Il progetto del file system

In un sistema operativo la gestione delle informazioni richiede la possibilità di memorizzare permanentemente grandi quantità di dati, che possono essere condivisi da elaborazioni simultanee o diluite nel tempo

Ci sono tre necessità da soddisfare:

- memorizzare (e accedere efficacemente) *grandi quantità* di dati
- le informazioni devono poter essere *persistenti*
- l'accesso deve essere garantito anche a processi *concorrenti*

La maggior parte dei sistemi consente di definire archivi con un nome, i *file*, che contengono in modo persistente programmi o dati

• I file **non** fanno parte dello spazio d'indirizzamento di un processo

Il s.o. mette quindi a disposizione alcune chiamate di sistema per:
crearli, distruggerli, leggerli, scriverli, ecc.

NOTA:
Il software di gestione dei file si chiama file system, così come la collezione di tutti i file di un sistema: non confondere i due concetti!

Nel seguito cercheremo di usare il termine *file system server* per definire il software che gestisce la collezione dei file che compongono un file system

Uso del file system nei programmi

Per usare i servizi del file system i processi usano le chiamate `open`, `read`, `write` e `close`.

Processo lettore:

```
/* process A */
#include <fcntl.h>

main()
{
    int fd;
    char buf[512];
    fd=open("/etc/passwd", O_RDONLY);
    read(fd, buf, sizeof(buf));        /* lettura */
    read(fd, buf, sizeof(buf));        /* lettura */
    close(fd);
}
```

Processo scrittore:

```
/* process B */

#include <fcntl.h>

main()
{
    int fd, i;
    char buf[512];
    for (i=0, i<sizeof(buf); i++)
       buf[i]= 'a';
    fd = open("/etc/passwd", O_WRONLY);
    write(fd, buf, sizeof(buf));  /* scrittura */
    write(fd, buf, sizeof(buf));  /* scrittura */
    close(fd);
}
```

Tipi di file

In astratto, un file è un insieme ordinato di record, accessibili in sequenza (file sequenziali) o in modo arbitrario (file relativi, file ad accesso diretto)

Dal punto di vista fisico esistono almeno 3 modi di rappresentare i file in un sistema operativo:

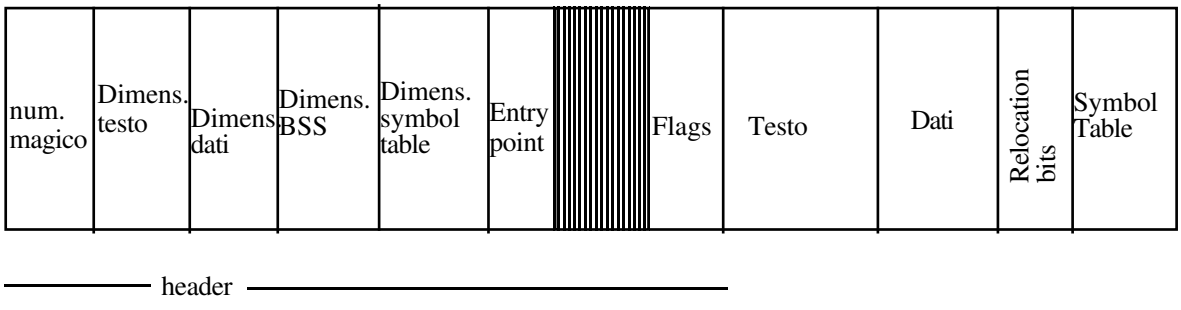
a) **sequenza di byte** (es.: Unix)

b) **sequenza di record di lunghezza fissa** (es.: CP/M)

In entrambi i casi, gli elementi di queste sequenze non possono essere inseriti o cancellati dal mezzo di un file (in pratica sono file sequenziali)

c) **albero di blocchi di disco**; ogni blocco contiene *n* record con chiave
Nuovi blocchi possono essere inseriti in qualsiasi posto dell'albero;
se viene aggiunto un record ad un blocco pieno, questo viene diviso in 2 blocchi, ciascuno dei quali viene aggiunto all'albero nell'ordine corretto
Questo metodo viene utilizzato in molti mainframe:
viene chiamato *ISAM* (*Indexed Sequential Access Method*)

La struttura di base può supportare qualsiasi sovrastruttura; per es., un file eseguibile in Unix è una sequenza di caratteri che ha una sovrastruttura particolare:



Nomi di file

L'attributo più importante di un file è il suo *nome*, che ne permette l'accesso da parte dei processi

Tutti i sistemi operativi cercano di supportare nomi *indipendenti dai dispositivi,* cioè a rendere la modalità d'accesso omogenea dovunque sia effettivamente memorizzato il file

In Unix un file system (fisicamente: un disco) può essere montato ovunque nell'albero dei file: comunque si accede ad ogni file mediante un **cammino**, senza riguardo al tipo di dispositivo che memorizza il file system

In MS-DOS l'utente deve invece specificare **esplicitamente** qual è il dispositivo in cui si trova il file (eccetto che nel caso del dispositivo attivo)

La maggior parte dei sistemi operativi ha molti tipi distinti di file: in MSDOS si distinguono per l'estensione:

`.sys` denota file di sistema,
`.exe` e `.com` file eseguibili,
`.txt` file di testo,
`.bas` file BASIC, ecc.

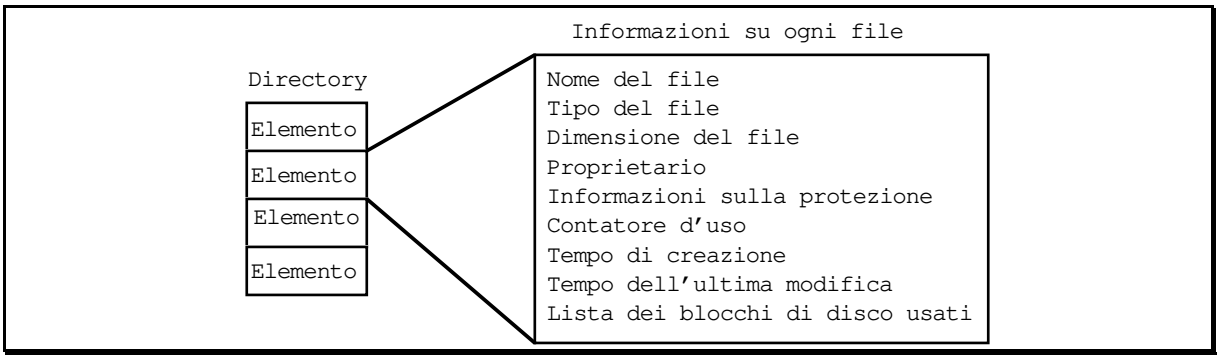
Unix ha file normali, directory, e file speciali (a blocchi e a caratteri)

- I file **normali** contengono dati e programmi
- I **directory** contengono coppie: <stringa,intero>; la stringa è il nome di un file, l'intero è un puntatore al suo descrittore
- I file **speciali** (a blocchi, a caratteri) rappresentano dispositivi o canali di comunicazione (pipe)

Nomi e directory

Per dare struttura al file system, in molti sistemi è possibile usare "contenitori" detti **directory** (che spesso sono essi stessi file)

Un *directory* contiene un certo numero di record, che rappresentano ciascuno un file; ciascun record elenca gli attributi del file, oppure punta ad un descrittore che li contiene (questa è la soluzione di Unix)



Quando un file system viene organizzato ad albero, come in Unix, esistono 2 metodi per specificare il nome del file, che di solito coesistono:

Nel primo metodo, ad ogni file viene assegnato un **cammino assoluto** che consiste del cammino nell'albero del directory radice al file

Esempio:

```
/etc/passwd
```

L'altro tipo di nome è il **cammino relativo**,
che presuppone il concetto di *directory di lavoro* (o *attivo*)

Esempio:

```
../passwd
```

In Unix, tutti i cammini che iniziano con uno slash sono assoluti;
tutti gli altri sono relativi al directory di lavoro

Nota: i nomi di file che iniziano con . sono invisibili per il comando `ls` "semplice". Per vedere i file "invisibili" occorre dare il comando `ls -a`

Operazioni sul file system

Vediamo alcune delle principali operazioni astratte che possono essere compiute su file (sistemi operativi diversi avranno nomi diversi per tali operazioni)

Operazioni su file

CREATE	crea un file vuoto e ne definisce alcuni attributi
DELETE	cancella il file dal file system
OPEN	porta in memoria il descrittore del file
CLOSE	elimina il descrittore dalla tabella dei descrittori
READ	a partire dalla posizione corrente, leggi n byte da un file e copiali in una certa area di memoria
WRITE	a partire dalla posizione corrente, scrivi n byte su un file, prendendoli da una certa area di memoria
APPEND	esegui una WRITE a partire dalla fine del file
SEEK	sposta arbitrariamente il puntatore della posizione corrente entro il file

