INAIL
ISTITUTO NAZIONALE PER L'ASSICURAZIONE CONTRO GLI INFORTUNI SUL LAVORO

INAIL FORMAZIONE

Materiale Didattico

# SISTEMI OPERATIVI

CONSULENZA PER L'INNOVAZIONE TECNOLOGICA

SERVIZIO FORMAZIONE


| | |
|---|---|
| Redattori | Augusto Albo |
| | |
| | |

# INDICE

## 0 INTRODUZIONE

Un sistema operativo è un insieme di programmi che operano sull'hardware di un calcolatore con l'obiettivo di rendere più semplice ed efficace lo sviluppo di programmi e di realizzare politiche di gestione delle risorse hardware.

Il sistema operativo agisce da intermediario fra l'utente e la struttura fisica del calcolatore.

I compiti specifici del sistema operativo dipendono dall'obiettivo del sistema di calcolo. Le funzioni svolte dal sistema operativo sono molteplici e strettamente legate all'architettura del calcolatore sul quale è eseguito.

# 1 DEFINIZIONE DI SISTEMA OPERATIVO

Il sistema operativo è un programma o un insieme di programmi che rendono agevole l'utilizzo del calcolatore.

Un sistema di calcolo si può suddividere in quattro componenti: i dispositivi fisici, il sistema operativo, i programmi d'applicazione e gli utenti (Figura 1.1).

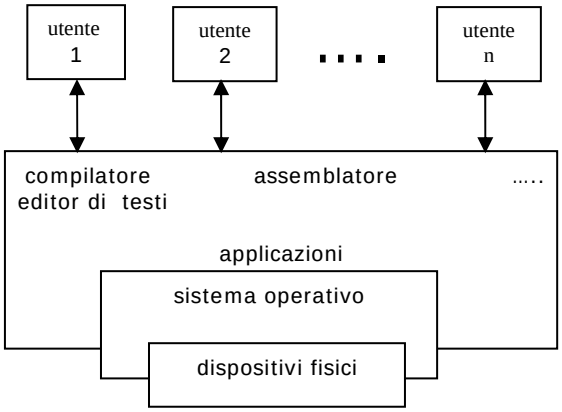



**Figura 1.1** Componenti di un sistema di calcolo.

La parte hardware del calcolatore è composta da tre moduli fondamentali (Figura 1.2), i dispositivi fisici (CPU, memoria, dispositivi di I/O), che comunicano tramite linee di collegamento (bus dati).

I programmi di applicazione vengono utilizzati per risolvere i problemi computazionali degli utenti (intesi non solo come persone ma anche i calcolatori, le macchine, i programmi ecc. possono essere utenti). Ogni utente ha esigenze diverse quindi ognuno farà utilizzo di programmi diversi.

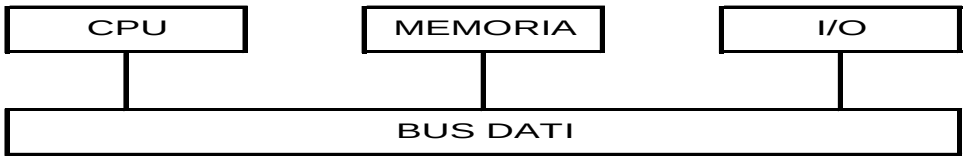



**Figura 1.2** I dispositivi fisici di un calcolatore.

I programmi utilizzano le risorse hardware per svolgere le funzioni che sono state codificate in esse per raggiungere l'obiettivo dell'utente.

Nasce il problema dell'utilizzo delle risorse quando si eseguono sulla stessa macchina più programmi. Per esempio se esiste una sola CPU non possono essere eseguiti due programmi contemporaneamente. Stesso

discorso vale per lo spazio di memoria limitato che quindi deve essere allocato ai programmi uno alla volta. Anche i dispositivi di I/O e tutte le altre risorse devono essere utilizzate da un programma alla volta.

Per evitare problemi che nascono con lo sviluppo della multiprogrammazione su un calcolatore, occorre implementare un insieme di funzioni che controllano il corretto svolgimento dei programmi. Utilizziamo quindi un SISTEMA OPERATIVO, un programma che rende *conveniente* l'uso delle risorse hardware della macchina, allocando in modo corretto le stesse risorse ai diversi programmi applicativi che sono in esecuzione. Il sistema operativo non produce risultati visibili al programmatore, ma si limita a svolgere funzioni di supporto.

Esso rende *efficiente* l'utilizzo delle risorse: alloca e coordina le risorse agendo come un gestore di tali risorse, in modo da svolgere i compiti in modo più efficiente possibile.

Quindi i due scopi principali del sistema operativo sono:

- *CONVENIENZA*
- *EFFICIENZA*

Questi due requisiti possono essere in alcuni casi contrastanti. Ci sono sistemi progettati per svolgere operazioni diverse: alcuni solo per soddisfare il requisito della convenienza (sistemi operativi progettati per i normali personal computer utilizzati da un unico utente magari non esperto, richiedono solo la facilità d'uso, realizzata mediante un'interfaccia grafica), altri per l'efficienza (per esempio i mainframe per elaborare dati utilizzati da utenti esperti, sono progettati in modo da massimizzare l'utilizzo delle risorse).

Il sistema operativo deve quindi realizzare politiche di gestione delle risorse hardware al fine di:

1. regolamentare l'impiego delle risorse evitando conflitti di accesso;
2. scegliere i criteri con cui assegnare una risorsa a fronte di più richieste contemporanee, dovuto alla disponibilità limitata di risorse;
3. nascondere all'utente i dettagli hardware legati al particolare dispositivo.

# 2 EVOLUZIONE DEI SISTEMI OPERATIVI

## 2.1 SISTEMI BATCH

Nei sistemi batch i programmi (*job*) non prevedono interattività con l'utente ma vengono suddivisi a LOTTI (BATCH) aventi caratteristiche simili, quindi letti dal calcolatore ed eseguiti. Se il programma non funziona, in caso di terminazione anomala del programma stesso, vengono stampati oltre ai risultati anche una mappa dei registri di memoria.

I sistemi di tipo batch prevedono l'esecuzione di un programma alla volta sempre residente nella memoria (Figura 2.1): il calcolatore legge un programma, lo carica in memoria e lo esegue fino alla fine o fino all'errore, poi stampa il risultato. Quindi passa ad un nuovo programma: lettura, memorizzazione, esecuzione stampa. E così via per tutti i programmi.


**Figura 2.1** Configurazione della memoria per un sistema batch.

Il sistema operativo utilizzato è molto semplice: non c'è interazione tra programma e utente, non si devono prendere decisioni su come allocare le risorse, né scegliere il successivo programma da eseguire.

Questi sistemi sono poco efficienti. La CPU è spesso inattiva poiché i dispositivi meccanici di I/O sono più lenti (Figura 2.2). Inoltre le risorse in ingresso e in uscita non sono sfruttate al massimo: in fase di lettura la CPU e le risorse di uscita non sono utilizzate; nella fase di stampa del risultato oltre alla CPU non sono utilizzate le risorse di ingresso; mentre in fase di elaborazione si utilizza solo la CPU.

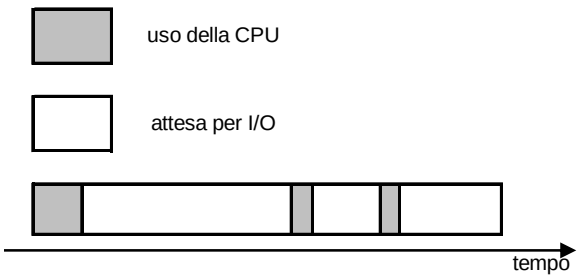



**Figura 2.2** Utilizzo della CPU nei sistemi batch.

Conseguentemente le risorse in ingresso possono essere utilizzate solo nella fase di lettura iniziale perché l'utente non può interagire con il programma durante la sua elaborazione, quindi tutti i dati devono essere inseriti all'inizio. Un discorso analogo può essere fatto per le risorse in uscita. L'inefficienza di questo tipo di sistema operativo è dovuta alla mancanza dell'utilizzo parallelo delle risorse che viene realizzata nei sistemi sviluppati successivamente.

### 2.2 SISTEMI MULTIPROGRAMMATI

Il problema dell'utilizzo inefficiente delle risorse viene pienamente superato con l'introduzione dei sistemi multiprogrammati.

La *multiprogrammazione* consente di aumentare l'utilizzo della CPU organizzando i lavori in modo tale da mantenerla in continua attività. Con la multiprogrammazione si è introdotto lo schema di gestione della memoria per organizzare i programmi che devono andare in esecuzione.

In questi sistemi sono presenti contemporaneamente in memoria centrale diversi programmi (Figura 2.3), uno dei quali viene selezionato e mandato in esecuzione. Il processo creato mantiene il controllo della CPU finché non si arresta perché termina o perché in attesa di qualche evento come il completamento di una operazione di I/O. A questo punto la CPU invece di rimanere inattiva viene assegnata ad un altro processo presente in memoria. Se il primo processo non era terminato, quando torna pronto riprende il controllo della CPU continuando la sua esecuzione.

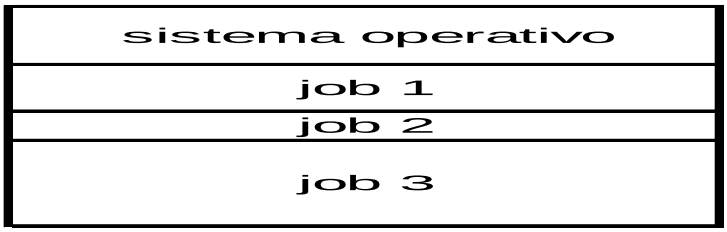



**Figura 2.3** Configurazione della memoria per un sistema multiprogrammato.

## 3 STRUTTURA DEI SISTEMI OPERATIVI

Il compito del sistema operativo è quello di implementare funzioni per gestire tutti i dispositivi che compongono il calcolatore e per fornire ai programmi utente un'interfaccia semplificata con l'hardware. Per ottimizzare le prestazioni del calcolatore, il sistema operativo viene suddiviso in diverse componenti, ognuna delle quali ha il compito di realizzare un sottoinsieme delle funzioni sopra citate.

Ogni sistema svolge funzioni diverse, a seconda dell'ambiente in cui viene implementato. Inoltre anche la struttura interna che li caratterizza può essere molto diversa da sistema a sistema.

### 3.1 FUNZIONI SVOLTE DAL SISTEMA OPERATIVO

Il sistema operativo offre un ambiente in cui eseguire i programmi e fornire servizi ai programmi stessi ed ai loro utenti. Il sistema operativo svolge quindi un insieme di funzioni che variano secondo l'architettura del calcolatore.

