ORGANIZZAZIONE DI UN SISTEMA OPERATIVO

Come sono organizzate le diverse **risorse** componenti di un S.O. e le modalità di interconnessione tra di esse

- **Sistemi monolitici**
- **Sistemi a livelli**
- **Macchine virtuali**
- **Modello cliente-servitore**

File System e memorizzazione e uso delle risorse

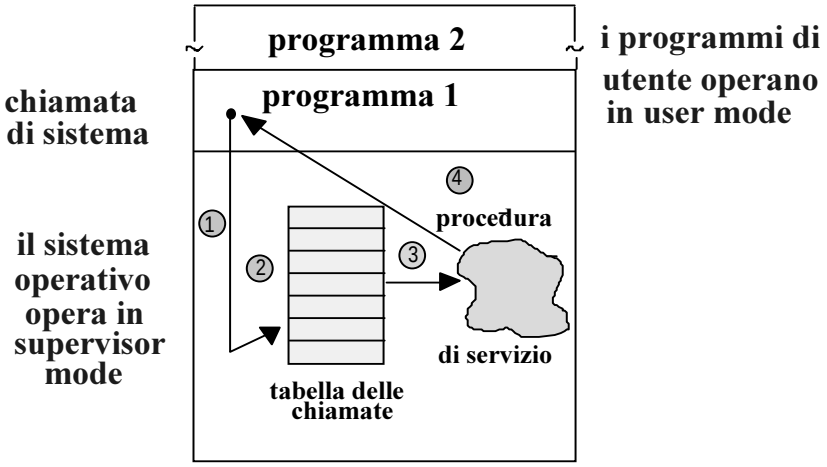
Processi e uso delle risorse

SISTEMI MONOLITICI

Il sistema operativo è scritto come un **insieme di procedur**e, tutte allo **stesso livello** di gerarchia.

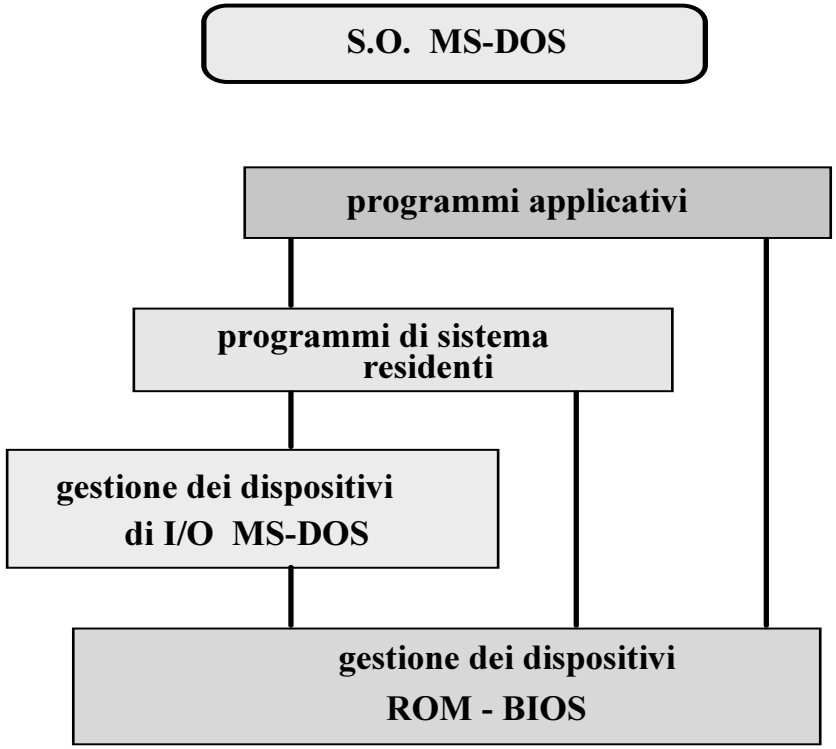
Le funzioni possono chiamarsi reciprocamente s sono invocabili attraverso **ben definite** interfacce
tipicamente **interrupt software** o **trap**

Requisiti di
protezione e **riuso**



Separazione tra **S.O.** e **Processi Utente**

ESEMPIO di SISTEMA (monoprogrammato)



Windows 3.1

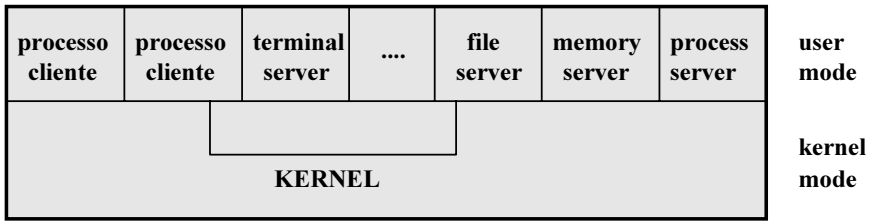
Windows 95 (multiprogrammato?)
Windows NT (multiprogrammato)