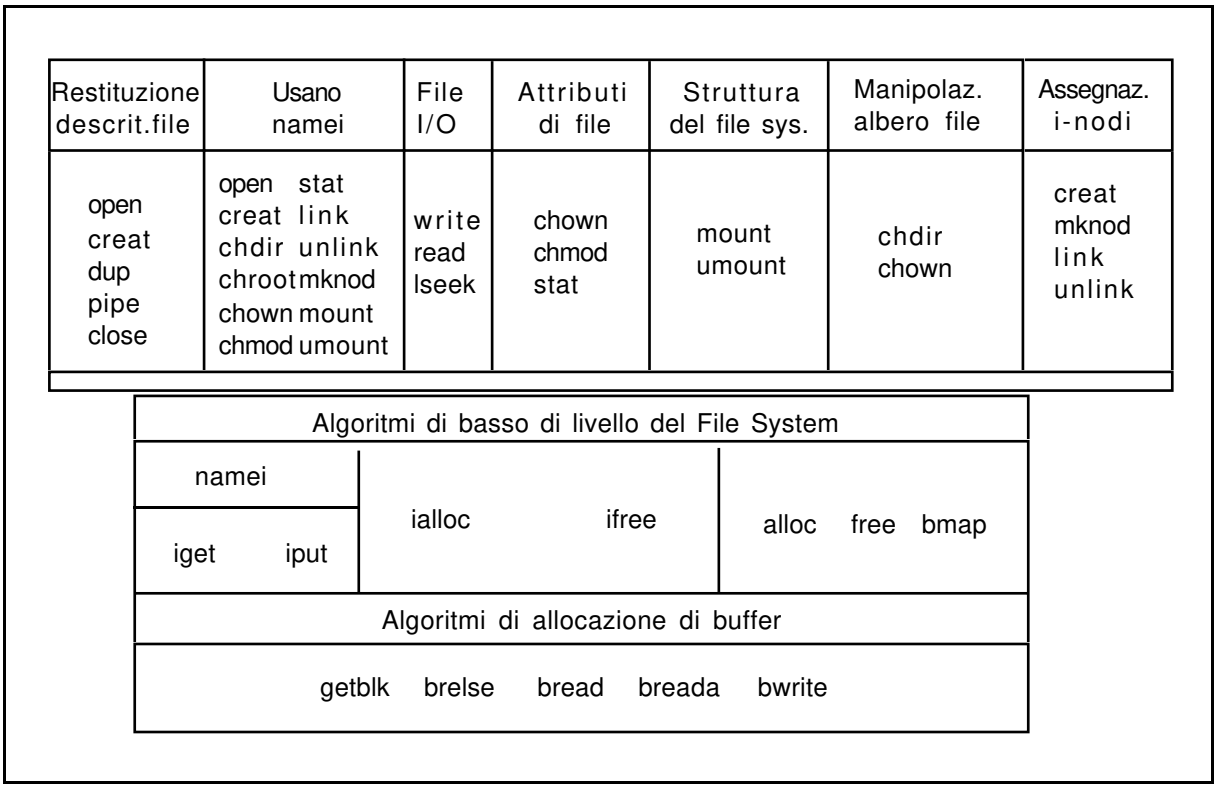
GET ATTRIBUTES leggi il descrittore
SET ATTRIBUTES modifica il descrittore
RENAME cambia il nome del file

Operazioni su directory

CREATE	crea un directory contenente due elementi: . e ..
DELETE	cancella un directory vuoto
OPENDIR	porta in memoria centrale il descrittore del file directory
CLOSEDIR	elimina dalla memoria centrale il descrittore del directory
READDIR	leggi il contenuto del file directory
RENAME	cambia il nome del file directory
LINK	condividi un file di un altro directory
UNLINK	elimina un file da questo directory

Chiamate di sistema del file system (Unix)

La struttura delle chiamate di sistema in Unix è raffigurata dallo schema seguente



Il file system di Unix

Il file system principale si chiama anche `root` (è sempre allocato sul disco RAM, in memoria centrale) e di solito contiene:

`/bin`	– contiene i più importanti programmi in binario
`/dev`	– contiene i file speciali per i programmi di I/O
`/etc`	– contiene vari programmi per la gestione del sistema
`/lib`	– contiene alcuni programmi di libreria
`/tmp`	– usato per gestire file temporanei
`/user`	– usato per montare altri file system
`/usr`	– qui viene montato il disco di sistema

In `/bin` risiedono i programmi usati più di frequente come `cat`, `cp` e `ls`, più altri programmi, come `login` e `sh`

Il directory `/dev` contiene i file speciali per i dispositivi di I/O:

`/dev/ram`	– il disco RAM
`/dev/mem`	– la memoria assoluta
`/dev/kmem`	– la memoria riservata al nucleo
`/dev/null`	– il dispositivo nullo
`/dev/fd0`	– il dispositivo di dischetto 0
`/dev/fd1`	– il dispositivo di dischetto 1
`/dev/hd0`	– il dispositivo minore del disco rigido 0
`/dev/hd1`	– il dispositivo minore del disco rigido 1
`/dev/lp`	– la stampante
`/dev/tty0`	– il terminale 0 (console)

Tutti i file speciali hanno un *numero di dispositivo maggiore* che si riferisce alla classe del dispositivo e un *numero di dispositivo minore* che specifica un particolare dispositivo tra quelli della stessa classe

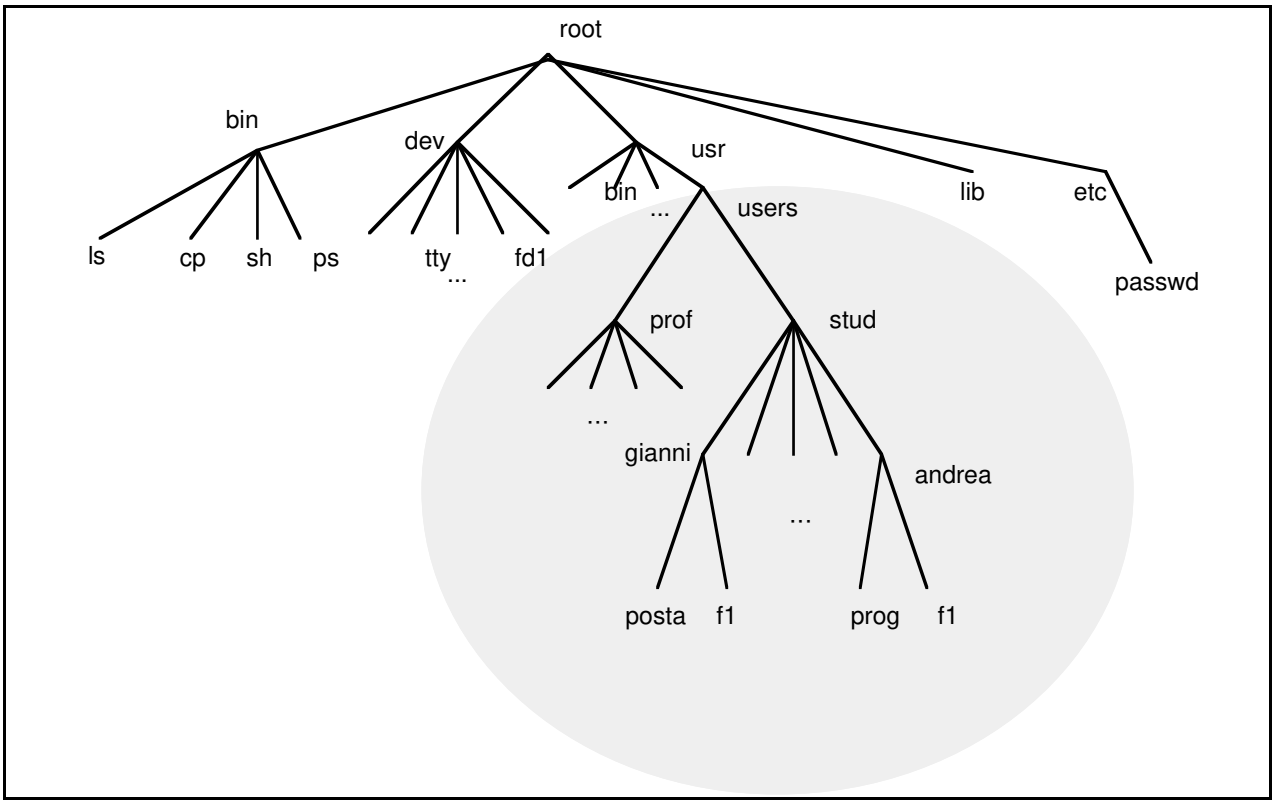
Il file system

Dal punto di vista del programmatore, i processi agiscono e comunicano unicamente tramite il file system

Il file system è un oggetto rappresentato fisicamente su dispositivi:

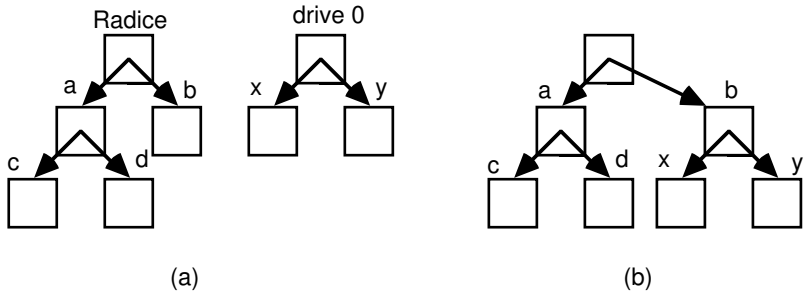
- dischi rigidi
- dischetti
- CD-ROM
- disco RAM

Di solito sul disco RAM risiede la parte alta del file system
(`/bin, /dev, /etc, ...`)
montata automaticamente al momento del bootstrapping



Montaggio di un file system

Un file system residente su un disco per essere reso accessibile va *montato*



(a) Prima del montaggio, i file sul drive 0 non sono accessibili
(b) Dopo il montaggio fanno parte del file system gerarchico

Alcuni file sono *file speciali* e rappresentano dispositivi

Esistono due tipi di file speciali:

I file speciali a **blocchi** vengono usati per i dispositivi costituiti da una serie di blocchi indirizzabili in modo diretto, come i dischi

I file speciali a **carattere** sono usati per i modelli di dispositivo costituiti da sequenze di caratteri, come i terminali, le stampanti e la interfacce di rete

File condivisi

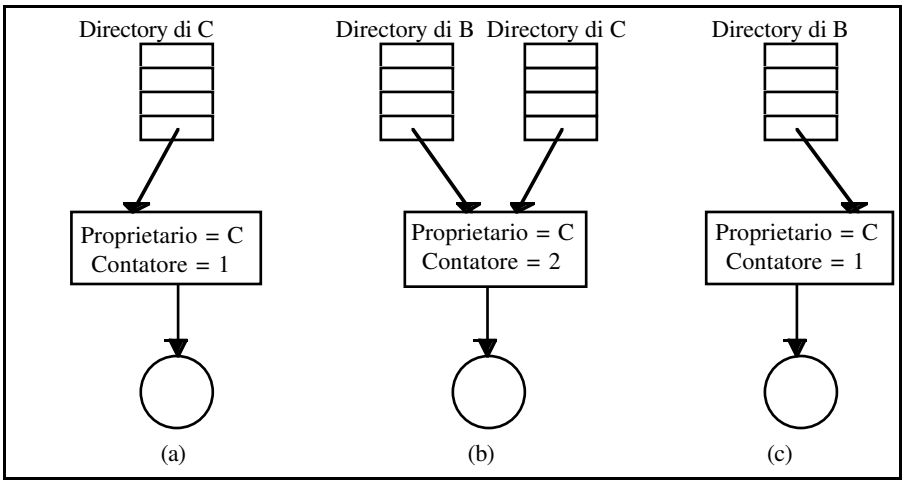
Spesso è conveniente che uno stesso file appaia simultaneamente in più directory, per esempio appartenenti a utenti diversi (condivisione)

