Sistemi operativi.

Definizione e funzioni principali di un S.O.

Un sistema operativo è un'insieme di programmi che sovrintendono al processo di elaborazione dei dati interagendo con le unità hardware da un lato e con l'utente dall'altro.
Esso svolge due funzioni principali:

- Interfaccia tra il sistema d'elaborazione e l'utente;
- Gestione delle risorse.

La funzione d'interfaccia o intermediazione resa dal sistema operativo, è evidentemente resa necessaria perché in ultima analisi il computer tratta i dati e le istruzioni in forma esclusivamente binaria. Sarebbe impensabile richiedere all'utente un'interazione con la macchina a così basso livello. Il sistema operativo pertanto risolve il problema dell'interazione permettendo all'utente di interagire con il computer ad un livello elevato utilizzando una forma di comunicazione evoluta.

Le risorse di un sistema possono essere definite come l'insieme di tutti i dispositivi hardware e software che concorrono al raggiungimento di determinati obiettivi d'elaborazione dei dati.
Il sistema operativo deve cercare di ottimizzare l'uso delle risorse a sua disposizione allo scopo di raggiungere nel modo più efficiente possibile gli obiettivi prefissi dall'utente.

Evoluzione storica dei sistemi operativi.

Molta della storia dei sistemi operativi è stata fatta secondo considerazioni di natura economica.
In particolare ai primordi dell'era informatica l'hardware era molto costoso mentre il lavoro degli operatori addetti al controllo e all'utilizzazione delle macchine era relativamente poco costoso, quindi la necessità di massimizzare l'utilizzo della macchina. Oggi invece l'hardware è poco costoso mentre il lavoro dei tecnici specializzati è molto costoso da cui la necessità di rendere le macchine facili da utilizzare allo scopo di massimizzare la produttività del lavoro umano.

Agli inizi dell'era dei computer questi erano macchine enormi molto costose sia da acquistare sia da far funzionare. Il computer era usato in modo interattivo da un singolo utente. Il programmatore interagiva con la macchina a livello molto basso prevalentemente attraverso switches (interruttori) su una console e mediante l'immissione di schede perforate. L'interfaccia è sostanzialmente "crudo" hardware.

- Problema: il codice necessario per gestire le operazioni di I/O da dispositivi esterni è complesso e rappresenta una sorgente di difficoltà per il programmatore il quale deve spendere molto tempo in operazioni di debugging.
- Soluzione: costruire una biblioteca di subroutine (costituita da driver di dispositivi) capaci di gestire tali operazioni. La biblioteca è caricata in memoria e rimane là si può considerare questo come il sistema operativo allo stato embrionale.

Il computer è costoso è necessario tenerlo occupato il più possibile.

- Problema: il computer rimane inattivo per molto tempo mentre il programmatore predispone per il suo corretto funzionamento. Pessimo uso di un grosso investimento.
- Soluzione: assumere personale specializzato capace di eseguire le necessarie attività di predisposizione della macchina in breve tempo. Tali figure

professionali sono più rapide del programmatore ma ancora molto più lente della macchina.

- Soluzione: sistemi batch. Per rendere l'elaborazione più veloce, i job con requisiti simili sono raggruppati insieme ed eseguiti in un unico blocco. Così i programmatori lasciavano i loro programmi all'operatore che li ordinava in lotti (batch) e quando il computer si rendeva disponibile li mandava in esecuzione, alla fine i risultati solitamente in forma di stampe erano quindi inviati al relativo programmatore. Con i sistemi batch non è possibile l'interazione con il sistema mentre un job è in esecuzione.

Problema: in un dato momento il job sta utilizzando o le periferiche di I/O oppure la CPU, una parte del sistema rimane inutilizzata mentre l'altra lavora.
Soluzione: permettere al job di sovrapporre le operazioni di I/O e di calcolo vero e proprio. Introducendo il buffering e la gestione dell'interruzione nelle subroutine.
Problema: un job non può tenere occupata contemporaneamente la CPU e le periferiche di I/O.
Soluzione: multiprogrammazione. L'idea alla base del concetto di multiprogrammazione è la seguente: il sistema operativo tiene contemporaneamente in memoria principale diversi job, uno di questi job è scelto dal S.O. e mandato in esecuzione, ad un certo momento il job richiederà delle operazioni di I/O, anziché tenere la CPU inattiva, come accadrebbe se non si avesse multiprogrammazione, il S.O. provvede a mandare in esecuzione un altro job, mentre il primo continua le operazioni di I/O, quando tali operazioni sono concluse, la CPU è nuovamente concessa a quel job, in tal modo la CPU non resta mai inattiva.

- Nuova fase.

I computer cominciano a diventare poco costosi, mentre il lavoro diventa più costoso in proporzione.
Diventa di fondamentale importanza rendere il computer facile da usare così da migliorare la produttività degli operatori. Diventa importante l'interattività.
Il costo dei computer diminuisce ma risulta ancora costoso averne uno per utente.
Soluzione: il time-sharing interattivo.
I vecchi batch scheduler erano progettati per eseguire un job fino a che esso utilizzava la CPU in modo effettivo in sostanza fino alla richiesta di I/O. Ora gli operatori richiedono un risposta in tempi ragionevoli dal computer. Con il time-sharing si ha l'impressione di avere più job in esecuzione contemporaneamente, grazie alla possibilità di accordare la CPU a diversi job per tempi molto brevi, questo permette l'interattività in quando ogni tanto il sistema sospende l'esecuzione del job e si mette ad ascoltare l'utente.
Soluzione: preemptive scheduling.
Problema: gli utilizzatori hanno bisogno di avere i programmi pronti da mandare in esecuzione, mentre utilizzano il computer.
Soluzione: aggiungere il file system per accesso rapido ai dati. Il computer diventa un magazzino dei dati, non è necessario ricorrere alle schede o a nastri.
Problema: il capo si collega e non riesce ad eseguire il suo lavoro perché il sistema è sovraccarico.
Soluzione: priorized scheduling.
I computer diventano sempre più economici. Diventa possibile dare un computer ad ogni utente. I sistemi sono minimali sia in hardware sia software.

Nasce la necessità di condividere le risorse ed i dati, le reti cominciano a diventare importanti, così come il tema della sicurezza.
I sistemi operativi diventano più sofisticati.
La rete si afferma. Internet diventa popolare. Il sistema operativo non è più un'interfaccia con l'hardware a basso livello.
Nasce il concetto di network computer. Cioè di un sistema che permette di utilizzare risorse sparse in rete.

Struttura modulare di un sistema operativo.

Si tratta di una struttura gerarchica nella quale ogni livello o modulo agisce sostanzialmente solo sul livello immediatamente sottostante. Ogni modulo ha compiti specifici.
Il modulo più interno è rappresentato dal nucleo o kernel che si occupa della gestione dei processi e della gestione della CPU, il modulo successivo, il gestore della memoria centrale o memory manager si occupa della gestione della memoria principale. Quindi troviamo il gestore delle periferiche i cui compiti fondamentali sono costituiti dalla gestione della memoria di massa. Lo strato successivo rappresentato dal file system si occupa della gestione dei file e della traduzione delle operazioni logiche sui file in operazioni fisiche sui dati contenuti nella memoria di massa.
Il modulo successivo è rappresentato dall'interprete dei comandi o shell la cui funzione è di accettare i comandi dell'utente e di provvedere alla loro esecuzione. Infine si ha il complesso dei programmi applicativi.

Nella parte che segue ci occuperemo diffusamente d'ogni singolo modulo del sistema operativo.

Gestione dei processi.

Definizione di processo.

Per processo s'intende l'insieme delle operazioni svolte dal processore, la CPU, allo scopo di eseguire un programma.
Un programma è un'entità passiva, mentre un processo è un'entità attiva. Un processo non è un programma ma perché un programma possa essere eseguito bisogna creare uno o più processi.
Quindi un processo può essere visto come un programma in esecuzione caratterizzato dal valore del program counter e dal contenuto dei registri del processore.

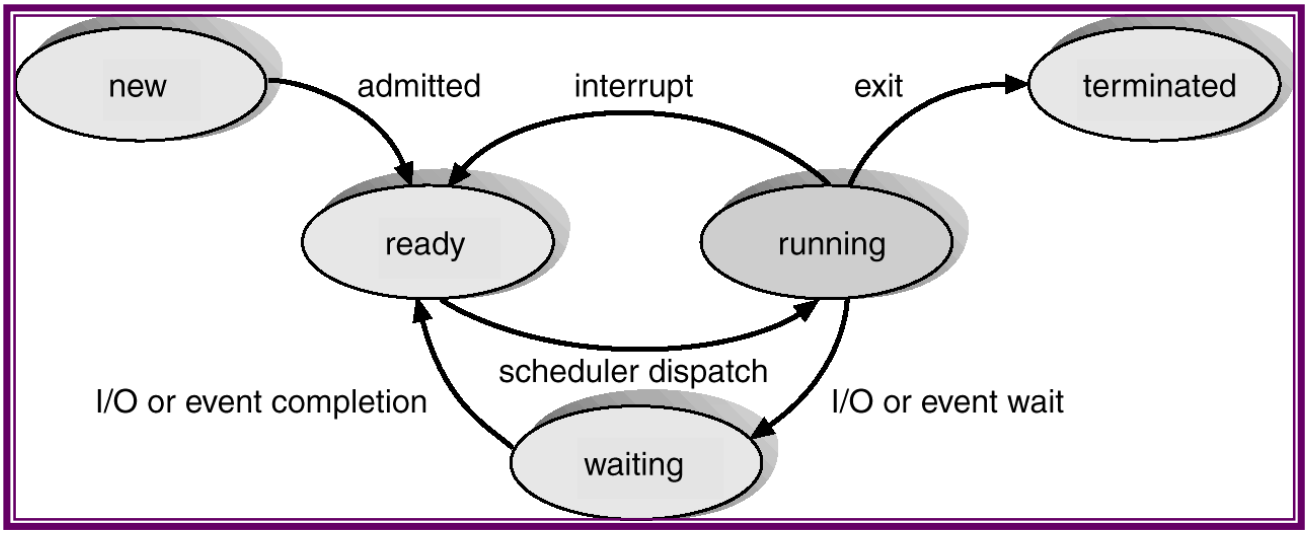
Stati di un processo.