SISTEMA A LIVELLI

- Le funzioni del S.O. sono organizzate per **livelli gerarchici**
- Ogni livello definisce un **tipo** di servizio e le **modalità** per essere utilizzato dai **livelli superiori**
- **Macchine virtuali -** Ogni livello definisce una nuova macchina, aggiungendo nuove funzionalità alla precedente
- Livelli ccome **aiuto** nel progetto per superare difficoltà pratiche nella realizzazione (gerarchia)

MODELLO CLIENTE/SERVITORE

- Sistemi operativi **moderni:** la maggior parte delle funzioni del sistema operativo sono realizzate come **processi di utente**



- Per richiedere un servizio un **processo cliente** (utente) invia una richiesta ad un **processo servitore** (es.: *lettura di un file*)

- **Vantaggi:**

☺ le funzioni sono suddivise in parti di dimensioni ridotte e ben specificate

☺ protezione contro errori: le funzioni non hanno accesso al kernel (operano in *user mode*)

CLASSIFICAZIONE DEI SISTEMI OPERATIVI

Organizzazione interna
Visibilità utente

* monoprogrammato
* batch
* multiprogrammato

* interattivo
* a divisione di tempo
* general purpose
* special purpose

SISTEMI MONOPROGRAMMATI

Si gestiscono i programmi in modo sequenziale nel tempo
Tutte le risorse hardware e software del sistema sono dedicate ad un solo programma per volta.

utilizzo della CPU = T_p / T_t

T_p = tempo dedicato dalla CPU alla esecuzione del programma

T_t = tempo totale di permanenza del programma nel sistema

throughput = numero di programmi eseguiti per unità di tempo

Per avere alta efficienza, si vorrebbe avere un utilizzo vicino ad 1 e throughput elevato

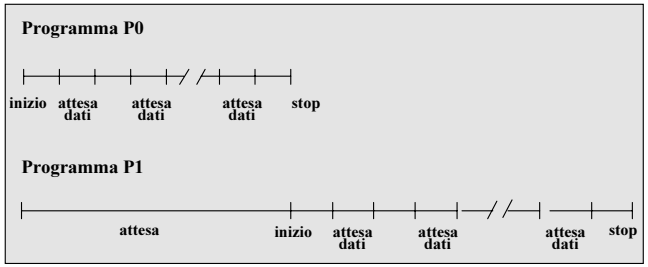
☹ **BASSA UTILIZZAZIONE DELLE RISORSE**

SISTEMI MULTIPROGRAMMATI

Si **gestiscono** "contemporaneamente" più programmi indipendenti presenti nella memoria principale

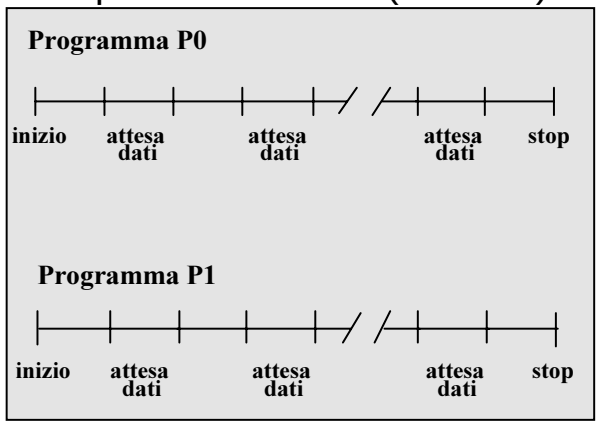
- Migliore **utilizzazione delle risorse** (riduzione dei tempi morti)
- **Maggiore complessità** del S.O.:
  - algoritmi per la gestione delle risorse (CPU, memoria, I/O)
  - protezione degli ambienti dei diversi programmi

Processi senza multiprogrammazione

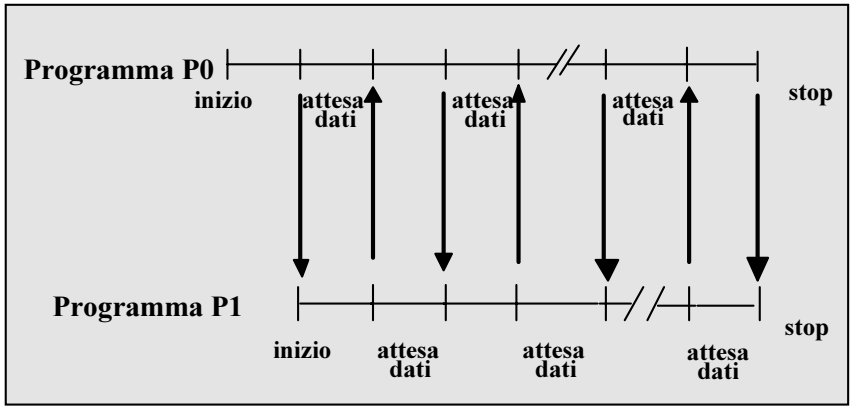


- Durata dell'esecuzione :
  2 minuti (1 minuti per programma)
- Occupazione della CPU:
  1 minuti (50%)

