SISTEMI OPERATIVI

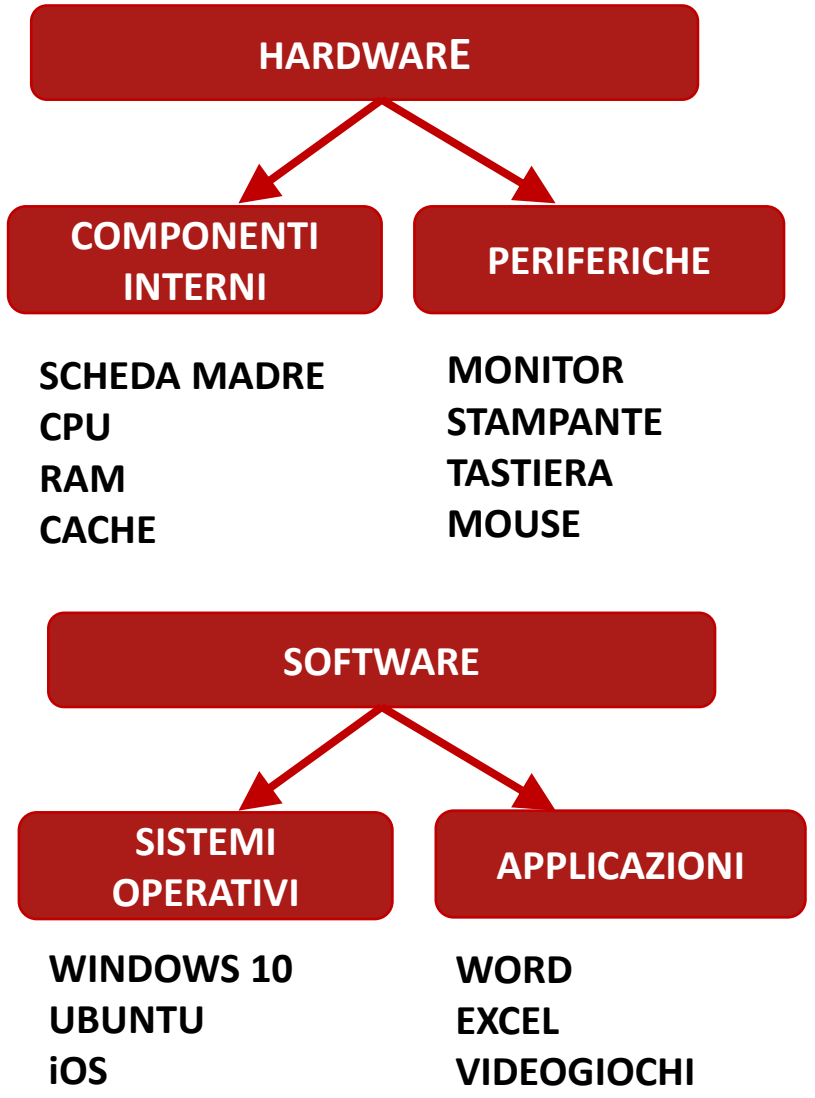
Prof. CAPEZIO Francesco

Quest'opera è soggetta alla licenza **Creative Commons**
Attribuzione – Non Commerciale

Introduzione

Un sistema operativo è un **software** il cui compiti principali sono:

- Fornire un'interfaccia all'utente per permettere di usare programmi e hardware (**shell**)
- Gestire l'esecuzione di tutti i processi software (**schedulazione**)
- Permettere alle applicazioni un'interfaccia con l'hardware (**system call**)
- Gestire l'hardware del computer (**driver**)
- Gestire l'utilizzo della memoria principale (RAM)
- Gestire il file system (insieme di file e cartelle)



Introduzione

Alcuni noti sistemi operativi sono:

Per Personal Computer

- Il primo, testuale, della Microsoft: MS-DOS
- Quelli della famiglia Microsoft Windows: 95, 98, XP, Vista, Seven, 8, 10
- Le distribuzioni Linux: Ubuntu, Redhat, Debian, Fedora → **Freeware, Open Source**
- Quelli della Apple: MacOS, OSX