In generale si possono individuare due classi di funzioni:

1. *Funzioni visibili all'utente.* Ne fanno parte tutte le funzioni che realizzano il criterio di convenienza. In particolare il sistema deve:

   - Rendere possibile l'esecuzione di programmi da parte dell'utente;

   - Rendere disponibili i dispositivi di I/O. Svolgere il coordinamento delle operazioni di I/O poiché le richieste arrivano da più programmi nello stesso momento;

   - Gestire il file system. I programmi manipolano i dati sui files (leggono, elaborano e stampano o memorizzano i dati): il sistema operativo deve rendere possibile il loro utilizzo attraverso la gestione dei file;

   - Gestire la comunicazione tra più programmi eseguiti su una stessa macchina o, nel caso di connessione di rete, su macchine diverse. La comunicazione si può realizzare tramite una memoria condivisa o attraverso lo scambio di messaggi. In questo caso il sistema operativo trasferisce pacchetti d'informazione tra i vari processi.

   - Gestire gli errori temporanei o permanenti nella CPU o nei dispositivi di memoria, che possono causare l'interruzione della comunicazione o produrre risultati errati. Il sistema deve garantire un'elaborazione corretta e coerente;

2. *Funzioni non visibili all'utente*. Ne fanno parte tutte le funzioni che realizzano il criterio di efficienza. In particolare il sistema deve:

   - Allocare le risorse in modo efficiente in modo da massimizzare l'obiettivo che il sistema operativo si prefigge. Per l'assegnazione della CPU il sistema fa uso di algoritmi di scheduling che tengono conto della velocità della CPU, dei processi da eseguire e di altri fattori. Le altre risorse vengono assegnate ai vari processi secondo

procedure che utilizzano tabelle per la memorizzazione degli utenti e delle risorse che utilizzano;

• Contabilizzare l'uso delle risorse. Il sistema deve tenere traccia delle risorse che i programmi utilizzano a scopi commerciali, statistici o per implementare sistemi di sicurezza;

• Realizzare funzioni per la gestione della protezione e della sicurezza. Nei sistemi multiprogrammati il sistema operativo deve evitare l'accesso di utenti alle risorse di altri utenti. La protezione assicura che l'accesso alle risorse del sistema sia controllato.

### 3.2 SYSTEM CALL

Le system call costituiscono l'interfaccia tra un processo e il sistema operativo. Tutte le funzioni svolte dal sistema operativo devono essere disponibili ai programmi scritti dagli utenti attraverso chiamate di sistema. Esse sono chiamate di funzioni in cui la funzione viene svolta dal sistema operativo (in assembler o linguaggi ad alto livello).

Le chiamate di sistema si possono classificare in cinque categorie principali:

• Controllo dei processi. Utilizzate per svolgere operazioni sui processi (creare, fermare, terminare e eseguire processi), per caricare il sistema in memoria, per attendere un certo evento o un intervallo di tempo;

• Gestione dei file. Su un sistema di elaborazione esistono diversi supporti di memorizzazione con caratteristiche e modalità di memorizzazione diverse. L'utente dovrebbe tenerne presente e considerare tutti i dettagli. Per questo motivo il sistema operativo realizza l'astrazione dei file introducendo l'unità di memorizzazione logica. I file sono poi mappati nei dispositivi di memoria di massa, così che l'utente si preoccupa solo di scrivere i dati su un file. Il sistema operativo mette a disposizione dell'utente un certo numero di chiamate di sistema per la gestione dei file;

• Gestione dei dispositivi di I/O. Come per i file, i dispositivi sono numerosi e ognuno con proprie caratteristiche interne. Per semplificare il lavoro dell'utente, il sistema mette a disposizione dei file evitando che l'utente colloqui direttamente con i dispositivi. Il sistema operativo offre all'utente una interfaccia semplificata;

• Gestione delle informazioni. System call poco significative per l'utente (riguardano l'impostazione della data e dell'ora attuale; numero degli utenti collegati);

- Comunicazione con altri processi in esecuzione sulla stessa macchina o su macchine diverse. Il sistema operativo gestisce la comunicazione tra processi: fornisce system call per l'apertura/ chiusura dei collegamenti su un altro processo, imposta i parametri per i collegamenti, gestisce l'invio/ricezione delle informazioni tra processi.

Spesso, l'insieme delle system call esportate dal sistema operativo risulta essere troppo di basso livello per essere utilizzabile in maniera semplice dall'utente. Per questo motivo, vengono talvolta fornite delle librerie che implementano funzionalità di più alto livello basandosi sulle chiamate di sistema.

## 3.3 COMPONENTI DI UN SISTEMA OPERATIVO

Il sistema operativo può essere suddiviso in moduli, ognuno dei quali fornisce una interfaccia e realizza funzioni ben precise. Le componenti fanno riferimento alle funzioni viste sopra.

### 3.3.1 GESTIONE DEI PROCESSI

Il gestore dei processi si occupa di tutte le operazioni che coinvolgono i processi.
Un processo è un programma in esecuzione. Per svolgere i suoi compiti necessita di alcune risorse (CPU, memoria, file dispositivi di I/O) che possono essere attribuite in diversi momenti, alla creazione o durante la sua esecuzione. Il sistema operativo si occupa di mandare un programma in esecuzione e allocare le risorse utilizzate.
In particolare il sistema operativo deve gestire l'allocazione della CPU in base alla politica di schedulazione adottata: deve decidere quale processo mandare in esecuzione, quando revocargli la CPU e quale sarà il prossimo processo a cui assegnare la CPU.

Il S.O. inoltre:

- crea e cancella i processi utente e di sistema;
- sospende e ripristina i processi;
- fornisce meccanismi per la sincronizzazione dei processi;
- fornisce meccanismi per la comunicazione tra i processi;
- fornisce meccanismi per la gestione dei deadlock.

### 3.3.2 GESTIONE DELLA MEMORIA CENTRALE

La memoria è condivisa dalla CPU e da alcuni dispositivi di I/O. Prima di eseguire un programma deve essergli allocata la memoria: il programma

deve essere associato a indirizzi assoluti e caricato in memoria; tali indirizzi vengono generati dalla CPU.
Se ci sono più programmi che richiedono l'elaborazione occorre scegliere quale mandare in esecuzione tenendo conto della memoria totale disponibile e di quella richiesta: se un programma occupa più memoria di quella disponibile si attua il processo di *virtualizzazione*.

Il sistema operativo deve quindi seguire degli schemi di gestione della memoria che variano in funzione dell'architettura del sistema. In particolare deve:

- tener traccia della memoria disponibile;
- selezionare i processi da caricare in memoria;
- assegnare e revocare la memoria ai processi;
- registrare per ogni processo la memoria allocata.

### 3.3.3 GESTIONE DEI FILE

I dispositivi di memorizzazione secondaria sono molteplici ed ognuno ha proprie caratteristiche (per esempio velocità di trasferimento dati, metodo d'accesso) e una organizzazione fisica diversa. Il sistema operativo si occupa di semplificare i compiti dell'utente e traduce le sue richieste per i dispositivi fisici.

Il sistema realizza effettivamente l'astrazione dei file, fornendo una visione logica astratta e uniforme delle informazioni memorizzate (i file). L'astrazione riguarda le particolari caratteristiche fisiche dei dispositivi.

Riguardo alla gestione dei file è responsabile di:

- creare e cancellare file e directory;
- fornire all'utente le funzioni fondamentali per la manipolazione di file e directory;
- creare una associazione tra i file e i corrispondenti dispositivi di memoria secondaria;
- mantenere un back up dei file su dispositivi di memorizzazione stabili, non volatili.

### 3.3.4 GESTIONE DELL' I/O

Il sistema operativo deve nascondere all'utente le caratteristiche degli specifici dispositivi hardware. Solo il driver del dispositivo software relativo al dispositivo fisico conosce le caratteristiche dello stesso.

Il sistema operativo realizza un'astrazione dei dispositivi di I/O definendo un'interfaccia comune che sfrutta i servizi dei driver (*sottosistema I/O*), che è composto da:

- componente per la gestione della memoria di un buffering, latching e spooling;
- un'interfaccia generale per i driver dei dispositivi;
- i driver per gli specifici dispositivi.

### 3.3.5 GESTIONE DELLA MEMORIA SECONDARIA

Prima di eseguire un programma deve essergli allocata la memoria centrale. Poiché i programmi eseguiti su un calcolatore sono molti, anche la richiesta di memoria sarà elevata. Per questo motivo i sistemi devono disporre di una memoria secondaria a sostegno della memoria centrale.

La maggior parte dei moderni sistemi di calcolo impiega i dischi come principale mezzo di memorizzazione secondaria, sia per i programmi sia per i dati a cui accedono.

Il gestore della memoria secondaria è utile perché tutti i sistemi hanno i supporti di memorizzazione secondaria non volatili e le prestazioni del sistema operativo spesso dipendono da questi sistemi di memorizzazione.

Questa componente si occupa di:

- gestire lo spazio libero del disco (per memorizzare da file a disco);
- allocazione e revoca dello spazio di memoria ai vari file;
- realizzare lo scheduling del disco a fronte di richieste multiple di utilizzo del disco.

### 3.3.6 GESTIONE DEL NETWORKING

In un sistema distribuito le unità di elaborazione sono collegate da una rete di comunicazione. I calcolatori che ne fanno parte condividono le risorse.

Quindi il sistema operativo deve gestire l'accesso a queste ultime e gestire la comunicazione con processi remoti. Per far questo è dotato di un dispositivo di interfaccia con la rete (*dispositivi di I/O*) e sviluppa la parte dei protocolli di rete nel sistema operativo stesso.

Normalmente i sistemi operativi generalizzano l'accesso alla rete come una forma d'accesso ai file, dove i particolari riguardanti l'interconnessione sono contenuti nel driver del dispositivo d'interfaccia della rete stessa.

### 3.3.7 GESTIONE DEL MECCANISMO DI PROTEZIONE

Se un sistema di calcolo ha più utenti e consente che più processi siano eseguiti in modo concorrente, i diversi processi devono essere protetti dall'attività di altri processi e gli utenti possono accedere solo alle risorse di cui dispongono i permessi.

Per questo motivo esistono meccanismi che assicurano che i file, i segmenti della memoria, la CPU e altre risorse possano essere controllate solo dai processi che hanno ricevuto l'autorizzazione dal sistema operativo. E' compito del sistema stesso amministrare la sicurezza del sistema in modo opportuno.