- **P_0** e **P_1** eseguono per 1 secondo, quindi attendono per 1 secondo (30 volte)



Esecuzione dei programmi **con** multiprogrammazione



- Durata dell'esecuzione:
  1 minuti
- Occupazione CPU:
  1 minuti (100%) (caso teorico)

☹ efficienza (in genere) più bassa e throughput inferiore a 1 al minuto
(2 uscite ogni 2 minuti)

SISTEMI a DIVISIONE di TEMPO

Astrazione di più **macchine virtuali**

- Ogni utente ha un proprio programma in memoria ed a ciascuno è dedicata una macchina virtuale
- Si abbreviano i tempi di attesa dei programmi corti
- Tempo impiegato dal S.O. per trasferire il controllo da un programma ad un altro (**overhead**)

SISTEMI BATCH

- I programmi sono inseriti **a lotti** nella memoria di massa e successivamente elaborati in multiprogrammazione
- **Obiettivo:** migliore utilizzazione possibile delle risorse (elevato ***troughput***)
- Scelta dell'insieme di programmi (***job mix***) in memoria principale che ottimizzano l'utilizzo delle risorse

SISTEMI INTERATTIVI

- Più utenti che utilizzano contemporaneamente il sistema di calcolo (a divisione di tempo)
- **Obiettivo:**
  migliorare i tempi di risposta per l'utente (in contrasto con l'ottimizzazione delle risorse)
- Scelta del processo che ha più urgenza di servizio

SISTEMI GENERAL-PURPOSE

- Centri di Calcolo di grandi dimensioni
- Utenze di tipo differenziato
- Modalità di uso batch ed interattivo

SISTEMI SPECIAL-PURPOSE

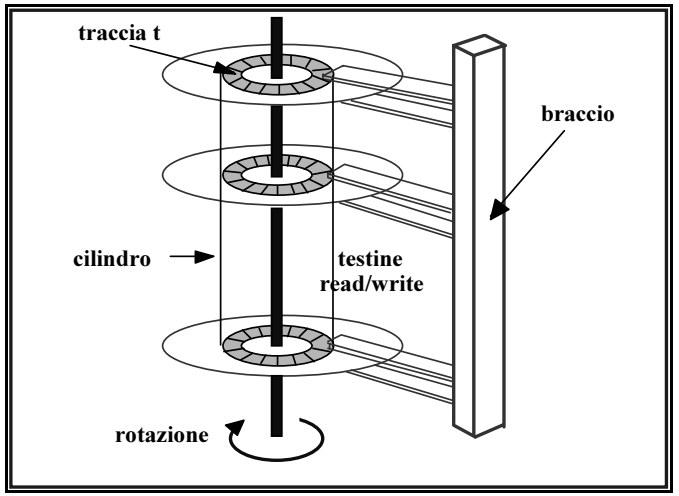
- Dedicati ad **applicazioni specifiche**, spesso con hardware speciale (dispositivi di I/O)

- **Sistemi in tempo reale:** il sistema di calcolo deve "rispondere" entro un limite di tempo (tempo di reazione)

  - tempo reale **stretto**: il tempo di risposta è molto stringente come vincolo e non può non essere rispettato (p.e. processi industriali)

  - tempo reale **debole**: il tempo di risposta deve essere "ragionevole" (p.e. sistema interattivo per la prenotazione voli)

IMPLEMENTAZIONE DEL FILE SYSTEM



Struttura del disco

Componenti:

- disk driver — parte meccanica (motore del dispositivo, testine di lettura e scrittura)
- disk controller — controlla interazione con la CPU. Trasforma le istruzioni di I/O in comandi al disk driver

indirizzo fisico:

- settore (blocco fisico) unità minima di informazione che può essere letta o scritta da/su disco

  settore 32 – 4096 byte

  traccia 4 – 32 settori

  superficie 20 – 1550 tracce

- per individuare un settore:

  seek time per raggiungere la traccia

  latency time per portare il settore sotto la testina

- il S.O. vede il disco come un vettore di blocchi fisici
- gli indirizzi dei blocchi si incrementano all'inizio di una traccia, lungo tutte le tracce di un cilindro e per tutti i cilindri
- l'indirizzo di un blocco viene espresso tramite ***numero di cilindro, numero di traccia nel cilindro (o numero di superficie), e numero di settore nella traccia***

GESTIONE dello SPAZIO LIBERO

- **BIT MAP**

Ogni blocco è rappresentato da un bit (**0** se libero, **1** se occupato)

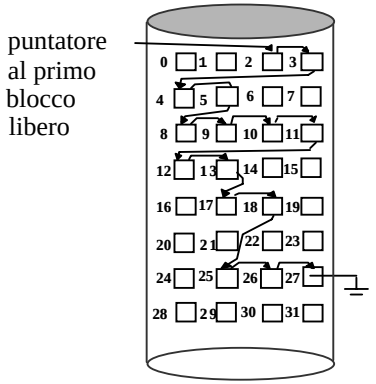
1100000111001110000011.....

