'indice, stack pointer. Quando si verifica una interruzione della CPU, tutte queste informazioni insieme al contatore di programma devono essere salvate (informazioni presenti nei registri di CPU durante l'esecuzione), in modo da permettere la corretta esecuzione del processo in un momento successivo.
- **Informazioni sullo scheduling di CPU**: Informazioni relative alla priorità del processo, i puntatori alle code di scheduling e i parametri di scheduling.
- **Informazione sulla gestione della memoria**: Valori dei registri base e di limite, tabelle delle pagine/segmenti.
- **Informazioni di contabilizzazione delle risorse**: tempo d'uso della CPU e il tempo reale di utilizzo...
- **Informazioni sullo stato di I/O**: La lista dei dispositivi I/O assegnati ad un processo, elenco file aperti.

# Descrittore di processo

Il S. O. gestisce i PCB di tutti i processi, disponendoli in strutture dati chiamate **code**; ogni coda contiene PCB caratterizzati dallo stesso stato: ready, waiting, …

**PCB**

| NOME |
|---|
| STATO |
| PROGRAM COUNTER |
| REGISTRI DI CPU |
| LIMITI DI MEMORIA |
| FILE APERTI |
| …… |

## Osservazione

Due istanze contemporaneamente attive di una specifica applicazione (per esempio di un programma per la redazione di documenti) hanno in comune lo stesso programma, ma hanno un distinto ambiente in quanto i dati (per esempio il testo visualizzato) e le risorse (per esempio il file aperto) sono diverse: sono quindi due processi distinti.

Allo scopo di ridurre l'occupazione della memoria del computer, in questo caso il codice del programma viene caricato una sola volta, ma l'ambiente è ovviamente replicato per ogni istanza del programma in esecuzione.

# CONTEXT SWITCH

# Context switch

**Il cambio di contesto (contest switch) è l'esecuzione di un insieme di operazioni che permette lo scambio tra il processo in esecuzione con un altro scelto dalla coda dei processi pronti.**
In questo lasso di tempo il S. O. realizza, ordinatamente, le seguenti operazioni:

- **Salvataggio dello stato del processo uscente**
- **Selezione del nuovo processo**
- **Ripristino dello stato del processo entrante**

# Context switch

Il context-switching è un'attività del sistema operativo – in particolare del gestore dei processi – che impiega la risorsa processore per essere svolta;

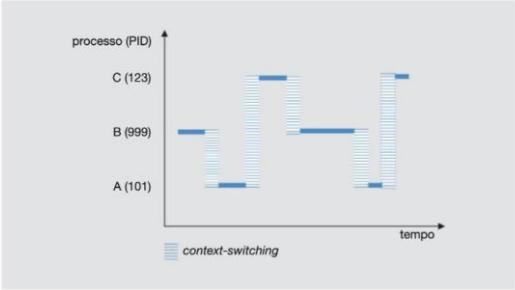
Il tempo utilizzato dal S. O. per le operazioni di context switch, in rapporto al **time-slice** (quanto di tempo di CPU), deve essere tale da non implicare:

- **riduzione dell'efficienza della CPU**
- **lunghi tempi di risposta**

A tal fine, un giusto compromesso si dimostra essere in cui il tempo di context switch non superi il 25% del time-slice

# Context switch

La sensazione dell'utente di una esecuzione contemporanea dei vari programmi presenta, quindi, il costo nascosto di un aggravio del tempo di esecuzione totale dovuta al tempo di context-switching da parte del sistema operativo.



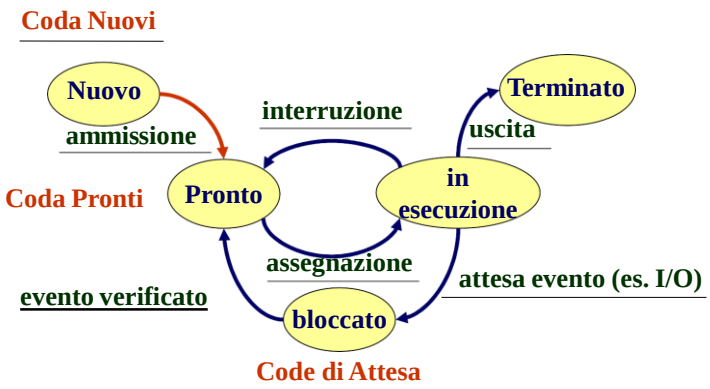

# STATI DI UN PROCESSO

# Stati di un processo

Ogni processo è caratterizzato da uno **stato**, cioè una condizione in cui esso si viene a trovare in un dato momento del suo ciclo di vita:

- **Nuovo (New)**: il processo viene creato
- **Pronto (Ready)**: il processo non è in esecuzione ma è pronto per farlo in quanto l'unica risorsa mancante è la CPU
- **In esecuzione (Running)**: il processo è in esecuzione
- **Bloccato, in attesa (Waiting, o Blocked o Sleeping)**: il processo è in attesa di un evento. Ha invocato un servizio del sistema operativo (ad esempio operazione di lettura e scrittura o in attesa di segnalazione da parte di un altro processo)
- **Terminato (Terminated)**: il processo rilascia la risorsa processore

# Stati di un processo



OSSERVAZIONE

Nello **stato di Run** vi possono essere al massimo **tanti processi quanti sono i processori** (in particolare, su computer con un solo processore solo un processo può trovarsi in stato di Run), ma negli **stati di Wait e di Ready** vi possono essere **numerosi processi** in attesa di essere sbloccati o selezionati dal sistema operativo.

# Schedulazione Dei Processi

# Schedulazione Dei Processi

Lo **scheduling** è una tecnica di allocazione delle risorse utilizzata dai sistemi operativi, per gestire le richieste derivanti da più processi che concorrono per il loro utilizzo.

*L'obiettivo della multiprogrammazione consiste nel disporre della possibilità di eseguire*, non inteso come materialmente eseguito dalla CPU, perché ciò non è possibile, *di più processi contemporaneamente, con lo scopo di massimizzare l'utilizzo della CPU*.

Se esistono più processi, quelli che non sono in esecuzione devono aspettare che la CPU si liberi e la loro esecuzione possa essere ripresa o iniziata

Esso mira a rendere il sistema

- efficiente
- produttivo
- rapido nelle risposte

# Schedulazione Dei Processi

**Nel sistema a singola CPU, potendo eseguire un solo processo alla volta, gli altri processi devono attendere che la CPU sia libera per poter essere assegnata ad uno di loro.**

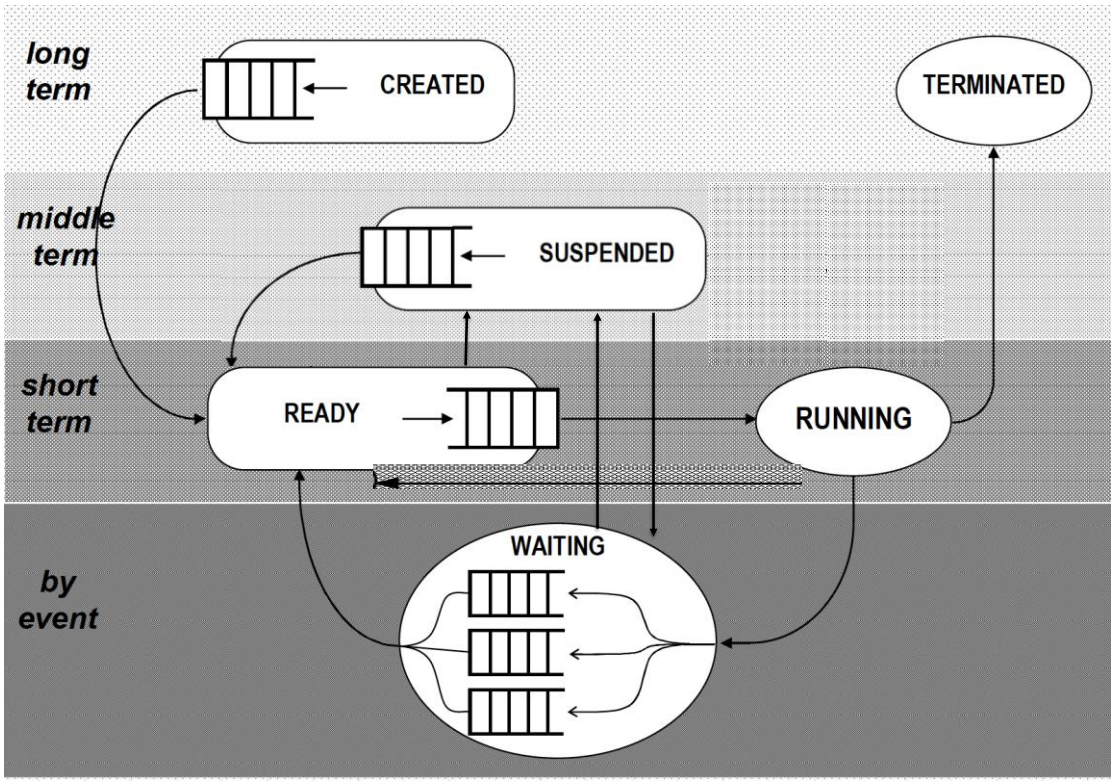
# La Schedulazione del Processore

- **Definizione delle politiche di ordinamento dei processi**

## Livelli di schedulazione

Nella gestione delle risorse di un elaboratore, una possibile classificazione dello scheduling può essere espressa in funzione del tempo:

- **Schedulazione a breve termine**
- **Schedulazione a medio termine**
- **Schedulazione a lungo termine**




# Livelli di schedulazione

In alcuni sistemi si introduce un livello di scheduling intermedio

Lo scheduler a medio termine può rimuovere processi dalla memoria (e dalla contesa attiva per la CPU) e cosi ridurre il grado di multiprogrammazione, successivamente il processo può essere reintrodotto, questo processo si chiama **swapping**.

Tale processo può servire a liberare memoria in caso di necessità o a migliorare la combinazione di processi.

- Coordina le operazioni di trasferimento temporaneo dei processi in memoria secondaria quando la memoria centrale è esaurita al fine di migliorare le prestazioni del sistema

- Interviene nella gestione dei processi in attesa di eventi da lungo tempo

# Scheduling della CPU

Lo scheduling costituisce una funzione fondamentale dei S. O., alla quale sono sottoposte quasi tutte le risorse di un elaboratore.

In particolare, lo **scheduling della CPU** consiste nella scelta, da parte del S. O., del processo a cui assegnare la CPU e tiene conto del fatto che l'esecuzione di un processo è una sequenza alternata di

- operazioni di CPU
- operazioni di I/O o attese di eventi

# Scheduling della CPU

Quando la CPU rimane inattiva, il **process scheduler,** modulo del kernel che seleziona dalla coda dei processi pronti il prossimo processo a cui assegnare la CPU.

Tuttavia, l'effettiva assegnazione della CPU al processo prescelto per l'esecuzione è affidata al **dispatcher**, modulo del kernel che gestisce

- **il context switch**
- **il passaggio al modo utente**
- **il salto alla giusta posizione del programma utente**

# OBIETTIVI DELLA SCHEDULAZIONE

## Processi I/O Bound

Alcuni processi – in particolare quelli che prevedono una continua interazione con l'utente – passano molto tempo nello stato di *Wait* ad attendere come input un'azione (per esempio l'attivazione del mouse in una specifica posizione, oppure la digitazione di un testo) che viene intercettata dal sistema operativo e successivamente inoltrata al processo interessato: processi di questo tipo sono definiti di tipo *I/O-bound*, cioè dipendenti principalmente dall'I/O (Input/Output) (FIGURA 6).

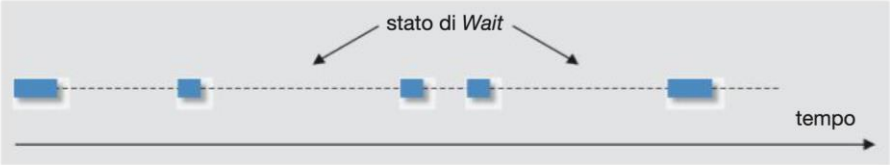



FIGURA 6 Esecuzione di un processo di tipo *I/O-bound*.

# Processi CPU-Bound

Altri processi – in particolare quelli che devono svolgere un'intensa attività di elaborazione di dati presenti in memoria – non rilasciano quasi mai il processore passando nello stato di *Wait* e sono definiti di tipo *CPU-bound*, cioè dipendenti principalmente dalla disponibilità del processore (FIGURA 7).

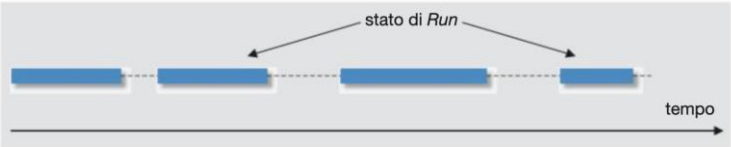



FIGURA 7 Esecuzione di un processo di tipo *CPU-bound*.