La protezione è definita da ogni meccanismo che controlla l'accesso da parte di programmi, processi o utenti alle risorse di un sistema di calcolo. Questo meccanismo deve fornire i mezzi che specifichino quali controlli si debbano eseguire e i mezzi per effettuarli.

Il sistema operativo deve realizzare il meccanismo di protezione considerando la sicurezza sotto due aspetti differenti: si parla infatti di *sicurezza di sistema*, intesa come tecnica per prevenire intrusi all'interno di un sistema, e *sicurezza di rete* (derivante dal fatto che i calcolatori sono connessi in rete) intesa come insieme di tecniche per garantire il controllo degli accessi provenienti dalla rete e per garantire la sicurezza dei dati che viaggiano sulla rete.

### 3.3.8 INTERPRETE DEI COMANDI

L'interprete dei comandi è la parte con cui il sistema operativo si interfaccia direttamente con l'utente. È il programma di sistema più importante, la cui funzione principale consiste nel ricevere i comandi impartiti dagli utenti ed eseguirli.

A seconda dell'architettura del sistema l'interprete viene implementato nel nucleo con tutte le altre funzioni o fornito al di fuori del nucleo. In questo ultimo caso l'interprete è come una tipica applicazione di sistema, quindi modificabile.

L'interprete realizza un piccolo numero di comandi, gli altri sono in più realizzati come funzioni: il sistema operativo cerca il programma (i programmi sono organizzati all'interno di file) contiene il comando richiesto dall'utente e lo manda in esecuzione. In questa ottica l'utente può aggiungere nuovi comandi.

Esistono interpreti di diverso tipo: alcuni efficienti ma con interfaccia di basso livello (linux); altri hanno interpreti più facili da utilizzare, realizzati mediante l'utilizzo di una interfaccia grafica.

# 4 MEMORIA CENTRALE

La parte del sistema operativo che gestisce la memoria è il gestore di memoria e si occupa di tutti i problemi legati all'assegnazione della memoria ai diversi processi che ne fanno richiesta per portare a termine la loro esecuzione.

Il gestore della memoria deve quindi tenere traccia di quali parti di memoria sono in uso e quali sono libere, deve allocare / deallocare memoria ai processi in modo opportuno. Infine, quando la memoria principale non è sufficiente a contenere tutti i processi, il gestore deve provvedere a fornire una memoria di supporto tramite un meccanismo di virtualizzazione della memoria.

Un programma per essere eseguito deve avere un'area di memoria a disposizione. Il sistema operativo deve implementare dei meccanismi che allochino spazi di memoria ai processi pronti ad andare in esecuzione e che revochino lo spazio ai processi che terminano.

Poiché la memoria principale non è sempre sufficiente a mantenere tutti i processi correntemente attivi, il sistema utilizza una memoria di supporto, generalmente i dischi, sulla quale risiedono i processi in eccesso e che verranno introdotti successivamente in memoria principale.

Se la politica di schedulazione è basata sulla priorità, un processo in esecuzione può essere scaricato dalla memoria prima che sia terminato. Questo accade se si presenta un processo con priorità maggiore di quello attualmente in esecuzione.

In alternativa a questo metodo esiste la strategia della memoria virtuale che consente anche ai programmi *parzialmente* caricati in memoria principale di essere eseguiti.

## 4.1 ASSOCIAZIONE DEGLI INDIRIZZI

Il sistema deve realizzare una corrispondenza tra istruzioni del programma e indirizzi di memoria. Tale associazione può essere stabilita in momenti diversi. Se il sistema sa già in fase di compilazione dove risiederà il programma genera un indirizzamento assoluto del programma. Il codice così generato non è rilocabile, cioè se ad un dato momento l'indirizzo del programma deve cambiare occorre ricompilare il programma.

Quando l'associazione è eseguita in fase di caricamento il codice generato dal compilatore è rilocabile. Quindi se l'indirizzo iniziale cambia si ricarica il programma in memoria. Le traduzioni opportune sono effettuate dal loader.

## 4.2 AVVICENDAMENTO DEI PROCESSI

Esistono due meccanismi con i quali il sistema sostituisce i processi in memoria.
Quello più semplice è definito avvicendamento dei processo, o swapping; il sistema carica *interamente* in memoria il processo che richiede l'esecuzione e solo alla fine viene spostato su disco. L'operazione mediante la quale il sistema trasferisce il processo dal disco alla memoria è definita *swap-in*, l'operazione contraria è definita *swap-out*.

## 4.3 SISTEMA DI BASE PER LA GESTIONE DELLA MEMORIA

Lo schema di gestione della memoria varia a seconda del tipo di sistema: nei sistemi monoprogrammati la gestione risulta semplice, ma tali sistemi sono ormai considerati obsoleti.

Nei sistemi multiprogrammati la gestione risulta più complicata, ma ottimizza le prestazioni del sistema e incrementa l'utilizzo della CPU.

### 4.3.1 Gestore della memoria nei sistemi monoprogrammati

Nei sistemi monoprogrammati è prevista l'esecuzione di un processo alla volta. La memoria disponibile è condivisa tra il sistema operativo e il programma attualmente in esecuzione. Esistono tre diverse rappresentazioni (Figura 4.3):

- una prevede che il sistema operativo si trovi nella parte bassa della RAM, la memoria ad accesso casuale (Figura 4.3(a)); questo modello era utilizzato principalmente sui mainframe e sui minicomputer;
- una variante lo rappresenta nella parte alta della memoria ROM, la memoria a sola lettura (Figura 4.3(b)); questo modello è utilizzato nei sistemi embedded;
- la terza alternativa prevede i gestori dei dispositivi di I/O nella parte alta della memoria ROM e il resto del sistema operativo nella parte bassa della memoria RAM (Figura 4.3(c)); questo modello era utilizzato nei vecchi personal computer (per esempio nel MS-DOS).

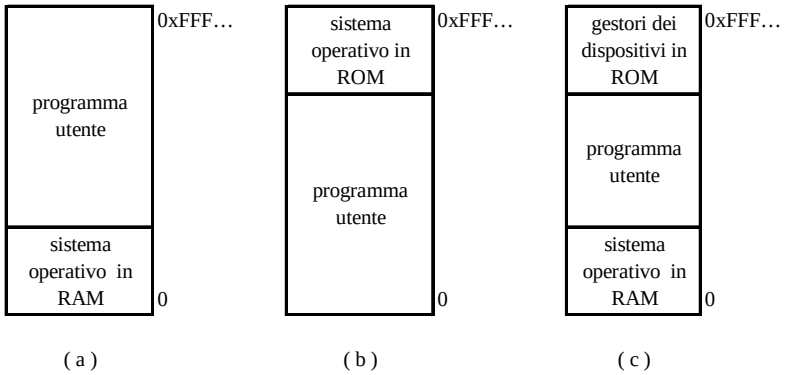



**Figura 4.3** Organizzazione della memoria nei sistemi monoprogrammati.

Nei sistemi monoprogrammati può essere in esecuzione un solo processo. Il sistema operativo carica il programma dal disco in memoria centrale e lo esegue. Solo al termine dell'esecuzione un nuovo processo può sovrapporsi al primo in memoria.

### 4.3.2 Gestore della memoria nei sistemi multiprogrammati

Nei sistemi multiprogrammati il sistema operativo consente a diversi processi di girare contemporaneamente, quindi la memoria deve essere suddivisa in un certo numero di partizioni in modo da soddisfare le esigenze dei processi.
Il modo più semplice prevede di dividere la memoria in partizioni fisse, ognuna dedicata ad un processo; terminato un processo la sua partizione torna ad essere libera per un altro processo. *Il grado di multiprogrammazione è limitato dal numero di partizioni.*
Quando viene caricato un processo viene inserito in una coda di ingresso. Esistono due diverse implementazioni:

- partizioni di memoria fisse con code di ingresso separate per ciascuna partizione (Figura 4.4(a));
- partizioni di memoria fisse con coda di ingresso unica (Figura 4.4(b)).

Con il primo approccio, il processo caricato viene inserito nella coda d'ingresso della partizione più piccola che lo possa contenere. Poiché le partizioni sono fisse tutto lo spazio di una partizione non utilizzato dal processo è sprecato. Il caso peggiore si ha quando molti processi sono in attesa di una partizione piccola mentre la coda di ingresso di una partizione grande è vuota: piccoli processi sono in attesa anche se gran parte della memoria risulta libera.
L'organizzazione dei processi in un'unica coda risulta più efficiente in termini di tempi di attesa: appena una partizione di memoria è libera, viene occupata dal primo processo in coda che può entrare nella partizione. Una ottimizzazione di questa implementazione consiste nel selezionare non il primo processo in coda, ma cercare il processo più

grande in coda che la può occupare. In questo modo si evita di sprecare partizioni grandi per far girare processi piccoli.

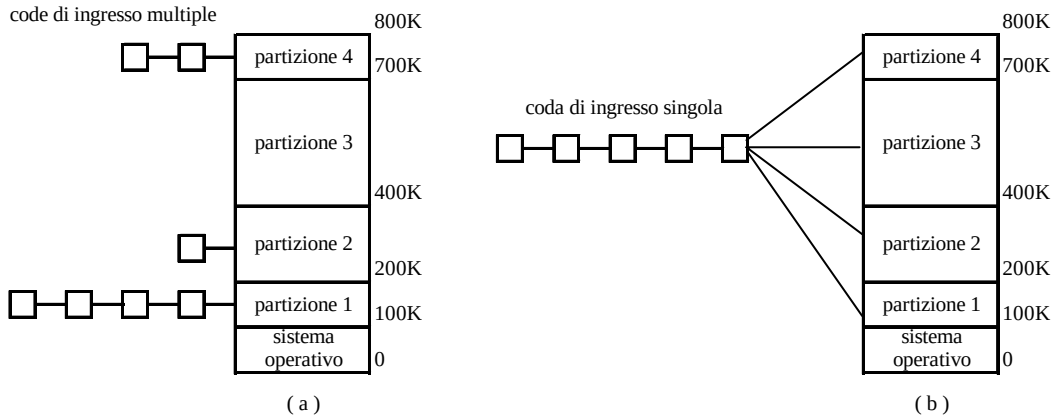



**Figura 4.4** Partizioni di memoria fisse.

Una soluzione più recente considera la memoria suddivisa in partizioni variabili. Il sistema operativo deve mantenere aggiornate due strutture dati: una per la gestione delle partizioni di memoria allocata, l'altra per lo spazio libero da allocare per i nuovi processi.
Inizialmente tutta la memoria è a disposizione dei processi utente, un unico grande blocco di memoria disponibile. Quando viene caricato un processo il sistema operativo cerca un sottospazio di memoria, generalmente definito "buco di memoria" (hole), sufficientemente grande da contenerlo. Al processo viene allocata solo la memoria strettamente necessaria all'esecuzione, la parte rimanente è utilizzata per soddisfare eventuali richieste future. Se la memoria disponibile non è sufficiente a contenere il processo, il sistema operativo può attendere che si renda disponibile oppure mette in attesa quel processo e ne seleziona uno più piccolo nella coda di ingresso. Se al contrario il buco è più grande delle richieste, viene diviso in due parti: una parte, di dimensioni della memoria richiesta dal processo, viene allocata, la parte rimanente viene riportata nell'insieme dei buchi liberi.