☺ è semplice trovare **n blocchi consecutivi**

😐 la mappa deve essere contenuta in **memoria principale** (efficienza)

LISTA DI BLOCCHI LIBERI

☹ **non è efficiente** occorre leggere un blocco per avere l'indirizzo del successivo



modifica: il primo blocco contiene gli indirizzi di **n** blocchi liberi. L'ultimo di questi contiene l'indirizzo di altri **n** settori liberi, etc.

METODI di ALLOCAZIONE dei BLOCCHI IMPEGNATI

- Un file è memorizzato in **blocchi di byte** di dimensione finita (blocchi logici)
- Tutte le operazioni di I/O avvengono in termini di blocchi logici

Esempio: **1 k byte** **blocco logico**
512 k byte **blocco fisico**

Ogni **lettura/scrittura** di **un blocco** comporta la **lettura/scrittura** di **due settori**

Sempre, il SO bufferizza i blocchi letti/scritti per ottmizzare i tempo di interazione con il disco e per fornire servizi più veloci alle richieste dei processi utente

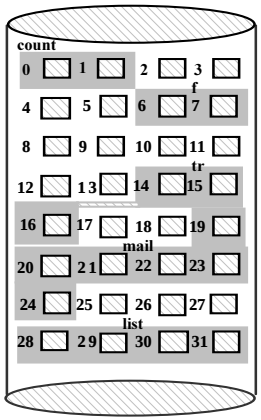
Quali processi utente sono favoriti?
¿Come si possono ottenere buoni servizi?

Allocazione **contigua,**
a lista linkata,
ad indice

Allocazione contigua

direttorio

File	Inizio	Fine
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2



- riduce il tempo di seek
- consente sia accessi sequenziali sia diretti

Accesso sequenziale: il file system memorizza l'indirizzo dell'ultimo blocco letto/scritto. L'accesso avviene per il blocco successivo

Accesso contiguo: se il file inizia al blocco **b**, l'accesso avviene per il blocco **b + i**

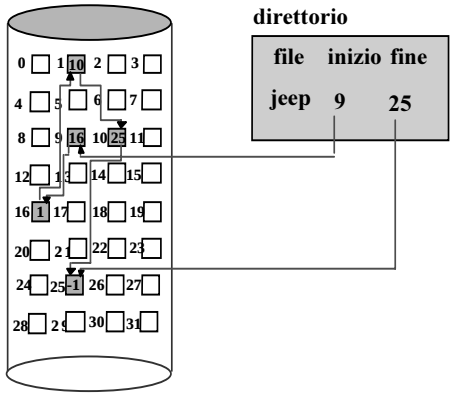
Problemi:

😐 **Determinazione** dello spazio contiguo
tecniche di ***first-fit, best-fit, worst-fit***

☹ **Frammentazione esterna**
esigenze di compattamento

☹ **Crescita dinamica** dei file

Allocazione a LISTA



- La scrittura in un file comporta **la ricerca** di un blocco nella **lista dei blocchi liberi** e l'inserimento alla fine del file
- **Non** c'è frammentazione esterna e **non necessario** conoscere la lunghezza del file
- Efficienza **per accessi sequenziali**

Problemi:

😐 **doppio link per evitare problemi** (perdita di un puntatore)

☹ Spazio richiesto per i puntatori

☹ Accesso diretto **non realizzabile** in pratica

Frammentazione

mancanza o non disponibilità di risorse che sono invece potenzialmente presenti

Frammentazione INTERNA

Frammentazione ESTERNA

Allocazione a lista linkata

Allocazione a lista risolve i problemi della allocazione contigua

☺ **frammentazione esterna**

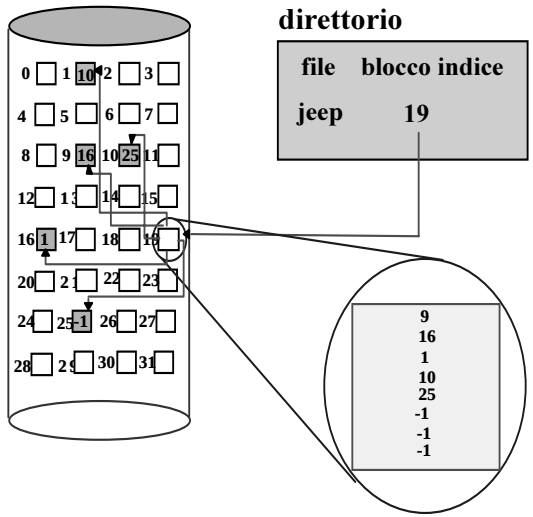
☺ **crescita dinamica di un file**

ma

☹ Non supporta un **accesso diretto**: i blocchi sono sparsi su tutto il disco

☹ I puntatori ai vari blocchi sono **sparsi** su tutto il disco

Allocazione a indice



Allocazione a indice: tutti i puntatori ai blocchi di file sono contenuti in un **blocco indice**