I processi possono assumere sostanzialmente solo tre stati:

- Stato d'esecuzione (Running)
- Stato di pronto (Ready)
- Stato d'attesa (Waiting).

Un processo si trova nello stato di running se ha la totale disponibilità della CPU, cioè le istruzioni di quel processo sono in esecuzione nella CPU.
Un processo si trova nello stato di ready se non può avere la disponibilità della CPU perché essa è occupata nell'esecuzione di un'altro processo. Cioè se aspetta che la CPU si renda disponibile.
Infine si dice che un processo è nello stato di waiting se è nell'attesa di una risorsa diversa dalla CPU in genere nell'attesa di completare un'operazione di I/O.

Transizioni permesse ai processi.



Osservazione. Lo stato di running è uno stato esclusivo, cioè un solo processo alla volta può stare nello stato di running su una data CPU. Cioè la CPU elabora un processo alla volta.
Mentre per gli stati di ready e waiting questo non vale, ovvero è possibile avere diversi processi disposti in code. Così si parlerà di coda ready o coda waiting.

Identificazione dei processi: PCB.

Ogni processo è riconosciuto dal sistema operativo grazie ad un descrittore detto PCB (Process Control Block) il quale lo identifica in modo univoco. Il PCB contiene gran parte delle informazioni relative ad un processo; in particolare contiene: stato del processo, il contenuto del program counter, il numero del processo, stato dei regstri, stato della memoria centrale, puntatori ai processi in coda etc...

Gestione della CPU (CPU Scheduling).

Caratteristiche della CPU.

La CPU ha la caratteristica di essere:

- Non divisibile. Significa che un solo processo alla volta può essere in esecuzione, cioè un solo processo alla volta può trovarsi nello stato di running in una data CPU.
- Interrompibile. Significa che è possibile sospendere la esecuzione di un processo o provocarne la interruzione forzata.

Grazie alla presenza di queste caratteristiche è possibile sviluppare delle strategie di gestione della CPU la cui funzione ultima e primaria consiste nella ottimizzazione dell'uso della risorsa CPU.

Parametri quantitativi.

Allo scopo di valutare numericamente la bontà di una strategia di gestione si utilizzano diversi parametri quelli che qui si descriveranno sono: la percentuale di utilizzo della CPU, il rendimento o throughput.

Percentuale di utilizzo della CPU.

La percentuale di utilizzo della CPU rappresenta una misura dell'utilizzo della CPU in termini di tempo in modo percentuale.

$$\%CPU = 100 * T_{CPU} / T_{in} + T_{out} + T_{CPU}$$

Nella formula T_{CPU} rappresenta il tempo durante il quale la CPU è attiva, T_{in} rappresenta il tempo durante il quale si hanno operazioni di input, T_{out} indica il tempo durante il quale si hanno operazioni di output. Il denominatore della frazione, se riferito ad un processo, rappresenta evidentemente il tempo necessario ad eseguire un processo. Mentre il numeratore rappresenta il tempo durante il quale il processo impegna effettivamente la CPU.

Rendimento o Throughput.

Indica il numero dei processi eseguiti nella unità di tempo.

$$\text{Throughput} = N / \sum_{i=1}^{N} (T_{in} + T_{out} + T_{CPU})_i$$

N rappresenta il numero di processi ed il denominatore rappresenta il tempo necessario ad elaborarli.

Al fine di illustrare alcune delle diverse strategie di gestione della CPU si procederà secondo uno schema che rappresenta l'evoluzione dei diversi algoritmi e mostra come nel tempo si sia resa necessaria l'introduzione di nuovi concetti e nuove tecniche.

Monoprogrammazione.

In monoprogrammazione si ha un solo programma in memoria centrale. Inoltre si ha un solo processo in coda ready il quale, quando si libera la CPU, passa in stato di running fino alla terminazione.
La monoprogrammazione offre basse percentuali d'utilizzazione della CPU e bassi valori di throughput.

Modo batch.

Il modo batch consiste nell'immettere più job in un unico blocco. Abbiamo ancora un solo processo in stato di running come sempre, ma più processi in stato di ready. La ready queue viene gestita secondo il criterio FIFO.
Si ha un miglioramento dei parametri e però si introduce un nuovo compito per la CPU la gestione della coda di ready.
Infatti va detto che il processore può trovarsi in due stati diversi:

- modo supervisore, esegue processi che sono di utilità per il sistema operativo, ma non riguardano esplicitamente i programmi utente.
- modo utente, se sono in esecuzione processi derivanti da programmi utente.

Le operazioni di gestione sebbene utili e necessarie per ottimizzare l'utilizzazione delle risorse in ultima analisi costituiscono un costo, infatti richiedono risorse che altrimenti potrebbero essere utilizzate in modo utente. Si ritiene accettabile un T_s minore o uguale al 10% del tempo totale (Ts + T_u) dove T_u e T_s rappresentano il tempo in cui la CPU si trova in modo utente e modo supervisore rispettivamente. In altre parole le strategie di gestione sono convenienti fintanto che non richiedono troppe risorse.

Con le strategie esposte finora cioè monoprogrammazione e metodo batch un processo rilascia la CPU solo quando è giunto a terminazione (normale o anomala), durante le operazioni di I/O il processore rimane inattivo.
Con l'introduzione di un nuovo meccanismo, quello delle interruzioni si compie un notevole passo avanti nella strada della ottimizzazione dell'uso della CPU.

Il meccanismo delle interruzioni.

Con il termine interruzione si intende un evento che causa la sospensione temporanea di un processo in esecuzione. Un evento può consistere nella richiesta di una operazione di I/O in tal caso si parla di interruzione esterna oppure può trattarsi di una richiesta di interrupt perchè il tempo concesso al processo è scaduto, o un errore di esecuzione e si parlerà di interruzione interna. La cosa importante è comunque che il meccanismo delle interruzioni permette di togliere la CPU ad un processo e di accordarla ad un altro. Perché si può rendere necessaria un'operazione del genere?
Supponiamo che un processo in esecuzione richieda un'operazione di I/O, le operazioni di questo tipo sono molto lente rispetto ai tempi della CPU, mentre il processo attende il completamento delle operazioni di I/O il processore è inattivo, se si vuole tenerlo attivo si può procedere in questo modo: si può interrompere il processo che richiede I/O e metterlo in uno stato tale che possa completare le operazioni di I/O ma che lasci libera la CPU (stato di waiting), quindi si può concedere la CPU ad un altro processo che quindi avanza, quando il processo precedente termina le operazioni di I/O passa nello stato di ready in attesa che gli si conceda la CPU e quando ciò avviene può giungere a terminazione, il secondo processo potrebbe essere nel frattempo giunto a terminazione o potrebbe avere esso stesso richiesto una operazione di I/O nel qual caso esso stesso subirebbe una interruzione; il risultato di tale procedura è comunque che la CPU non è stata mai inattiva. Questo è

la meta che ci si prefigge con il meccanismo delle interruzioni: fare in modo che la CPU sia sempre attiva.
Con l'introduzione del meccanismo delle interruzioni si può parlare di parallelismo virtuale infatti le interruzioni permettono di concedere la CPU a diversi processi con grande rapidità dando così l'impressione che essi procedano contemporaneamente o parallelamente, anche se in realtà solo uno alla volta si trova in stato di running.
Ora ci si trova a dover risolvere un problema: secondo quale criterio si deve concedere la CPU ai diversi processi in coda di ready?
Si distinguono due politiche di gestione:

- Politica event driven,
- Politica time driven.

Nella politica event driven è il verificarsi di un'evento a determinare la transizione di stato di un processo, per evento si intende ad esempio la terminazione di un processo, o la richiesta di una operazione di I/O.
Nella politica time driven invece, la transizione avviene sulla base del tempo assegnato ad un processo.

I diversi criteri nell'ambito della politica event driven.

I criteri più comunemente adottati sono sostanzialmente tre.