### 4.3.3 Assegnazione dinamica della memoria

La gestione dell'allocazione dei buchi ai vari processi è detta allocazione dinamica della memoria. Il sistema operativo organizza i buchi liberi in una struttura, dalla quale seleziona secondo un particolare algoritmo il buco da allocare al processo che richiede la memoria. Supponendo che il gestore della memoria sappia quanta memoria allocare, esistono tre diversi algoritmi di allocazione della memoria:

1. First fit: il gestore alloca il primo buco libero sufficientemente grande. Lo spazio libero è organizzato in una lista concatenata

semplice, o anche crescente. Il gestore ricerca il buco partendo dall'inizio della lista oppure dal punto in cui era terminata la ricerca precedente. Se il buco è più grande delle richieste del processo, la parte rimanente viene reinserita nella lista dei buchi liberi;

2. Best fit: il gestore alloca il più piccolo buco in grado di contenere il processo. La ricerca del buco avviene scorrendo tutta la lista se non è ordinata per dimensione. Con questo criterio si riducono al minimo le dimensioni dei buchi inutilizzati;
3. Worst fit: il gestore alloca il buco più grande. Anche con questo criterio si deve scorrere tutta la lista se non è ordinata per dimensione. Le parti di buchi inutilizzati risultano più grandi e possono essere più utili delle parti ottenute con il criterio precedente.

I migliori algoritmi in termini di tempo e di utilizzo della memoria risultano i primi due.

### 4.3.4 Frammentazione

La frammentazione della memoria può essere interna e esterna.
La frammentazione interna riguarda spazi di memoria allocati ai processi ma non utilizzati. Supponiamo che un processo richieda 15.998 byte e che l'unico buco che lo può contenere abbia dimensione 16.000 byte. Il sistema operativo può decidere di allocare al processo solo la parte strettamente necessaria e mantenere nella lista dei buchi liberi un buco di 10 byte. Il costo per il mantenimento di questa informazione risulta più grande del buco stesso. Per questo motivo in generale il sistema operativo prevede di suddividere la memoria fisica in blocchi di dimensione fissata, che costituiscono le unità di assegnazione (Figura 4.5). Al processo verrebbe quindi allocato l'intero blocco e i 2 byte non utilizzati creano frammentazione interna.

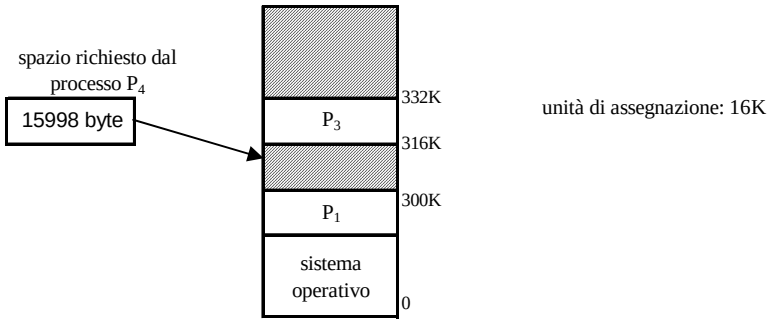



**Figura 4.5** Al processo P4 è allocata più memoria di quella richiesta.

La frammentazione esterna è causata dal meccanismo di swapping: caricando e scaricando processi di dimensioni diverse in memoria principale si creano molti buchi liberi. La soluzione a questo problema è

data dalla compattazione con la quale si riordina il contenuto della memoria libera in un unico grande blocco. Questo procedimento richiede un tempo non trascurabile e non è sempre possibile: il codice deve essere rilocabile perché la compattazione prevede lo spostamento dei processi e di conseguenza occorre rilocare tutti gli indirizzi interni. Ciò è possibile solo se la rilocazione è dinamica e a tempo di esecuzione, non se è statica e a tempo di assemblaggio o di caricamento.

### 4.3.5 Paginazione

La paginazione consente di assegnare ad un processo spazi di memoria non contigui ovunque essi siano disponibili, risolvendo quindi il problema della frammentazione esterna.

Ogni programma genera un insieme di indirizzi di memoria chiamati indirizzi logici e formano lo spazio di indirizzamento logico. La memoria logica è divisa in blocchi di memoria, le pagine, ognuna con dimensioni fissate e la memoria fisica in blocchi di memoria di uguali dimensioni, chiamati frames. Quando un processo richiede lo spazio di memoria, vengono caricate nei frames disponibili le pagine che gli occorrono per andare in esecuzione.

Quando viene utilizzata la memoria logica gli indirizzi logici non vengono messi direttamente sul bus di memoria ma vengono mandati alla memory management unit (MMU), un modulo del sistema operativo che mappa gli indirizzi logici sugli indirizzi della memoria fisica.

Il sistema operativo deve mantenere una struttura dati implementata in hardware, la tabella delle pagine, che consente di mappare pagine logiche in pagine fisiche: contiene per ogni pagina della memoria logica l'indirizzo di base del relativo frames in memoria fisica. Quindi la tabella delle pagine è una struttura che dispone di tante entrate quante sono le pagine logiche in cui è stato diviso lo spazio logico e ogni processo ha una propria tabella delle pagine, memorizzata nel proprio descrittore.

la CPU genera un indirizzo logico composto da due campi: selettore e offset. Tramite il selettore il sistema seleziona un ingresso nella tabella delle pagine che insieme all'offset determinano l'indirizzo della memoria fisica. Questo meccanismo corrisponde *all'allocazione della memoria con la rilocazione dinamica degli indirizzi.*

## 4.4 MEMORIA VIRTUALE

Il problema fondamentale della multiprogrammazione consiste nella limitata disponibilità della memoria: la dimensione combinata di

programmi, dati e stack può eccedere la dimensione della memoria fisica per essi disponibile.

Si introduce quindi il meccanismo di virtualizzazione della memoria che permette di eseguire processi non contenuti interamente nella memoria, basandoci sul fatto che l'intero programma non è necessario tutto in una volta.

Il sistema operativo mantiene in memoria principale solo le parti del programma in uso e il resto su una memoria ausiliaria. Spesso ci sono parti di programmi che non vengono quasi mai eseguiti (per esempio il codice relativo alla gestione degli errori) e lo spazio occupato può essere utilizzato per mantenere contemporaneamente in memoria più programmi.

Per realizzare la virtualizzazione è necessaria la presenza di una zona di memoria dove le pagine risiedono in attesa di essere allocate per i diversi processi: esiste quindi un'area di swap, che contiene le pagine temporaneamente non in memoria.

L'area di swap è una zona della *memoria di massa.*

### 4.5 MEMORIA SECONDARIA

La memoria secondaria rappresenta la parte più bassa del file system.

La parte del sistema operativo che si occupa della gestione della memoria deve implementare degli algoritmi di scheduling che ordinano la sequenza delle operazioni di I/O ai fini di migliorare le prestazioni del sistema. Inoltre deve gestire le condizioni di errore che si verificano sulle unità di memorizzazione.

# 5 I DISCHI

I dischi sono presenti nella maggior parte dei sistemi di elaborazione.

Dal punto di vista logico, i dischi possono essere considerati come un lungo array unidimensionale di blocchi logici di dati con dimensioni diverse (tipicamente 512 byte, ma facendo una formattazione a basso livello si può modificare tale dimensione).

Fisicamente un disco è diviso in cilindri aventi due superifici, ognuna contenente un certo numero di tracce concentriche. Ogni traccia è a sua volta divisa in settori.

Il numero di settori per traccia non è costante: più la traccia è lontana dal centro del disco, maggiore è la sua circonferenza e il numero di settori che può contenere (a parità di dimensione di settori).

Ogni settore è diviso in tre parti (Figura 5.1):

- Preambolo: contiene le informazioni di controllo (numero della traccia, numero del settore…);
- Dati: contiene i dati;
- ECC: contiene il codice a correzione di errore. Contiene sufficienti informazioni affinché il controllore possa identificare i bit danneggiati e ricalcolare il loro corretto valore.

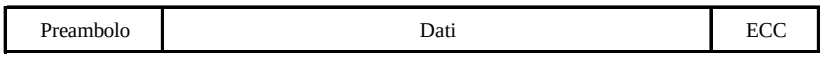



**Figura 5.1** Settore di un disco.

## 5.1 FORMATTAZIONE DEL DISCO

La formattazione del disco prevede cinque fasi:

1. Inizialmente il disco è vuoto;
2. *Formattazione a basso livello*: in questa fase vengono scritti i settori all'interno delle tracce del disco. La formattazione a basso livello dà un formato al disco (da qui deriva il termine "formattazione") in modo che il controllore possa leggere o scrivere sul disco;
3. Creazione delle *partizioni* del disco. Il sistema operativo suddivide il disco in gruppi di cilindri (le partizioni) e tratta ogni gruppo come una unità a disco a se stante;
4. *Formattazione ad alto livello*: in questa fase viene inserito un file system all'interno delle partizioni del disco;
5. Installazione del sistema operativo.

## 5.2 ALGORITMI DI SCHEDULING

Il disco è gestito dal controllore. Quando arriva una richiesta di lettura o scrittura il driver controlla prima di tutto se il dispositivo è libero o occupato (o fermo: in questo caso deve essere prima avviato), quindi esegue opportune operazioni di traduzione. Il controllore deve inoltre gestire la coda delle richieste pendenti da cui estrae i processi secondo determinate politiche.

I dischi sono dispositivi lenti e la durata delle operazioni di lettura o scrittura che lo coinvolgono dipende da diversi fattori. Esaminiamo l'operazione di lettura: il driver riceve la richiesta, controlla lo stato del dispositivo e traduce la richiesta. La parte del file system che fa parte del software indipendente dal dispositivi tramite la struttura di directory individua il blocco fisico.