- Per leggere il blocco **i-esimo** si usa il puntatore contenuto nella **posizione i-esima** del blocco indice
- Alla **creazione** di un file, il suo blocco indice contiene tutti **nil**. Quando **si scrive** il blocco i-esimo, l'indirizzo viene scritto nella posizione i-esima del blocco indice
- Possibilità di **accesso diretto** senza **frammentazione esterna**

☹ Spazio per il blocco indice

Allocazione a indice

- Se il file è di dimensioni elevate, il blocco indice può occupare **due o più blocchi**
- L'ultimo elemento di un blocco indice è **nil** (per file piccoli) o **l'indirizzo** di un altro blocco indice (per i file grandi)

- Indice a più livelli. Il primo blocco indice contiene indirizzi di altri blocchi indice che contengono gli indirizzi dei blocchi **(due livelli**)

Esempio:

blocco indice I° livello
256 puntatori a blocchi indice

blocco indice II° livello
256 puntatori a blocchi
X 256 puntatori a blocchi

Totale: 65536 blocchi

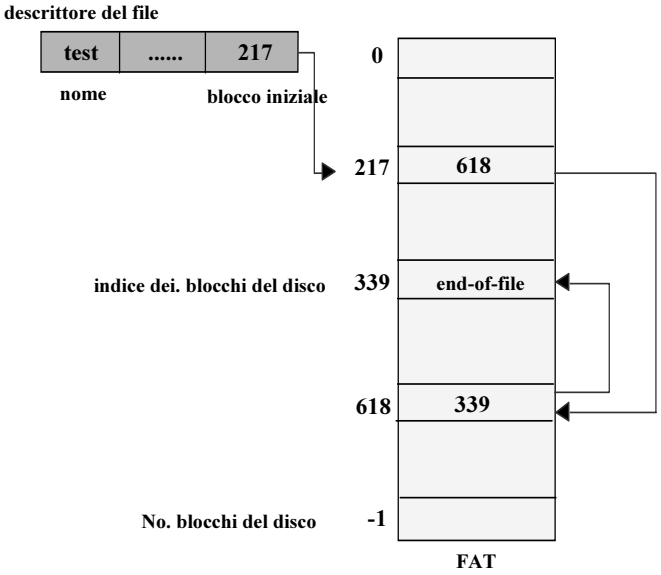
1 blocco 1024 byte 67.108.864 byte

Metodi di allocazione

Proprietà indicative

- **grado di utilizzazione della memoria**
- **tempo necessario per accedere ad un blocco sul disco**
- **metodi di accesso consentiti**

Tabella di allocazione dei file di dicco (FAT di MS-DOS)

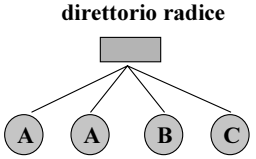


Indice del disco intero

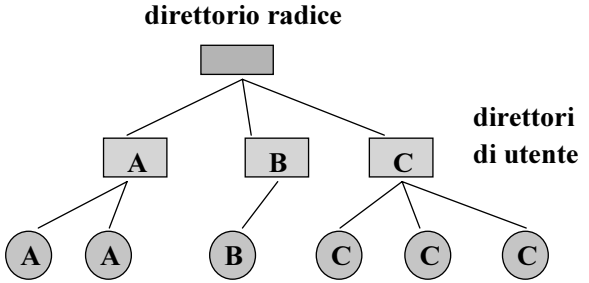
ORGANIZZAZIONE dei DIRETTORI

I direttori consentono l'accesso ai blocchi (o al primo blocco del file, o a un descrittore del file) e sono file a loro volta

Diverse organizzazioni storicamente ...
La **prima organizzazione** mantiene un **file system unico** per tutti gli utenti

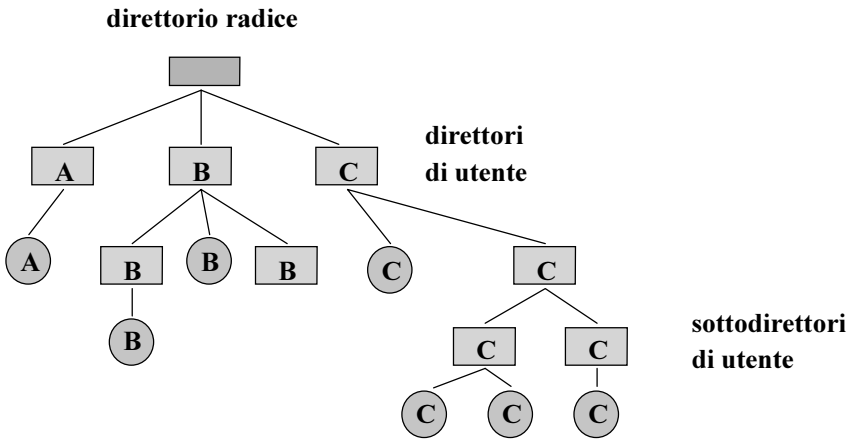


La **seconda organizzazione** riconosce e mantiene un **file system per ogni utente** (a ciascuno viene dato uno spazio unico)