- **Criterio FIFO o First Come First Served**. Secondo questo criterio viene servito dalla CPU per primo quel processo della coda ready che è entrato per primo in coda. Vantaggi: semplicità dell'algoritmo di gestione che determina poco overhead. Svantaggi grande variabilità dei tempi di attesa in coda e possibilità che processi molto brevi debbano attendere troppo a lungo la terminazione di processi molto lunghi serviti prima.
- **Criterio SJF Shorthest Job First**. In base a tale criterio il processo che ottiene la CPU cioè che passa dallo stato di ready allo stato di running è il processo della coda che ha il minor numero di istruzioni. Vantaggi riduzione del tempo medio di attesa in ready per i processi brevi. Svantaggio eccessiva penalizzazione per i processi lunghi.
- **Criterio SPTF Shortest Process Time First**. Il criterio prevede di servire per primi quei processi che hanno un T_{CPU} più breve cioè vengono serviti per primi i processi con molte operazioni di I/O. Questi processi passano in waiting con grande frequenza quindi consentono ad altri processi di ottenere la CPU.

Osservazione. Tutti i criteri determinano un certo overhead di sistema, ma alcuni cioè quelli che implementano algoritmi particolarmente complessi richiedono molte risorse quindi come sempre il migliore dei criteri è quello che presenta il più basso valore del rapporto tra overhead di sistema e vantaggi derivanti dalla gestione.

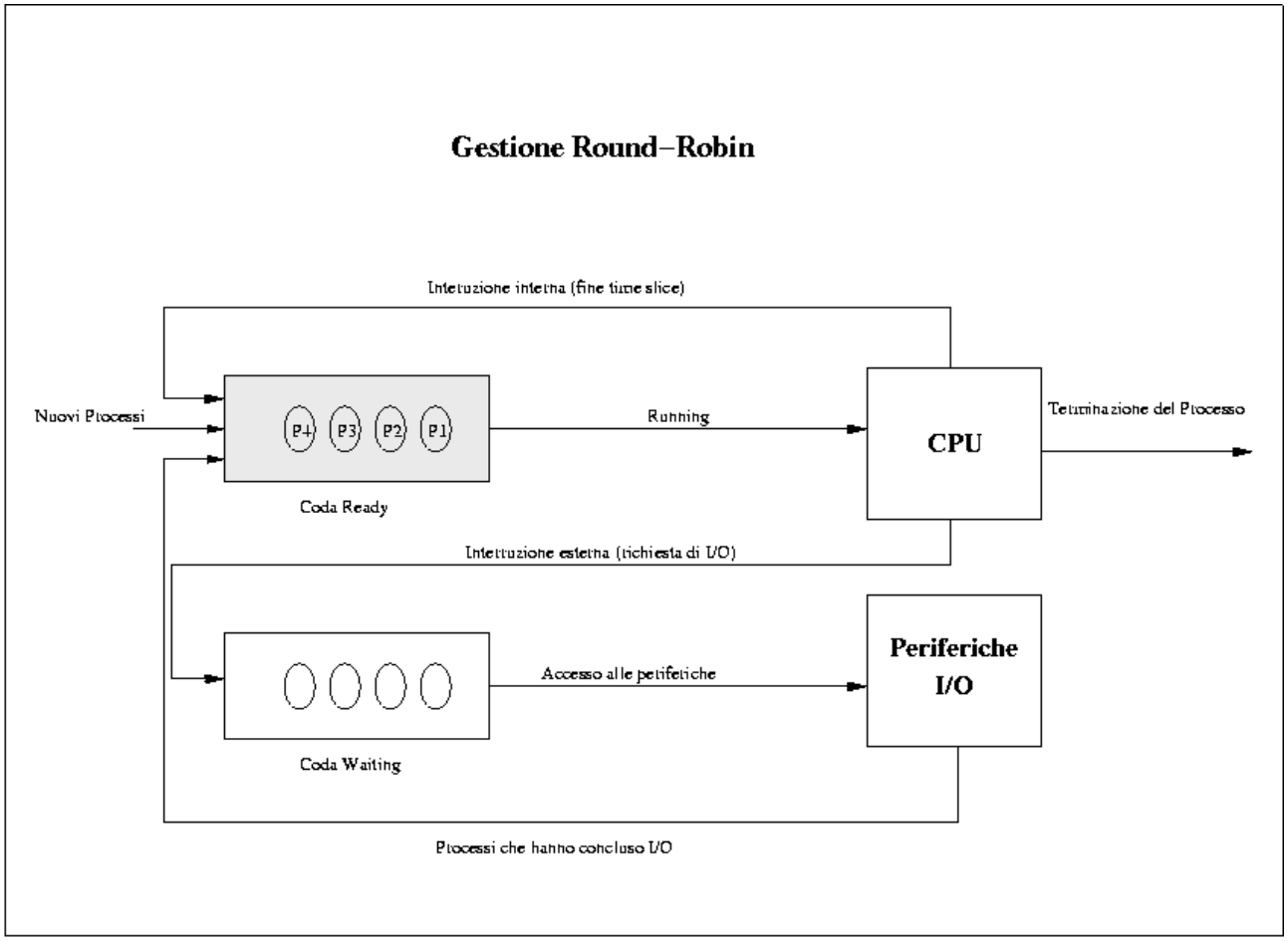
I diversi criteri nell'ambito della politica time driven.

I criteri time driven sono noti anche con il nome di preemptive o a prelazione in quanto forzano il rilascio della CPU dei processi prima che raggiungano la terminazione o vengano mandati in stato di waiting.

- **Criterio round-robin**.

In base a questo criterio di gestione ad ogni processo viene assegnata la CPU per un tempo massimo prefissato detto quanto di tempo o time slice, al termine del quale il processo viene sospeso e quindi posto in stato di ready.
La coda ready viene gestita con criterio FIFO. I nuovi processi vengono posti in coda e serviti ciclicamente per un quanto di tempo, se un quanto di tempo è sufficiente per portare il processo a terminazione esso rilascerà la CPU spontaneamente, se invece un quanto di tempo non è sufficiente per il completamento del processo questo riceve un'interrupt e viene posto

forzatamente alla fine della coda ready mentre si assegna la CPU. Le prestazioni del criterio RR dipendono molto dalla dimensione del quanto di tempo se esso è molto lungo cioè tende all'infinito il criterio RR coincide con il criterio FCFS, se il time slice è molto breve il criterio RR viene chiamato processor sharing e gli utenti hanno l'impressione che ognuno degli n processi della coda abbia un proprio processore con velocità 1/n.



Gestione della memoria centrale.

Struttura della memoria centrale.

La memoria centrale può essere vista come un insieme di locazioni contigue singolarmente indirizzabili. Ad ogni locazione corrisponde un indirizzo.
A sua volta ogni locazione è formata da un insieme di elementi bistabili in numero di 8, 16, 32 o 64 a seconda della macchina.
Ad ognuno dei due stati permessi si fa corrispondere uno stato logico. Cioè ogni elemento bistabile permette di rappresentare un singolo bit. L'insieme dei bit corrispondenti ad una singola locazione si chiama word o parola.

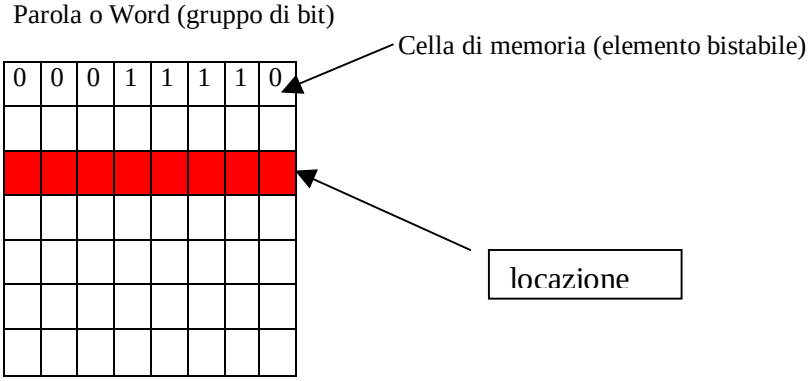
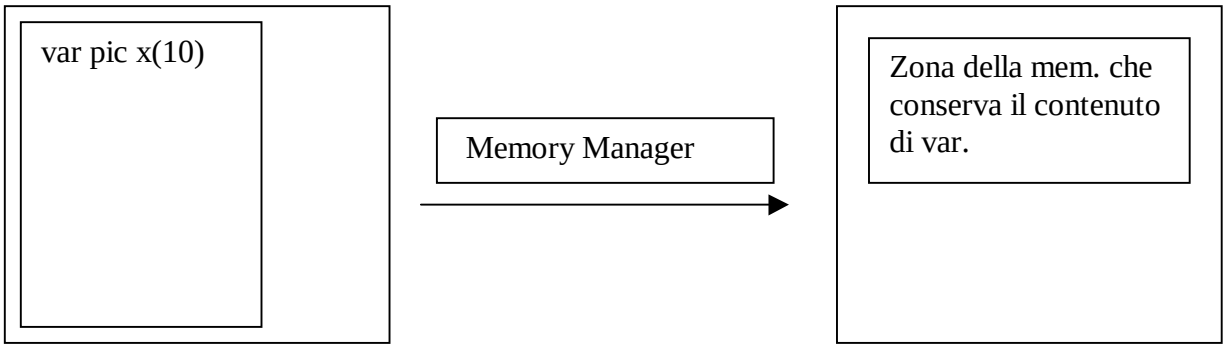


Fig. 1 Struttura della M.C.

Piano logico e piano fisico.