# Problematiche interazione
# Processi IO-bound e CPU-bound

Compito del gestore dei processi del sistema operativo è quello di ottimizzare l'uso del processore: avere più processi di tipo *I/O-bound* contemporaneamente in esecuzione aumenta l'utilizzazione del processore, in quanto i lunghi periodi in cui un processo resta in stato di *Wait* possono essere utilizzati per eseguire altri processi nello stato di *Run*. Ma l'esecuzione di uno o più processi di tipo *CPU-bound*, che rilasciano raramente il processore uscendo dallo stato di *Run*, ha come conseguenza un progressivo ritardo nell'esecuzione degli altri processi che, pur avendo terminato l'attesa della richiesta avanzata al sistema operativo ed essendo quindi passati nello stato di *Ready*, devono nuovamente rimanere a lungo in attesa che sia disponibile il processore. In particolare, per i processi che interagiscono con l'utente, questo ritardo viene percepito come un malfunzionamento e non è quindi accettabile.

# Criteri di scheduling

Esistono diversi algoritmi per lo scheduling della CPU (breve termine), hanno proprietà differenti e possono favorire particolari classi di processi.
Per classificare questi algortmi per poi fare un accurata scelta di uso, esistono diversi criteri:

- **Utilizzo della CPU**: la CPU deve essere più attiva possibile
- **Produttività**: Una misura del lavoro è data dal numero di processi che si riescono a completare in una unità di tempo, tale misura viene detta throughput.
- **Tempo di completamento**: L'intervallo di tempo che intercorre tra la sottomissione di un processo e il suo completamento viene detto tempo di completamento (turnaround time).
- **Tempo di attesa:** L'algoritmo di scheduling della CPU, influisce solo sul tempo di attesa nella coda dei processi pronti. Il tempo d'attesa è la somma degli intervalli d'attesa passati nella coda dei processi pronti.
- **Tempo di risposta**: tempo che intercorre tra la sottomissione di una richiesta e la prima risposta prodotta.

# Obiettivi dello schedulazione

Non potendo realizzare tutti gli obiettivi contemporaneamente, ogni algoritmo di scheduling della CPU cerca di ottenere un giusto compromesso oppure al raggiungimento di alcuni di essi. Tipicamente,

- nei **sistemi batch** (elaborazione a lotti) si cerca di massimizzare il throughput.

- nei **sistemi interattivi** (tempi di risposta immediati) è essenziale minimizzare i tempi di risposta e di attesa dei processi.

# CLASSIFICAZIONE DELLE POLITICHE DI SCHEDULAZIONE

## Classificazione delle politiche

In generale, le politiche di scheduling della CPU si differenziano in base a tre criteri di classificazione:

- **non-preemptive/preemptive**
- **senza priorità/con priorità**
- **statiche/dinamiche**

Gli algoritmi di scheduling, nell'assegnare la CPU ai processi pronti, utilizzano opportune combinazioni di tali criteri, in base a precise strategie di gestione delle risorse.

# Classificazione delle politiche

- **Politiche senza prelazione** (non preemptive):

Una volta che la CPU è stata assegnata ad un processo, essa non può più essergli tolta se non quando il processo ha esaurito il suo quanto di tempo o richieda un'operazione di I/O.

- **Politiche con prelazione** (preemptive):

La CPU può essere forzatamente tolta al processo in esecuzione per essere assegnata ad un altro processo.

# Classificazione delle politiche

- **Politiche senza priorità**:

I processi sono sullo stesso piano dal punto di vista dell'urgenza di esecuzione.

- **Politiche con priorità**:

Suddividono in code di livello diverso i processi in stato di pronto, a seconda della loro importanza. Sono necessarie nei sistemi real-time e interattivi.

# Classificazione delle politiche

- **Politiche statiche**:

Un processo conserva nel tempo i suoi diritti priorità per ciò che riguarda l'accesso all'unità centrale.

- **Politiche dinamiche**:

I processi modificano nel tempo i propri diritti di accesso all'unità centrale.

# POLITICHE DI SCHEDULAZIONE

- **FCFS:First Come First Served**
- **SJF: Shortest Job First**
- **Priorità**
- **Round Robin**

## FCFS: First Come First Served

La CPU è assegnata ai singoli processi in base al loro ordine nella coda dei processi pronti (ready queue), stabilito in base al tempo di arrivo oppure dal numero identificativo (PID).