In Unix:
La connessione fra un directory ed un file si chiama *legame* (*link*)

(a) Situazione prima della creazione del legame

(b) Dopo la creazione del legame

(c) Dopo che il proprietario originale rimuove il file



Questo meccanismo funziona se i due directory ed il file condiviso sono sullo stesso disco; nel caso di dischi diversi si usano i *link simbolici*, che sono veri e propri file

In MacOS 7.0:
il meccanismo degli *alias* è simile a quello dei link simbolici in Unix

Gestione dello spazio del disco

Sono possibili le seguenti strategie per la memorizzazione di un file di *n* byte:

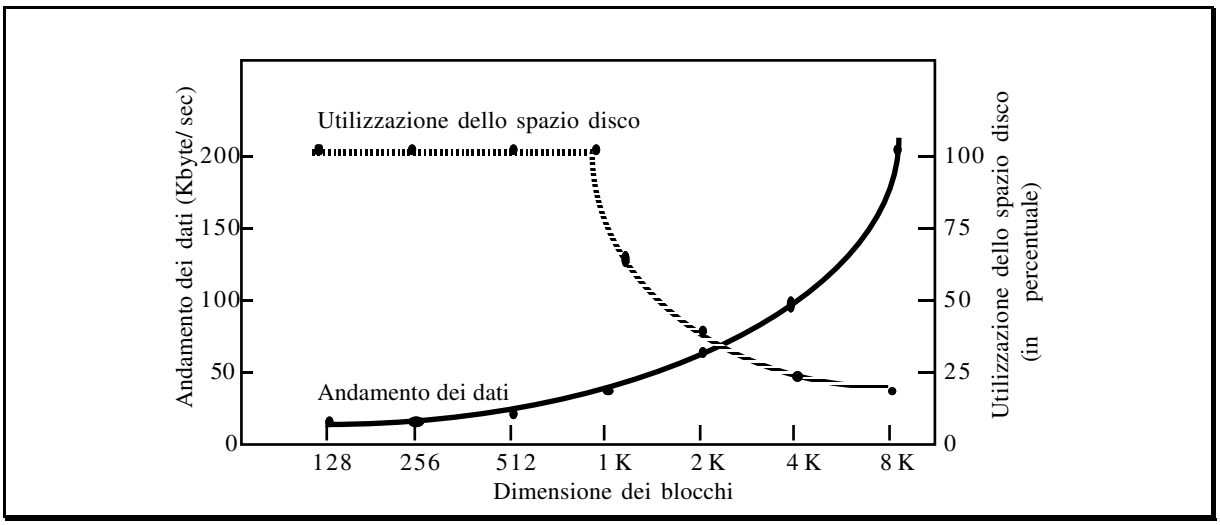
1) vengono allocati *n* byte consecutivi di spazio del disco
2) il file viene diviso in un lista di blocchi non necessariamente contigui
3) alla lista si associa un indice
4) invece dell'indice, si usa un i-nodo per gestire la lista dei blocchi

Immagazzinare un file in una sequenza di byte contigui presenta il problema che se il file cresce deve essere spostato sul disco. Quindi conviene partizionarlo in blocchi; ma quanto devono essere grossi? un settore? una traccia? un cilindro?

È una decisione che va riferita alla particolare situazione

La curva continua (scala a sinistra) dà la velocità di accesso ai dati sul disco in funzione della dimensione dei blocchi (lungh. media dei file = 1K)

La curva tratteggiata (scala a destra) dà l'efficienza dello spazio sul disco: notare il conflitto tra tempo e spazio

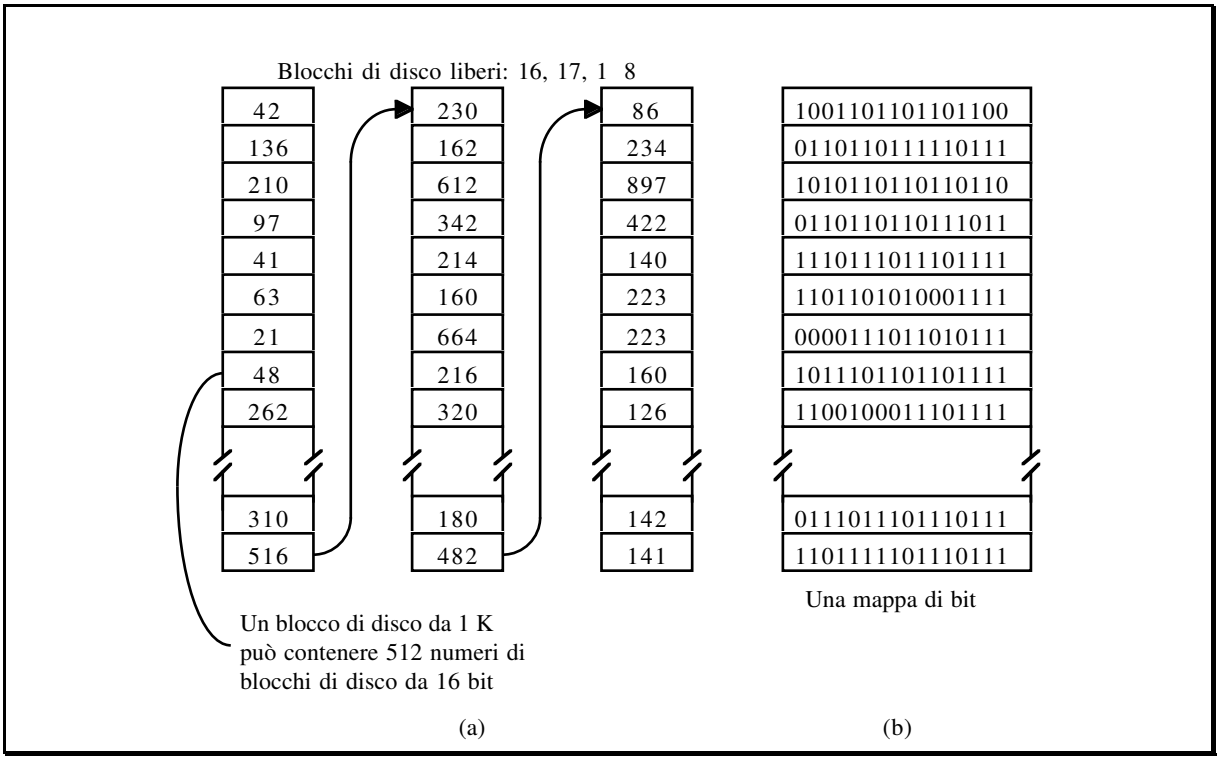


Gestione dei blocchi liberi

Per la gestione dei blocchi liberi vengono normalmente usati 2 metodi:

(a) **lista concatenata** di blocchi; ogni blocco contiene id. di blocchi liberi

(b) **mappa di bit**; un disco con *n* blocchi richiede una mappa con *n* bit



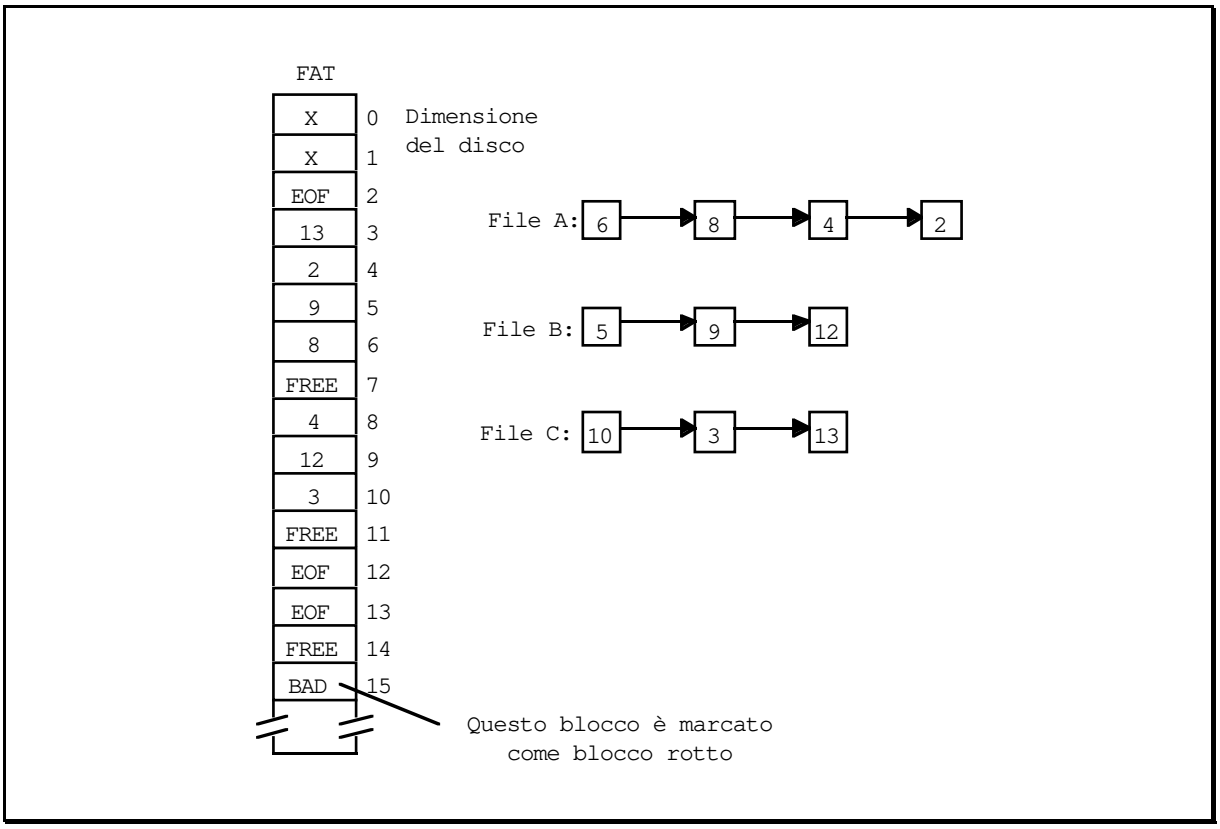
Memorizzazione dei file con una lista

Il file system server deve tenere traccia dei blocchi di ciascun file

Un metodo accettabile è la memorizzazione di un file come una lista

Ogni blocco da 1024 byte contiene 1022 byte di dati ed un puntatore da 2 byte al blocco successivo nella catena.