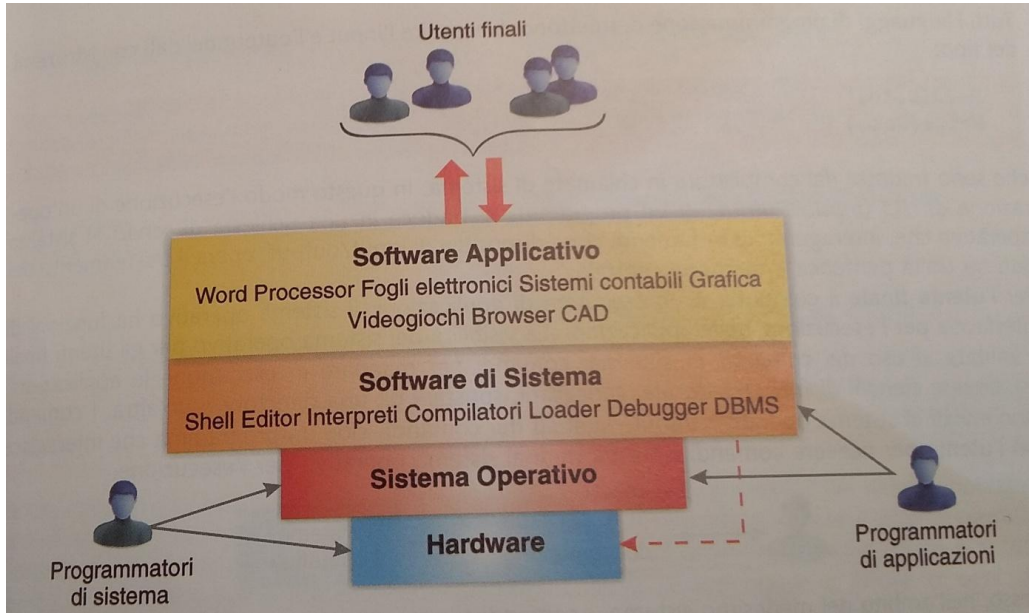
Per Dispositivi Mobili

- Microsoft: Windows Phone, Windows 8
- Android (basato su kerne Linux) → **Freeware, Open Source**
- Apple iOS

Il **Sistema Operativo (S.O.)** serve per interfacciare utente e applicazioni alla risorse del calcolatore. In particolare i suo compiti sono:
Gestire l'interfaccia con l'utente
Gestire il processore e le applicazioni
Gestire la memora centrale (principale e cache)
Gestire le periferiche
Gestire i dati e le informazioni (File System)

Software applicativo: sono i programmi che svolgono compiti utili all'utente (*Foglio Elettronici, Editor di Testo, Videogiochi etc*)
Software di sistema: sono gli applicativi utilizzati dai programmatori per funzionalità di sistema (*Compilatori, Software di sistema, Debugger, etc*)

L'interfacciamento del SO con l'hardware avviene attraverso i **driver**



SHELL e KERNEL

Un S.O.è composto principlamente da due parti:

SHELL
È l'interfaccia verso l'utente, quella che vediamo tutti i giorni quando accendiamo il computer.
Può essere **grafica** o **testuale** e permette all'utente di:

- Eseguirie i programmi
- Navigare nel file system
- Operare su file e cartelle
- Cambiare impostazioni e preferenze dell'interfaccia stessa

KERNEL
E' il nocciolo del S.O. ed è composto ed è composto da software in grado di assolvere ai compiti esposti in precedenza
Tipicamente è scritto in linguaggio C e, come ordine di grandezza, è composto da 5000000 di righe di codice.

MONOKERNEL E MICROKERNE

Il Kernel di un sistema operativo può essere realizzato con due filosofie diverse:

KERNEL MONOLITICI

Il kernel esegue tutte le funzionalità (gestione processi, della RAM, dell'hardware, del file system)

VANTAGGI:

- Tipicamente è più veloce e più facile da progettare

SVANTAGGI

- Sono complicate le modifiche (anche la semplice aggiunta di un driver richiede la ricompilazione del kerne)
- Un errore in una qualche funzionalità compromette l'operatività del S.O.

