APPUNTI SO2

DISPOSITIVO I/O

I dispositivi di I/O sono componenti HW che devono fornire ingressi o dirigere uscite. Il S.O. controlla tutti i dispositivi di I/O del computer:

-inviando comandi ai dispositivi

-intercettando gli interrupt che sono dei segnali inviati dal dispositivo al S.O.

-gestendo gli errori

I dispositivi di I/O si suddividono in:

-**DISPOSITIVI A BLOCCHI→** ovvero dispositivi che archiviano informazioni in blocchi di dimensione fisse ognuno con il proprio indirizzo. (es dischi fissi, penne USB)

-**DISPOSITIVI A CARATTERE→** ovvero dispositivi che rilasciano o accettano un flusso di caratteri senza alcuna struttura a blocchi e inoltre non sono indirizzabili. (es Tastiera)

Questa classificazione non è completa poiché alcuni dispositivi come i dispositivi di rete, orologi ecc non si adattano.

Un dispositivo di I/O è formato da due parti:

-una parte meccanica che è il dispositivo stesso.

-una parte elettronica che viene detta controller attraverso il quale il dispositivo comunica con il computer tramite un punto di connessione detto porta. Il controller è un chip o un insieme di chip che controlla il dispositivo, infatti, quest'ultimo accetta i comandi che li vengono inviati dal S.O. e li esegue.

COME COMUNICANO CPU E DISPOSITIVI DI I/O?

Modello di riferimento del sistema

CPU, memoria e dispositivi di I/O sono tutti collegati da un unico bus di sistema, per esempio, il bus PCIe e comunicano tra loro attraverso il bus.

Modello di riferimento del dispositivo

Si divide in:

-Interfaccia che è l'interfaccia HW che il controller del dispositivo presenta al resto del sistema che può essere usata per controllare il funzionamento del dispositivo.

-**Struttura interna** che è formata in genere da dei registri interni di controllo:

- **Registri dei dati:** usati dalla CPU per controllare il trasferimento dei dati in ingresso e in uscita dal dispositivo.

- **Registri di stato:** usati dalla CPU per scoprire cosa sta succedendo sul dispositivo.

- **Registri dei comandi:** possono essere scritti dalla CPU per controllare cosa farà il dispositivo.

Oltre a questi registri molti dispositivi usano un buffer di dati che viene usato per memorizzare i dati letti dal dispositivo che non possono essere passati direttamente al S.O.

Per consentire la comunicazione tra CPU-Dispositivo, i registri del dispositivo devono essere accessibili dalla CPU e ci sono due approcci per farlo:

A-I/O mappato alle porte

Ad ogni registro del dispositivo viene assegnato un numero di porta di I/O. Questo numero è un numero intero che può essere predefinito oppure assegnato all'avvio del S.O. La raccolta di tutti i numeri di porta I/O forma lo spazio della porta I/O che è ben diverso dallo spazio degli indirizzi della memoria. Tale spazio è protetto poiché è accessibile solo in mode kernel perciò solo dal S.O.

Con tale metodo si usano delle istruzioni speciali di I/O per comunicare.

Vantaggi I/O mappato alle porte

1-I dispositivi non devono consumare intervalli di indirizzi di memoria fisica.

2-Non è necessario gestire la memorizzazione nella cache.

B-I/O mappato in memoria

I registri del dispositivo vengono mappati nello spazio di memoria. Ad ogni registro viene assegnato un indirizzo di memoria univoco (all'avvio del S.O.) a cui non è assegnata memoria.

Vantaggi I/O mappato in memoria

1-I driver di dispositivo possono essere implementati in linguaggi di programmazione di alto livello come il C, perciò i registri sono visti come le variabili nei linguaggi di programmazione.

2-Non è necessario alcun meccanismo di protezione speciale per impedire ai processi utente di eseguire I/O di basso livello.

Come viene eseguito L'I/O?

1-I/O programmato (PIO) noto anche come polling

La CPU chiede al controller del dispositivo di eseguire una richiesta di I/O e si trova in un loop ristretto interrogando di continuo il dispositivo per vedere se ha completato la richiesta. Al termine della richiesta di I/O, la CPU può eseguire l'istruzione successiva. Questo metodo viene detto Polling oppure busy waiting.

Vantaggi I/O programmato (PIO) noto anche come polling

1-Facile da implementare

2-Ideale per piccoli trasferimenti di dati

Svantaggi I/O programmato (PIO) noto anche come polling

1-La CPU non fa altro che controllare lo stato del dispositivo di I/O finché non è pronto e quindi la CPU non può essere usata per svolgere un lavoro più utile.

2-I/O comandati da interrupt

La CPU chiede al controller del dispositivo di eseguire una richiesta di I/O e blocca il processo che ha richiesto l'operazione di I/O e va a prendere un altro processo da eseguire. Quando il controller del dispositivo di I/O vede che l'operazione di I/O è terminata genera e invia un interrupt alla CPU. Il segnale viene rilevato dal controller di interrupt. Il controller di interrupt ignora l'interrupt se ne sono presenti altri con priorità più alta, altrimenti, lo inizia a gestire in mode kernel e la CPU inserisce lo stato del processo nello stato di interrotto. Quando il gestore di interrupt è terminato si esegue un'istruzione di ritorno dall'interrupt e si elimina quest'ultimo e si ritorna all'esecuzione del processo attualmente in esecuzione.

Esistono due tipi di interrupt:

INTERRUPT PRECISO

Questo è un interrupt che lascia la macchina in uno stato ben definito:

1-Il PC viene salvato in luogo noto.

2-Tutte le istruzioni precedenti a quella indicata dal PC sono state completate.

3-Nessuna istruzione oltre a quella indicata dal PC è terminata.

4-Inoltre è noto lo stato di esecuzione dell'istruzioni oltre a quella indicata dal PC.

INTERRUPT IMPRECISO

Questo è un interrupt in cui almeno una delle quattro proprietà indicate negli interrupt precisi è violata.

3-DMA (Direct Memory Access)

La CPU scarica parte del lavoro su un processore per scopi speciali chiamato DMA controller.

1-La CPU programma il dispositivo I/O e configura il controller DMA; dopodiché, può svolgere altri lavori utili.

2-Il controller DMA esegue i trasferimenti di dati tra la memoria e il dispositivo I/O tramite PIO.

3-Infine il controller DMA invia un interrupt alla CPU quando ha finito di eseguire i trasferimenti.

Vantaggi DMA

1- Nessuna perdita di tempo per la CPU.
2- Ideale per grandi trasferimenti di dati.