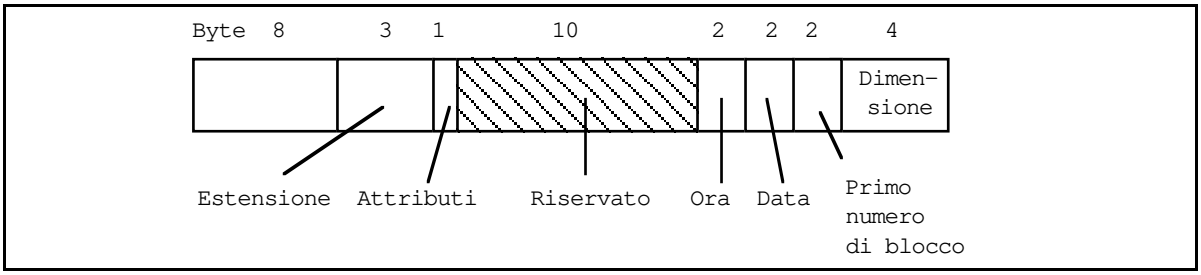
Lo schema di allocazione con lista connessa è usato da MS-DOS
Le caselle 0 ed 1 si usano per specificare la dimensione del disco
I codici EOF e FREE sono usati per Fine del File e Elemento Libero



Il file system di MS-DOS

Associata ad ogni disco MS-DOS c'è la *tabella d'allocazione dei file* (FAT): ha un elemento per ogni blocco del disco

In MS-DOS un elemento di directory è lungo 32 byte e contiene il nome del file ed il numero del suo primo blocco, che può essere usato come indice interno per la FAT, per trovare il numero del secondo blocco, ecc.



L'elemento nel directory di ogni file dà il numero del primo blocco del file; l'elemento associato a questo blocco nella FAT contiene il numero del blocco successivo

Questo schema è stato progettato originariamente per dischetti da 320Kb che usavano blocchi di 1Kb: i numeri dei blocchi sono di 12 bit, quindi una FAT di 320 elementi necessita di 480 byte, che entra bene in un settore di 512 byte

Quando l'IBM decise di ridimensionare il dischetto a 360Kb, (versione DOS 2.0), la FAT crebbe a 540 byte: non rientrava più in un settore

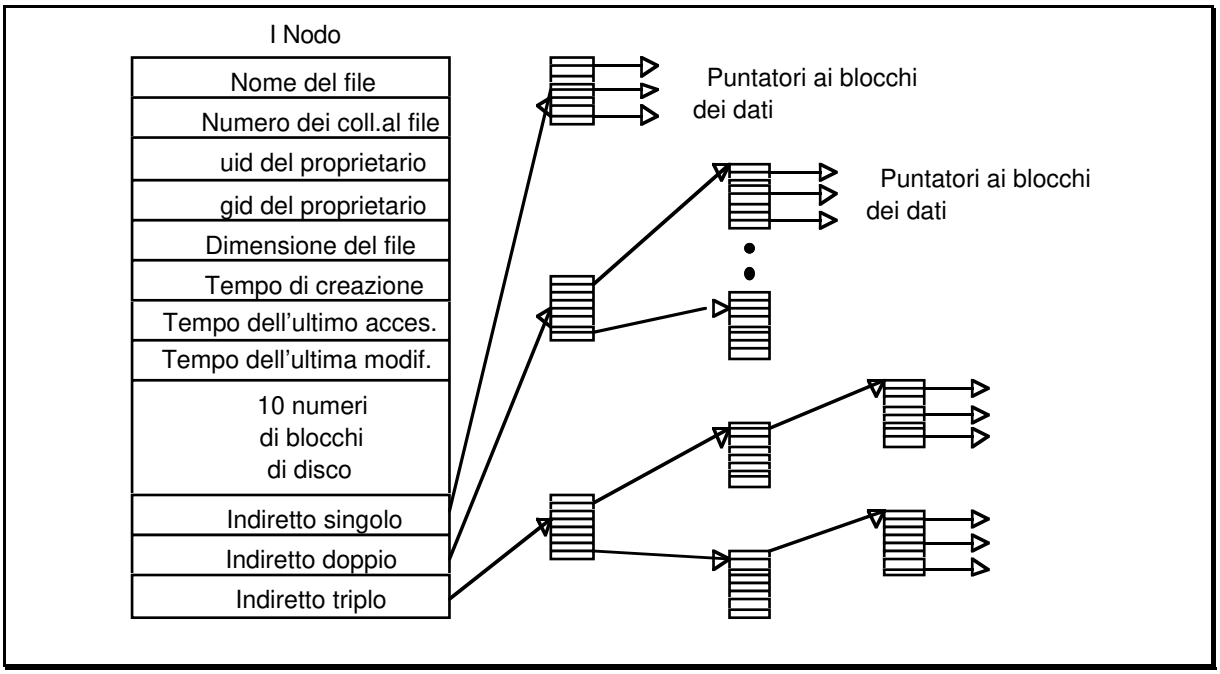
Quando furono introdotti i dischi rigidi con più di 4096 blocchi, il numero di blocco da 12 bit divenne inadeguato, e si cambiò di nuovo la FAT

Con dischi grandi, questo schema diventa meno attraente: il problema è che i puntatori per tutti i file sono mescolati nell'unica FAT: questo significa che potenzialmente è necessario conoscere tutta la FAT, anche se è aperto un solo file

I-nodi (Unix)

Un metodo migliore tiene in posti differenti le liste dei blocchi per ogni file

Unix associa ad ogni file un descrittore chiamato **i-nodo**, che contiene gli attributi del file più altre informazioni sulla sua rappresentazione in memoria secondaria



Struttura di un i-nodo

Il descrittore contiene 13 puntatori: 10 puntano direttamente blocchi del file su di disco mentre 3 puntano blocchi "indiretti" (blocchi di puntatori): per file lunghi fino a 10 blocchi, tutti gli indirizzi sono conservati nell'i-nodo

Quando un file cresce oltre i 10 blocchi di disco, viene acquisito un blocco libero cui punta il puntatore indiretto *singolo*; tale blocco viene usato per contenere puntatori ai blocchi del disco

Con un blocco da 1Kb ed indirizzi di 32 bit, un blocco indiretto singolo può contenere 256 indirizzi (Questo schema è sufficiente per file fino a 266 blocchi: 10 nell'i-nodo, 256 nel blocco indiretto)

Il blocco indiretto *doppio* è sufficiente fino a $266 + 256^2 = 65802$ blocchi

Per file più lunghi di 64Mb, viene usato il puntatore indiretto *triplo* per puntare ad un blocco con 256 puntatori a blocchi indiretti doppi

File più lunghi di 16 Gb (gigabyte) non possono essere gestiti

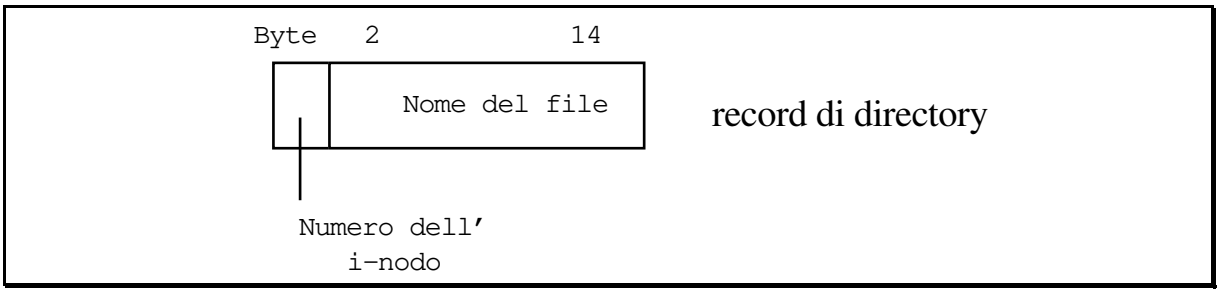
La forza dello schema Unix è che i blocchi indiretti vengono usati solo quando sono necessari: per file sotto i 10Kb non sono necessari

Anche per il file più lungo sono necessari al più 3 riferimenti al disco per localizzare l'indirizzo di ogni byte del file

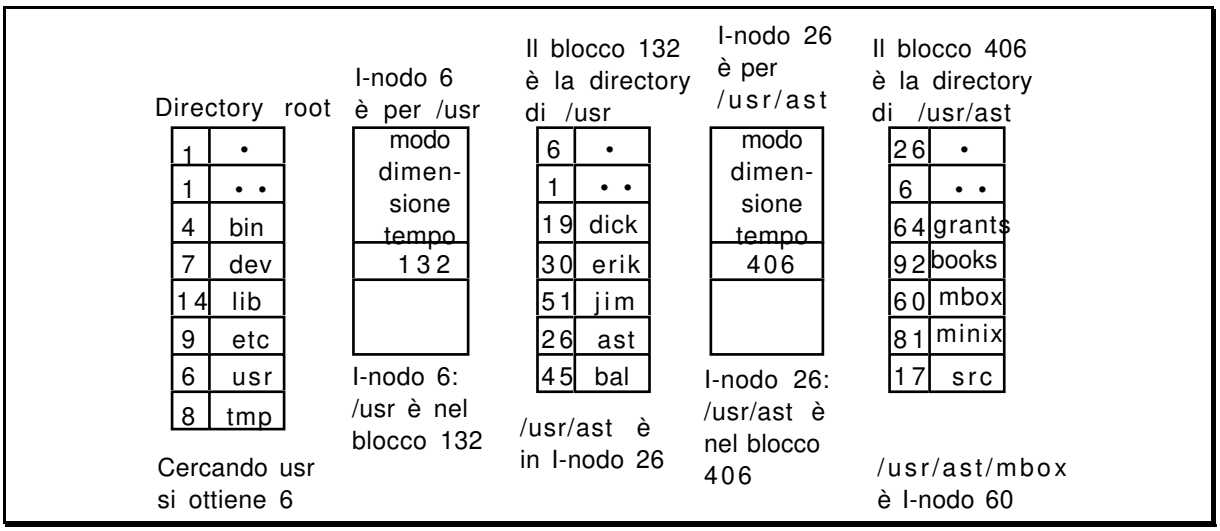
Struttura del directory in Unix

La struttura dei directory usata in Unix è semplice: i directory sono file e contengono un numero arbitrario di record (16 byte); *Ogni record contiene il nome del file ed il suo numero di i-nodo;*