Quando si scrive un programma utilizzando un linguaggio di programmazione evoluto, si definiscono le variabili e le istruzioni da un punto di vista esclusivamente logico. Nessun riferimento infatti è presente alla effettiva posizione in memoria delle variabili e delle istruzioni.
La traduzione dal piano logico al piano fisico è a carico del sistema operativo ed in particolare del memory manager o gestore della memoria.
Si parla anche di spazio logico cioè quello relativo alle variabili ed alle istruzioni del programma sorgente, e di spazio fisico ovvero quello relativo agli indirizzi fisici delle locazioni in memoria che conterranno le variabili e le istruzioni.



Rilocazione e Allocazione.

La traduzione delle istruzioni dal piano logico al piano fisico si esplica attraverso due fasi.

Fase di rilocazione dei dati e delle istruzioni. Consiste nel calcolare gli indirizzi effettivi delle regioni in memoria che dovranno contenere i dati.

Fase di allocazione dei dati e delle istruzioni. Consiste nel riservare ed attribuire effettivamente lo spazio calcolato alle variabili.

La fase di rilocazione può essere classificata in base al momento nel quale avviene il calcolo degli indirizzi in:

Rilocazione assoluta. In questo caso il calcolo degli indirizzi avviene in fase di compilazione. Cioè la attribuzione degli indirizzi fisici alle variabili ed istruzioni avviene durante la compilazione del programma. Gli indirizzi effettivi sono sempre gli stessi ogni volta che si lancia il programma.

Rilocazione statica. In questo caso il calcolo degli indirizzi avviene al momento del caricamento in memoria del programma, e rimangono gli stessi durante tutto il periodo di esecuzione del programma.

Rilocazione dinamica. In questo caso il calcolo degli indirizzi avviene in fase di esecuzione, quindi la posizione e lo spazio allocato ai singoli processi può cambiare in seguito alla disponibilità o meno di aree libere in memoria.

Le partizioni o block.

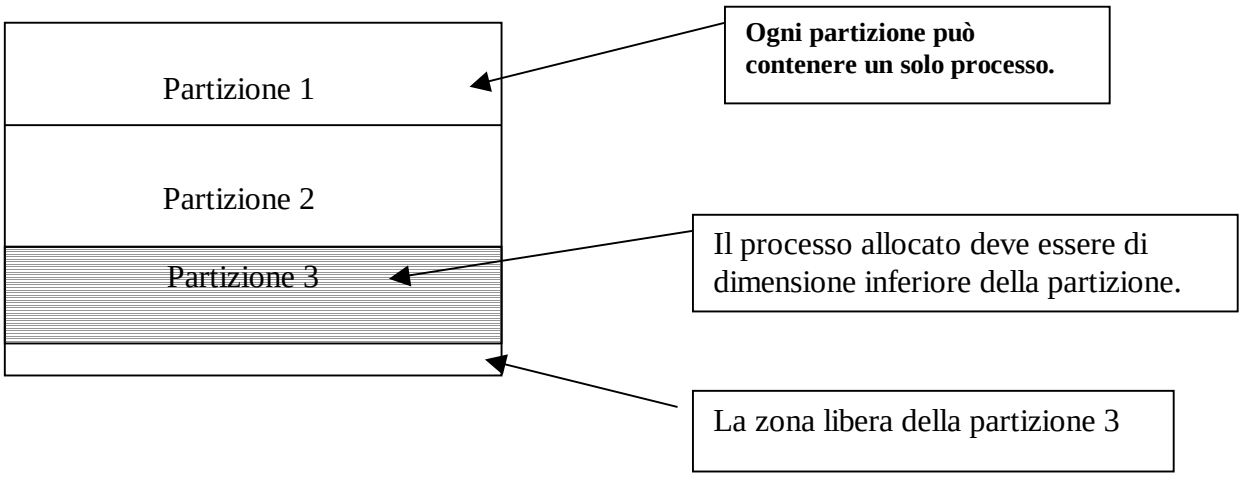
Per partizione si intende una area in memoria capace di accogliere cioè allocare un processo soltanto.

La suddivisione della memoria può avvenire secondo due modalità:

Partizioni fisse. La memoria principale viene suddivisa in un numero fisso di partizioni di dimensioni variabili. Si procede quindi ad allocare un processo in una partizione, lo spazio libero residuo non può essere utilizzato.

Partizioni dinamiche. Le partizioni non sono fissate a priori ma vengono create man mano che i processi vengono allocati. In tal caso le partizioni possono variare sia in numero che dimensioni.

Bisogna osservare che la possibilità di modificare le partizioni si basa sulla presenza di due tipi di operazioni possibili. La operazione di splitting o suddivisione della partizione, e la operazione di coalesce o merging che consiste nella fusione di due o più partizioni adiacenti allo scopo di creare una partizione più grande.



Il concetto di frammentazione.

Per frammentazione si intende la incapacità di utilizzare la memoria libera a causa della configurazione dei processi allocati. Si intende cioè il fenomeno dello spreco di memoria.

Si distinguono due tipi di frammentazione:

frammentazione esterna. Si verifica quando non è possibile utilizzare la memoria libera in quanto questa è suddivisa in tante piccole partizioni non adiacenti, che non possono essere riunite in una grande partizione libera utilizzabile.

Frammentazione interna. Si verifica quando viene concesso ad un processo molto più spazio di quello effettivamente necessario. Pertanto lo spazio residuo essendo all'interno di una partizione e potendo una partizione ospitare un solo processo determina la impossibilità di utilizzare tale spazio residuo.

Lista delle partizioni libere.

Il sistema è a conoscenza delle partizioni libere e della loro posizione in quanto esse sono raccolte in una lista delle partizioni libere.
Quando si presenta la necessità di allocare un nuovo processo il sistema si riferisce alla lista delle partizioni libere.
Perché un processo possa essere allocato in una partizione libera basta che essa sia di dimensioni sufficienti ad accoglierlo.
In realtà però si utilizzano criteri particolari nella assegnazione delle partizioni libere ai diversi processi.
I criteri più noti sono i seguenti:
Criterio First Fit. Consiste nel assegnare ad un processo la prima partizione libera di dimensioni sufficienti nella lista delle partizioni libere. Questo è il più rozzo dei criteri.
I suoi vantaggi sono il fatto che data la sua semplicità permette una assegnazione veloce delle partizioni libere ai processi e non determina un overhead di sistema.
Svantaggi: Nessuna considerazione sulla frammentazione.

Criterio Best Fit. Consiste nel scegliere tra tutte le partizioni libere la più piccola delle partizioni libere che permettono di allocare il processo, cioè quella che determina il minor spazio libero residuo.
Vantaggi: Riduce la frammentazione interna.
Svantaggi: Tende a lasciare troppi piccoli pezzi che difficilmente potranno essere utilizzati per allocare nuovi processi, perché troppo piccoli.

Criterio Worst Fit. Consiste nel assegnare ad un processo quella partizione libera che lascia più spazio libero residuo. Combatte la frammentazione esterna perché con grande probabilità lo spazio lasciato sarà sufficiente ad allocare nuovi processi. Però tende ad eliminare le partizioni di grosse dimensioni e pertanto può diventare difficile allocare processi di grosse dimensioni, cioè che richiedono molta memoria.

Partizioni rilocabili.
Un metodo ancora differente è quello delle partizioni rilocabili. Esso si basa sulla possibilità di spostare i processi in memoria. Lo scopo è di compattare la memoria libera e pertanto permettere attraverso la fusione di partizioni libere adiacenti di creare nuove partizioni che consentano di allocare nuovi processi.
Riduce fortemente la frammentazione ma richiede un continuo ricalcolo degli indirizzi e pertanto richiede un notevole dispendio di risorsa CPU. Grosso overhead di sistema.

Tutti i criteri illustrati finora utilizzano sempre le operazioni di splitting e di coalesce allo scopo di operare su partizioni adiacenti, sia creando nuove che fondendo per darne delle nuove.

Paginazione o paging.

Nella paginazione o paging il processo viene suddiviso in pagine logiche tutte delle stesse dimensioni. Ogni pagina logica viene allocata in una pagina fisica che è invece una suddivisione della memoria centrale in grado di allocare una pagina logica. Le pagine fisiche sono di dimensioni fisse molto spesso 4k byte.
Ora per conoscere l'indirizzo di una istruzione in memoria bisogna dare due coordinate la prima è l'indirizzo di pagina e la seconda è l'indirizzo entro la pagina (offset).
La traduzione dal piano logico al piano fisico avviene grazie alla tabella di traduzione degli indirizzi che contiene l'indirizzo base di ogni pagina fisica ovvero l'indirizzo dell'inizio di ogni pagina fisica.
Inoltre esiste una tabella di occupazione delle pagine fisiche che permette al memory manager di sapere quali pagine sono occupate e quali no, così da poter allocare le nuove richieste.
Il paging permette di eliminare la frammentazione esterna infatti scomponendo un processo in pagine lo si può allocare anche in zone non adiacenti, si vanno in qualche modo a tappare i "buchi" di memoria libera che altrimenti resterebbero inutilizzati.
Riduce il problema della frammentazione interna infatti lo spazio residuo può essere al massimo di mezza pagina per processo.
Tanto più piccole sono le pagine tanto più piccolo sarà lo spazio sprecato per allocare un processo, però saranno necessarie un numero maggiore di pagine e quindi la gestione delle pagine diventerebbe una attività troppo dispendiosa in termini di risorsa CPU.