VERSO UNIX

Una **organizzazione** gerarchica consente agli utenti di decidere e mantenere la **propria strutturazione** con i **propri direttori**



In UNIX il file system è unificato e non ci sono limiti ai direttori

Ci sono sistemi operativi in cui il **file system** può anche essere non gerarchico ma a **grafo anche ciclico**

Si ricordi la differenza tra link hardware e software

ASSEGNAZIONE del PROCESSORE

(CPU SCHEDULING)

- **Scheduler:** quella parte del S.O.che decide a quale dei **processi pronti** presenti nel sistema assegnare il controllo della CPU
- I processi possono essere o in memoria principale o in memoria secondaria
- **Algoritmo di scheduling:** realizza un particolare **criterio di scelta** tra i processi pronti

Possibili criteri di scelta:

- utilizzo della CPU
- produttività
  throughput come uscite per unità di tempo
- tempo di "turnaround" (sistemi batch)
  permanenza nel sistema
- tempo di risposta (sistemi interattivi)
  garantendo risposte pronte
- non privilegio (fairness)
  giustizia nel servizio

Possibilità degli algoritmi di scheduling

scheduling preemptive

un processo in esecuzione perde il controllo della CPU anche se logicamente può proseguire

scheduling nonpreemptive

un processo in esecuzione prosegue fino al rilascio spontaneo della CPU

(terminazione, richiesta di I/O, sospensione per attesa di evento)

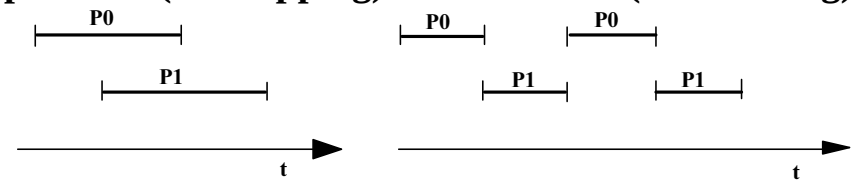
Relazioni tra PROCESSI

Processi **concorrenti**
Processi **interagenti**
Processi **indipendenti**

Processi CONCORRENTI

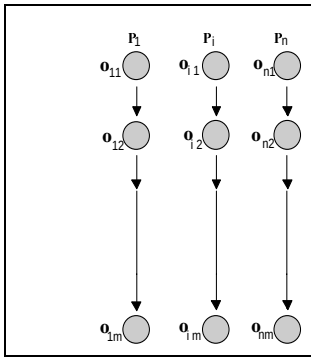
Due processi sono concorrenti se la loro esecuzione si sovrappone nel tempo

più CPU (overlapping) **una sola CPU (interleaving)**



Due processi sono **concorrenti** se la prima operazione di uno inizia prima dell'ultima dell'altro

Processi INDIPENDENTI



Proprietà

stato non condiviso

esecuzione deterministica e riproducibile

Processi INTERAGENTI

- modificano variabili comuni (**stato condiviso**)
- esecuzione **non deterministica**: il risultato dipende dalla sequenza di esecuzione relativa e non è predicibile
- esecuzione **non riproducibile**: il risultato non è sempre lo stesso per gli stessi dati di ingresso

Processi CONCORRENTI e INTERAGENTI

Competizione per l'uso di risorse comuni che non possono essere usate contemporaneamente

Cooperazione nell'eseguire un'attività comune mediante scambio di informazioni (comunicazione)

Interazione tra processi

competizione sull'uso delle risorse

comunicazione reciproca se interagenti (attraverso file, mailbox, etc.)

Livelli di scheduling

Long term scheduling (job scheduling)
determina quali programmi devono essere caricati dalla memoria di massa alla memoria principale

* controlla il **grado di multiprogrammazione** (numero di processi in memoria principale)
* possibile criterio di scelta: insieme (mix) equilibrato di processi "**CPU-bound**" e "**I/O-bound**"
* interviene ad intervalli di tempo relativamente lunghi; algoritmi più complessi