Tutte le informazioni sul tipo, la dimensione, il tempo di creazione, la proprietà ed i blocchi di disco usati sono contenute nell'i-nodo



Vediamo come si cerca un file che ha il nome assoluto */usr/ast/mbox*



Il file system recupera `root` (il directory radice) (in Unix il suo i-nodo è posto in una posizione precisa sul disco); quindi si cerca lì la prima componente del cammino, *usr*, per trovare l'i-nodo del file */usr*

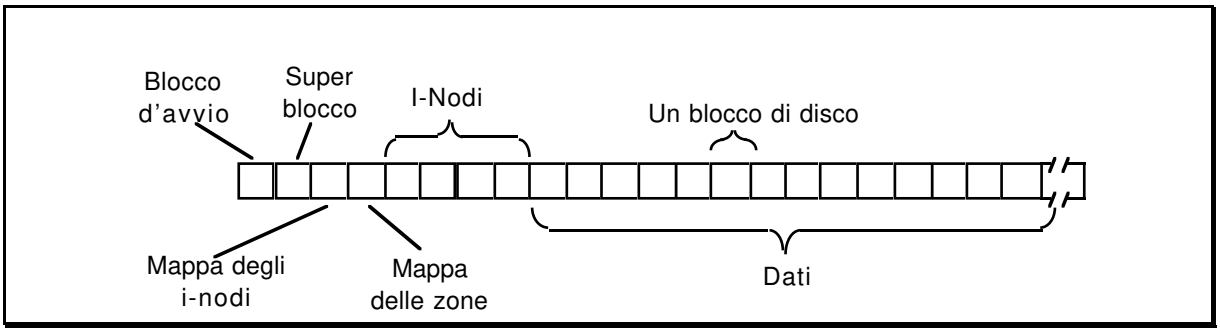
Dal suo i-nodo, il sistema individua il directory di `/usr` e cerca in tale directory la componente successiva, `ast`, e così via; tale i-nodo viene letto e mantenuto in memoria fino a che il file non viene chiuso

Rappresentazione di un file system su disco

Un file system è una entità logica autosufficiente con i-nodi, directory, e blocchi di dati

Può essere memorizzato su un dispositivo a blocchi, come un dischetto o (una partizione di) un disco rigido

Esempio:
La figura illustra questa disposizione per un dischetto da 360Kb con 127 i-nodi e blocchi di dimensione di 1Kb
(cioè due settori consecutivi da 512 byte trattati come un singolo blocco)



Ogni file system comincia con un blocco d'avvio

Quando il computer viene acceso, l'hardware legge il blocco d'avvio nella memoria ed esegue un salto alla prima parola

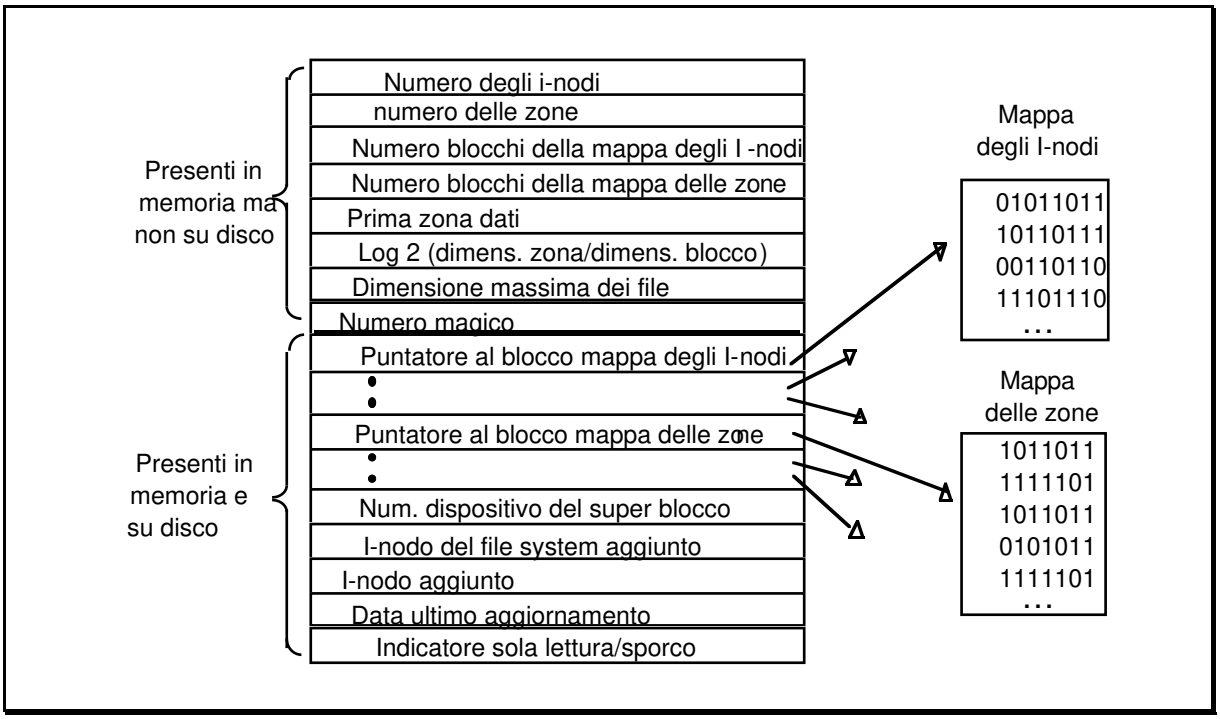
Non tutti i gestori di disco possono essere usati come dispositivi d'avvio, ma per mantenere una struttura uniforme ogni dispositivo ha un suo blocco d'avvio

I superblocchi

Ogni super blocco contiene le informazioni proncipali di descrizione della struttura di un file system

Dato un file system su un dischetto, la funzione principale del relativo super blocco è di dire al gestore del file system quanto sono grandi i vari componenti del dischetto

Data la dimensione del blocco ed il numero del suo i-nodo, è facile calcolare la dimensione della mappa degli i-nodi ed il numero di blocchi con i-nodi



Uso dei superblocchi

Quando viene avviato Unix, viene letto il super blocco del disco del dispositivo che contiene la radice e messo in una tabella in memoria

Quando vengono montati altri file system, anche i loro super blocchi vengono portati in memoria

Nota: i tentativi di montare un file system non in formato Unix verranno rifiutati dalla chiamata `mount`, che controlla il super blocco per vedere il numero magico ed altre cose

La tabella dei super blocchi in memoria centrale contiene alcuni campi che non sono presenti nei superblocchi dei dischi:

dispositivo da cui provengono,
flag che dice se possono essere avviati in sola lettura,
flag che dice se la versione in memoria viene modificata

Costruzione di un file system
Il programma di servizio `mkfs` serve per costruire un file system vuoto (oppure con una struttura di directory mediante un file prototipo) su un disco formattato

Esempio:

```
mkfs /dev/fdl 360
```

questo comando costruisce un file system con 360 blocchi vuoti su dischetto formattato inserito nel driver 1

Gestione dei link (puntatori a file)

La realizzazione dei link può avvenire in 2 modi

```
ln dir1/file1 dir2/file2
ln –s dir1/file1 dir2/file2
```

- i blocchi di disco vengono elencati da una piccola struttura dati associata al file; i directory hanno un puntatore a questa struttura dati (approccio usato in Unix, dove la struttura dati è l'i-nodo)
- *B* si lega ad uno dei file di *C* con la creazione da parte del sistema di un nuovo file, di tipo *link*, e l'inserzione di questo file nel directory di *B* (approccio chiamato *collegamento simbolico)*

Ognuno di questi metodi ha i suoi punti deboli

Nel primo metodo, al momento in cui *file2* si lega a *file1* la creazione del legame incrementa il contatore dei legami nell'i-nodo del file, in modo che il sistema sappia quanti directory contengono un elemento che punta al file

Purtroppo in questo modo è costoso cercare tutti i directory che condividono un file, se occorre cancellarlo

Con i legami simbolici questo problema non sorge perché solo il vero proprietario ha un puntatore all'i-nodo; gli altri directory collegati al file conoscono solamente il cammino d'accesso, non il puntatore

Il problema con i legami simbolici è il lavoro extra richiesto: il file che contiene il cammino deve essere letto, quindi il cammino deve essere analizzato e seguito, componente per componente

Inoltre è necessario un i-nodo in più per ogni legame simbolico, e un blocco di disco in più per memorizzare il cammino

Efficienza del file system

L'accesso al disco è molto più lento dell'accesso alla memoria centrale

La tecnica usata più comunemente per ridurre gli accessi al disco implica l'uso della memoria cache dei blocchi (*buffer cache*)

Una *memoria cache* è una collezione di blocchi che appartengono logicamente al disco, ma che vengono conservati in RAM

Vari algoritmi possono essere usati per gestire la cache;

• il modo di DOS era di leggere tutti i blocchi "vicini" a quello da leggere (variabile buffers in config.sys)

• un altro modo molto comune è controllare tutte le richiese di lettura per vedere se il blocco necessario a soddisfare la richiesta è nella cache
Se sì, la richiesta di lettura è soddisfatta senza accessi al disco
Se no, il blocco va in cache e poi la richiesta viene soddisfatta
Altre richieste per tale blocco possono essere ora soddisfatte direttamente dalla cache

Quando un blocco deve essere caricato in una cache piena, qualche blocco deve essere rimosso e riscritto sul disco se è stato modificato da quando è stato introdotto

Una strategia utile è quella LRU

Sono state studiate anche tecniche predittive

Consistenza del file system

Il gestore del file system legge e scrive blocchi **in modo virtuale**, ovvero usando come buffer di transito la *cache*: se il sistema collassa prima che tutti i blocchi modificati siano stati scritti nel disco (da `sync`), il file system può restare in uno stato inconsistente

Alla riaccensione del sistema il file system server può effettuare almeno due tipi di controllo di consistenza: dei blocchi e dei file

Per controllare la consistenza dei *blocchi* si costruisce una tabella con 2 contatori per blocco, entrambi inizialmente a 0; Il primo contatore registra quante volte il blocco è presente in un file; il secondo registra quante volte è presente nella lista libera (o nella mappa dei blocchi liberi)