Per ridurre i tempi il sistema sfrutta il *principio di località* per diminuire gli accessi al disco. Quando riceve una richiesta il sistema legge non solo il settore che contiene la richiesta, ma anche alcuni settori successivi. Il driver riceve le informazioni utili a soddisfare la richiesta, mentre tutto ciò che è stato letto negli altri settori viene memorizzato in una cache. Quando arriva una richiesta che coinvolge settori presenti nella cache il sistema non accede al disco, ma i dati da passare al driver vengono prelevati dalla cache.

### 5.2.1 Algoritmo FCFS

L'algoritmo FCFS (first-come first-served) organizza la lista delle richieste pendenti secondo l'ordine di arrivo. L'inserimento e l'estrazione delle richieste risulta semplice, ma questo algoritmo non permette di migliorare le prestazioni del sistema dal punto di vista del seek time.
Supponiamo di avere un disco con 200 tracce e di avere le seguenti richieste nella coda:

| traccia richiesta | 98 | 183 | 37 | 122 | 14 | 124 | 65 | 67 |
|---|---|---|---|---|---|---|---|---|
| ordine di arrivo | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Le richieste di lettura o di scrittura indicano la traccia su cui si trova il blocco che coinvolge l'operazione.
Supponiamo inoltre che la testina sia posizionata inizialmente sulla traccia 53. Con l'algoritmo FCFS il sistema serve le richieste nell'ordine di arrivo, quindi la testina si sposta dalla traccia 53 alla traccia 98, poi sulla traccia 183, poi sulla traccia 37 e così via.
Dopo aver servito l'ultima richiesta presente nella coda, la testina si è spostata complessivamente di 640 cilindri.

Questo algoritmo offre basse prestazioni perché il numero di spostamenti della testina è elevato: la testina ogni volta si deve spostare casualmente da una parte all'altra del disco.

### 5.2.2 Algoritmo SSTF

L'algoritmo SSTF (Shortest Seek Time First) minimizza i tempi di ricerca. Il sistema serve le richieste in base alla posizione della testina: seleziona dalla coda delle richieste pendenti quella più vicina alla testina in quell'istante. L'inserimento delle richieste nella coda risulta ancora semplice, mentre nell'estrazione il sistema deve scorrere tutta la lista per individuare la richiesta.
Utilizzando la situazione ipotizzata nel descrivere l'algoritmo FCFS, l'ordine di esecuzione delle richieste con l'algoritmo SSTF diventa (si specificano le tracce su cui la testina si sposta):
$65 \rightarrow 67 \rightarrow 37 \rightarrow 14 \rightarrow 98 \rightarrow 122 \rightarrow 124 \rightarrow 183$. Il risultato finale calcolato in numero di cilindri visitati, è di 236 cilindri.

Questo algoritmo introduce *starvation*: se una richiesta è troppo lontana rispetto alle altre richieste che continuano ad arrivare in coda, non verrà mai servita. Al contrario le richieste favorite sono quelle che richiedono blocchi posizionati nelle tracce centrali al disco.

### 5.2.3 Algoritmo SCAN

L'algoritmo SCAN effettua una scansione di tutte le tracce. Il braccio del disco parte da un estremo, si sposta nella sola direzione possibile servendo tutte le richieste che incontra sui cilindri, fino a raggiungere l'estremo opposto del disco. A questo punto viene invertita la direzione (verso l'estremo da cui era partita) e riparte a servire le richieste. Le testine attraversano continuamente il disco nelle due direzioni.

Questa soluzione elimina la *starvation*, perché sicuramente tutte le richieste vengono servite. Il tempo massimo di attesa di una richiesta risulta, nella peggiore delle ipotesi, uguale al tempo necessario alla testina ad attraversare per due volte il disco.
Se ripetiamo l'esempio precedente applicando questo algoritmo otteniamo 208 cilindri, risultato migliore del caso precedente, ma in generale lo SCAN offre prestazioni peggiori del SSTF.

### 5.2.4 Algoritmo C-SCAN

Con l'algoritmo SCAN quando il braccio del disco torna indietro la zona vicino all'estremo è stata visitata da poco, quindi probabilmente ci saranno poche richieste, mentre la maggior densità di richieste sarà concentrata all'estremo opposto. Una variante di questo algoritmo (ottimizzazione) prevede una *scansione circolare* del disco. Si parla quindi di C-SCAN: la testina si sposta da un estremo all'altro come con lo SCAN, ma quando

raggiunge un estremo riparte dall'inizio, senza servire richieste nel viaggio di ritorno.
Il C-SCAN tratta il disco come una lista circolare.

### 5.2.5 Algoritmo LOOK

Esistono altri due algoritmi che derivano dall'algoritmo a scansione SCAN e C-SCAN che si basano sull'idea che risulta inutile arrivare fino in fondo al disco se non ci sono più richieste oltre una determinata traccia: con il LOOK (per lo SCAN) e il C-LOOK (per il C-SCAN) la testina si sposta finché ci sono richieste da servire, quindi torna all'inizio (Figura 5.2).

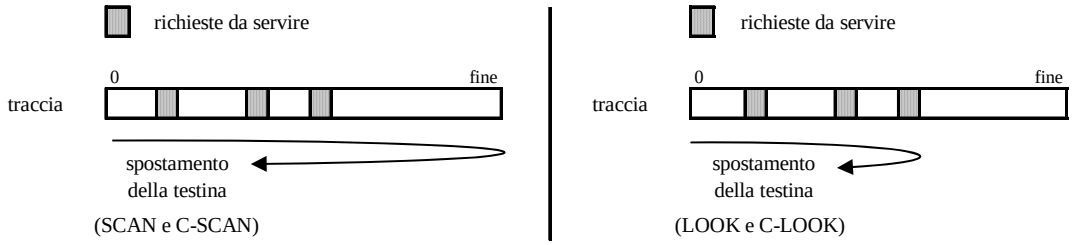



**Figura 5.2** Spostamento della testina nei diversi algoritmi.

## 6 IL FILE SYSTEM

La parte del sistema operativo che interagisce direttamente con l'utente è il file system. Permette all'utente di gestire e memorizzare i dati tramite l'utilizzo dei file e delle directory, che sono parte integrante del file system.

Tutte le applicazioni necessitano di memorizzare e recuperare informazioni. Una possibile soluzione a questo problema prevede che un processo in esecuzione memorizzi una quantità limitata di informazioni nel suo spazio di indirizzamento. Ovviamente questa implementazione è in conflitto con le richieste delle applicazioni:

- i processi devono poter memorizzare una grande quantità di informazioni, mentre questa soluzione offre una capacità di memorizzazione limitata dalla dimensione dello spazio di indirizzamento virtuale;
- le informazioni memorizzate occorrono anche quando il processo che le sta utilizzando termina, mentre se vengono memorizzate nel suo spazio di indirizzamento del processo vengono perse;
- i processi devono poter condividere le informazioni, ciò non è possibile con questa soluzione.

L'alternativa è quella di memorizzare le informazioni sui dischi o altri supporti esterni, in unità chiamate file. Questa implementazione risolve

tutti i problemi introdotti con la soluzione precedente: le informazioni nei file sono permanenti e un file viene eliminato solo su esplicita richiesta dell'utente. I file sono gestiti dal sistema operativo che si occupa della loro implementazione e di tutte le problematiche relative a tali unità.

La parte del Sistema Operativo che si occupa dei file è il File System ed è composto:

- dall'insieme dei file;
- dalle informazioni necessarie alla gestione dei file; queste informazioni sono contenute nella struttura di directory, organizzate in modo da permettere lo svolgimento di determinate operazioni sui file. La struttura di directory è situata in una zona del disco (quindi gli accessi risultano lenti) e contiene per ogni file del File System un elemento che descrive il file.

## 6.1 IL FILE

Il sistema operativo definisce una unità di informazione logica (il file) e la rende disponibile all'utente, che può far riferimento al file tramite il nome e degli attributi.
L'utente interagisce direttamente con il File System e non si occupa di come sono memorizzate le informazioni all'interno del file.
Il file può essere rappresentato:

- senza struttura, il file è visto come una sequenza non strutturata di byte;
- con una struttura minima, il file è composto da una sequenza di record di lunghezza fissa, ognuno con una propria struttura interna;
- con una struttura più forte, il file è formato da un albero di record, non tutti necessariamente della stessa dimensione.

Anche i dati che contiene possono essere diversi: un file può contenere informazioni numeriche, alfabetiche, alfanumeriche o in formato binario.

Gli attributi che il sistema associa a un file organizzati nella struttura di directory sono diversi e variano a seconda del sistema:

- *Nome*: attributo mediante il quale l'utente identifica il file. Il sistema operativo lo identifica tramite un ID, ovvero una sequenza di numeri;
- *Tipo*: attributo utilizzato dal sistema operativo e specifica al sistema stesso l'applicazione da utilizzare per operare su quel file. Alcuni sistemi operativi impongono un tipo predefinito. Il tipo è indicato come estensione del nome del file;

- *Locazione*: specifica la posizione del file all'interno del disco o in qualsiasi altro supporto di memorizzazione;

- *Dimensione*: specifica la dimensione del file;

- *Protezione*: specifica chi può accedere al file e quali operazioni può eseguire sul file;

- *Proprietario*: proprietario corrente del file;

- *Tempo*: specifica la data di creazione del file.

### 6.1.1 Operazioni sui file

Le operazioni possibili sui file vengono eseguite tramite system call e variano in funzione della struttura del sistema operativo utilizzato. Il file system mantiene un *puntatore* al file: quando si effettua una scrittura tale puntatore è posizionato alla fine del file, in caso di lettura punta la locazione di memoria specificata. Quando è richiesta una operazione di read o write il sistema deve individuare l'elemento di directory nella struttura tramite il nome del file passato come parametro della system call. Quindi individua la posizione corrente del puntatore.

Oltre alla lettura e scrittura, le operazioni più utilizzate sono le seguenti:

- Create: alloca un elemento nella struttura di directory. Il sistema individua lo spazio opportuno sul disco e assegna agli attributi del file gli opportuni valori; in particolare memorizza la posizione di partenza del file sul disco nell'attributo locazione;
- Delete: cancella il file. Il sistema operativo individua la locazione del file sul disco e dealloca lo spazio;
- Open: apre il file. Restituisce un identificatore utilizzato per effettuare le operazioni successive sul file;
- Close: chiude il file;
- Write: scrive dei dati nel file. Ha come parametri il nome del file e le informazioni da scrivere. Se la posizione corrente è alla fine del file, la dimensione del file viene estesa. Se la scrittura avviene nel mezzo del file, i dati esistenti vengono soprascritti dai nuovi;
- Read: legge i dati dal file. Ha come parametri il nome del file e la locazione di memoria da leggere;
- Seek: posiziona il puntatore alla posizione corrente;
- Truncate: elimina una quantità di dati dal file.