MICROKERNEL

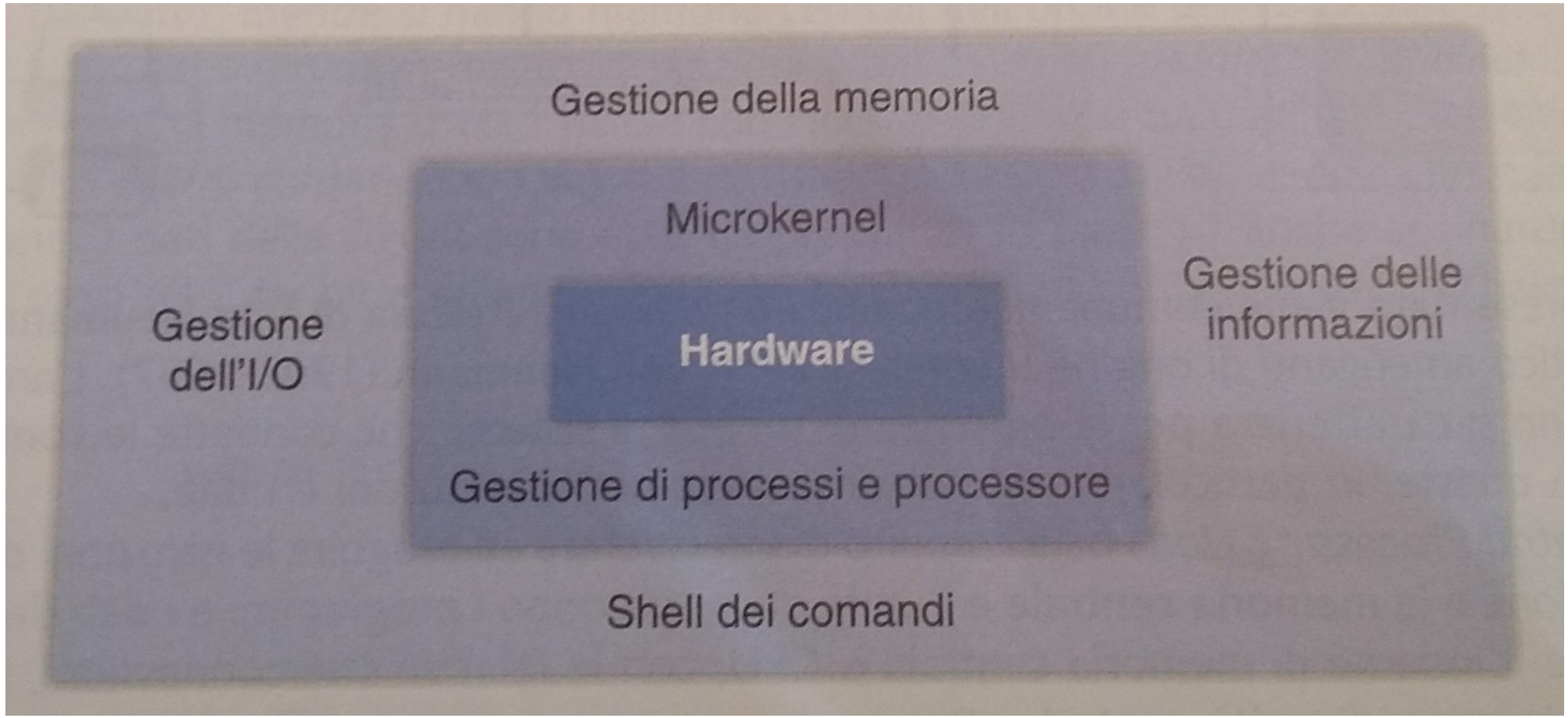
il Kernel esegue solo le funzionalità di base (gestione dei processi e della RAM) delegando a **Servizi** esterni i compiti considerati secondari (driver, file system etc.)