Sii leggono tutti gli i-nodi: partendo da un i-nodo, è possibile costruire una lista di tutti i blocchi usati; non appena viene letto un numero di blocco, il suo contatore nella prima tabella viene incrementato
Si esamina la lista libera per trovare tutti i blocchi liberi che non sono in uso: ogni occorrenza di un blocco nella lista libera risulta nel contatore della seconda tabella, che viene incrementato

Se il file system è consistente, ogni blocco avrà 1 o nella prima o nella seconda tabella ma non in entrambe

Dopo un crash, un blocco potrebbe essere marcato: 0 0. I blocchi mancanti sprecano spazio e riducono la capacità del disco: il controllo di consistenza del file system li deve aggiungere alla lista libera

Può capitare che un blocco risulti 2 o più volte nella lista libera; anche qui la soluzione è semplice: ricostruire la lista libera

Il caso peggiore è che uno stesso blocco sia presente in 2 o più file: l'azione corretta è di allocare un blocco libero, copiarci il contenuto del blocco difettoso, ed inserire la copia in uno dei due file; la struttura del file system diventa consistente, ma il file resta danneggiato: l'errore va riportato all'utente

Consistenza del file system

Il gestore del file system quando controlla la consistenza verifica anche l'organizzazione dei *directory*: usa 2 tabelle di contatori, ma questi riguardano i file, e non i blocchi

Il gestore parte dalla radice e discende ricorsivamente l'albero, ispezionando ogni directory; di ogni file in ogni directory, incrementa il contatore per l'i-nodo; quando ha terminato la visita, si ha una lista indiciata con i numeri degli i-nodi, che conta quanti directory puntano a quell'i-nodo; quindi confronta tali numeri con il contatore di legami memorizzato nell'i-nodo

In un file system consistente, entrambi i contatori devono essere uguali; possono capitare 2 tipi di errori: il contatore dei legami nell'i-nodo può essere 1) troppo alto o 2) troppo basso

Se il contatore dei legami è maggiore del numero di elementi di directory, allora anche se tutti i file vengono rimossi dal directory, il contatore sarà ancora diverso da 0 ed l'i-nodo non sarà rimosso
Questo non è un errore grave, ma fa sprecare spazio sul disco per i file che non sono in alcun directory; può essere risolto mettendo nel contatore dei legami nell'i-nodo il valore giusto

L'altro errore è potenzialmente catastrofico: se 2 elementi di directory sono legati allo stesso file, ma l'i-nodo dice che ce n'è uno solo, quando un elemento viene rimosso il contatore nell'i-nodo va a 0; allora FS marca i blocchi del file come inutilizzati e li libera: quindi in uno dei directory si ha un puntatore ad un i-nodo inutilizzato, i cui blocchi possono essere riassegnati ad altri file: bisogna assegnare al contatore di legami il numero attuale degli elementi di directory

Altri controlli euristici sono possibili: per es., i directory hanno un formato definito, con un numero di i-nodo ed un nome ASCII; se un numero di i-nodo è maggiore del numero di i-nodi sul disco, il directory è stato danneggiato

Sicurezza e protezione del file system

Sicurezza:
- eventi naturali: (incendi, terremoti, animali, ecc.)
- errori hw e sw
- errori umani
- intrusioni maliziose

Famosi problemi di sicurezza
- programmi maliziosi
- virus e cavalli di Troia
- internet worm
- covert channels

Meccanismi di gestione della sicurezza: basati su sistemi di identificazione certa dell'utente (password, one time passwd, identificaz. fisica)

Meccanismi di protezione nel sistema operativo:

- domini di protezione
- liste di controllo degli accessi
- capabilities

Modelli di protezione

SICUREZZA

I file system spesso contengono informazioni di valore per i loro utenti. La protezione di queste informazioni contro usi non autorizzati è un importante compito del file system

Sicurezza vuol dire difesa dai rischi: ne esistono moltissimi.
Due dei rischi più importanti sono la perdita di dati e le intrusioni

Alcune delle cause più comuni per la perdita dei dati sono:

–*Errori umani*: inserimento dati scorretti, dischi o nastri montati erroneamente, esecuzione di programmi sbagliati, dischi o nastri smarriti.

–*Caso*: incendi, inondazioni, terremoti, guerre, rivolte, dischetti o nastri o carte magnetiche rosicchiate dai topi

–*Errori hardware o software*: malfunzionamenti della CPU, dischi o nastri illeggibili, errori nelle telecomunicazioni, errori nei programmi

Molti di questi rischi possono essere superati conservando copie di riserva aggiornate, preferibilmente lontano dai dati originali

Intrusioni

Gli intrusi *passivi* vogliono solo leggere un file che non sono autorizzati a leggere

Gli intrusi *attivi* sono più maliziosi: vogliono fare cambiamenti non autorizzati ai file, per esempio vogliono distruggerli

Alcune categorie comuni di intrusioni:

– Intrusioni casuali da parte di utenti inesperti

– Intrusioni per sfida

– Spionaggio commerciale o militare

Deve essere chiaro che cercare di impedire al KGB di rubare segreti militari è diverso dal cercare d'impedire ad uno studente d'inserire un divertente messaggio-del-giorno nel sistema. La quantità di sforzi che uno impiega per la sicurezza e la protezione dipendono chiaramente da quale nemico pensa di avere.

Un altro aspetto della sicurezza è la *privacy*: proteggere gli individui dall'abuso di informazioni su di essi. Questo porta a questioni legali e morali.

Famose imperfezioni della sicurezza

Il comando Unix *lpr* ha l'opzione che rimuove il file dopo che è stato stampato. In una versione di Unix era possibile utilizzare *lpr* per stampare, e rimuovere dal sistema, il file delle parole d'accesso.

Un'altra imperfezione in Unix era legata al comando *mkdir*, che normalmente è posseduto da root (superuser).
L'utente prima creava l'i-nodo per un directory *d* con la chiamata di sistema `mknod`, e quindi cambiava il proprietario di *d* dalla sua uid effettiva (cioè root) alla uid reale (la uid dell'utente).
Se il sistema era sovraccarico e quindi lento, era talvolta possibile rimuovere rapidamente l'i-nodo del directory e creare un link con il file delle passwd dopo la `mknod` ma prima di `chown`. Quando *mkdir* faceva `chown` rendeva l'utente proprietario del file delle passwd

Molti sistemi operativi sono colabrodi dal punto di vista della sicurezza.

Il modo normale per controllare la sicurezza di un sistema è ingaggiare un gruppo di esperti, noto *gruppo di penetrazione*, per vedere se riescono ad aprire una breccia

Attacchi tipici alla sicurezza

Alcuni degli attacchi più comuni sono

– Richiedere risorse come pagine di memoria, blocchi di disco o segmenti di nastro e leggerli a livello fisico. Molti sistemi non li cancellano prima di allocarli: possono dunque essere pieni d'informazioni interessanti scritte dal proprietario precedente

–Tentare chiamate di sistema illegali, o chiamate di sistema legali con parametri illegali: può succedere qualcosa di interessante

–Iniziare il collegamento e quindi battere DEL, RUBOUT o BREAK a metà della sequenza di login. In alcuni sistemi, il controllo della passwd viene annullato ed il collegamento viene considerato terminato con successo!

– Cercare di modificare strutture del sistema operativo tenute nello spazio dell'utente. In molti sistemi, per aprire un file un programma costruisce una struttura dati che contiene il nome del file e molti altri parametri e li passa al sistema. Quando il file viene letto o scritto, il sistema talvolta aggiorna la struttura stessa. Cambiare questi campi può provocare una falla nella sicurezza

– Ingannare l'utente lanciando un programma che scrive il prompt "login:" sullo schermo e continua registrando la passwd

– Cercare dove i manuali dicono "Non fare X". Provare quante più variazioni è possibile di X.

– Convincere un sistemista a modificare il sistema in modo che salti i controlli di sicurezza per un certo userid

– Quando tutto fallisce, si può provare a corrompere la segretaria del direttore del centro di calcolo, che ha probabilmente facile accesso a tutti i tipi di informazioni, ed è di solito pagata poco.
Non bisogna sottovalutare i problemi provocati dal personale!

Principi di progettazione per la sicurezza

Saltzer e Schroeder (1975) hanno identificato i principi generali che possono essere usati come guida per la progettazione di sistemi sicuri

• il progetto del sistema deve essere pubblico. Assumere che l'intruso non conosca il funzionamento del sistema serve solo a deludere il progettista

• il default dovrebbe essere "nessun diritto di accesso". Gli errori in cui accessi legittimi vengono rifiutati saranno riportati molto più velocemente degli errori in cui accessi non autorizzati vengono permessi

• controllare dinamicamente l'autorizzazione all'accesso. Molti sistemi controllano i permessi quando viene aperto un file, e dopo non più. Questo significa che un utente che apre un file, e lo tiene aperto per settimane, continuerà ad avervi accesso, anche se nel frattempo il proprietario ha cambiato la protezione del file

• dare ad ogni processo il minor numero possibile di privilegi. Se un editor ha solo l'autorità per accedere a file che devono essere editati, un editor con un cavallo di Troia non sarà capace di fare molto danno

• il meccanismo di protezione deve essere semplice, e costruito nello strato più interno del sistema. Tentare di inserire meccanismi di sicurezza in un sistema esistente insicuro è quasi impossibile. La sicurezza, come la correttezza, non è una proprietà aggiungibile

• lo schema scelto deve essere psicologicamente accettabile. Se gli utenti sentono che la protezione delle loro risorse richiede troppa fatica, la eviteranno

Autenticazione

Molti schemi di protezione sono basati sull'assunzione che il sistema conosce gli identificatori di ogni utente. Il problema di identificare gli utenti quando si collegano è chiamato *autenticazione dell'utente*

Parola d'accesso (password)
La forma di autenticazione usata più diffusamente è richiedere all'utente di battere una parola d'accesso (*password*)

L'autenticazione con parola d'accesso è facile da aggirare

Non stupirà scoprire che mentre viene battuta la parola d'accesso, il computer non deve mostrare i caratteri che vengono battuti, per proteggerli da occhi indiscreti vicini al terminale. Meno ovvio è che la parola d'accesso non deve essere mai memorizzata nel computer in forma non crittografata; e neanche il centro di gestione del computer deve avere copie non crittografate. Tenere copie di parole d'accesso non crittografate in qualsiasi posto significa cercare guai

Domande personali
Una variante della password è che ogni utente definisce una lunga lista di domande e risposte che vengono immagazzinate nel computer in forma crittografata. Le domande possono essere scelte in modo che l'utente non debba scriversi le risposte. Interrogazioni tipiche sono:

- Chi è la sorella di Alice?
- In quale strada era la sua scuola elementare?
- Cosa insegnava la S.ra Crucioli?