### 6.1.2 Modalità di accesso ai file

Si distinguono due modi di accedere ai file:

1. Accesso sequenziale. Un processo che opera in questo modo deve accede alle informazioni del file in modo sequenziale, leggendo tutti i record di un file in ordine partendo dall'inizio. Non è possibile saltare qualche informazione né leggerle fuori dall'ordine in cui sono state memorizzate;

2. Accesso diretto. Permette ai processi di accedere ad un particolare elemento del file senza rispettare alcuna sequenza.

Le operazioni di lettura e scrittura dei file sono diverse a seconda della modalità di accesso. Nella *modalità sequenziale* esistono le seguenti operazioni:

- Read next, write next: un file è visto come un insieme di elementi o record con contenuto diverso. Queste operazioni permettono di leggere o scrivere sull'elemento successivo;
- Reset: Apre il file in lettura. Questa system call posiziona il puntatore dell'operazione corrente all'inizio del file, sulla prima istruzione, e si predispone per la lettura;
- Rewrite: operazione equivalente alla reset. Apre il file in scrittura.

Se l'accesso al file avviene in modo *diretto*, le operazioni possibili sono:

- Read n, write n: quando si esegue una operazione su un file, si fa riferimento ad ogni singolo elemento e non all'intero file. Generalmente i record hanno una lunghezza fissa e sono numerati all'interno del file, quindi è facile individuare ogni elemento tramite l'indice (n): la lettura e la scrittura specificano la posizione dell'elemento all'interno del file;
- Position n: posiziona il puntatore dell'operazione corrente in corrispondenza dell'n-esimo elemento del file.

## 7 REALIZZAZIONE DEL FILE SYSTEM

Il sistema operativo per gestire le informazioni memorizzate dagli utenti utilizza il file system, che risiede nella memoria secondaria progettata appositamente per contenere grandi quantità di dati. Il sistema organizza il file system su più livelli, perché la stratificazione permette una migliore gestione dell'intero sistema.

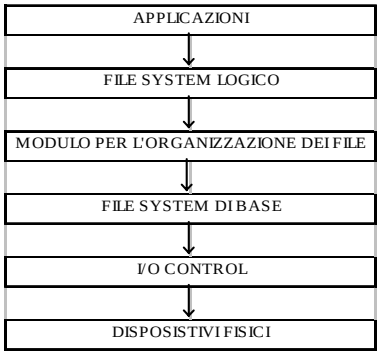
## 7.1 ORGANIZZAZIONE DEL FILE SYSTEM

Il file system risiede permanentemente nella memoria secondaria, sui dischi. Essendo una entità complessa, il file system non viene realizzato come un unico blocco, ma su strati diversi (tipicamente di 512 byte) per migliorare l'efficienza del sistema.

Una organizzazione di questo tipo consente una migliore gestione del sistema, infatti i dischi permettono di accedere direttamente a qualunque blocco con un facile accesso sequenziale o casuale ai file. Inoltre i dischi possono essere riscritti localmente: il sistema legge un blocco, lo modifica e lo riscrive nella stessa posizione.

Per gestire le informazioni sul disco il sistema sfrutta la stratificazione del file system (layering): ogni livello implementa servizi per gli strati superiori, utilizzando le informazioni fornite dagli strati inferiori.

I livelli del file system sono sei:




1. Applicazioni
2. File system logico
3. Modulo per l'organizzazione dei file
4. File system di base
5. I/O control
6. Dispositivi fisici

Il livello più alto è rappresentato dalle applicazioni, composto dagli utenti e dai programmi.

Sotto le applicazioni c'è il file system logico che implementa l'astrazione dei file e delle directory. Questo livello rende disponibile alle applicazioni le call system per eseguire le operazioni sui file e tutte le funzioni realizzate a livello di file system logico.

Il modulo per l'organizzazione dei file traduce i blocchi logici nei corrispondenti indirizzi fisici. Può realizzare questa funzione perché conosce come il file è allocato su disco e la corrispondenza tra indirizzo logico e indirizzo fisico. Il modulo per l'organizzazione dei file traduce le richieste che gli passa il file system logico nel formato "leggi / scrivi il blocco logico 1": il modulo individua all'interno del file il blocco logico, gli associa il corrispondente blocco fisico e passa il messaggio al livello sottostante.

## 7.2 STRUTTURA DEL FILE SYSTEM

Un disco è suddiviso in partizioni, ognuna della quale rappresenta un disco virtuale. In generale una partizione indica una parte del disco, ma può anche accadere il contrario, cioè esistono partizioni che occupano più dischi.

In generale esistono diverse partizioni per separare le informazioni dai programmi. Una partizione è dedicata all'installazione del sistema operativo e per i programmi, un'altra memorizza i dati utilizzati dai programmi. Con una organizzazione di questo tipo, se si verifica un crash del sistema la partizione dei dati è salva.
Inoltre la suddivisione del disco in partizioni permette di installare più sistemi operativi su una stessa macchina, ognuno su partizioni diverse.

La prima partizione contiene sempre il *file system* (Figura 7.1). Il primo blocco contiene il master boot record (MBR), utilizzato per l'avvio della macchina. Il secondo blocco contiene la tabella delle partizioni che memorizza le informazioni sulle partizioni del disco; per ogni partizione viene indicato l'inizio e la fine. All'interno della tabella una partizione è contrassegnata *attiva*: è la partizione che contiene il sistema operativo che deve essere caricato su quella macchina.

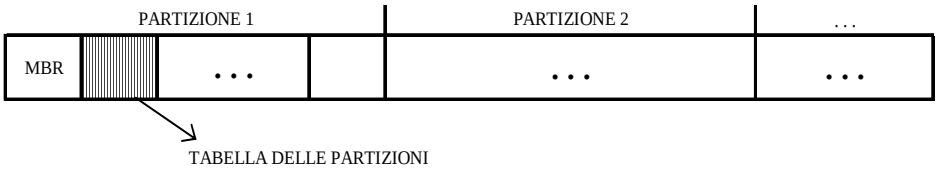



**Figura 7.1** Struttura della partizione che contiene il file system.

All'avvio della macchina parte il programma *bios* che esegue MBR e carica la tabella delle partizioni, indicando anche la partizione attiva. Quindi viene caricato il sistema operativo tramite l'esecuzione di un apposito programma presente nella partizione attiva. Da questo momento in poi la macchina è sotto il controllo del sistema operativo.

Nei sistemi basati sull'uso della memoria virtuale esiste una partizione (la seconda) dedicata all'area di swap, la cui grandezza dipende dal tipo di programmi che vengono eseguiti nel sistema. Poiché è difficile da dimensionare, si indica lo spazio di swap grande due volte lo spazio di memoria RAM.

### 7.2.1 Struttura delle partizioni

Le partizioni del disco possono contenere il file system, un sistema operativo o altre informazioni (area di swap, dati…). Ogni partizione è composta da cinque blocchi (Figura 7.2):

- Blocco di avvio; se la partizione contiene un sistema operativo, il blocco di avvio contiene il programma di caricamento del sistema, altrimenti è vuoto;
- Super blocco; contiene le informazioni relative al file system della partizione in questione, caricate in memoria in fase di avvio della macchina;
- Gestione dello spazio libero; contiene la struttura dati per la gestione dello spazio libero su disco (non tutti i blocchi sono occupati) controllata dal sistema ogni volta che riceve una richiesta di allocazione dello spazio su disco da parte di un file;
- Directory radice, unica directory creata in fase di formattazione del disco;
- File system; i rimanenti blocchi della partizione contengono il file system (vuoto).

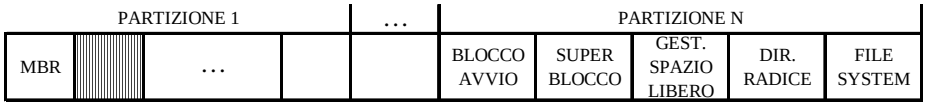



**Figura 7.2** Struttura delle partizioni di un disco.

### 7.2.2 Strutture dati utilizzate dal file system

Il sistema utilizza diverse strutture dati memorizzate in memoria per agevolare la gestione dei file e del sistema stesso. In particolare mantiene nella tabella delle partizioni le informazioni relative alle partizioni montate sul sistema operativo e altre strutture dati per la memorizzazione delle informazioni relative alla struttura di directory.

La struttura di directory si trova sul disco, che comporta accessi lenti.

Quindi tutte le volte che un processo riferisce un file non viene sempre fatto un accesso sul disco perché comporterebbe un enorme spreco di tempo.

Il sistema operativo si avvale del supporto di alcune strutture presenti in memoria:

- tabella globale dei file aperti**:** tabella gestita dal sistema operativo e memorizza tutti i file aperti da tutti i processi nel sistema. Ogni entrata della tabella contiene le stesse informazioni presenti nella struttura di directory;
- tabella dei file aperti dal processo: contiene gli elementi prelevati dalla struttura di directory relativi ai file aperti dal processo. Esistono tante tabelle quanti sono i processi in esecuzione. Ogni entrata della tabella contiene solo una parte delle informazioni presenti nella struttura di directory

Supponiamo inizialmente di operare in un *sistema monoutente*. In questa ottica il sistema mantiene in memoria solo una tabella dei file aperti dall'unico processo in esecuzione.

Quando un file è aperto per la prima volta il sistema accede alla struttura di directory, individua l'elemento relativo al file su cui è stata fatta la open e lo trasferisce nella tabella dei file aperti dal processo. I successivi accessi a quel file non comportano più un accesso al disco perché il corrispondente elemento è già stato caricato in memoria.

Quando un file non è più utile per l'elaborazione, il processo effettua la close del file: il sistema individua l'elemento nella tabella dei file aperti dal processo e lo copia nella corrispondente entrata della tabella di directory su disco.

Questo modo di operare presuppone la presenza di un solo utente.

Per estendere questo meccanismo ad un *sistema multiprogrammato* il sistema deve mantenere in memoria più strutture che consentono ai processi di operare sequenzialmente sugli stessi file: occorre una tabella dei file aperti per ogni processo in esecuzione e una tabella globale dei file aperti.

Quando un processo effettua la open su un file il sistema prima di tutto controlla se esiste già l'elemento che descrive quel file nella tabella globale dei file aperti. Se è già presente significa che precedentemente un altro processo lo ha aperto: il sistema memorizza le informazioni necessarie, prelevate dalla tabella globale, nella tabella dei file aperti dal processo che ha effettuato la open.