VANTAGGI:

- Più semplici da ampliare e modificare
- Un errore in un servizio non compromette la funzionalità del S.O.

SVANTAGGI

- Risulta più lento e meno ottimizzato nelle funzionalità


Gestione della memoria
Microkernel
Hardware
Gestione di processi e processore
Gestione dell'I/O
Gestione delle informazioni
Shell dei comandi

DAL PROGRAMMA AL PROCESSO

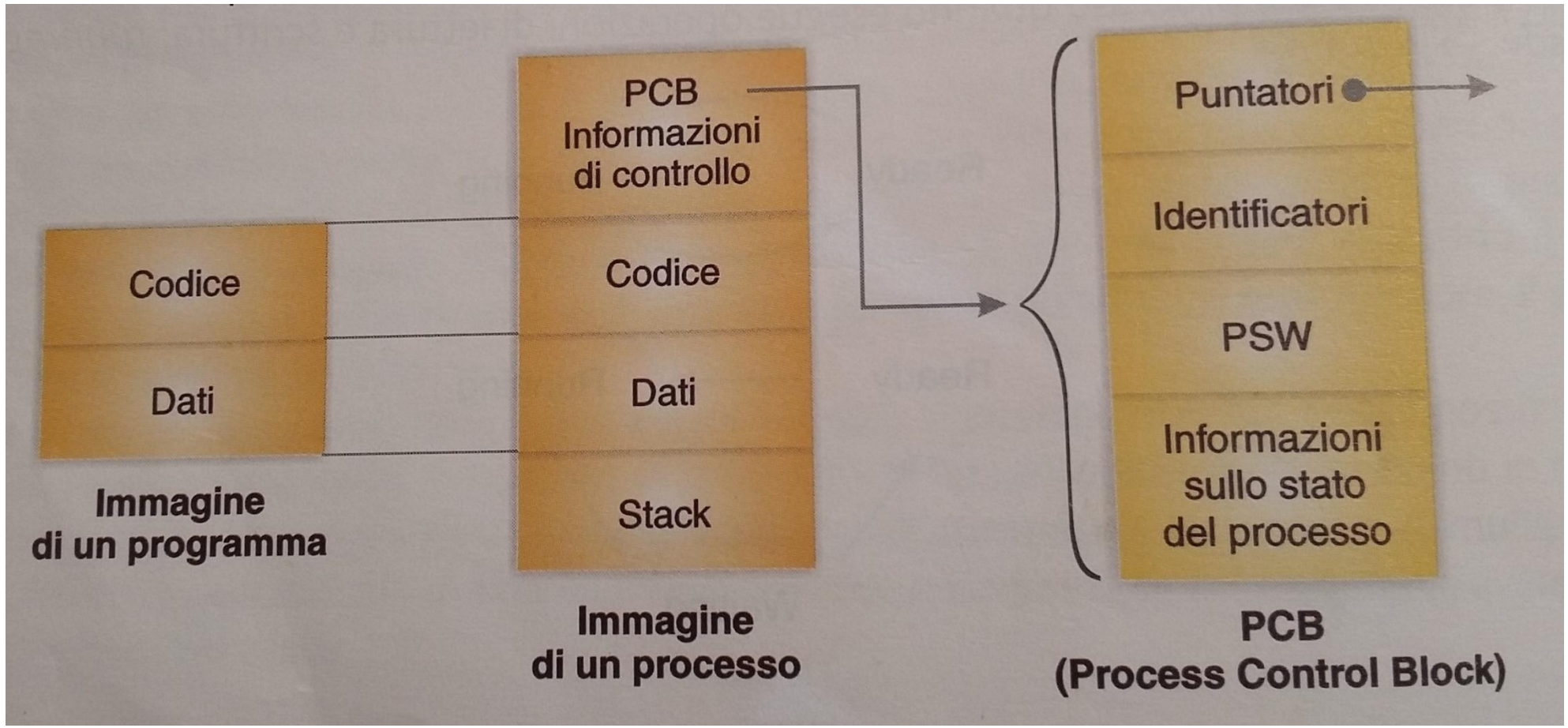
Un **programma** è una serie di istruzioni che un calcolatore (in particolare una CPU) deve svolgere per implementare un algoritmo e risolvere un problema. Le istruzioni possono essere scritte un C o in un altro linguaggio ad alto livello, ma anche scritte in linguaggio macchna (se ad esempio il programma è già un eseguibile compilato)