Svantaggi DMA

1- Richiede dispositivi supportati da DMA.
2- Complessità extra per gestire il controller DMA.
3- Alcuni cicli di CPU sono usati per configurare il DMA.

Per semplicità assumiamo che il DMA e la CPU accedano al bus di sistema in modo indipendente.

Diversi approcci del DMA

1-Modalità Fly-By

In questa modalità i trasferimenti dati non passano attraverso il DMA. Ogni trasferimento di dati viene eseguito direttamente dal controller del dispositivo, infatti, il controller del DMA dice solo al controller del dispositivo quale operazione eseguire e l'indirizzo di memoria dove andare a leggere/scrivere i dati.

2-Modalità Flow-Through

I trasferimenti di dati passano attraverso il controller DMA. In questo caso i dati trasferiti vengono prima letti dal dispositivo o dalla memoria in un buffer interno al controller DMA. Dopodiché i dati vengono quindi scritti nella memoria o nel dispositivo.

Per questa modalità ci sono due possibili approcci attuabili:

1-CYCLE STEALING

Per ogni parola trasferire il DMA deve prima acquisire il bus e quindi rilasciarlo al termine del trasferimento. Il vantaggio di questo approccio è che impedisce alla CPU di rimanere inattiva per un lungo periodo mentre, lo svantaggio è che è inefficiente poiché dobbiamo acquisire e rilasciare il bus per ogni singolo trasferimento.

2-Modalità Burst

Il DMA dice al dispositivo di acquisire il bus, emettere una serie di trasferimenti, e infine di rilasciare il bus. Il vantaggio è che efficiente per i trasferimenti possono essere eseguiti con un'acquisizione di un bus. Lo svantaggio è che può bloccare la CPU e altri dispositivi per un certo periodo di tempo se vengono trasferiti molti dati e il dispositivo di I/O è molto lento.

IN CHE MODO IL S.O. GESTISCE LE RICHIESTE DI I/O?

Esistono tre approcci:

1-I/O programmato (PIO)

1-Il S.O. legge/scrive sul dispositivo alcuni dati.

2-Il S.O. interroga continuamente il dispositivo per vedere se è pronto a fornire/accettare nuovi dati. (chiamato pooling o busy waiting).

3-Il S.O. ripete i passaggi precedenti fino a quando non è necessario leggere alcun dato sul dispositivo/scrivere dati sul dispositivo.

Vantaggi PIO

1-Semplice da implementare

2-PIO va bene se il tempo di attesa ovvero il tempo che ci mette il dispositivo a completare l'operazione è molto breve.

Svantaggi PIO

1-Impegna la CPU a tempo pieno fino al completamento di tutte le operazioni di I/O.

2-I/O comandati da interrupt

1-Il S.O. invia la richiesta di I/O e blocca il processo che ha richiesto l'operazione di I/O.

2-Il S.O. fa qualcos'altro, per esempio, pianifica un altro processo fino a quando non arriva un interrupt dal dispositivo di I/O per segnalare il completamento oppure un errore.

3-Dopo che l'interrupt è stato inviato, il S.O. deve catturarlo e poi gestirlo se non ha altri interrupt con priorità maggiore.

4-Una volta che è stato gestito il S.O. ritorna ad eseguire l'istruzione successiva del processo che stava eseguendo prima di cominciare a gestire l'interrupt.

Vantaggi I/O comandati da interrupt

1-La CPU può fare qualcos'altro quando la richiesta di I/O viene eseguita dal dispositivo.

Svantaggi I/O comandati da interrupt

2-L'elaborazione di ogni interrupt richiede molto tempo, quindi troppi interrupt sprecano una certa quantità di CPU.

3-I/O tramite DMA

1-Il controller DMA continua la richiesta di I/O senza che la CPU venga disturbata. In pratica, l'I/O che utilizza DMA è PIO, solo che in questo caso è il controller DMA che esegue tutto il lavoro, invece della CPU principale.

Vantaggi I/O tramite DMA

1-Meno spreco di tempo e risorse della CPU.

Svantaggi I/O tramite DMA

1-Richiede dispositivi supportati da DMA.

2-Complessità extra per gestire il controller DMA.

3-Alcuni cicli della CPU vengono spesi per configurare il controller DMA.

4-Il DMA controller e la CPU possono contendersi l'accesso al sistema.

OBIETTIVI DI PROGETTAZIONE DEL SOFTWARE DI I/O

1-INDIPENDENZA DAL DISPOSITIVO (DEVICE INDIPENDENCE)

Questo obiettivo richiede che i programmi possano accedere a qualsiasi dispositivo di I/O senza dover specificare in anticipo il dispositivo.

2-DENOMINAZIONE UNIFORME DEI NOMI (UNIFORM NAMING)

Il nome di un file o di un dispositivo non deve dipendere in alcun modo dal dispositivo, inoltre, l'utente non deve essere a conoscenza di quale nome corrisponde a quale dispositivo.

3-GESTIONE TRASPARENTE DEGLI ERRORI

Gli errori in genere vanno gestiti il più possibile a livello HW, dove, sono disponibili maggiori dettagli su un errore. In molti casi gli errori sono temporanei ed inoltre solo se gli errori non riescono ad essere risolti a livello HW se ne dovrebbe parlare a livelli superiori.

4-MODALITA' DI TRASFERIMENTO

La maggior parte degli I/O fisici è asincrona ovvero viene guidata da interrupt. Tuttavia, i programmi utente sono molto più facili da scrivere se le operazioni di I/O sono sincrone ovvero bloccanti, quindi una volta che il programma invia una richiesta di I/O il programma si blocca finché la richiesta non viene soddisfatta. È il S.O. che dovrebbe essere in grado di rendere sicure le operazioni sincrone che sono effettivamente asincrone rispetto al programma.

5-GESTIRE IL BUFFERING

Spesso i dati che escono dal dispositivo non possono essere memorizzati direttamente nella loro destinazione finale. Il S.O. non sa dove inserire i dati parziali finché non li ha analizzati e memorizzati. Per questo motivo i dati vengono memorizzati in un buffer.

6-DISPOSITIVI CONDIVISI E NON CONDIVISI (O DEDICATI)

I dispositivi condivisi possono essere usati da molti processi contemporaneamente. I dispositivi non condivisi devono essere utilizzati da un processo alla volta. Questi dispositivi possono portare a deadlock. Il S.O. deve essere in grado di gestire sia i dispositivi condivisi e sia quelli dedicati.