- La ready queue è gestita in **modalità FIFO** (First In First Out), al processo che giunge nella ready queue viene attribuito l'ultimo posto della coda
- La CPU è assegnata al processo situato alla testa alla ready queue

## Politiche di schedulazione

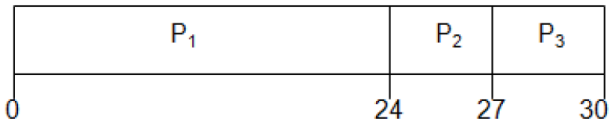
La politica **FCFS**

- È senza prelazione: Il processo rimane in esecuzione sino al termine del suo tempo di servizio o fino a quando non necessita di un'operazione di I/O.
- Tende a penalizzare i processi I/O bound e quelli di breve durata, poiché costretti a lunghe attese dietro processi CPU bound.
- Prevede tempi di attesa molto variabili e generalmente lunghi. La gestione dei dispositivi è poco efficiente ed è adeguata per sistemi batch.

# Esempio

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

Suppose that the processes arrive in the order: $P_1$ , $P_2$ , $P_3$
The Gantt Chart for the schedule is:

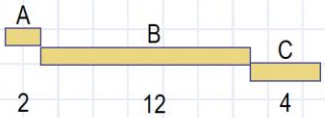



Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27
Average waiting time: (0 + 24 + 27)/3 = 17

# Esempio 2

processo A: tempo di esecuzione = 2 [u.t.]
processo B: tempo di esecuzione = 12 [u.t.]
processo C: tempo di esecuzione = 4 [u.t.]

N.B. trascuriamo per semplicità i tempi di scambio di contesto




**TEMPO DI ATTESA**

$T_{att}$ (A) = 0

$T_{att}$ (B) = 2

$T_{att}$ (C) = 2 + 12 = 14

$T_{att}$ (medio) = ( 0 + 2 + 14 ) / 3 = 5,3 [u.t.]

**TEMPO DI *TURN AROUND***

$T_{ta}$ (A) = 2

$T_{ta}$ (B) = 2 + 12 = 14

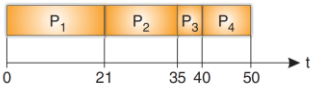
$T_{ta}$ (C) = 2 + 12 + 4 = 18

$T_{ta}$ (medio) = ( 2 + 14 + 18 ) / 3 = 11,3 [u.t.]

## PROGETTO 2

**Verificare che il tempo medio di attesa e il tempo medio di risposta si riducono privilegiando i processi brevi, cioè eseguendoli per durate crescenti.**

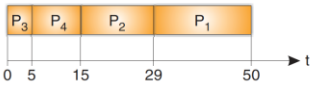
Si considerino quattro processi **P1**, **P2**, **P3**, **P4**, di durata pari a 21, 14, 5, 10 ms, che arrivano quasi contemporaneamente nell'ordine specificato.
Supponendo che siano eseguiti in modalità FCFS, i processi iniziano e terminano negli istanti indicati dal seguente schema temporale.




$\mathbf{P_1}$ inizia al tempo 0 e termina dopo 21 ms, $\mathbf{P_2}$ deve attendere 21 ms per iniziare e termina al millisecondo 35, e così via. Noti i tempi di inizio e di fine di ogni processo, si può calcolare $\mathbf{T_{ma}}$, il tempo medio di attesa per l'inizio del processo, e $\mathbf{T_{ms}}$, il tempo medio di fine esecuzione.

$T_{ma} = ( 0 + 21 + 35 + 40 ) / 4 = 24.0$ ms
$T_{ms} = ( 21 + 35 + 40 + 50 ) / 4 = 36.5$ ms

Si noti che il tempo medio, $T_{ms}$, sopra calcolato non è altro che il valore medio del tempo di *turnaround* dei quattro processi.
Calcoliamo ora il valore dei due tempi medi quando i quattro processi sono eseguiti non più in ordine di arrivo ma in ordine di durata, privilegiando i processi brevi. Lo schema temporale cambia nel seguente modo:




I nuovi valori del tempo medio di attesa $\mathbf{T_{ma}}$ e del tempo medio di uscita dei processi $\mathbf{T_{ms}}$ sono:

$T_{ma} = ( 0 + 5 + 15 + 29 ) / 4 = 12.25$ ms
$T_{ms} = ( 5 + 15 + 29 + 50 ) / 4 = 24.75$ ms

che sono notevolmente ridotti rispetto ai precedenti valori. Si può infatti dimostrare che il minor tempo di *turnaround* si ottiene eseguendo i processi per valori crescenti di durata. Questa idea è alla base dell'algoritmo di schedulazione detto *Shortest Job First*.