Al momento del collegamento, il sistema propone una di queste domande. Altre domande possono essere fatte a caso durante il collegamento

Un'altra variante è la *risposta-sfida*: l'utente sceglie un algoritmo quando gli viene assegnato uno userid, ad esempio x^2. Quando l'utente si collega, il computer batte un argomento, diciamo 7, nel qual caso l'utente deve rispondere 49. L'algoritmo può essere differente la mattina o al pomeriggio, in giorni diversi della settimana, a terminali differenti e così via

Autenticazione

Identificazione fisica

L'utente usa una carta plastificata con una striscia magnetica o chip (*badge*). Le macchine che consentono di prelevare soldi di solito operano in questo modo

Un altro approccio è misurare caratteristiche fisiche difficili da imitare. Per esempio, un'impronta digitale o vocale possono essere analizzate da dispositivi speciali per verificare l'identità dell'utente

Contromisure

Le installazioni che devono essere affidabili e sicure spesso prendono misure per rendere più difficile l'ingresso non autorizzato

I collegamenti su linee telefoniche possono essere come segue: chi chiama fa il numero e si collega, ma dopo che il collegamento ha avuto successo, il sistema interrompe la connessione e richiama l'utente ad un numero concordato

Tutti i collegamenti devono essere registrati. Quando un utente si collega, il sistema deve riferire l'ora ed il terminale del collegamento precedente, in modo da poter scoprire eventuali intrusioni

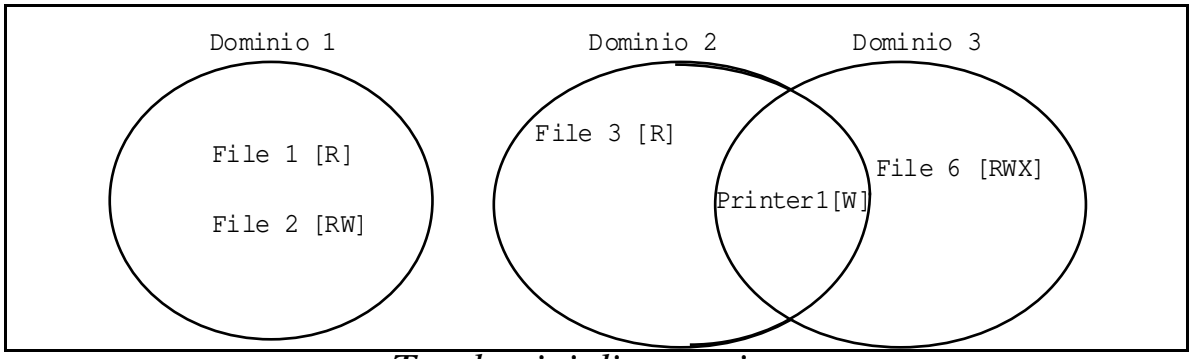
Il passo successivo è lasciare trappole per catturare l'intruso. Uno schema semplice è avere uno userid con passwd ovvia (ad esempio, userid: *guest*, passwd: *guest*). Quando qualcuno si collega usando questo userid, vengono subito avvisati i sistemisti

Domini di protezione

Un sistema di elaborazione contiene molti *oggetti* che devono essere protetti. Questi oggetti possono essere parti hw, come la CPU, segmenti di memoria, terminali, dischi o stampanti, o possono essere componenti software, come processi, file, o descrittori

Ogni oggetto ha un nome unico attraverso il quale può essere denotato, ed un insieme di operazioni che possono essere compiute su di esso

Gli oggetti, nei sistemi operativi, sono equivalenti a quello che sono nei linguaggi di programmazione i *tipi di dato astratto*



Tre domini di protezione

È necessario un meccanismo per impedire ai processi di accedere ad oggetti su cui non sono autorizzati ad operare. Inoltre, questo meccanismo deve anche rendere possibile di limitare l'attività di un processo ad un sottoinsieme delle operazioni legali che sono necessarie

Un *dominio* è un insieme di coppie (oggetto, diritti). Ogni coppia specifica un *oggetto* ed un sottoinsieme delle operazioni che possono essere eseguite su di esso. Un *diritto* in questo contesto specifica permesso di compiere una delle operazioni

La figura illustra tre domini, mostrando in ogni dominio gli oggetti ed i diritti [Read, Write, eXecute] disponibili su ogni oggetto. Si noti che *Printer1* è in due domini allo stesso tempo. Nonostante non sia mostrato in questo esempio, è possibile avere lo stesso oggetto in diversi domini, con diritti *differenti* in ogni dominio

Meccanismi di protezione dei file in Unix

Ad ogni file è associata una stringa che determina una serie di diritti d'accesso

```
d     r   w    x     r   w   x     r   w   x
      proprietario   gruppo         altri
```

primo carattere: tipo del file (directory, speciale, normale, ecc.)

9 bit: tre classi (proprietario, gruppo del proprietario, altri utenti)

	file normale	directory
r	lettura	lettura del contenuto
w	scrittura	inserimento e cancellazione di file
x	esecuzione	accesso con `cd`

proprietario	(u)	Chi ha creato il file
gruppo	(g)	Gruppi cui appartiene il proprietario
altri	(o)	Tutti gli altri utenti

Esempio:

```
$> ls -la
-rwxr--r-- root   .login  13.00  24-5
-rwxrwxr-- root   FILE1   11.22  12-4
-rwxr--rwx andrea FILE2   13.00  04-1
-rwxr----x root   FILE3   13.00  22-5
drwxr--r-x root   POSTA   13.00  21-3
```

I diritti possono essere cambiati solo dal proprietario o dal super-utente (col comando `chmod`)

Esempio: `chmod go-rwx POSTA`

Domini di protezione

Ad ogni istante ogni processo gira avendo accesso ad un preciso dominio di protezione. In altre parole, c'è qualche collezione di oggetti a cui può accedere, e per ogni oggetto ha un insieme di diritti

In Unix il dominio di un processo è definito dalle sue uid e gid. Data la coppia (uid, gid) di un processo, è possibile fare una lista completa di tutte le risorse (cioè i file, inclusi i file speciali di I/O) cui può accedere in lettura, scrittura o esecuzione

Due processi con la stessa coppia (*uid*,*gid*) avranno accesso alla stessa collezione di oggetti. I processi con valori differenti di (*uid*,*gid*) avranno accesso ad insiemi differenti di file, nonostante in molti casi questi possano sovrapporsi in misura sostanziale

In Unix ogni processo ha due metà: la parte dell'utente e la parte del nucleo. Questa ha accesso ad un insieme differente di oggetti rispetto alla parte dell'utente. Per esempio, il nucleo può accedere a tutte le pagine in memoria fisica, all'intero disco ed a tutte le altre risorse protette. Quindi una chiamata di sistema provoca il cambiamento di dominio

Quando un processo fa una `exec` su di un file con il bit SETUID o SETGID ad uno, acquisisce una nuova uid o gid. Con una combinazione (*uid*,*gid*) differente, ottiene l'accesso ad un diverso insieme di file e di operazioni

Anche far girare un programma che include le chiamate `setuid` o `setgid` rappresenta un cambiamento di dominio

Matrice di protezione

Una questione importante è come il sistema tiene traccia dela relazione tra oggetti e domini: si può usare una matrice, con le righe per i domini e le colonne per gli oggetti. Ogni casella elenca i diritti che il dominio contiene per l'oggetto

Esempio: la matrice per la fig. del lucido 26 è la seguente

	Oggetti							
Dominio	File 1	File 2	File 3	File 4	File 5	File 6	Printer 1	Plotter 2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Data questa matrice ed il numero del dominio corrente, il sistema può sempre dire se è consentito l'accesso in un particolare modo ad un certo oggetto del dominio.

	Oggetti										
Dominio	File 1	File 2	File 3	File 4	File 5	File 6	Printer 1	Plotter 2	Dom. 1	Dom. 2	Dom. 3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Il cambiamento del dominio può essere facilmente incluso nel modello della matrice con l'operazione ENTER, considerando che il dominio è lui stesso un oggetto.
Nella seconda figura i tre domini sono essi stessi oggetti. I processi nel dominio 1 possono passare al dominio 2, ma una volta lì, non possono tornare indietro. Questa situazione modella l'esecuzione del programma SETUID in Unix (in questo esempio non è consentito nessun altro cambiamento di dominio)

Liste di controllo degli accessi

La matrice degli accessi non viene memorizzata tutta perché è grande e sparsa; i metodi pratici sono la memorizzazione della matrice per righe o per colonne: i due approcci sono sorprendentemente differenti

La memorizzazione per colonne consiste nell'associare ad ogni oggetto una lista ordinata contenente tutti i domini che possono accedere all'oggetto: la lista si chiama *lista di controllo degli accessi* (*ACL)*. Se fosse da realizzare in Unix, la via più semplice sarebbe mettere l'ACL di ogni file in un blocco di disco, ed includere il numero del blocco nell'i-nodo del file

Immaginiamo che le ACL siano utilizzate in un ambito Unix, dove un dominio è specificato come coppia (*uid*,*gid*). Assumiamo di avere 4 utenti, (cioè *uid*) Qui, Quo, Qua e Paperino, che appartengono, rispettivamente, ai gruppi *system*, *staff*, *student* e *student*. Alcuni file hanno le seguenti ACL:

```
File0: (Qui, *, RWX)
File1: (Qui, system, RWX)
File2: (Qui, *, RW-), (Qua, staff, R--), (Pap, *, R--)
File3: (*, student, R--)
File4: (Quo, *, ---), (*, student, R--)
```

Ogni elemento dell'ACL specifica, tra parentesi, una uid, una gid e gli accessi consentiti. Un asterisco significa "tutte le uid o le gid".

File0 può essere letto, scritto o eseguito da qualsiasi processo con uid=Qui e qualunque gid. Il *File1* può essere acceduto solo da processi con uid = Qui e a gid = system. Un processo con uid = Qui e gid = staff può accedere al *File0* ma non al *File1*. Il *File2* può essere letto e scritto da processi con uid=Qui e qualunque gid, da processi con la uid=Qua e la gid = staff o da processi con la uid=Pap e qualunque gid. Il *File3* può esser letto da ogni studente.
Il *File4* è speciale. La uid=Quo, in ogni gruppo, non può accedervi, ma tutti gli altri studenti possono leggerlo. Usando le ACL è possibile proibire a uid o gid specifiche di accedere ad un oggetto, anche autorizzando qualsiasi altro in qualche classe.