SOFTWARE I/O: LIVELLI DEL SOFTWARE

Il software I/O è organizzato in quattro livelli e ogni livello ha una funzione ben definita da eseguire ed un'interfaccia ben definita per i livelli adiacenti:

1-GESTORI DI INTERRUPT

Questo livello nasconde i dettagli per la gestione degli interrupt ai livelli superiori. Quando si verifica un interrupt, il relativo gestore dell'interrupt fa tutto il necessario per gestirlo.

1-Il S.O. salva il contesto del processo corrente (registri, tabella pagine...).
2-Il S.O. prepara il contesto e lo stack per gestire l'interrupt.
3-Il S.O. manda un acknowledgement (ovvero un segnale di conferma) al controller interrupt avvisandolo della presa in carico dell'interrupt che li aveva inviato.
4- Il S.O. attraverso il gestore degli interrupt gestisce l'interrupt.
5- Il S.O. sceglie il prossimo processo da eseguire.
6- Il S.O. prepara il prossimo contesto per il processo scelto ed avvia la sua esecuzione.

2-DRIVER DI DISPOSITIVO

I driver di dispositivo nascondono i dettagli per il controllo dei dispositivi ai livelli superiori.

Ciascun driver di dispositivo in genere gestisce un tipo di dispositivo o una classe di dispositivi strettamente correlati tra di loro. I driver di dispositivo sono eseguiti di solito in mode kernel. Esso necessità di accedere ai registri del controller.

Esistono due modalità d'installazione per i driver di dispositivo:

1-collegato staticamente al S.O.

In questo caso si deve ricompilare il S.O. ogni volta che è necessario aggiungere un nuovo driver. Questo metodo veniva usato in passato quando i dispositivi di I/O cambiavano raramente poiché è molto costoso ricompilare l'intero S.O.

2-caricato dinamicamente in fase di esecuzione

Viene caricato a runtime da notare che alcuni S.O. richiedono un riavvio per rendere effettive le modifiche.

3-alcuni S.O. consentono i driver di dispositivo in modalità utente (non sono molto usati però)

STRUTTURA GENERALE DI UN DRIVER DI DISPOSITIVO: Come fa un driver le operazioni I/O?

1-Accetta le richieste di I/O dai livelli superiori
2-Verifica se i parametri sono validi traduco la richiesta per un livello più basso
3-Controllo se il dispositivo è attualmente occupato e se è così, si accoda la richiesta altrimenti la si esegue subito.
4-Invio una serie di comandi di basso livello al dispositivo: ogni comando viene scritto, controllo che sia stato accettato dal controller del dispositivo e che sia pronto per accettare un nuovo comando.
5-Una volta impartiti i comandi se è PIO: aspetta fine dell'operazione; se è basato su interrupt si blocca fino all'arrivo di un interrupt.
6-Al termine dell'operazione controllo se ci sono errori.
7-Si restituiscono le informazioni sullo stato degli errori se ci sono altrimenti si ritorna l'output.
8-Dopodichè sì seleziona la prossima richiesta in coda se è presente oppure si blocca l'attesa della richiesta successiva.

3-SOFTWARE I/O: SOFTWARE DEL S.O. INDIPENDENTE DAI DISPOSITIVI

Questo livello include funzioni comuni a tutti i dispositivi e fornisce un'interfaccia uniforme ai livelli superiori e inferiori. In pratica è il confine tra il driver del dispositivo e il software indipendente dal dispositivo.

Le funzioni tipiche in questo livello sono:

1-interfaccia uniforme per i driver di dispositivo

È necessario fornire un'interfaccia uniforme tra il S.O. e i driver di dispositivo per la stessa classe di dispositivi. Infatti, senza un'interfaccia uniforme del driver di dispositivo, il S.O., deve conoscere le funzioni specifiche di ciascun driver di dispositivo. Invece con un'interfaccia uniforme del driver di dispositivo è più facile collegare nuovi driver di dispositivo in un S.O. ed inoltre è più facile sviluppare nuovi driver di dispositivo per un S.O.

2-buffering

È un problema sia per i dispositivi a blocchi sia per quelli a caratteri, sia per le operazioni di lettura e scrittura. →La domanda che sorge è come gestire i byte in entrata/in uscita verso un dispositivo?

Esempio: Lettura di dati

Primo caso nessun buffering

In questo caso il processo utente esegue una system call "read" e legge un carattere e poi si blocca. Ogni carattere in arrivo causa un interrupt. Dopodiché la procedura degli interrupt presenta il carattere al processo utente. Questo caso è troppo inefficiente poiché ci sono troppi interrupt e quindi ci sono troppi context switch.

Secondo caso buffering nello spazio utente

In questo caso il processo fornisce un buffer di n caratteri nello spazio utente e fa una lettura di n caratteri. La procedura degli interrupt mette i caratteri in ingresso in questo buffer finché non è pieno. Solo a quel punto risveglia il processo utente.

Terzo caso buffering nello spazio kernel

Un ulteriore approccio è creare un buffer all'interno del kernel e avere il gestore degli interrupt che vi mette i caratteri. Quando questo buffer è pieno, la pagina con il buffer utente è portata dentro, se necessario, e il buffer viene copiato in una sola operazione. Questo schema è più efficiente degli altri due precedenti.

Quarto caso doppio buffering nello spazio kernel

In questo caso si usano due buffer nello spazio del kernel. Quando un buffer è pieno si usa l'altro. In alcune situazioni il buffer doppio è inadeguato, ad esempio con i burst rapidi di I/O.

Quinto caso buffering circolare

In questo caso il buffer è composto da una zona di memoria e due puntatori, uno dei quali punta alla parola libera successiva, dove poter mettere i nuovi dati, mentre l'altro punta alla prima parola dei dati nel buffer che non è stata ancora rimossa.

Esempio: Scrittura di dati

Primo caso nessun buffering

In questo caso il processo utente esegue una system call "write" e scrive un carattere alla volta e lo blocca. Un interrupt per ogni carattere quindi è inefficiente.

Secondo caso buffering nello spazio utente

In questo caso si scrivono blocchi di caratteri alla volta.

Terzo caso buffering nello spazio kernel

In questo caso si scrivono blocchi di caratteri alla volta e non li si blocca. I dati vengono copiati nel buffer dello spazio kernel. L'utente può riutilizzare immediatamente il buffer dello spazio utente.

→Il buffering è una tecnica molto usata, ma ha anche una controindicazione ovvero se i dati vengono messi troppe volte nel buffer, ne soffrono le prestazioni.

3-segnalazione errori

Gli errori sono molto comuni nel contesto dell'I/O e quando si verificano il S.O. deve gestirli nel migliore dei modi. Molti errori nello specifico del dispositivo devono essere gestiti dal giusto driver. Esistono due tipi di errori:

-**errori di programmazione:** si verificano quando il chiamante richiede operazioni non valide o specifica parametri non validi.