# Politiche di schedulazione

### SJF: Shortest Job First

La CPU è assegnata al processo, tra quelli in stato di pronto, con intervallo di tempo di utilizzo della CPU più breve (tempo di servizio più piccolo).

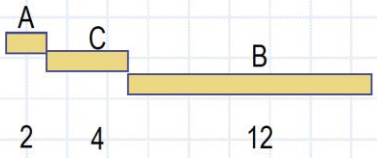
Nel caso in cui più processi pronti presentino gli stessi tempi di servizio, la scelta viene fatta secondo la politica FCFS.

# Politiche di schedulazione

La politica SJF prevede due casi:

- **SJF senza prelazione**: il processo in esecuzione trattiene la CPU sino al termine del suo quanto di tempo o fino a quando non necessita di un'operazione di I/O.
- **SJF con prelazione**: se, nella coda dei processi pronti giunge un processo con un tempo di servizio più breve di quello che rimane al processo in esecuzione, la CPU viene rilasciata e ceduta al processo appena arrivato (**Shortest Remaining Time First = SRTF**).

## Shortest Job First senza prerilascio




**TEMPO DI ATTESA**

$T_{att}(A) = 0$

$T_{att}(B) = 2 + 4 = 6$

$T_{att}(C) = 2$

$T_{att}(medio) = (0 + 6 + 2) / 3 = 2{,}6$ [u.t.]

**TEMPO DI *TURN AROUND***

$T_{ta}(A) = 2$

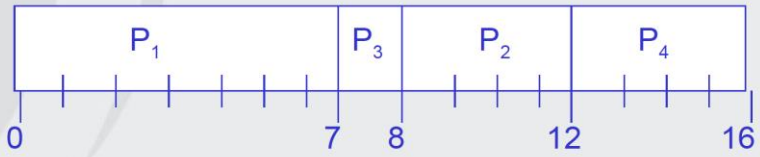
$T_{ta}(B) = 2 + 4 + 12 = 18$

$T_{ta}(C) = 2 + 4 = 6$

$T_{ta}(medio) = (2 + 18 + 6) / 3 = 8{,}6$ [u.t.]

## SJF Non-Preemptive

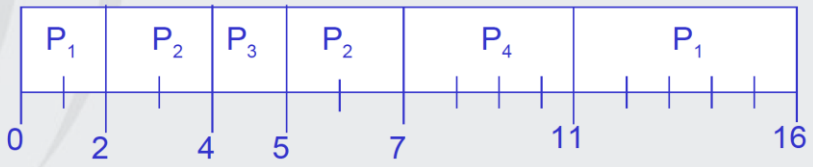
| Processo | Tempo di arrivo | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |




**Tempo di attesa medio: (0 + (8-2) + (7-4) +(12-5))/4 = 4**

## SJF Preemptive

| Processo | Tempo di arrivo | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |




**Tempo di attesa medio: (9 + 1 + 0 +2)/4 = 3**

# Politiche di schedulazione

### Scheduling con priorità

A ciascun processo, in fase di generazione, viene attribuito un livello di priorità, espresso da un numero intero; convenzionalmente, all'intero più piccolo corrisponde la priorità più elevata.

Nella politica di scheduling per priorità, la CPU è assegnata al processo, tra quelli in stato di pronto, che presenta la massima priorità.

A parità di priorità, si ricorre alla politica FCFS

# Politiche di schedulazione

La politica **con priorità** può essere:

- **senza prelazione**: il processo in esecuzione trattiene la CPU sino al termine del suo tempo di servizio o fino a quando non necessita di un'operazione di I/O.
- **con prelazione**: se nella coda dei processi pronti arriva un processo con priorità maggiore di quella del processo in esecuzione, questo rilascia la CPU, cedendola al processo appena arrivato.

# Politiche di schedulazione

Le priorità possono essere:

- **definite dal S. O.**, secondo i criteri:
  - limiti di tempo
  - requisiti di memoria
  - numero di file aperti
  - tipi di processo (CPU bound o I/O bound)

  ……
- **definite dall'utente** , secondo:
  - l'importanza del processo
  - il tempo di esecuzione previsto

  ……

# Politiche di priorità

➢**statiche**: una volta stabilite, non variano nel tempo.

➢**dinamiche**: variano nel tempo, per evitare che alcuni processi possano essere bloccati per un tempo indefinito (**starvation**), a causa della costante presenza nella ready queue di processi a più elevata priorità.