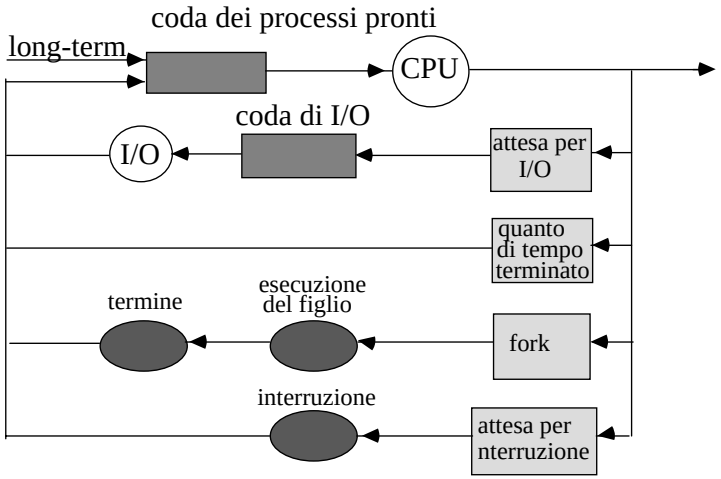
Short term scheduling seleziona tra i processi pronti in memoria principale quello cui assegnare la CPU

* elevata frequenza di intervento
* efficienza
* **politica**: FIFO, priorità, etc...

Dispatcher realizza le operazioni necessarie per assegnare il controllo della CPU al processo selezionato dal **short-term scheduler**:

* cambio di contesto
* ritorno allo stato di utente
* mette in esecuzione il processo selezionato

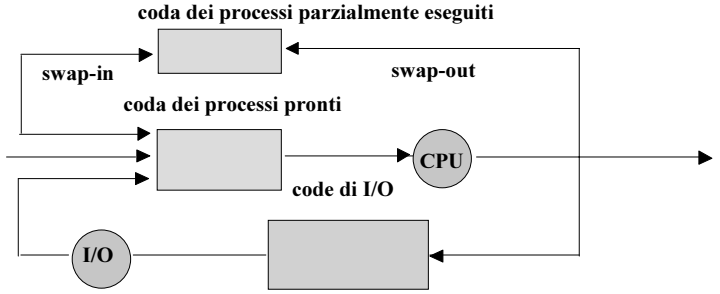
Modello a stati



Medium term scheduling rimuove temporaneamente dalla memoria principale un processo in esecuzione

* riduzione del grado di multiprogrammazione
* modifica dell'insieme di programmi in memoria principale

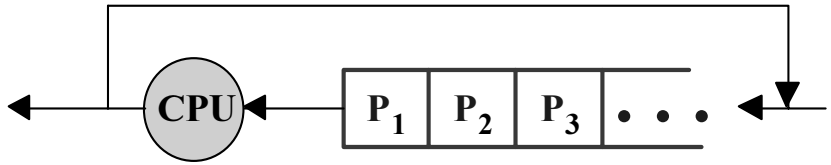
Swapping (***"swap-in, swap-out"***)



Algoritmi di short-term scheduling

Scheduling Round-Robin (RR)

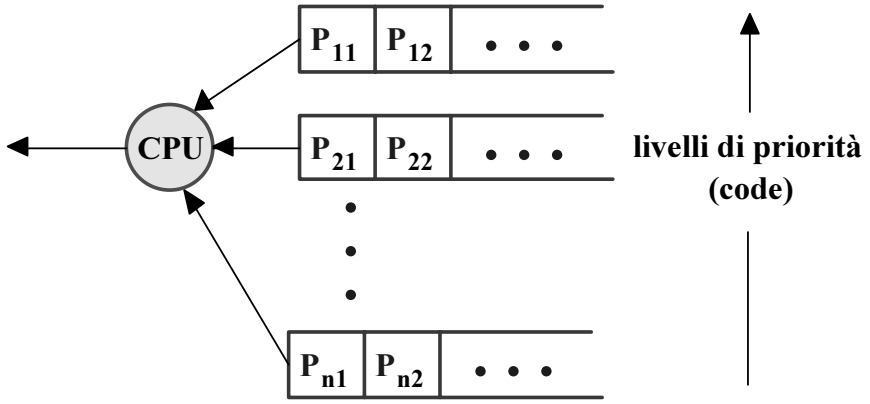
- La coda dei processi pronti è gestita FIFO, ma ad ogni processo è assegnata la CPU per un **quanto di tempo** (**QT**) prefissato



- Se il processo P_1 non termina durante QT, la CPU viene assegnata a P_2 e P_1 è inserito in fondo alla coda

- **Scelta di QT:** deve essere **sufficientemente** grande rispetto al tempo di cambio del contesto (ordine delle decine di msec)

Scheduling a Priorità



In ogni istante, è in esecuzione il processo pronto a **priorità massima**

- **Priorità statica** fissata alla creazione dei processi in base alle loro caratteristiche
  Es.: processi di I/O con priorità più elevata
- **Processi foreground** (sistema, interattivi) **alta**
  Processi background (batch) **bassa priorità**

- Problema della **starvation**: ad un processo a bassa priorità può non venire mai assegnata la CPU perchè vi sono sempre processi a **priorità più elevata**

Scheduling a Priorità dinamica

può essere modificata durante l'esecuzione dei processi:

- per **penalizzare** processi che impegnano troppo la CPU
- per evitare fenomeni di **starvation**
- per favorire processi **I/O bound**