-errori di I/O: si verificano quando non va qualcosa durante la richiesta di I/O.

4-assegnazione/rilascio di dispositivi dedicati

Alcuni dispositivi sono dedicati e possono essere usati solo da un singolo processo alla volta per esempio due o più processi non possono stampare sulla stessa stampante allo stesso momento.

Il S.O. deve:

-esaminare le richieste di utilizzo del dispositivo e deve accettare o rifiutare la richiesta a seconda che il dispositivo richiesto sia disponibile o meno.

Esistono due tipi di approcci:

-primo approccio: i processi utente tentano di aprire il file del dispositivo

In questo approccio se il dispositivo non è disponibile la chiamata all'apertura non riesce ed è responsabilità del programma utente gestire tali errori.

-secondo approccio: meccanismi speciali forniti dal S.O. per richiedere e rilasciare dispositivi dedicati

Con questo approccio se un processo richiede al S.O. di usare il dispositivo dedicato che però non è disponibile verrà allora presa la richiesta e verrà messa in coda e il chiamante potrà essere bloccato fino a quando la sua richiesta non viene soddisfatta.

5-Dimensione del blocco indipendente dal dispositivo

Diversi dispositivi a blocchi possono avere dimensioni di blocco diverse, infatti, diversi dischi possono avere dimensioni di settore diverse. È compito di questo livello nascondere questi dettagli e fornire una dimensione del blocco uniforme agli strati superiori.

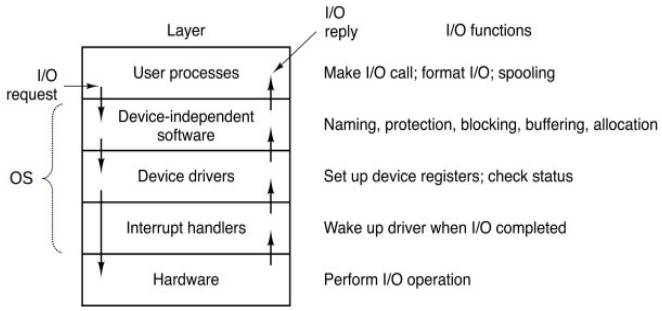
4-SOFTWARE PERL'I/O A LIVELLO UTENTE

La maggior parte dei software I/O funzionano in kernel, ma una piccola parte sono sistemi di librerie linkate insieme con programmi utente o programmi interi che si trovano nello spazio utente.

- **Librerie di sistema collegate con programmi utente:** le funzioni di libreria invocano le system call e a volte settano variabili di stato per segnalare errori.
- **Programmi interi in spazio-utente:** solitamente sono processi demoni che vengono eseguiti continuamente in background, spesso sono associati con il sistema di spooling.
    - ♦ Spooler: programma di sistema daemon
    - ♦ Spooling directory: usata dallo spooler per gestire inputs e/o outputs
    - ♦ File speciali: file di dispositivo come I/O
    - ♦ Deadlock-free: solo lo spooler accede ai file dispositivo

  ES: print spooling system:
  processo manda richiesta stampa al print spooler daemon->richiesta accodata->stampa pronta-> seleziono next.

LIVELLI DEL SISTEMA DI I/O E PRINCIPALI FUNZIONI DI OGNI LIVELLO



DISCHI

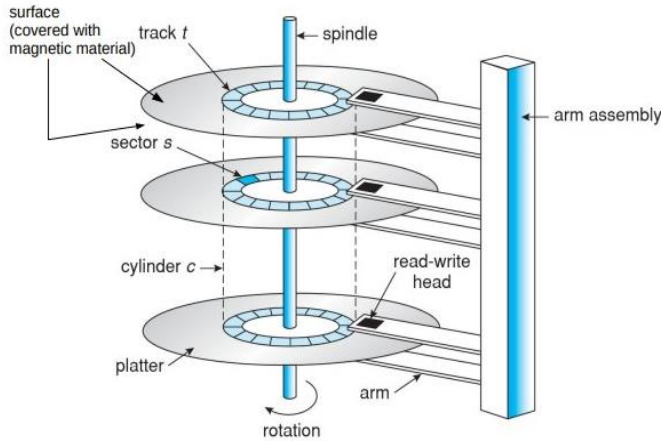
Ne esistono diversi tipi:

Magnetici: lettura e scrittura alla stessa velocità, usati come memorie secondarie.
Solid-state disk (SSD): velocità di lettura è maggiore della velocità di scrittura, sono usati come memoria secondaria e sono costosi.
Dischi ottici: lettura + volte ma 1 scrittura. Per distribuzione di programmi, film... (es: DVD, BLU-RAY).

DISCHI MAGNETICI

Ogni superficie di un piatto è logicamente divisa in tracce circolari, divise in settori (1 settore = 512B).
I settori sono separati l'uno dall'altro da gaps in cui non sono salvati bits.
Le tracce che sono in una certa posizione del braccio formano un cilindro.
Il controller legge/scrive dati come multipli di un settore: per cambiare 1 byte in un settore, devo leggere, aggiornare e riscrivere il vecchio settore controllando l'error correction code.
Il controller può leggere/scrivere tutti i dati in una traccia senza muovere il braccio inoltre è da notare che lavorare su una sequenza di settori della stessa traccia è molto più veloce di lavorare su settori di tracce diverse.
Vedi immagine pagina dopo di un disco magnetico

Disks: Magnetic Disks (cont'd)



I trasferimenti dati su I/O bus sono fatti dai controller:

-Host controller: controller alla fine del bus

-Disk controller: microcontroller presente in ogni disk drive

-Old disks: il controlller host esegue la maggior parte del lavoro e il disk controller fornisce un semplice flusso di bit seriale.

-Modern disk: il disk controller fa molto lavoro e consente all'host controller di inviare una serie di comandi di alto livello.

Svolgimento operazioni di disco I/O:

-SO inserisce un comando nel registro dei comandi dell'host controller.

-L'host controller manda un comando al controller del disco.

-Il controller del disco aziona il disk-drive HW per eseguire il comando.

-il controller del disco ha una cache incorporata.

-I trasferimenti dei dati sull'unità disco succedono tra la cache e la superfice del disco, mentre, i trasferimenti dei dati all'host avvengono tra la cache e l'host controller.