Una possibile soluzione allo starvation dei processi è data dalla procedura di **aging** (invecchiamento): la priorità di un processo può

- ✓ aumentare al crescere del suo tempo di attesa

*Esercitazioni su Scheduling della CPU*

- **Esercizio 1**

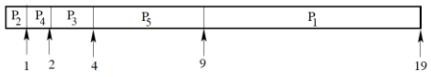
  Si considerino cinque processi caratterizzati dai seguenti tempi di esecuzione (in millisecondi) e priorità date esternamente (un codice di priorità più piccolo indica una priorità più alta):

| *Processo no.* | *Tempo di esecuzione* | *Priorità* |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 3 |
| $P_4$ | 1 | 4 |
| $P_5$ | 5 | 2 |

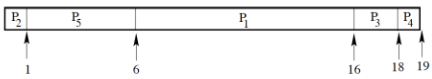
  I processi usano solo la CPU ed arrivano tutti al tempo 0 nell'ordine $P_1$, $P_2$, $P_3$, $P_4$, $P_5$. Si illustri quale risulta l'ordine di esecuzione nel caso delle politiche SJF ed a priorità. Si calcoli il tempo di attesa medio nei due casi.

  *Soluzione*

SJF




Scheduling a Priorita'




$$t_a(SJF) = \frac{9+0+2+1+4}{5} = 3.2$$

$$t_a(Pr) = \frac{6+0+16+18+1}{5} = 8.2$$

# Politiche di schedulazione

## Round Robin

La CPU viene assegnata, a turno, ad ogni processo, tra quelli in stato di pronto, per un certo periodo di tempo (time-slice o quanto di tempo).

Se, alla fine di tale periodo, il processo in esecuzione non è terminato, esso viene ricondotto nella coda dei processi pronti, nella posizione corrispondente alla sua priorità o in ultima posizione.

# Round Robin

Progettato per i sistemi a partizione di tempo, **ciascun processo riceve una piccola quantità di tempo di CPU**, chiamato *quanto di tempo*, e la coda dei processi pronti è trattata come una **coda circolare**. Viene pertanto lasciato il quanto di tempo se il processo termina prima si passa al successivo altrimenti il processo è forzato a rilasciare.

Per il funzionamento la coda di priorità viene gestita come una coda FIFO, i nuovi processi si aggiungono alla fine della coda.
Nell'algoritmo RR si assegna la CPU ad un processo per non più di un quanto di tempo per volta. Se la durata della sequenza di operazioni eccede il quanto di tempo, il processo viene sottoposto a prelazione e riportato nella coda dei processi pronti, pertanto RR è un algoritmo con prelazione.

# Politiche di schedulazione

La politica di scheduling **Round Robin**, nel caso dei sistemi time-sharing:

- è con prelazione
- gestisce la ready queue in modo circolare
- garantisce tempi di risposta bassi senza penalizzare l'avanzamento di processi I/O-bound

# Politiche di schedulazione

Il **rendimento** del Round Robin dipende molto dalla dimensione del time-slice:

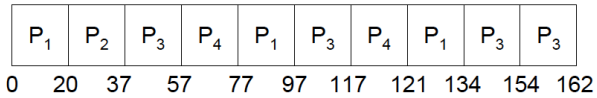
- un time-slice troppo grande rischia di far tendere la politica Round Robin a quella FCFS.
- un time-slice troppo piccolo consente tempi di risposta brevi ma impegna eccessivamente la CPU per le operazioni di context switch, con conseguente calo della produttività e dell'efficienza del sistema.

Nei sistemi reali il valore del time-slice che garantisce buone prestazioni è compreso tra 20 e 50 millisecondi.

## Esempio di RR con Q = 20

| Processi | tempo di burst |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

- Gantt:



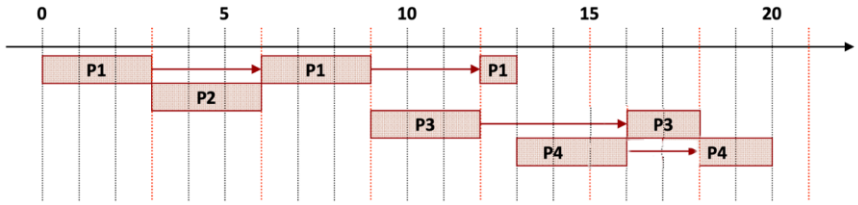

- Tempo di *turnaround* maggiore di SJF, ma migliore *tempo di risposta.*

## RR – Round Robin

- **Processi**

| Processo | Arrival time | Service (run) time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 3 |
| P3 | 5 | 5 |
| P4 | 10 | 5 |

- **Quanto di tempo pari a 3 unità**




Tempo di attesa: P1= 0+3+3, P2=1, P3=4+4, P4=3+2
Tempo di attesa medio: (6+1+8+5)/4=4

# Politiche di schedulazione

### Scheduling a code multiple

La coda dei processi pronti è suddivisa in code distinte, ciascuna delle quali ha priorità più elevata rispetto alle successive.