Un **processo** è una singola istanza di un programma in esecuzione su un calcolatore

(ad esempio Microsoft Word è un programma; ogni finestra di word con un documento in elaborazione è un diverso processo –instanza del programma-)

Un processo, oltre alle istruzioni, deve contenere dei dati che servono alla CPU per la sua esecuzione: questo è il **Process Control Block** che contiente:

- Identificativo del processo (serve al S.O. per riferirsi al processo)
- Stato del processo e puntatori agli altri processi in coda
- Puntatore al processo padre (che ha generato il processo) e ai processi figli (generati dal processo)
- Area di memoria dedicata al processo (stack)
- Valore dei registri (in particolare del Program Counter)


Codice
Dati
Immagine
di un programma
PCB
Informazioni
di controllo
Codice
Dati
Stack
Immagine
di un processo
Puntatori
Identificatori
PSW
Informazioni
sullo stato
del processo
PCB
(Process Control Block)

ESECUZIONE DELLE APPLICAZIONI

- ANNI ‘40 primi anni ‘50: non c’è sistema operativo. I programmi vengono scritti in linguaggio macchina e caricati uno alla volta sulla ram
- Successivamente nasci il linguaggio Assembly: il programmatore scrivi in assembly, un assemblatore (assembler) traduce il programma in linguaggio macchina (su schede perforate), per poi trasferirlo in RAM per l’esecuzione.

Si nota coma la maggior parte del tempo viene sprecata per operazioni manuali, come il setup della macchina e la traduzione delle schede

ESECUZIONE DELLE APPLICAZIONI
I SISTEMI BATCH

Dalla seconda metà degli anni ‘50 nascono i primi rudimentali sistemi operativi.

il programma viene scritto con un linguaggio simbolico (FORTRAN). Successivamente viene compilato e diversi JOB (processi) vengono caricati in RAM per essere eseguiti uno alla volta.

Un linguaggio apposito (JCL – Job Control Language) viene utilizzato per gestire i diversi processi nella RAM

SISTEMA OPERATIVI INTERPRETE JCL
JOB 1
JOB 2
JOB 3
JOB 4

ESECUZIONE DELLE APPLICAZIONI
I SISTEMI BATCH

Prevedere processi diversi in esecuzione introduce alcune problematiche da affrontare:

- Proteggere la memoria RAM, per evitare che i processi vada a scrivere in aree di memoria a loro non concesse
- Prevedere un modo per velocizzare l'accesso alle informazioni provenienti dall'HW (interruzioni) → DMA (Direct Memory Access), ovvero l'accesso, per alcune periferiche come i dischi, alla memoria RAM in modo diretto, senza interruzioni dei processi
- Prevedere delle code per l'accesso alle periferiche (SPOOL), in modo da evitare il sovrapporti di dati provenienti da processi diversi (es. stampante

MONOTASKING E MULTITASKING

PROCESSI CPU-BOUNDED: processi nei quali la parte di elaborazione da parte della CPU è preponderante rispetto all'interazione con l'utente – es. Antivirus

PROCESSI I/O-BOUNDED: processi nei quali la parte di interazione con l'utente è preponderante rispetto alla parte di elaborazione da parte della CPU (processi interattivi) – es. Word

MONOTASKING

I primi calcolatori avevano una architettura **batch** nella quale i processi venivano caricati in memoria ed eseguiti uno dopo l'altro. Questo metodo andava bene in quanto i processi erano per lo più CPU-Bounded.

Con l'introduzione di processi interattivi la CPU restava inutilizzata in attesa dell'interazione con l'utente.

MULTITASKING

Il S.O. può interrompere l'esecuzione di un processo per assegnare la CPU ad un altro (ad esempio se il processo in esecuzione è in attesa di una risorsa o dell'interazione con l'utente.