Controller Moderni

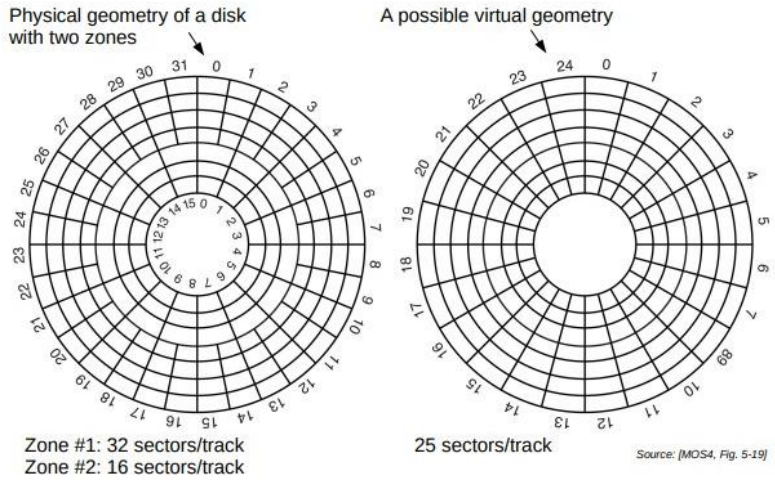
I controller moderni permettono i **Seek sovrapposti (ricerche sovvrapposte) (seek=ricerca)**: mentre il controller aspetta il completamento del seek su un drive, inizia il seek di un altro drive. Le operazioni di ricerca possono anche sovrapporsi alle operazioni di trasferimento dati.

Il trasferimento dei dati tra CONTROLLER e MEMORIA PRINCIPALE non può sovrapporsi.

GEOMETRIA DEL DISCO:

Vecchi dischi: Il numero di settori per traccia era lo stesso per tutti i cilindri (stessa geometria)
Dischi moderni: La geometria fisica è diversa dalla geometria virtuale.

Zone bit recording (ZBR): chiamato anche multiple zone recording.
La lunghezza della traccia fisica (circonferenza) incrementa allontanandoci dal centro.
L'idea→ per aumentare la capacità di archiviazione è, quella di suddividere i dischi in zone di registrazione con più settori nelle zone esterne rispetto alle zone più interne.
I cd sono divisi in *recording zone*: settori nella stessa zona hanno lo stesso numero settori.
Per nascondere questo al S.O., i controller usano la geometria virtuale.
Geometria virtuale: S.O. richiede un settore usando un indirizzo virtuale e il controller lo traduce in un indirizzo reale/fisico CHS (cilindro, testa e settore).



Seagate Cheetah 15K.4

Zone number	Sectors per track	Cylinders per zone
(outer) 0	864	3201
1	844	3200
2	816	3400
3	806	3100
4	795	3100
5	768	3400
6	768	3450
7	725	3650
8	704	3700
9	672	3700
10	640	3700
11	603	3700
12	576	3707
13	528	3060

MODALITA' DI INDIRIZZAMENTO REALE comuni
-Cylinder-Head-Sector (CHS) (settore-clindro-testa): Il S.O. pensa che ci siano NC cylinders, NH heads/cylinder, and NS sectors/track. Il numero di cilindro e di testa iniziano a essere numerati da 0, i numeri di settore iniziano ad essere numerati da 1.
-Logical block addressing (LBA): I settori (chiamati blocchi) hanno una semplice numerazione consecutiva che parte da 0. Il disk controller traduce indirizzo fisico in numero.

Cylinder-Head-Sector	**Logical block addressing (LBA)**
SO pensa ci siano NC cylinders, NH heads/cylinder, and NS sectors/track	Visto tutto allo stesso modo
Numerazione parte da 1 per settori, 0 per altri	Numerazione unica completa
Segue ordine C-H-S (001->002->003)	Segue ordine blocchi 0-1-2-3-...

Formule per conversione:

(c,h,s) → lba:
$lba = (c \times N_H + h) \times N_S + (s-1)$
lba → (c,h,s):
$c = lba \text{ div } (N_H \times N_S)$
$h = (lba \text{ div } N_S) \bmod N_H$
$s = (lba \bmod N_S) + 1$

LBA value	CHS tuple (c,h,s)
0	(0,0,1)
1	(0,0,2)
62	(0,0,63)
63	(0,1,1)
1007	(0,15,63)
1008	(1,0,1)
32256	(32,0,1)

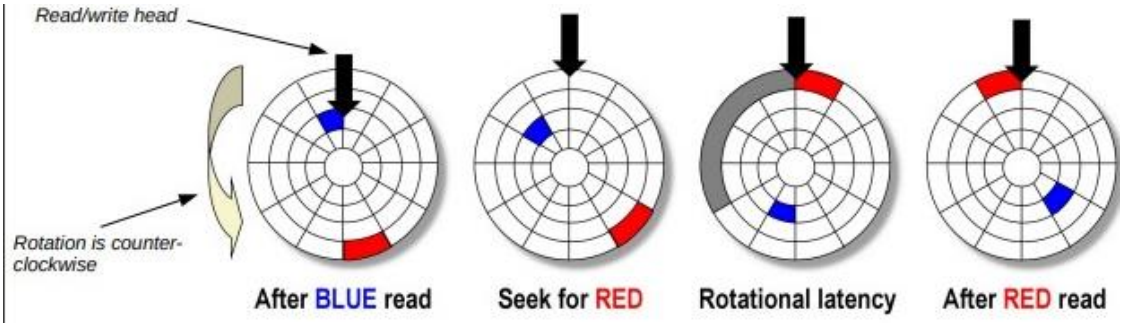
PRESTAZIONI DEL DISCO

CHE COSA SUCCEDE QUANDO SI ESEGUE UNA RICHIESTA DI I/O?

1-Si attiva il motore del gruppo braccio (assembly arm) e si posiziona la testa sopra il binario destro. (seek Ts)

2-Si attende che il settore appropriato ruoti per allinearsi con la testa (la rotazione è in senso antiorario), in pratica si attende che ci si posizioni sul giusto settore del disco. (Rot. lat. Tr)

3-Si eseguono le operazioni di lettura/scrittura mentre il settore si sposta sotto la testina e trasferisce i dati da/verso il buffer. (transf. Tt)



FORMULE

1-SEEK TIME: secondi che ci mette il braccio del disco a posizionarsi sul cilindro desiderato. Ne esistono tre tipi:

-Minimo: (seek traccia-a-traccia): per spostarsi nella traccia adiacente

-Massimo: per spostarsi dal più interno al più esterno o viceversa.

-Medio: la media tra le ricerche tra ciascuna possibile coppia di tracce su disco.

2-LATENZA/RITARDO DI ROTAZIONE: secondi per ruotare alla head del disco.