Segmentazione.

Nella segmentazione il processo viene ancora diviso in parti logiche dette segmenti ma diversamente dalla paginazione queste unità logiche non sono tutte delle stesse dimensioni.
Ogni segmento nasce da un modulo di programma cioè ogni segmento nasce da una procedura del programmatore così per esempio ci saranno i segmenti che derivano dalle variabili locali, quelli che provengono dalle variabili globali, quelle che provengono dalle procedure etc.
La conversione del programma in segmenti dipende dal linguaggio che si utilizza nella codifica del programma stesso. Ovvero diversi linguaggi di programmazione individuano diversi criteri per generare i segmenti.

Il concetto di memoria virtuale.

Consiste nella capacità di alcuni sistemi di simulare la condizione di avere più memoria fisica di quella realmente disponibile.
A tale scopo una parte della memoria di massa, memoria sul disco viene trattata come se fosse una estensione della memoria centrale.
Ovvero un'area della memoria di massa (area di swap) viene utilizzata per effettuare il dump della memoria del processo, cioè il contenuto della memoria relativo al processo viene scaricata su disco in modo tale da potere successivamente ricaricarla in RAM quando una parte della memoria si libera.
Questa strategia permette di caricare in memoria più programmi di quelli che sarebbe possibile in assenza della memoria virtuale o di caricare programmi più grossi.
Bisogna tenere presente che l'accesso alla memoria su disco è molto più lento dell'accesso alla RAM quindi se un programma per poter girare richiede un uso massiccio della memoria virtuale girerà in modo terribilmente lento.

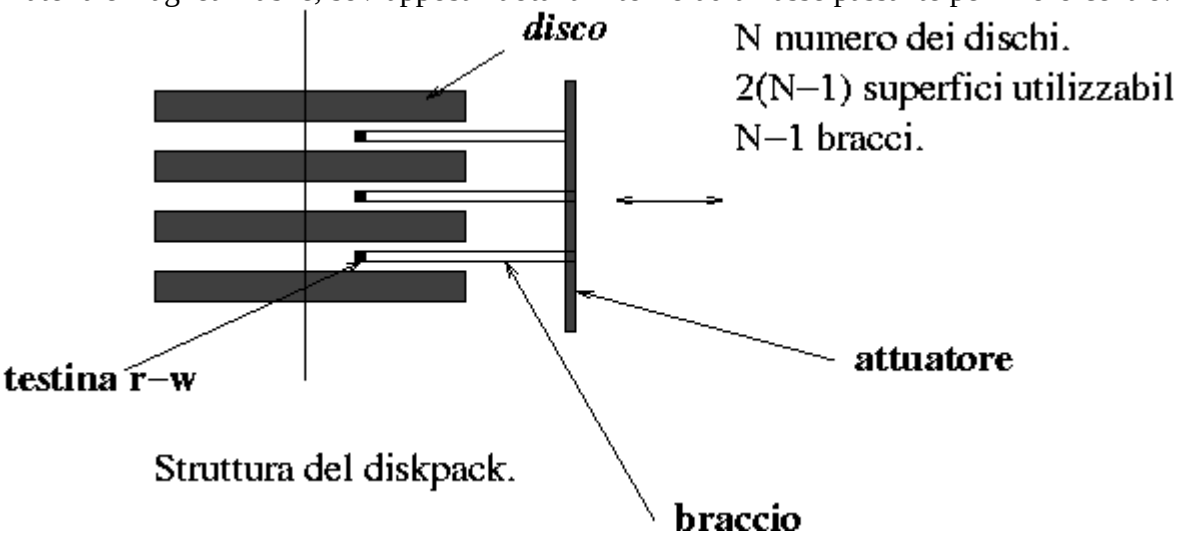
Gestione della memoria di massa.

Gestione del disco rigido.

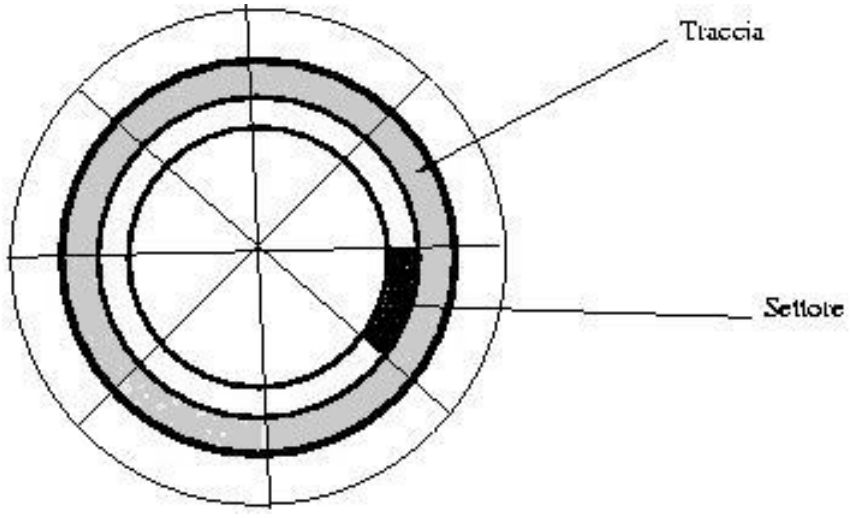
Dal punto di vista del S.O. la gestione della memoria di massa ed in particolare del disco, significa ottimizzare i tempi di registrazione e di reperimento dei dati.

Struttura del disco rigido.

Il disco rigido (Hard Disk) è costituito da un insieme di piatti, di alluminio ricoperti da uno strato di materiale magnetizzabile, sovrapposti ruotanti intorno ad un asse passante per il loro centro.



La superficie del disco è divisa in regioni anulari concentriche, il cui centro è rappresentato dall'asse di rotazione, dette tracce.
Ogni traccia è indirizzabile direttamente.
Le tracce sono poi divise in settori o blocchi. Per accedere ad un blocco è necessario attendere che il blocco passi sotto o sopra le testine di lettura-scrittura.
Il settore costituisce la più piccola parte indirizzabile di un disco.
L'insieme delle tracce corrispondenti ad una posizione del braccio costituisce un cilindro. In altre parole un cilindro è costituito da tutte le tracce che hanno lo stesso raggio. Quindi se il disk-pack consiste di 4 dischi, quindi di 6 superfici utilizzabili, un cilindro è costituito da 6 tracce, una per ogni superficie.



La lettura e scrittura dei dati sul disco avviene tramite le apposite testine.
Se volessimo individuare un singolo blocco su un hard-disk dovremmo specificare 3 coordinate:

- Numero di faccia o superficie.
- Numero di traccia o cilindro.
- Numero di settore.

Se si numerano i settori in progressione, il sistema individua il disco come un vettore unidimensionale di blocchi.
Ad ogni blocco corrisponde un numero, che lo individua in modo univoco.
In base alla convenzione di numerazione alle 3 coordinate (cilindro i, superficie j, settore k) si fa corrispondere l'indirizzo unidimensionale I (i, j, k) = k + NS * (j + i * NT), dove NS indica il numero di settori per traccia e NT indica il numero di tracce per cilindro ovvero il numero di superfici o facce distinte del diskpack.
Esempio. Si abbia un diskpack che consiste di N=3 dischi, con 3 traccie per faccia, con NS = 6 settori per traccia. Poichè il numero di facce risulta essere 2 * (N –1) risulta NT = 4.
Il numero di settori singolarmente indirizzabili vale allora NS*NT*#tracce = 72.
Se si numerano le tracce in modo progressivo si avranno le tracce 0, 1, 2 ovvero che è lo stesso i cilindri 0, 1, 2.
Ad ogni superficie si può assegnare il numero di superficie o numero di faccia che può assumere i valori 0, 1, 2, 3.
Ad ogni settore si associano i numeri 1, 2, 3, 4, 5, 6. Avevamo supposto NS = 6.
Un blocco o settore lo individuo come già detto con (i, j, k) rispettivamente (#cilindro, #superficie, #settore).
Quindi l'indirizzo del 1° blocco del 1° cilindro della 1° faccia ovvero I(0, 0, 1), sostituendo nella formula per I(i, j, k), vale 1 proprio come ci si aspetta.
Mentre l'indirizzo dell'ultimo blocco dell'ultimo cilindro dell'ultima faccia corrisponde a
I(2, 3, 6) = 6 + 6 * (3 + 2 * 4) = 72 proprio quanto volevamo infatti il numero massimo di settori singolarmente indirizzabili risultava 72.

Il problema che deve essere risolto da parte del sistema operativo è:

- sapere dove trovare i blocchi liberi.
- ottimizzare i tempi di accesso ai blocchi.

Gestione dello spazio libero.

Il problema di distinguere dove trovare i blocchi liberi può essere convenientemente risolto in diversi modi i due più utilizzati sono:

- **Vettore di bit.**

  Ad ogni blocco individuato mediante l'indirizzo unidimensionale appena descritto si associa un bit di stato.

  Esempio. Blocco libero bit di stato di valore 1, blocco occupato bit di stato di valore 0.