Questo meccanismo, detto di **schedulazione** dei processi, permette l'esecuzione in contemporanea di più processi (anche se la CPU esegue una istruzione alla volta e, dunque, un processo alla volta.

SCHEDULAZIONE e STATI DI UN PROCESSO

La schedulazione prevede l'assegnazione (da parte del S.O.) della CPU ad un processo in base ad una precisa politica. I processi, dunque possono trovarsi in stati diversi durante la loro permanenza in memoria.

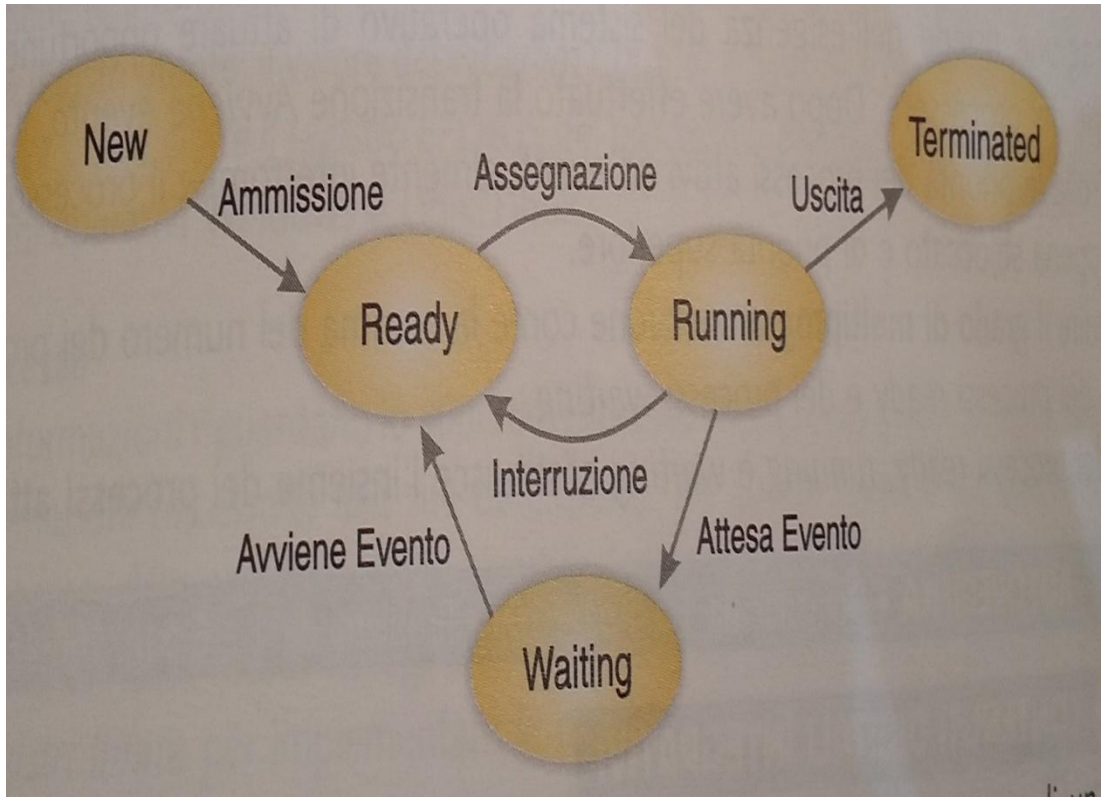
NEW: stato nel quale un processo è appena stato caricato in RAM ma necessita ancora dell'assegnazione di alcune risorse di cui ha bisogno per la sua esecuzione (dispositivi HW, spazio in memoria etc.)