-Maximum rotational latency: tempo impiegato dal disco per completare un intero periodo di rotazione.

$$\rightarrow \textit{Maximum rotational latency} = \frac{1}{velocità\ di\ rotazione}$$

-Average rotational latency: tempo impiegato dal disco per completare la metà dell'intero periodo di rotazione.

$$\rightarrow \textit{Average rotational latency} = \frac{Maximum\ rotational\ latency}{2}$$

3-THROUGHPUT: è il tempo di trasferimento dati per un settore

-Tempo medio di trasferimento: è il tempo medio per il trasferimento di un settore tra la superficie e il controller.

$$\rightarrow \textit{Average transfer time} = \frac{Maximum\ rotational\ latency}{average\ sectors\ for\ tracks}$$

average sectors for tracks=media settori per traccia

-Tempo di posizionamento = ***→Tempo di posizionamento = seek time + rotational latency***

-Tempo medio di accesso per un settore: posizionamento +tempo di trasferimento. È il tempo per accedere a un intero settore del disco.

→Average access time= average seek time + average rotational latency + average transfer time

ES:
DATI→ rotational speed R:7200RPM; Average Ts:9msec; Average sector/track N:400
INCOGNITA→ Average rotational latency (Tr)? Average transfer time (Tt)? Average access time(Ta)?
CALCOLI:
Tr = 1/2 x 1/R x 60sec= 0.5 * 60.000 msec/7200 = 4170msec=4.17sec
Tt = 1/R x 1/N x 60sec = 60.000msec/7200 x 1/400=0.02 msec
Ta = Ts+Tr+Tt=9msec+4.1msec+0.02msec=13.19msec4
Nota→ 7200 RPM è un tempo in minuti per trasformali in millisecondi devi fare
60 diviso 7200 ovvero la velocità di rotazione moltiplicato per 1000 in questo modo passi dai minuti ai secondi e dai secondi ai millisecondi.

AFFIDABILITA' DEI DISCHI

-Fallimento completo: Il disco non è più capace di leggere/scrivere per crash pesanti/potenza/smagnetizzazione quindi la soluzione è rimpiazzare il disco.

-Failure rate: È il numero di fallimenti che mi aspetto in un'unità di tempo. Esiste quello annuale (AFR→ numero di guasti ogni anno) o quello orario (HFR→numero di guasti ogni ora).

-Mean time to failure: È il tempo medio prima che fallisca il disco ovvero l'inverso della percentuale di fallimento. Il MTTFY tempo medio prima che il disco fallisca annuo mentre MTTFH tempo medio prima che il disco fallisca orario.

$$\rightarrow MTTFY = \frac{1}{AFR}$$

$$\rightarrow MTTFH = \frac{1}{HFR}\ ,\ MTTFH = 24*365*MTTFY$$

Se è possibile riparare un disco dobbiamo considerare il **Mean Time To Repair (MTTR)** ovvero il tempo medio per riparare o sostituire il disco e il **Mean Time Between Failures (MTBF)** ovvero il tempo medio tra i guasti.

→MTBF = MTTF + MTTR

-Disponibilità(A): Frazione di tempo in cui il sistema è disponibile per soddisfare le richieste degli utenti.

$$\rightarrow A = \frac{MTTF}{MTBF}$$

-**downtime →** tempo durante il quale il sistema non è disponibile.
-uptime → tempo per il quale il sistema è disponibile.

Esercizio: dato un disco con AFR λA = 0.005 failure/year calcola il MTTFH
DATI→ AFR=0.005 failure/year
INCOGNITA→ MTTFH?
CALCOLI
Modo 1: MTTFY=1/0.005=200 anni , MTTFH = MTTFY*24h*356giorni=$1.7*10^6$ h

Modo 2: HFR=λH = λA /(24 h × 365gg)= 5.71×10^{-7} MTTFH=1/λH=1.7×106

Modello vasca da bagno

-Mortalità infantile→ I guasti sono per lo più dovuti a difetti di fabbrica.

-Tasso pubblicizzato→ La vita utile prevista, in cui i guasti sono per lo più casuali.

-Usura→Parti che si usurano, causando guasti dopo un uso prolungato.

Failure distribution:

La distribuzione di guasto è una distribuzione di probabilità che descrive la probabilità del prossimo guasto.

La probabilità del prossimo fallimento (ratio costante di fallimento). Può avere una distribuzione esponenziale negativa con $\lambda > 0$.

X~Exp(λ) = var per tempo tra 2 fallimenti, distribuzione esponenziale con parametro λ.

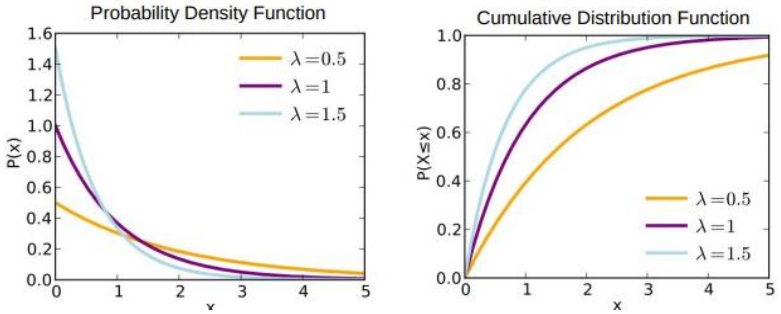
F(t)=Pr{X ≤ t}= funzione di distribuzione cumulativa, prob che il t per il prox fallimento sia <=t

Distribuzione esponenziale con λ>0:

Funz di densità di probab.: f(x, λ)={if(x<0)=0; else: λe^(- λx)

Funzione di distrib. cumulativa: F(x, λ)={if(x<0)=0; else:1-e^(-λx) FUNZIONE FALLIMENTO.

Funz sopravvivenza: 1-F(x)



La distribuzione esponenziale è **MEMORYLESS**: il tempo per il prossimo fallimento non dipende da per quanto tempo abbia funzionato. Pr{X < s+t | X > s} = Pr{X < t} o Pr{X > s+t | X > s} = Pr{X > t}.

{ES: se un bus passa ogni 10 minuti, le % che io debba aspettare 1 min o 9 sono =.*}

Se ho k dischi la probabilità che qualcuno fallisca>prob che fallisca uno specifico disco. Il fallimento di un disco su k è pari almeno al tasso di fallimento del disco peggiore.

FY (y)≥ maxi {FXi(y)}, i=1,…,k.

Esercizio:

100dischi identici indipendenti. AFR λdisk = 0.00585 annuali (MTTFdisk ≈1.5×106 hours ≈ 171 years).