Ciascun processo viene posto permanentemente in una delle differenti code, in base a qualche sua caratteristica (tipo di processo, priorità, memoria richiesta, …), e potrà entrare in esecuzione solo quando le code a priorità maggiore saranno tutte vuote.

# Politiche di schedulazione

**Schema a tre code multiple**


Ogni coda ha il suo algoritmo di scheduling; ad esempio,

- ✓ Coda dei processi interattivi → Round Robin
- ✓ Coda dei processi batch → FCFS

# Politiche di schedulazione

Dato che l'applicazione di questa politica di scheduling potrebbe bloccare indefinitamente un processo a bassa priorità, per il continuo arrivo di processi a più elevata priorità, è stato ideato un meccanismo correttivo, detto **scheduling a code multiple con retroazione (feedback)**.

Tale meccanismo consente ai processi lo spostamento dinamico fra le code, a più bassa o a più alta priorità, a seconda delle condizioni di retroazione prestabilite.

## Lo scheduling in Windows

- Lo scheduling in Windows è basato su priorità con retroazione e prelazione.
- Lo scheduler usa uno schema a 32 livelli di priorità: i processi real-time hanno una priorità da 16 a 31, gli altri processi hanno una priorità da 1 a 15 (0 è un valore riservato).
- Nel seguito ci limitiamo ai soli processi non real time: quando un processo nasce gli viene assegnata la priorità 1.
- In generale, lo scheduler sceglie il processo a priorità più alta e gli assegna la CPU. Se ci sono più processi con la stessa priorità, si usa RR.

## Lo scheduling in Windows

- Se il processo che ha la CPU va in wait prima di esaurire il suo quanto di tempo, la sua priorità viene alzata (fino al massimo a 15)
- L'entità dell'incremento dipende dal tipo di evento che il processo attende: se è un dato dalla tastiera (quindi siamo in presenza di un processo interattivo) si avrà un grosso incremento della priorità.
- Se il processo attende un dato dal disco, l'incremento è minore.
- Al contrario, se il processo che ha la CPU esaurisce il proprio quanto di tempo, la sua priorità viene abbassata (ma mai sotto la soglia 1).

## Lo scheduling in Windows

- Questa strategia favorisce ovviamente i processi che interagiscono con il mouse e la tastiera, per i quali è importante un tempo di risposta molto basso.
- Per la stessa ragione, Windows distingue tra i processi in background e il processo in foreground.
- Quando un processo passa in foreground il quanto di tempo assegnatogli viene moltiplicato per 3, cosicché il processo può continuare l'esecuzione per un tempo tre volte più lungo, prima di abbandonare la CPU.

## Lo scheduling in Linux

Anche il sistema operativo Linux ha due classi di priorità per i thread: quella comprendente le priorità da 0 a 99 è denominata real-time, mentre i thread di tipo normal hanno priorità da 100 a 139.
Per i thread di tipo real-time il kernel del sistema operativo Linux utilizza uno scheduling in cui l'esecuzione di un thread termina solo se esso rilascia volontariamente il processore, o se deve essere schedulato un thread avente una priorità superiore; per i thread aventi la stessa priorità lo scheduling è di tipo round-robin.
Per i thread di tipo normal invece il kernel del sistema operativo Linux impiega un sofisticato algoritmo di scheduling denominato Completely Fair Scheduler (CFS).

# Lo scheduling in Linux

Peri thread di tipo normal invece il kernel del sistema operativo Linux impiega un sofisticato algoritmo di scheduling denominato Completely Fair Scheduler (CFS): l'idea è quella di suddividere il tempo di esecuzione reso disponibile dal/i processore/i tra tutti i thread in modo equo. A questo scopo il concetto di quanto di tempo è sostituito dal concetto di virtual runtime che rappresenta il tempo di reale esecuzione del thread dal momento della sua creazione: il gestore dei processi schedula i singoli thread in modo che abbiano per quanto possibile tutti lo stesso virtual runtime.

Il criterio di scheduling è poi molto semplice: la CPU viene data al processo con il più basso valore di P.vruntime, ossia al processo che finora ha usato la CPU per meno tempo.