READY: il processo ha tutte le risorse di cui ha bisogno e aspetta di essere eseguito (manca l'assegnazione della CPU al processo)

RUNNING: il processo ha ottenuto l'assegnazione della CPU ed è in esecuzione

WAITING: il processo è in attesa di un evento (hardware o interazione con l'utente)

TERMINATED: il processo ha concluso e viene rimosso dalla RAM



SCHEDULAZIONE e STATI DI UN PROCESSO

Schedulare i processi vuol dire gestire la loro alternanza nello stato di **running** e, in particolare, gestire la **coda dei processi in ready.**

Alternare i processi in running ha diversi obiettivi:

- Massimizzare il **throughput** (num. di processi finiti nell'unità di tempo)
- Minimizzare i tempi di attesa (da quando entrano nel sistema a quando finiscono)
- Evitare la **starvation** (condizione nella quale un processo non accede mai allo stato di ready)
- Favorire i processi interattivi (che interagiscono con l'utente)

SCHEDULAZIONE

FCFS – First Came First Served

La coda dei processi in ready vengono gestiti con un politi FIFO (First In First Out) ovvero i processi vengono messi in ordine dal primo che arriva e con quest'ordine accedono allo stato di running.

PRO:

- Evita la starvation: tutti i processi vengono prima o poi eseguiti

CONTRO:

- Aumenta i tempi di attesa se processi lunghi sono in cima alla fila

SCHEDULAZIONE

SJF – Shortest Job First

I processi nella coda dei ready vengono ordinati secondo il loro tempo di esecuzione (i più brevi in cima alla coda)

PRO:

- Aumenta il througput
- Minimizza i tempi di attesa

CONTRO:

- Può comportare la starvation di processi lunghi se continuano ad arrivare processi più brevi
- E’ particolarmente difficile stabilire il tempo di esecuzione di un processo

SCHEDULAZIONE

SCHEDULAZIONE A PRIORITA'

I processi nella coda dei ready vengono ordinati secondo una priorità (importanza) dando una priorità maggiore ai processi interattivi

PRO:

- Favorisce i processi interattivi

CONTRO:

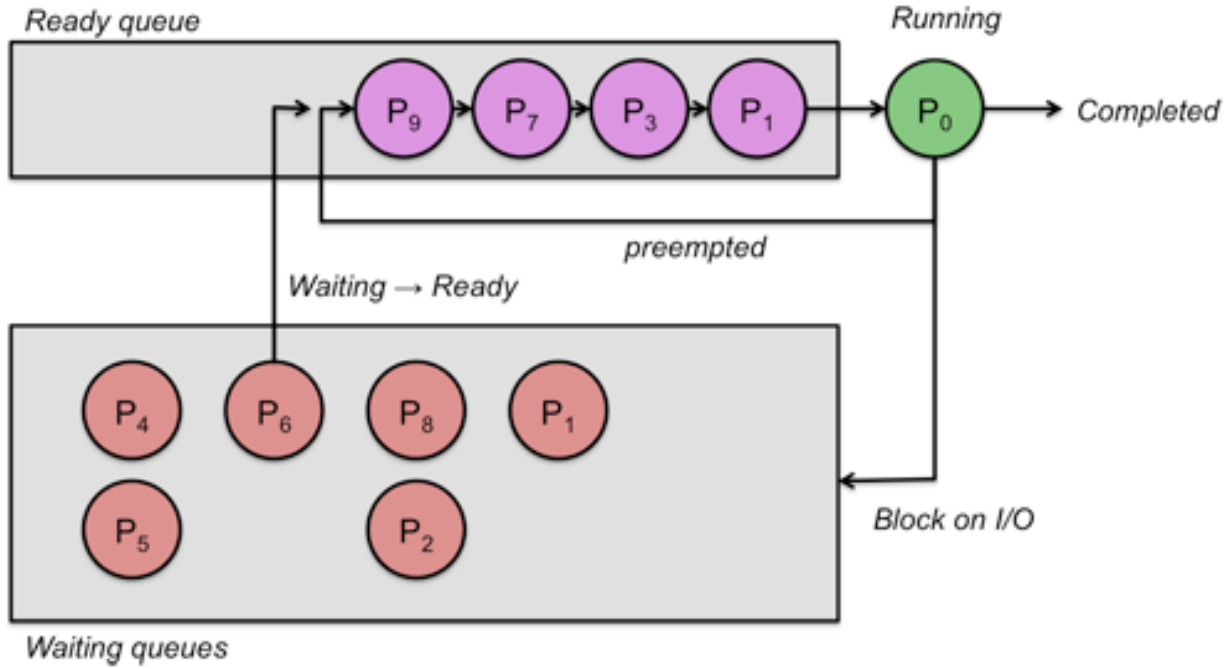
- E' difficile fornire la giusta priorità ai processi (ad esempio bisogna dare maggiore priorità a processi in attesa da tanto, per evitare la starvation)
- E complicato gestire la coda dei processi in ready