Se non trova nessun entrata nella tabella globale corrispondente al file aperto, significa che quel file non è utilizzato da nessuno; quindi il sistema accede alla struttura di directory e memorizza l'elemento corrispondente al file nella tabella globale dei file aperti e nella tabella del processo che ha effettuato la open.

Quando un processo chiama la close su un file il sistema copia gli aggiornamenti nell'entrata della tabella globale, prelevati dalla tabella dei file aperti dal processo.

La tabella globale dei file aperti da tutti i processi viene ricopiata su disco solo quando tutti i processi hanno chiuso i file. Nella tabella sarà quindi presente un contatore per ogni file che indica il numero di open eseguite dai processi su ogni file: ad ogni open il contatore viene incrementato, con la close si decrementa. Quando il contatore è 0 significa che tutti i processi hanno chiuso il file, quindi il sistema copia l'elemento dalla memoria al disco.

## 7.3 MEMORIZZAZIONE DEI FILE SU DISCO

I file e il disco vengono suddivisi rispettivamente in blocchi logici e blocchi fisici. Il sistema può implementare diversi modi di allocare il file all'interno del disco.

### 7.3.1 Allocazione contigua

Il modo più semplice è chiamato allocazione contigua (Figura 7.3): il sistema alloca un file in blocchi logici contigui.

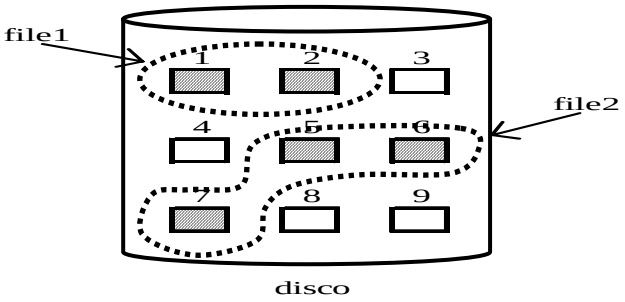



**Figura 7.3** Allocazione contigua dei blocchi fisici ad un file.

Nella struttura di directory oltre all'attributo "nome" deve essere memorizzato anche l'indirizzo fisico del primo blocco (*start*) e il numero dei blocchi allocati per quel file (*size*):

| Nome | Start | Size |
|---|---|---|
| file1 | 1 | 2 |
| file2 | 5 | 3 |

Queste informazioni sono sufficienti a localizzare correttamente tutti i file; infatti data la dimensione del file e il primo blocco fisico occupato si possono determinare quali sono i blocchi occupati da ogni file.

L'allocazione contigua offre al sistema vantaggi sotto molti aspetti:

- E' concettualmente semplice: quando un sistema deve allocare un file deve solo scegliere un numero opportuno di blocchi in base alle richieste ed allocarli;
- L'accesso sequenziale ad un file avviene normalmente: non è necessario spostare la testina del disco da un blocco all'altro perché sono consecutivi;
- E' possibile anche accedere direttamente ai file: l'indirizzo fisico a cui corrisponde l'indirizzo logico è calcolato facilmente.

La frammentazione interna è trascurabile ed eliminabile, mentre è pesante la frammentazione esterna che si crea quando i file vengono cancellati. Per questo motivo l'allocazione può talvolta risultare un problema: non è semplice individuare i blocchi liberi perché si creano buchi vuoti tra un file e l'altro non sufficientemente grandi da soddisfare le richieste. Si potrebbe pensare di eliminare la frammentazione esterna tramite il meccanismo di ricompattazione come accade nella memoria. Questo meccanismo è sconsigliato poiché applicato al disco comporta un enorme perdita di tempo. Il sistema può invece implementare degli algoritmi specifici, come nel caso dell'allocazione della memoria.

La dimensione dei file può variare nel tempo. Il sistema deve quindi tener presente che la richiesta dei blocchi fisici occupati da un file può aumentare. Supponiamo che un file aumenti le sue dimensioni dopo che il sistema lo ha allocato. Se l'allocazione è contigua ed è libero il blocco successivo all'ultimo allocato a quel file, il sistema estende l'allocazione normalmente apportando le opportune modifiche agli attributi del file (Figura 7.4 (a)).

Se il blocco successivo è occupato da un altro file, per evitare di abortire il programma, il sistema deve trovare un insieme di blocchi contigui liberi in grado di soddisfare la nuova dimensione del file e spostare tutto il file nel buco individuato, rilasciando quello vecchio (Figura 7.4 (b)).

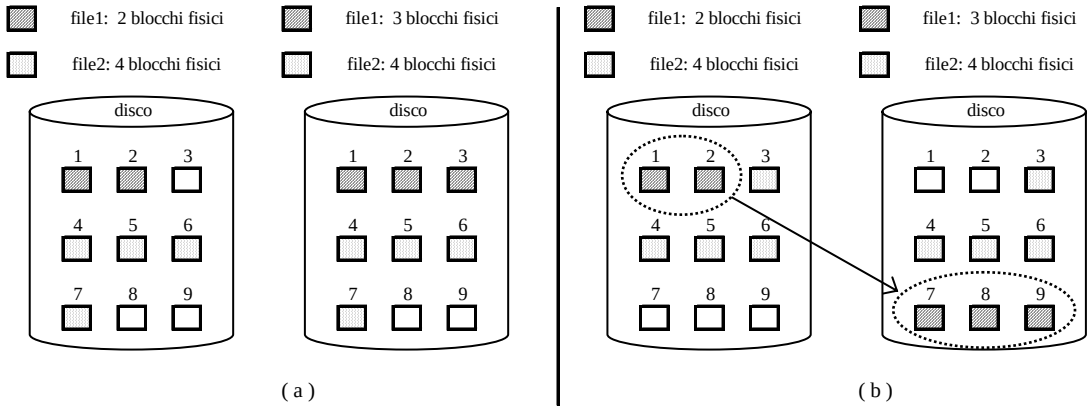



**Figura 7.4** Estensione di un file in un sistema che implementa l'allocazione contigua.

Con questo tipo di organizzazione il sistema impiega molto tempo nel trasferimento dei file da una parte all'altra del disco.
Una alternativa consiste nel gestire il problema preventivamente, allocando il file in uno spazio maggiore di quello richiesto. Il sistema dovrebbe prevedere la variazione della dimensione del file nel tempo, inoltre questa soluzione introduce frammentazione interna e riduce la percentuale di occupazione del disco (piccole parti dello spazio allocato ai file non verranno mai utilizzate).
La soluzione più efficiente al problema dell'estensione della dimensione dei file introduce il concetto di extent (Figura 7.5). Il sistema indica con extent una porzione del disco costituita da un certo numero di blocchi fisici allocati ad un file. Ogni extent può avere dimensioni diverse. Quando un file richiede l'allocazione il sistema gli concede uno o più extent. Quando il file cresce i blocchi precedentemente allocati rimangono invariati, a questi viene aggiunto un altro extent collocato in qualsiasi parte del disco. Gli extent di uno stesso file vengono collegati tramite link.

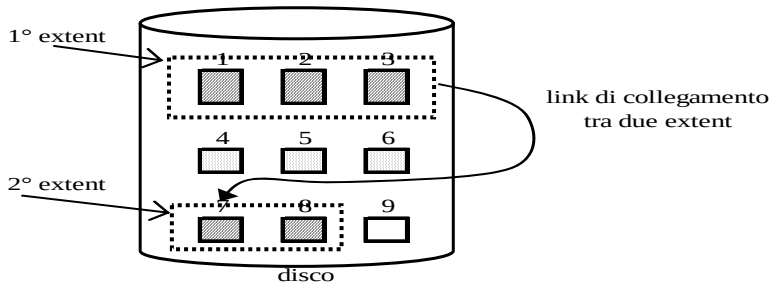



**Figura 7.5** Allocazione contigua con extent.

### 7.3.2 Allocazione con indice

L'allocazione con indice raggruppa tutti i puntatori in una sola locazione del disco, a formare il blocco indice (i-node). Questo tipo di organizzazione risolve l'esigenza di non dover scorrere tutti i blocchi per realizzare l'accesso diretto ad un file.
Il primo blocco allocato ad ogni file è il blocco indice e contiene un array in cui l'elemento i-esimo memorizza l'indirizzo del blocco fisico a cui corrisponde il blocco logico del file (Figura 7.8).
La struttura di directory dovrà essere modificata opportunamente: non conterrà più l'indirizzo del primo ed ultimo blocco allocato al file, ma indicherà l'indirizzo del blocco indice. Tale informazione è sufficiente a individuare i blocchi fisici che compongono il file.

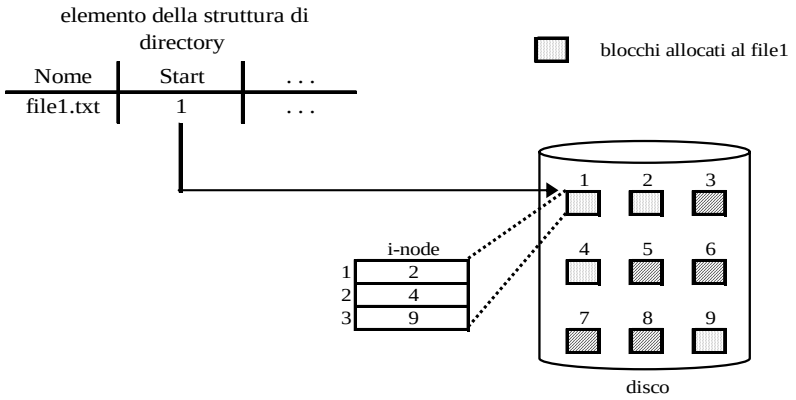



**Figura 7.8** Allocazione con indice.

L'allocazione con indice permette sia l'accesso diretto sia l'accesso sequenziale ai file perché il sistema tramite la struttura di directory è a conoscenza degli indirizzi di tutti i blocchi che compongono il file.

Ovviamente ad ogni file deve essere allocato un numero maggiore di blocchi rispetto alle richieste: un file composto da n blocchi, è allocato in (n+1) blocchi fisici.
Quindi la presenza del blocco indice comporta l'introduzione di overhead che risulta maggiore per i file di dimensioni minori. Inoltre un file con dimensione molto estesa può richiedere un *numero maggiore di blocchi indice*.
La frammentazione interna è limitata come accade con qualsiasi tipo di allocazione utilizzata, mentre non c'è frammentazione esterna.