  Lo stato di occupazione del disco risulta quindi completamente descritto dal vettore di stato.

  Il numero dei bit del vettore è uguale al numero di blocchi del disco. Nell'esempio precedente 72.

  Esempio: 000010101111000000111111111111000000110011001100111111111111100

Con il vettore di bit è immediato vedere se ci sono blocchi liberi consecutivi.
Questo metodo è realizzabile in pratica solo per dischi piccoli altrimenti il vettore di bit diventa di dimensioni proibitive per una gestione efficiente dovendo infatti risiedere in memoria centrale
.

- **Lista concatenata o linkata.**

  Si ottiene collegando tra loro i blocchi liberi tramite dei puntatori e memorizzando il primo blocco libero (Testa della lista).

  La lista è in sostanza costituita dagli indirizzi dei blocchi liberi. Il primo elemento della lista mi rimanda al blocco libero successivo della lista, il secondo al terzo e così via.

  Nella lista linkata per trovare più blocchi contigui bisogna scorrere tutta la lista.

- **Raggruppamento.**

  Il raggruppamento è una variante della lista linkata.

  Consiste nel memorizzare nel primo blocco di una serie di blocchi liberi contigui, gli indirizzi di tali blocchi.

  L'ultimo elemento del gruppo contiene l'indirizzo del primo elemento del gruppo successivo.

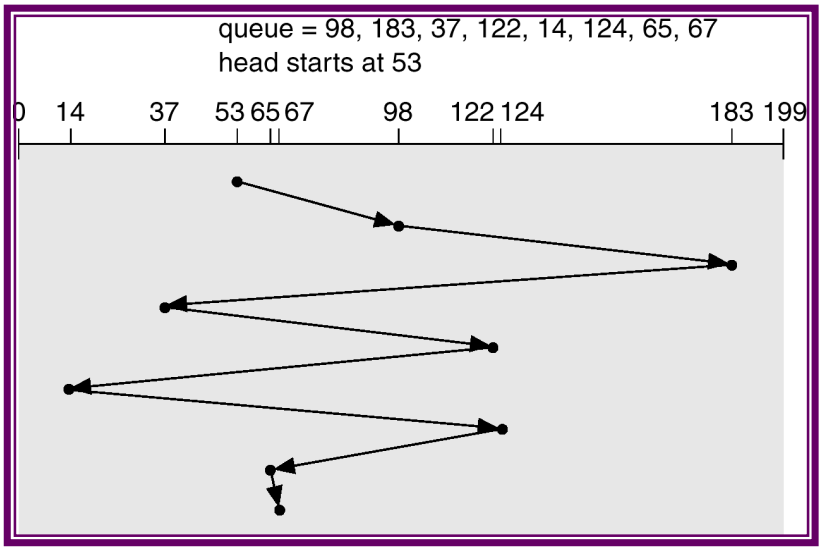
Scheduling del disco.(Pianificazione dell'accesso al disco).

Tecniche di minimizzazione dei tempi di accesso ai blocchi.

Il tempo di accesso ad un blocco risulta: $T_{accesso} = T_{seek} + T_{latenza}$.
Il tempo T_{seek} rappresenta il tempo necessario perché la testina si disponga sulla traccia richiesta, si tenga presente che l'accesso alla traccia è un accesso diretto.
Il tempo $T_{latenza}$ indica il tempo necessario perché il blocco desiderato della traccia sulla quale la testina si trova passi in corrispondenza della testina stessa. Questo tempo dipende dalla velocità di rotazione del disco.

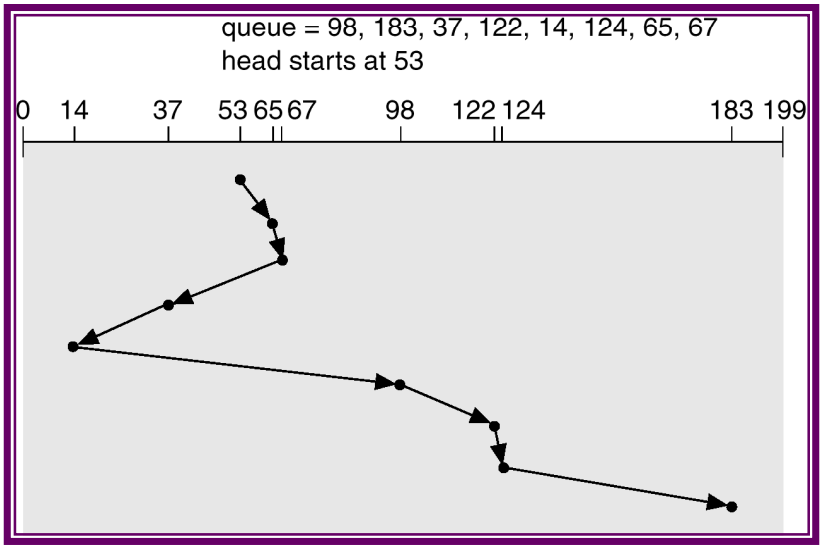
Se un processo richiede una operazione di I/O da disco, poiché il $T_{accesso}$ è grande rispetto ai tempi della CPU, il processo viene sospeso e posto in coda di waiting.
Quali criteri si adottano nel soddisfare le richieste dell'accesso al disco di processi in coda che lo richiedono?
Esistono diversi criteri.

- **Criterio FIFO o Scheduling FCFS** (First Come, First Served). Secondo questo criterio le richieste vengono soddisfatte in ordine di arrivo. Se si adotta questo criterio può capitare che si richieda l'accesso a regioni molto distanti in rapida successione, quindi la testina deve muoversi avanti e indietro tra regioni anche molto distanti, senza che nessuna considerazione sulla ottimizzazione dei tempi di accesso, risultanti dalla minimizzazione del percorso totale, venga realmente effettuata.



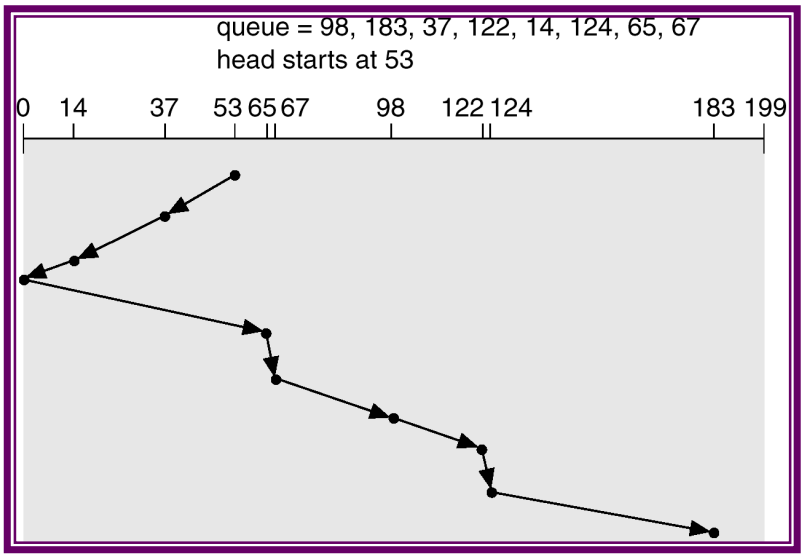
Nell'esempio illustrato la testina deve visitare un totale di 640 cilindri. Un miglioramento si ottiene con il criterio seguente.

- **Criterio SSTF** (Shortest Seek Time First). In base a questo criterio vengono serviti per primi i processi che richiedono un accesso a tracce vicine a quella corrente. Si ha un notevole miglioramento del tempo di accesso ma a discapito di un certo overhead di CPU richiesto per eseguire i calcoli necessari.



Nell'esempio della figura risulta che il numero dei cilindri visitati si riduce a 208, si nota quindi un notevole miglioramento.

- **Scan**. Secondo questo algoritmo il braccio dell'unità a disco parte da un estremo del disco e si sposta verso l'altro estremo servendo le richieste mentre attraversa i cilindri, fino a che non raggiunge l'estremo opposto del disco, quindi inverte il senso di marcia; la procedura va avanti in questo modo. Le testine attraversano il disco in continuazione nelle due direzioni in modo ciclico. Cioè operano una scansione del disco.



Altri aspetti della gestione del disco.

Formattazione.

Un disco magnetico nuovo non può essere utilizzato se non si provvede a formattarlo.
La **formattazione fisica** consiste nel dividere il disco in settori, in ogni settore viene scritta una speciale struttura di dati che consiste di una intestazione, un'area per la memorizzazione dei dati ed una coda, l'intestazione e la coda contengono informazioni relative al settore (numero del settore, e un codice per la correzione degli errori, ECC (Error Correcting Code)) queste informazioni vengono utilizzate dal controller del disco nelle operazioni di lettura e scrittura. Quando il controller scrive dei dati in un settore calcola il valore del ECC che dipende dai dati contenuti nella area dei dati di quel settore e lo memorizza nella intestazione e nella coda, quando il controller va a leggere i dati

contenuti nel settore prima di tutto ricalcola il valore del codice ECC che dipende dai dati memorizzati e lo confronta con il valore già memorizzato e se sono uguali provvede alla lettura, se sono diversi significa che il settore è difettoso, ed in alcuni casi cioè se il numero di bit alterati è piccolo provvede alla loro correzione quindi alla lettura.
La formattazione fisica del disco viene fatta dal costruttore del disco nella maggior parte dei casi.
Per memorizzare dati nel disco però non basta la formattazione fisica, infatti ogni sistema operativo utilizza un proprio file system, è necessario quindi memorizzare sul disco delle strutture di dati tipiche del file system questo processo si chiama **formattazione logica**.