SCHEDULAZIONE

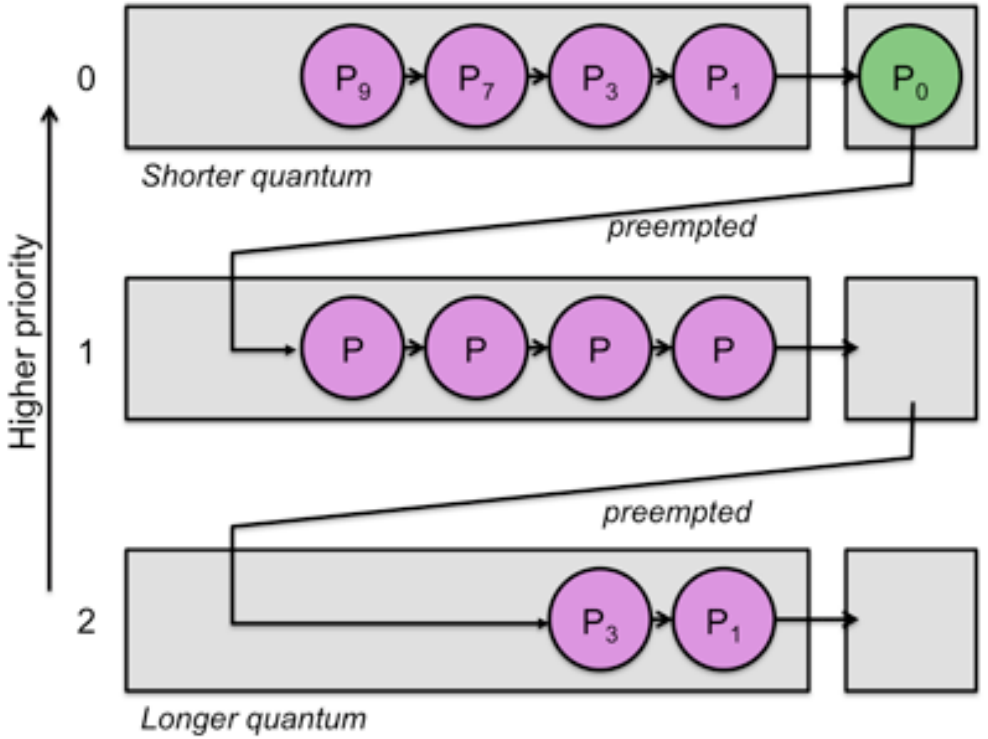
SCHEDULAZIONE ROUND ROBIN

Questa schedulazione gestisce la coda dei ready come una coda FIFO (facile da gestire) ma prevede la preemption, concedendo ad ogni Job in Running sono un determinato **time-slice** (quanto di tempo) al termina del quale viene spostato in coda

Questa schedulazione permette una buona alternanza dei processi nella CPU, evita la starvation e permette ai processi brevi di non attendere l'esecuzione di processi troppo lunghi



FEEDBACK SCHEDULING

Questa è la base della schedulazione effettivamente usata sui moderni calcolatori. Si tratta di una schedulazione **round robin** ma con code di ready a priorità diverse (ovvero i processi delle code a priorità minore verranno scelti meno di frequente pre l'esecuzione)

- Tutti i processi nuovi arrivano nella coda a priorità maggiore
- Se un processo ha bisogno di una risorsa (waiting) resta nella coda da cui è arrivato.
- Se un processo utilizza tutto il quanto di tempo scende ad una coda a priorità minore.
- I processi nelle code a priorità minore hanno a disposizione un quanto di tempo più lungo