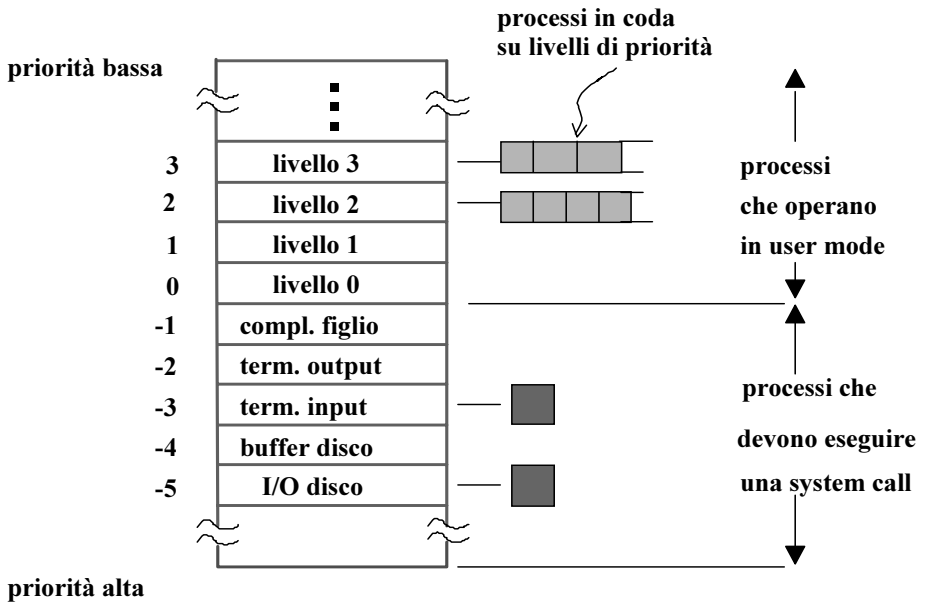
Esempio: UNIX

- **Più livelli di priorità** (crescente dall'alto al basso)
- **Round robin** tra i processi allo stesso livello
  (quanto di tempo = 100 msec)
- **Aggiornamento dinamico** della priorità
  ad esempio **ad ogni secondo**

Nuova Priorità = Base + Uso CPU

- **base:** normalmente = **0**, assume un valore > **0** tramite la ***system call*** **NICE**
- **uso CPU:** contatore per ogni processo incrementato di **1** ad ogni clock (5 o 6 interruzioni per QT). Diviso per **2**, per ridurre la penalizzazione per uso elevato di CPU.

UNIX - Struttura a code di priorità



Esempio: UNIX

- Quando un processo esegue una ***system call***, può **venire bloccato** (es.: richiesta di uso del disco)
- Al verificarsi dell'evento atteso, il processo diventa **pronto** e viene inserito nel relativo livello di priorità (priorità negativa significa priorità alta)
- **si favoriscono** i processi interattivi (attesa per un terminale) e che eseguono I/O

First Come First Served (FCFS)

CPU assegnata ai processi in accordo **all'ordine temporale di arrivo** nel sistema

- coda dei processi pronti servita **FIFO**
- il processo mantiene la CPU fino al rilascio **spontaneo** (terminazione, blocco); ***no-preemption***
- prestazioni basse in termini di tempo medio di attesa
- Utilizzabile in sistemi **batch**

Esempio FCFS:

Processi	**Tempo di esecuzione**
P_1	**30**
P_2	**55**
P_3	**5**

- I processi arrivano nell'ordine **P_1, P_2, P_3** e sono serviti FCFS

tempo di turnaround	P_1	30
" "	P_2	85
" "	P_3	90

Ordine: **P_1, P_2, P_3** produce
tempo medio di turnaround (30+85+90) / 3 = 68.33

Ordine: **P_3, P_1, P_2** produce
tempo medio di turnaround = 43.33

Shorted-Job-First (S.J.F.)

Tra tutti i processi nella coda dei pronti si sceglie quello **di durata minore**

- occorre conoscere le carattersitiche dei processi (**carico stabile**)
- fornisce la **soluzione ottima** (tempo medio di attesa **minimo**)

P_1	**a**	Se eseguiti nell'ordine:
P_2	**b**	P_1 finisce al tempo **a**
P_3	**c**	P_2 " " **a + b**
P_4	**d**	P_3 " " **a + b + c**
		P_4 " " **a + b + c + d**

Tempo medio di attesa = (4a + 3b + 2c + d) / 4

a incide più degli altri, arrivando per primo e ottenendo il servizio per primo

Quindi deve essere quello minimo

Come calcolarlo? A priori.

Esempio: S.J.F.

Processi	Tempo di esecuzione
P_1	8
P_2	4
P_3	4
P_4	4

Sequenza di esecuzione

FCFS (8 + 12 + 16 + 20) / 4 = 14

SJF

P_2	4	**tempo medio di turnaround**
P_3	4	(4 + 8 + 12 + 20) / 4 = 11
P_4	4	
P_1	8	