Gestione dei blocchi difettosi.

Può capitare che uno o più settori del disco smettano di funzionare correttamente perché per diverse ragioni, la superficie stessa del disco si alteri. I blocchi difettosi non possono essere utilizzati per la memorizzazione dei dati pertanto devono essere esclusi nel processo di allocazione. In dischi dotati di controller EIDE (quelli che normalmente abbiamo nei computer) si deve attuare manualmente un controllo e isolamento dei settori danneggiati Esistono comandi nel DOS chkdsk o scandisk in Windows che sono capaci di identificare e provvedere ad isolare i settori danneggiati.
Dischi dotati di controller SCSI seguono strategie più evolute che automatizzano queste operazioni.

Il File System.

Il File System rappresenta la parte più visibile del sistema operativo da parte degli utenti.
Fornisce il meccanismo per la memorizzazione e l'accesso di dati e programmi.
Il file system consiste in generale di tre parti:

- collezione di file,
- struttura di directory,
- insieme di partizioni.

Concetto di file.

Si può definire file come un'insieme di informazioni, correlate e registrate nella memoria secondaria, a cui è stato assegnato un nome.
Dal punto di vista dell'utente un file rappresenta la più piccola porzione di memoria secondaria logica, i dati possono essere scritti nella memoria secondaria soltanto all'interno di un file. Sebbene sia possibile memorizzare sia record che caratteri questi devono comunque trovarsi all'interno di un file.

Attributi dei file.

Gli attributi dei file sono:

- **Nome**, indica il nome simbolico del file che permette all'utente di individuarlo in modo univoco.
- **Tipo**, un sistema può supportare diversi tipi di file.
- **Locazione**, si tratta di un puntatore al dispositivo e alla locazione del file su quel dispositivo, indica in sostanza dove si trovano i dati del file sulla memoria di massa.
- **Dimensione**, si riferisce alla dimensione del file in byte o blocchi.
- **Protezione**, indica chi può accedere al file ed in che modo lettura, scrittura, esecuzione.
- **Ora, data identificazione del proprietario**, le date possono riguardare la creazione del file, l'ultima modifica, ultimo accesso.

Tutte le informazioni relative ai file sono memorizzate nella struttura di directory. Per ogni file possono essere necessari da 16 fino a più di 1000 byte per memorizzare i suoi attributi. Se i file sono molti la struttura di directory può occupare molti megabyte.

Operazioni sui file.

Il sistema operativo mette a disposizione delle system call che permettono di eseguire delle operazioni sui file, vediamo quali sono:

- Creazione di un file. Per creare un file sono necessari due passaggi: trovare lo spazio necessario nel file system, aggiungere un elemento alla directory che contenga i suoi attributi.
- Scrittura di un file. Per scrivere su un file è necessaria una system call che specifichi il nome del file e le informazioni che si vogliono scrivere su di esso. Un puntatore di scrittura punta alla locazione nel file nella quale si deve effettuare la successiva operazione di scrittura, dopo ogni singola operazione di scrittura il puntatore viene aggiornato.
- Lettura di un file. Per leggere da un file serve una system call che specifichi il nome del file da leggere e la posizione del blocco successivo del file. Serve un puntatore di lettura che punta alla posizione dove deve aver luogo la successiva operazione di lettura. Il puntatore

viene aggiornato dopo ogni singola operazione di lettura. Spesso si utilizza un solo puntatore sia per la lettura che per la scrittura.

- Riposizionamento in un file. Consiste nel riposizionamento del puntatore all'interno del file.
- Cancellazione di un file. Avviene in 3 passaggi, prima si cerca l'elemento della directory associato al file in questione, quindi si rilascia lo spazio associato al file ed infine si elimina l'elemento della directory relativo al file da cancellare.
- Troncamento di un file. Consiste nella possibilità di mantenere un file vuoto senza doverlo ricreare, ovvero si cancella il contenuto del file ma non il file stesso il file troncato avrà tutti gli attributi del file originario tranne la dimensione che sarà, nel file troncato, nulla.

Queste sono le operazioni di base sui file, ma possono rendersi necessarie altre operazioni come: rinomina di un file, aggiunta di dati a fine file, lettura degli attributi di un file, modifica degli attributi di un file, ecc..

Tipi di file.

Allo scopo di impedire operazioni inutili come la visione o la stampa di un file binario, oppure il tentativo di esecuzione di un file di testo, il S.O. deve riconoscere i diversi tipi di file.
Un metodo utilizzato di frequente per informare sia l'utente che il S.O. del tipo di file che si sta considerando è quello di introdurre informazioni relative alla natura del file nel nome stesso.
Il nome del file viene così suddiviso in due parti, un nome ed un'estensione, di solito separata da un punto. La tabella sottostante illustra alcuni dei più comuni tipi di file e relative estensioni.

Tipo di file	**Usuale estensione**	**Funzione**
Eseguibile	exe, com, bin, o nessuna	Programma in linguaggio macchina, eseguibile
Oggetto	obj, o	Compilato, in linguaggio macchina ma, non sottoposto a link.
Codice sorgente	c, p, pas, cpp, cc, f77, cbl, java	Codice sorgente in vari linguaggi di programmazione.
Batch	bat, sh	Script della shell
Testo	txt, doc	Testi, documenti.
Elaboratore di testi	tex, rrf, wp	Vari formati per elaboratori di testi.
Libreria	lib, a, dll, so	Librerie di routine per programmatori
Stampa o visualizzazione	ps, dvi, gif, pdf	File ASCII o binari in formato per stampa o visualizzazione.
Archivio	arc, zip, tar, tgz, rpm	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione e memorizzazione.

I diversi tipi di accesso ai file.

Accesso Sequenziale.
Rappresenta il più semplice metodo di accesso ai dati di un file. Secondo questo tipo di accesso i record vengono letti uno dopo l'altro secondo l'ordine con cui sono scritti nel file.
Per quanto riguarda la scrittura questa può avvenire solo alla fine del file.
Questo tipo di accesso si rifà alla struttura fisica del nastro magnetico che è intrinsecamente sequenziale. I compilatori utilizzano questo metodo di accesso ai file.

Accesso Diretto.
Il metodo di accesso diretto permette di leggere e scrivere i record senza un ordine particolare. Il file viene considerato come una sequenza numerata di record e quindi è possibile accedere sia in lettura sia in scrittura ad un record particolare.
I database utilizzano questo tipo di accesso ai file.
Ovviamente i file ad accesso diretto sono molto più flessibili dei file ad accesso sequenziale infatti consentono anche l'accesso sequenziale oltre che quello diretto.

Struttura di directory.

Il file system di un computer può contenere migliaia di file su centinaia di gigabyte di spazio su disco. Per gestire tutti questi dati in modo conveniente è necessario avere una buona organizzazione dello spazio nella memoria di massa.
Tale organizzazione si attua in due parti:

- la suddivisione del file system in partizioni, ogni partizione rappresenta una sorta di disco virtuale, sono considerate dal sistema operativo come se fossero dei dischi fisicamente diversi.
- la suddivisione delle partizioni in directory.

Ogni partizione contiene le informazioni sui file in essa contenuti. Tali informazioni sono mantenute negli elementi della directory del dispositivo o tabella dei contenuti del volume.
La directory registra tutte le informazioni (gli attributi) di tutti i file della partizione.
La directory può essere considerata come una tabella di simboli che traduce i nomi dei file negli elementi in essa contenuti. Questa visione è più corretta di quella più intuitiva che vede la directory come un contenitore di file.

Su una directory possono essere eseguite diverse operazioni:

- Ricerca di un file. Poiché la directory è una tabella la ricerca di un file consiste nella scansione della tabella dei contenuti di volume.
- Creazione di un file. La creazione di un file consiste nella aggiunta di un elemento alla tabella suddetta.
- Cancellazione di un file. La cancellazione consiste nella rimozione dell'elemento corrispondente dalla tabella.
- Listare una directory. Si possono elencare tutti gli elementi della tabella.

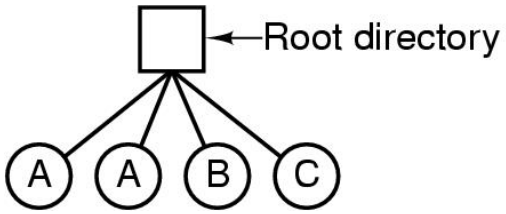
- Rinomina di un file. Consiste nella possibilità di associare ad un insieme di dati su disco un nome simbolico diverso nella tabella.
- Attraversamento del file system. Consiste nella possibilità di accedere a tutti i file e tutte le directory.

Le diverse tipologie delle strutture di directory.

Directory a livello singolo.

La struttura più semplice per una directory è quella a livello singolo. Tutti i file sono contenuti nella stessa directory. I limiti di una directory a livello singolo si manifestano in modo evidente se il numero dei file è grande oppure se deve servire più utenti.