DATI

-100 dischi, -AFR di un disco=0.00585 guasti/anno, - MTTF di un disco ≈1.5×106 ore ≈ 171 anni

INCOGNITA

-AFR di 100 dischi, -MTTFH di 100 dischi

CALCOLI

AFR di 100 dischi = 100*AFR di un disco = 100*0.00585=0.585 fallimenti annuali.

MTTFH di 100 dischi = (24*365) /AFR di 100 dischi = (24*365) /0.585 = 1.5*10^4=2 anni.

Tasso fallimento AFR di 100disks > AFR di un disk, mentre il MTTF100disks < MTTFdisk.

DISCHI RAID

Redundant Arrays of Independent Disk = I RAID sono una collezione di tecniche di organizzazione di dischi usati per aumentare le prestazioni e l'affidabilità dei dischi.
Inizialmente erano un'alternativa ai Single Large Expensive Disk (SLED).
Internamente il RAID è un sistema computerizzato con:

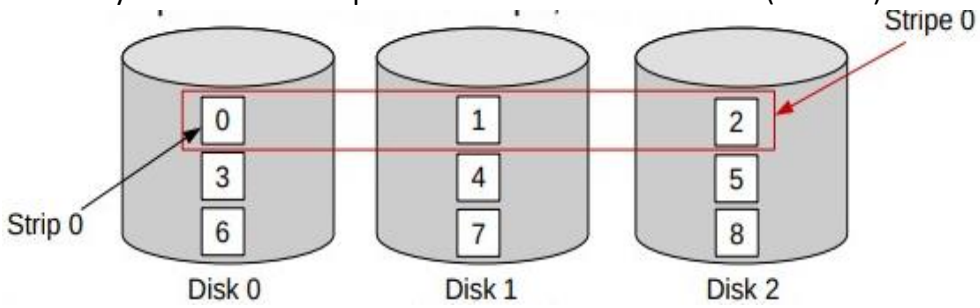
- un Array di dischi fisici
- Controller integrato per dirigere le operazioni I/O
- Memoria volatile per bufferizzare blocchi di dati.
- Possibile memoria non-volatile per bufferizzare scritture in modo sicuro
- Possibile logica specializzata per correggere gli errori.

Esternamente per il S.O. un RAID appare come un singolo disco logico: si presenta al S.O. come un array lineare di settori e i controller traducono ogni richiesta logica in richiesta fisica.

Per migliorare le prestazioni si usa il **parallelismo**:

- uso tanti dischi e gli accedo in parallelo
- **data striping:** è una tecnica per distribuire dati tra dischi fisici, disco diviso in **"strips" (blocchi fisici, settori o unità)** divise in dischi fisici consecutivi in modo round-robin.
  Stripe: set di strips logiche consecutive che mappano una strip a ogni disco.

Con array di n dischi: le strip i=0...m-1 si trovano nel disco (i mod n) dell'array e nella stripe (i/n)



Strip 1: Disk=1 mod 3=1, Stripe=1/3=0; Strip 6: Disk=0, Stripe=2...
Lo **striping** è gestito dal **raid controller** che totalmente trasparente all'esterno.
-bit-level striping: ogni bit in un disco diverso, ovvero, i dati vengono sottoposti a striping in modo tale che ogni bit sequenziale si trovi su un disco diverso.
-Byte-level striping: ogni byte in un disco diverso, ovvero, lo striping di dati viene eseguito in modo tale che ogni byte sequenziale si trovi su un disco diverso.
-Block-level striping: ogni blocco di byte contigui in un indirizzo diverso, ovvero, lo striping di dati viene eseguito in modo tale che ogni blocco sequenziale di byte contigui si trovi su un disco diverso.

Vantaggi del parallelismo:

- In un array di n dischi posso gestire in parallelo fino ad n strips.
- Una richiesta riguardante dischi diversi può essere servita in parallelo-> + transfer rate.
- Più richieste per strip su dischi diversi possono essere servite in parallelo ->+ rate richieste I/O eseguite

-) Richieste riguardanti strips sullo stesso disco devono essere svolte in sequenza
-) Pochi benefici se lavoro con poche richieste I/O o richieste in sequenza.
-) Non migliora l'affidabilità: + dischi = + affidabile. Dovrei essere ridondante e tenere copie extra

Per migliorare l'affidabilità di usa la **ridondanza:**

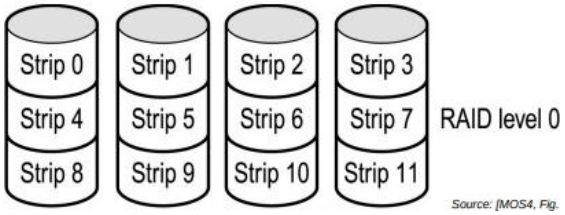
Mirroring: Con questo metodo ogni disco viene duplicato, è il metodo più semplice ma anche il più costoso. In questo caso rischio se perdo anche il clone (Mean time to repair). Con questo metodo si duplicano le richieste di scrittura/lettura poiché per esempio nel caso della scrittura vado a scrivere l'aggiornamento di un dato, per esempio, non solo sul disco principale ma anche sul disco clone.

In conclusione, possiamo dire che il mirroring offre un'elevata affidabilità, ma è costoso mentre, lo stripping, offre una velocità di trasferimento dati elevata ma non è affidabile.

I LIVELLI RAID

(Capacità desiderata = Cdisk x n.)

-LIVELLO RAID 0: In questo livello il raid usa solo lo stripping e non usa il mirroring.



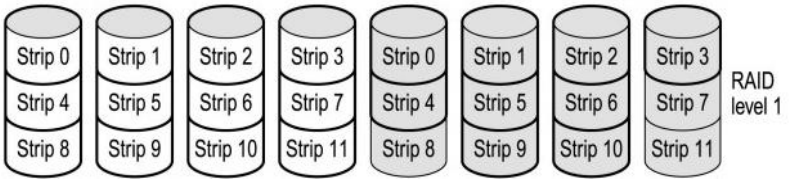
Vantaggi RAID 0) Performance e capacità ottime poiché per memorizzare n dischi bastano ne servono appunto n.

Vantaggi RAID 0) Siccome non è presente ridondanza senza di essa non sono tollerati errori sui dischi, infatti, se anche solo un disco ha un errore l'intero sistema è inutilizzabile, inoltre, è pessimo per le piccole richieste.