Quando un blocco indice non è più sufficiente a soddisfare le richieste di un file, il sistema deve implementare dei metodi per assegnare e collegare più blocchi indice. Esistono tre varianti alla allocazione indicizzata:

1. Indice concatenato. Il sistema utilizza più blocchi indice collegati tramite puntatori a costituire una lista concatenata di blocchi indice. L'ultima entrata di ogni blocco indice ha un puntatore al successivo blocco indice. Tutte le altre entrate puntano a blocchi di dati. L'ultimo blocco indice conterrà nell'ultima entrata il valore "null";

2. Indice multilivello. Il sistema realizza questo tipo di organizzazione tramite una *struttura gerarchica* dei blocchi indice. E' possibile utilizzare un blocco indice di primo livello che punta a un insieme di blocchi indice di secondo livello i quali a loro volta puntano ai blocchi del file;

3. Schema combinato. Il sistema utilizza gli i-node, blocchi indice a quindici elementi. La struttura di directory contiene il nome del file e l'i-node, mentre l'i-node è così composto:

- Nella prima parte ci sono le informazioni relative al file, ovvero tutti gli attributi escluso il nome.

- I primi dodici elementi puntano direttamente ai blocchi dei file; per i file di piccole dimensioni sono sufficienti questi elementi a contenerli, l'overhead è più contenuto.

- Il tredicesimo elemento è un puntatore a un blocco indiretto singolo, cioè un blocco indice che punta a blocchi di dati (due livelli di indicizzazione).

- Il quattordicesimo elemento è un puntatore a un blocco indiretto doppio, ovvero un blocco indice che punta a blocchi indice che a sua volta puntano blocchi di dati (tre livelli di indicizzazione).

- L'ultimo elemento è un blocco indiretto triplo (quattro livelli di indicizzazione).

L'accesso diretto è più veloce e si adatta ai file dimensioni grandi e piccoli.

## 8 LA STRUTTURA DI DIRECTORY

Una directory è rappresentata come un contenitore di file (i descrittori dei file). Alcuni File System (per esempio Unix) trattano la directory come un file con un contrassegno (bit) per indicare che è una directory; quindi le operazioni sui file valgono anche per le directory. Altri sistemi (per esempio Windows) le trattano in modo diverso ai file, quindi anche le operazioni che si possono fare sulle directory sono diverse dalle operazioni sui file. Questi sistemi presuppongono system call che fanno riferimento alle directory.
In generale le operazioni che si possono eseguire su una directory, oltre alla creazione e alla cancellazione della struttura, riguardano la creazione o la cancellazione di un file, la ricerca di un file tramite lo scorrimento della directory e la possibilità di elencare tutti i file contenuti in una directory, con le relative informazioni.

Le directory sono utilizzate per organizzare i file all'interno del File System ed esistono diversi modelli implementativi; il sistema deve prima di tutto tenere conto delle richieste dell'utente, ovvero:

1. *Efficienza*: le operazioni devono essere veloci;
2. *Nominazione*: attributo riferito ai nomi dei file. All'interno di una directory i nomi dei file devono essere unici. Il sistema operativo deve controllare che non ci siano conflitti;
3. *Raggruppamento logico dei file*: l'utente dispone di diversi file che devono essere raggruppati logicamente in classi di categorie diverse.

## 8.1 STRUTTURA A SINGOLO LIVELLO

L'organizzazione logica delle directory più semplice è rappresentata dalla struttura a singolo livello, in cui tutti i file sono contenuti in una directory comune (Figura 8.1). Il file system è organizzato su un unico livello, composto da una unica directory i cui elementi sono i descrittori dei file. Tra gli attributi di ogni file è presente anche il puntatore alla posizione sul disco.

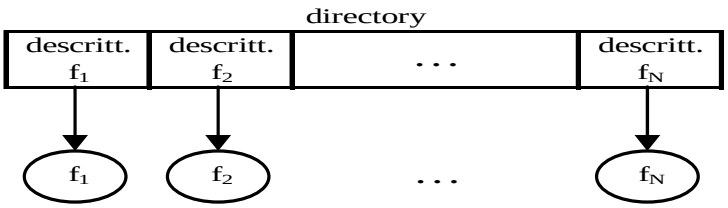



**Figura 8.1** Struttura di directory a singolo livello.

Per creare un file il sistema deve prima di tutto controllare se esiste già un file con il nome specificato dall'utente, scorrendo tutta la struttura di directory. Se non esiste, compie una sequenza di operazioni:

1. alloca lo spazio sul disco per il file;
2. introduce un nuovo elemento nella struttura di directory, inizializzando gli attributi secondo le specifiche dell'utente;
3. alloca il puntatore del file allo spazio su disco.

Per la cancellazione di un file, il sistema esegue analoghe operazioni: il file system cerca nella struttura di directory l'elemento con il nome specificato dall'utente, libera lo spazio sul disco e il descrittore del file.

## 8.2 STRUTTURA A DOPPIO LIVELLO

Una alternativa all'organizzazione a singolo livello è fornita dalla struttura delle directory su due livelli, comunemente chiamata struttura a doppio livello (Figura 8.2): in questa struttura compare una *directory principale* costituita da tanti elementi quanti sono gli utenti; ogni elemento della directory principale contiene un puntatore ad una *directory secondaria* (o directory utente o sottodirectory) organizzata come nella struttura a singolo livello.
Rispetto alla struttura a singolo livello, con questa organizzazione la gestione dei nomi dei file risulta semplificata per l'utente e per il sistema, poiché ogni directory contiene meno file. All'interno della directory secondaria si ripetono i problemi visti nella struttura a singolo livello: il sistema non fornisce un supporto per i file del singolo utente, ma li raggruppa in una unica directory.

Per fare riferimento ad un file il sistema deve fare un doppio scorrimento: il primo per trovare l'utente a cui appartiene il file, il secondo per trovare il file. In generale il primo scorrimento può essere evitato, caricando in memoria la directory dell'utente che sta effettuando un accesso al sistema. Le volte successive il sistema non accede più al disco e le operazioni risulteranno più veloci.

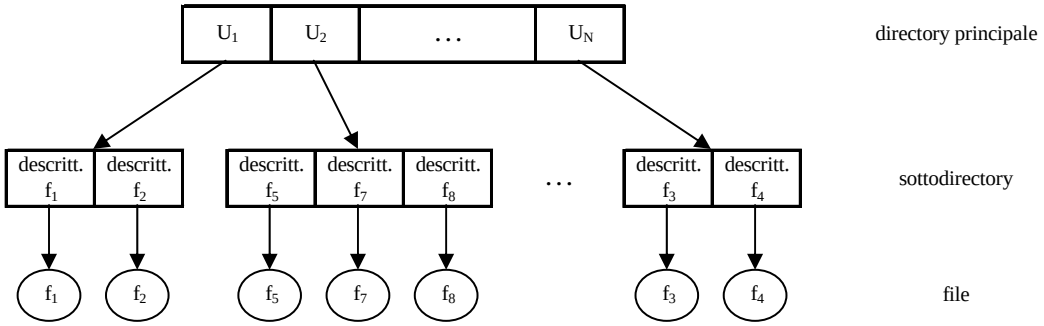



**Figura 8.2** Struttura di directory a doppio livello.

## 8.3 STRUTTURA AD ALBERO GENERICO

Generalizzando l'organizzazione a doppio livello delle directory, si ottiene la struttura ad n livelli (Figura 8.3), in cui il primo elemento è sempre la radice, il livello successivo è composto dagli utenti ed infine per ogni utente possono essere previste più directory a formare un albero.

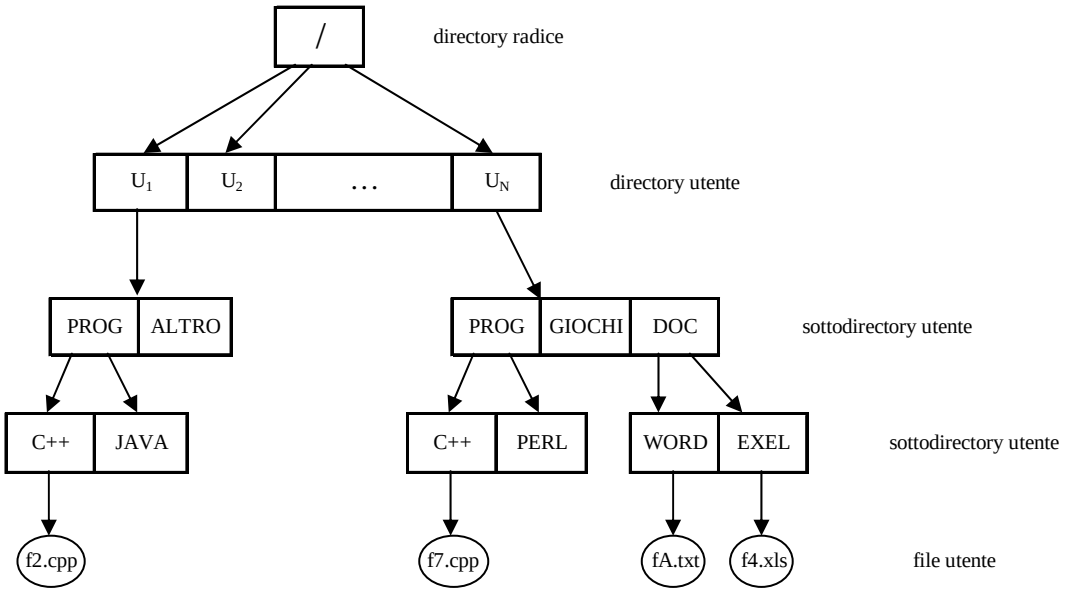



**Figura 8.3** Struttura di directory ad albero generico.

Questa struttura soddisfa le richieste dell'utente: la gestione dei nomi è semplificata e l'organizzazione logica è avvantaggiata dal fatto che ogni utente può riorganizzare le directory come vuole. L'unico aspetto negativo riguarda la struttura più complica rispetto alle organizzazioni precedenti: è più difficile da realizzare e risulta più complicata anche la ricerca dei file da parte del sistema che deve analizzare tutti i sottoalberi degli utenti.

Se due utenti hanno nella propria directory un file con lo stesso nome il sistema deve implementare un meccanismo con il quale distingue i due file. Per individuare univocamente i file vengono quindi definiti i concetti di:

- *path assoluto*: ad ogni file è assegnato un path name costituito dal percorso dalla directory radice (root) al file;
- *path relativo*: viene designata una *directory corrente* (o *di lavoro*) e ad ogni file è assegnato un path name costituito dal percorso dalla directory corrente al file. In molti sistemi ogni processo ha la propria directory corrente.