Infatti i nomi dei file devono essere unici e se si presentano le condizioni appena indicate può risultare molto difficile soddisfare tale richiesta. Inoltre la confusione determinata dall'accesso di molti utenti alla stessa directory determina senz'altro un alto grado di confusione.



Directory a due livelli.

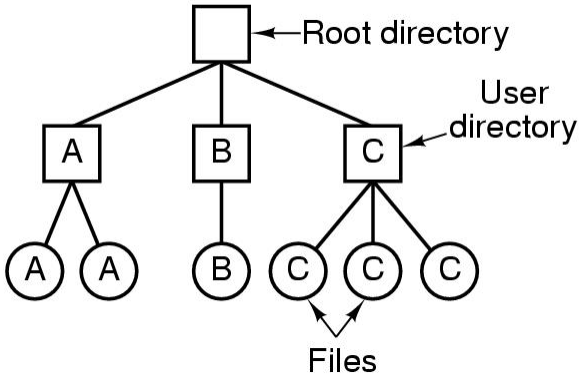
La soluzione ai problemi derivanti da una struttura a livello singolo vengono risolti introducendo una directory separata per ogni utente.

In tal caso ogni utente dispone della propria file directory UFD (User File Directory). Tutte le UFD hanno una struttura simile. Esiste la MFD (Master File Directory) che contiene la lista di tutti gli utenti e dei rispettivi UFD. Quando un'utente effettua il login viene effettuata la ricerca in MFD e se il login ha successo l'utente si trova nella propria UFD.

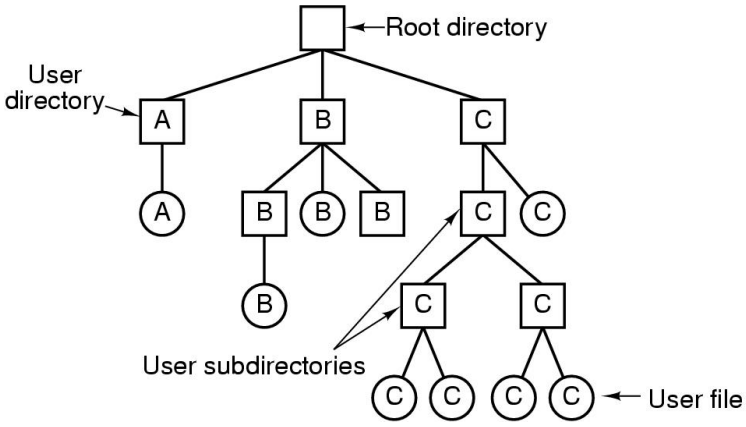
Spetta all'amministratore di sistema la manutenzione della MFD che prevede la creazione di un nuovo elemento nel caso di concessione di un nuovo account ad un nuovo utente o la eventuale cancellazione di un elemento già presente.

La directory a due livelli risolve il problema della collisione dei nomi di file appartenenti ad utenti diversi ma isola gli utenti, questo rappresenta un problema se essi lavorano ad uno stesso progetto.

Come si risolve il problema? Esistono diverse possibilità: se l'accesso è autorizzato un utente B può accedere ad un file di un'altro utente A specificando il percorso, ad esempio nella forma /utenteA/file1, o se richiede l'accesso ad un file appartenente al proprio UFD /utenteB/file1. Ma in alcuni sistemi l'accesso alla UFD di un diverso utente non è consentito, quindi per condividere un file esistono due possibilità o lo si ricopia in ogni UFD di ogni utente interessato a quel file, oppure si crea una directory utente particolare che contiene i file ai quali si richiede l'accesso da parte di diversi utenti; questa soluzione viene di solito adottata quando ci sono file da condividere tra tutti gli utenti del sistema come il caso dei file di sistema, come compilatori, routine di utilità ecc.



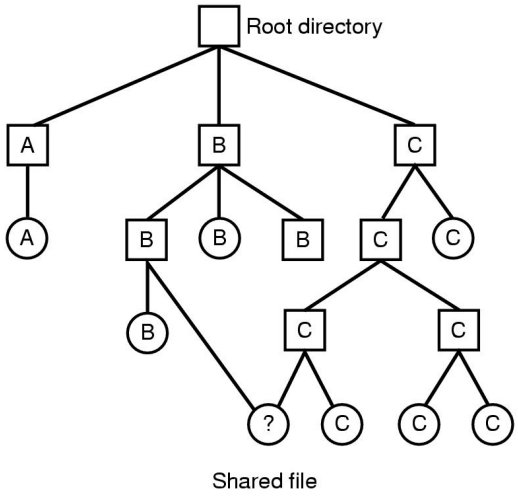
Directory con struttura ad albero.
Secondo questa struttura gli utenti possono creare le loro directory. Quindi una directory contiene sia sottodirectory che file. L'albero ha una directory radice (root directory). Ogni file del sistema ha un unico path name o nome di percorso che parte dalla radice, passa attraverso tutte le sottodirectory ed arriva al file specifico. Ogni utente ha una directory corrente che contiene i file di uso corrente per quell'utente.
Nelle directory con strutture ad albero devono esistere delle sistem call per gestire la creazione e la cancellazione delle directory. In particolare la cancellazione merita alcune considerazioni. Infatti una directory può essere vuota ed in tal caso la cancellazione non presenta difficoltà, basta cancellare il termine che la rappresenta dalla directory che la contiene. Ma se non è vuota esistono diversi approcci: il sistema MS-DOS non permette la cancellazione di una directory non vuota, se la si vuole cancellare bisogna prima cancellare i file e le sottodirectory che contiene in modo ricorsivo; questo approccio risponde a criteri di sicurezza ma può richiedere molto lavoro.
UNIX invece permette la cancellazione di una directory non vuota con un solo comando ad esempio **rm –r nome_dir** il flag r attiva infatti l'opzione di ricorsività. **NB. Questo comando è pericoloso!!!**



Directory con struttura a grafo aciclico.
La struttura ad albero proibisce la **condivisione** di file o directory. Bisogna chiarire cosa si intende per condivisione. Si supponga che due programmatori stiano lavorando ad uno stesso progetto ed

abbiano la stessa responsabilità sul progetto, entrambi vogliono avere l'accesso ai file del progetto, che possono essere memorizzati in una sottodirectory, vogliono inoltre che la sottodirectory specificata si trovi nella propria directory, si parla allora di directory condivisa.
Bisogna tenere ben presente che avere un file condiviso è ben diverso da avere due copie dello stesso file, infatti se un programmatore modifica un file che è copia del file originale l'altro file non rispecchierà tali modifiche. Se invece il file è condiviso esiste un solo file effettivo e le modifiche apportate da un programmatore ad un file sono immediatamente visibili anche all'altro.
La struttura a grafo aciclico permette la condivisione dei file e delle directory. Cioè uno stesso file o una directory possono trovarsi in due directory diverse.

UNIX risolve questa situazione con i link simbolici. Un link simbolico è un puntatore ad un file o per essere più precisi è un puntatore ad un path name. Un link non è un file realmente esistente ma un rimando ad un file realmente esistente.
La possibilità di creare dei link introduce alcuni problemi. Infatti cosa accade se viene cancellato il file a cui il link si riferisce? Il link continua ad esistere ma punta ad un file che non esiste più. Questo può creare dei problemi, per esempio se venisse creato un nuovo file con lo stesso nome il link punterebbe a questo senza preoccuparsi minimamente che esso il link era stato creato in riferimento ad un'altro file. Una strategia che risolve questo problema consiste nel conservare il file fino a che non sono stati rimossi tutti i link che puntano ad esso, questo può essere fatto ricorrendo ad un contatore di riferimenti, ogni cancellazione di un link decrementa di 1 il contatore dei riferimenti, quando esso diventa 0 il file che non contiene più link può essere cancellato, (modo utilizzato per implementare gli hard link in UNIX). Una struttura di directory con la possibilità della condivisione dei file è costituita dalle directory con struttura a grafo aciclico.



Protezione.

Con il termine protezione si intende il controllo dell'accesso ai file. L'accesso ai file ed alle directory deve avvenire soltanto nei casi in cui esiste una autorizzazione.

In generale è possibile controllare l'accesso alle seguenti diverse operazioni:

- Lettura del file.
- Scrittura del file.
- Esecuzione.
- Aggiunta.
- Cancellazione.
- Lista.

Il controllo dell'accesso ai file avviene in base all'identità dell'utente. Uno schema generale prevede la possibilità di associare una lista d'accesso ad ogni file o directory, questa lista deve contenere il nome degli utenti ed il tipo di accesso consentito ad ogni utente.
Le liste di accesso però sono in generale piuttosto lunghe e difficili da creare e gestire.
Per risolvere il problema si ricorre ad una versione condensata delle liste. Tale versione condensata si basa sul raggruppamento degli utenti in tre classi distinte:

- **Proprietario**. L'utente che ha creato il file.
- **Gruppo**. Un gruppo di utenti che richiede l'accesso agli stessi file perché ad esempio lavora ad uno stesso progetto.
- **Universo**. Tutti gli altri utenti.

Ora per definire l'accesso bastano tre campi di bit. Per esempio UNIX definisce 3 campi di 3 bit ciascuno rwx, r controlla l'accesso alla lettura, w controlla l'accesso alla scrittura e x controlla l'accesso alla esecuzione del file.