Capability

L'altro modo di sezionare la matrice degli accessi è per righe.
Associata ad ogni processo c'è una lista di oggetti cui può accedere, assieme all'indicazione di quali operazioni sono consentite su ognuno: in altre parole, il suo dominio. Questa lista è chiamata *lista delle capability*, e gli elementi individuali sono chiamati *capability*

Ogni capability ha un campo *tipo*, che dice che tipo di oggetto è, un campo *diritti*, che una mappa di bit che indica quali sono le operazioni legali sono consentite su quel tipo, ed un campo *oggetto*, che è un puntatore all'oggetto stesso (cioè il suo numero di i-nodo).

Le liste di capability sono esse stesse oggetti, e possono essere puntate da altre liste di capability, facilitando la condivisione di sottodomini

	Tipo	Diritti	Oggetto
0	File	R – –	Puntatore al file 3
1	File	R W X	Puntatore al file 4
2	File	R W –	Puntatore al file 5
3	Printer	– W –	Puntatore al printer 1

La lista delle capability per il dominio 2

Sono stati proposti 3 metodi per proteggere le liste di capability

• uso di una architettura hw in cui ogni parola di memoria ha un bit extra che indica quando la parola contiene una capability. Il bit non viene usato per istruzioni ordinarie, e può essere modificato solo dal sistema operativo

• conservare la lista all'interno del sistema operativo, e far sì che i processi facciano riferimento alle capability con il loro numero di casella

• conservare la lista nello spazio utente, ma crittografarla con una chiave segreta sconosciuta all'utente. Questo approccio è particolarmente indicato per sistemi distribuiti, ed è utilizzato in Amoeba

Capability

In aggiunta a diritti specifici dipendenti dall'oggetto, come lettura ed esecuzione, le capability hanno di solito *diritti generici* che sono applicabili a tutti gli oggetti. Esempi di diritti generici sono

1. Copia una capability: crea una nuova capability per lo stesso oggetto
2. Copia un oggetto: crea un duplicato dell'oggetto con una nuova capability
3. Rimuovi una capability : cancella un elemento dalla lista-C; l'oggetto non è interessato
4. Distruggi un oggetto: rimuove permanentemente un oggetto e la capability

Molti sistemi a capability sono organizzati come collezione di moduli, con *moduli gestori di tipo* per ogni tipo di oggetto. Le richieste per eseguire operazioni su un file vengono spedite al gestore dei file, mentre le richieste di fare qualcosa con un mailbox vanno al gestore del mailbox. Queste richieste sono accompagnate dalle capability rilevanti

Problema: siccome il modulo gestore di tipo è un processo normale, e il proprietario di un file ha una capability per cui può eseguire alcune operazioni sul file, ma non può avere il descrittore, è essenziale che il modulo gestore di tipo sia capace di fare con le capability più che un processo ordinario

Questo problema veniva risolto, in Hydra, con una tecnica chiamata *amplificazione dei diritti*, in cui al gestore di tipo venivano dati diritti-tipo che gli conferivano più diritti per un oggetto di quanti fossero concessi dalla capability stessa

Anche altri sistemi a capability che hanno un controllo forte dei tipi degli oggetti necessitano di qualcosa di analogo.

Capability

Un importante commento da fare sui sistemi a capability è che revocare un accesso ad un oggetto è difficile.

È complesso per il sistema trovare tutte le capability che afferiscono ad un qualsiasi oggetto per prenderle, dato che esse sono memorizzate in liste sparse su tutto il disco

Una soluzione è che ogni capability punti ad un oggetto indirettamente, mediante un puntatore. In tal caso, il sistema può rompere questa connessione, invalidando la capability tra il puntatore e l'oggetto

Un altro modo di effettuare la revocazione è quello usato in Amoeba. Ogni oggetto contiene un lungo numero casuale, che è anche presente nella capability. Quando viene presentata per l'uso una capability, i due numeri vengono confrontati. L'operazione è permessa solo se sono uguali.

Il proprietario di un oggetto può richiedere che il numero casuale nell'oggetto venga cambiato, e quindi invalidare le capability esistenti

Nessuno di questi schemi tuttavia permette revocazioni selettive (es.: revocare il permesso a John e non agli altri).

Modelli di protezione

Le matrici di accesso non sono statiche: cambiano quando vengono creati nuovi oggetti o distrutti vecchi oggetti, oppure quando un proprietario decide di modificare l'insieme di utenti dei suoi oggetti

Harrison ed al. (1976) indentificarono sei operazioni primitive sulla matrice di accesso che possono essere usate come base per modellare ogni sistema di protezione.

Queste operazioni sono:
CREATE OBJECT, DELETE OBJECT, CREATE DOMAIN, DELETE DOMAIN, INSERT RIGHT e REMOVE RIGHT.

Le ultime due primitive inseriscono e rimuovono diritti da elementi specifici della matrice, come assegnare al dominio 1 il permesso di leggere il *File6*.

Queste sei primitive possono essere combinate in *comandi di protezione*. È in questi comandi di protezione che i programmi utente possono eseguire cambiamenti alla matrice. Essi non possono eseguire direttamente le primitive.

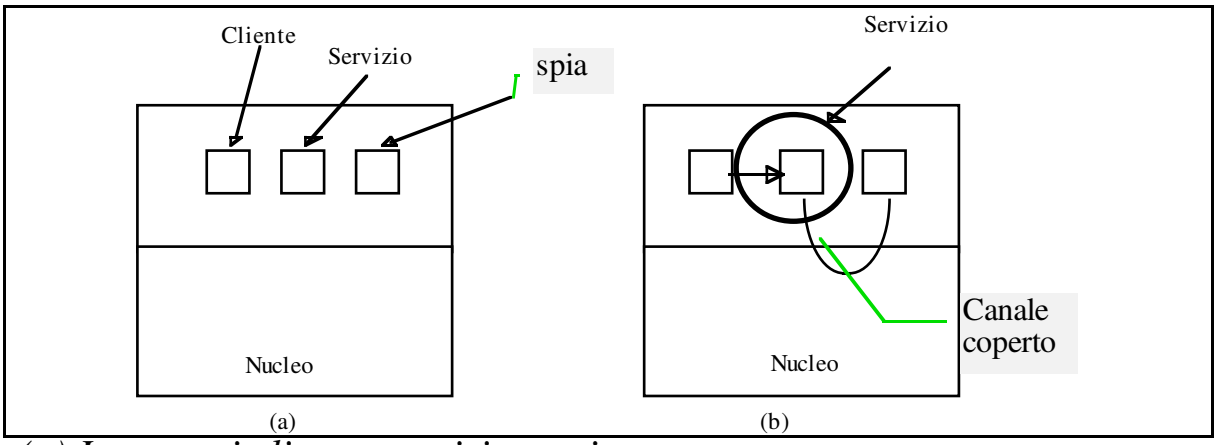
Per esempio, il sistema può avere un comando per creare un nuovo file, che controlla se il file esiste già e, se no, crea un nuovo oggetto e dà al proprietario tutti i diritti su questo.

Ci può essere un comando che permette al proprietario di dare il permesso di leggere il file a chiunque inserendo, in effetti, il diritto "lettura" nella casella del file in ogni dominio

Confinamento

Anche in un sistema che sia stato provato rigorosamente come assolutamente sicuro, la fuga di informazioni tra processi che in teoria non possono comunicare affatto è semplice

Il modello (Lampson) coinvolge tre processi ed è applicabile a sistemi a tempo condiviso. Il primo processo è il cliente, che necessita di alcuni compiti affidati al secondo, il servizio. Il cliente ed il servizio non possono avere intera fiducia uno dell'altro.



(a) I processi cliente, servizio e spia
(b) Il servizio anche se incapsulato può passare informazioni alla spia attraverso un canale coperto

Il terzo processo è la spia, che cospira con il servizio per rubare dati confidenziali al cliente. La spia ed il servizio sono posseduti solitamente dalla stessa persona.

Lampson ha chiamato questo il *problema del confinamento*.

Dal punto di vista del progettista, l'obiettivo è di confinare il servizio di modo che non possa passare informazioni alla spia.

Usando una matrice di protezione possiamo facilmente garantire che il servizio non possa comunicare con la spia mediante file. Possiamo anche garantire che il servizio non possa comunicare usando un meccanismo di comunicazione tra processi.

Sfortunatamente può essere disponibile un canale di comunicazione più subdolo.

Canali coperti

Il servizio può comunicare un flusso di bit come segue. Per mandare un 1, esegue più calcoli che può in un intervallo fissato di tempo. Per mandare uno 0, rimane inattivo per lo stesso intervallo di tempo

La spia può leggere il flusso di bit controllando i tempi di risposta. In generale, avrà una risposta migliore quando il servizio sta mandando uno 0 piuttosto che quando il servizio sta mandando un 1. Questo canale di comunicazione è conosciuto come *canale coperto*

Il canale coperto è disturbato, ma le informazioni possono essere spedite correttamente usando un codice correttore d'errori. L'uso di un codice riduce ancora la già bassa ampiezza di banda di un canale coperto, ma può essere sufficiente per la trasmissione di informazioni essenziali

La modulazione dell'uso della CPU non è il solo canale coperto. Può essere modulata anche la percentuale degli errori di paginazione In effetti, sono candidati quasi tutti i modi per degradare l'efficienza del sistema in maniera osservabile

L'acquisizione ed il rilascio di risorse dedicate (gestori di nastro, di plotter, ecc.) possono essere usate per segnalazioni. Il servizio acquisisce una risorsa per mandare un 1 e la rilascia per mandare uno 0. In Unix, il servizio può creare un file per indicare 1 e rimuoverlo per indicare uno 0; il collaboratore può usare la chiamata di sistema ACCESS per vedere se il file esiste

Sfortunatamente, dunque, esistono molti canali coperti: chiuderli tutti è impossibile. In pratica c'è poco da fare. L'introduzione di processi che provocano errori casuali di paginazione, oppure che usano il loro tempo per degradare l'efficienza del sistema per ridurre l'ampiezza di banda dei canali coperti, non è una prospettiva accettabile