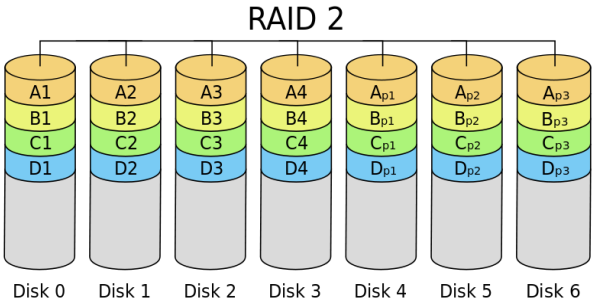
-LIVELLO RAID 1: In questo livello si usa il mirroring (dati striped e con copia). **La capacità** richiesta in questo livello è Cdisk×n: n+n, ovvero, è presente una copia per ogni disco di dati. **La tolleranza** agli errori è fino a n è sufficiente che non si guastino un disco e la sua copia affinché il sistema si utilizzabile.

Vantaggi RAID 1) È presente un'elevata tolleranza agli errori, il ripristino degli errori è semplice, ed inoltre le richieste di lettura possono essere fino al doppio della velocità.

Svantaggi RAID 1) È costoso, c'è poco miglioramento rispetto a RAID 0, ed inoltre è tanto costoso per avere a disposizione i cloni dei dischi.



-LIVELLO RAID 2: In questo livello di RAID si dividono i dati a livello di bit (non blocco) e si usa un codice ECC (error correction code) per correggere gli errori sui singoli bit e rilevare errori doppi.



-La capacità richiesta è di Cdisk x n: n+m che è minore di 2*n (n+m<2*n) (con n dischi di dati e m dischi di parità).

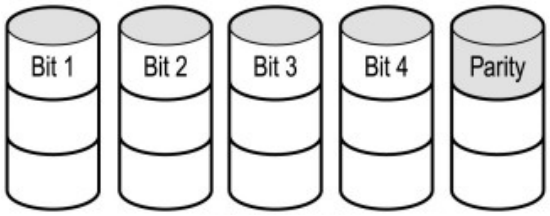
-Tolleranza: dipende dal ECC (>=1).

Richieste di scrittura e di lettura: Le richieste di lettura riguardano solo il disco dati mentre, quelle di scrittura anche il disco di parità.

Vantaggi RAID 2) È molto affidabile grazie all'ECC e ha buone prestazioni di lettura per lo stripping.

Svantaggi RAID 2) È possibile completare una richiesta I/O per volta, è costoso ed è complesso da implementare per tali motivi **NON È MOLTO USATO NELLA PRATICA.**

LIVELLO RAID 3: In questo livello il RAID è organizzato a striping a livello di bit più bit di parità. **(bit-interleaved parity organization)** striping + bit di parità (versione semplificata dei liv. 2).



-La capacità di questo livello è pari a: Cdisk * n dischi: n +1 disco di parità, perciò a n dischi di dati più un disco di parità.

-La Tolleranza: il numero di dischi guasti tollerati a questo livello è 1.

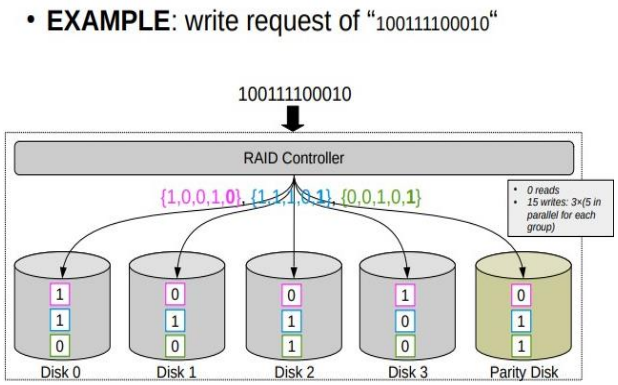
-Bit di parità: è un bit P aggiunto alla fine di una stringa binaria, che indica se il numero di bits nella stringa con valore 1 è pari o dispari. Viene calcolato il bit da aggiungere per far raggiungere alla stringa la parità e lo si confronta con quello a inizio operazione. Sono la forma più semplice di ECC.

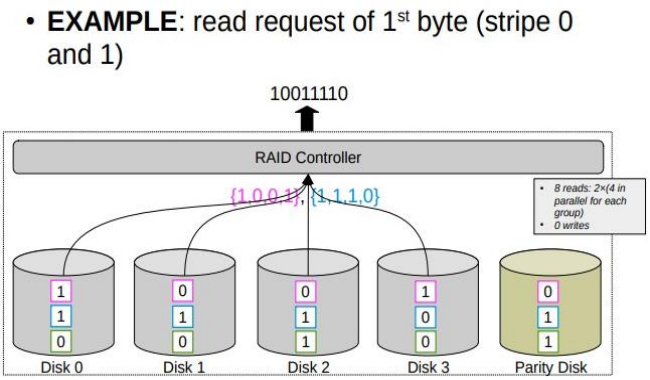
(Parità pari 2n) Se il numero di 1 presenti nella stringa è dispari il bit di parità è 1 altrimenti se il numero di 1 è pari il bit di parità è 0.

(Parità dispari 2n+1) Se il numero di 1 presenti nella stringa è dispari il bit di parità è 0 altrimenti se il numero di 1 è pari il bit di parità è 1.

Vantaggi RAID 3) Affidabile grazie ai bit di Parità, infatti, può tollerare un arresto anomalo di un singolo disco. Inoltre, ha le stesse prestazioni di RAID 2 ma un costo inferiore.

Svantaggi RAID 3) Come il RAID 2 è troppo complesso per essere implementato a causa dei requisiti di sincronizzazione ed ha ancora il problema che è possibile eseguire solo una richiesta di I/O alla volta. **PER TALI RAGIONI È USATO DI RADO NELLA PRATICA.**





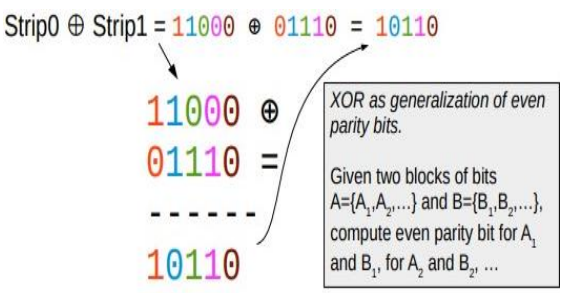
Nel disegno a sinistra ovvero nella richiesta di scrittura vengono fatte 0 letture e 15 scritture tra o quali 5 in parallelo, mentre, nel disegno di destra ovvero nella richiesta di lettura vengono fatte 0 scritture e 8 letture tra i quali 4 in parallelo.

LIVELLO RAID 4: In questo livello si utilizza lo stripping a livello di blocco più un blocco di parità. Questo tipo di organizzazione viene detta organizzazione di parità intervallati a blocco.

-La capacità richiesta è di n+1 (con l'1 che rappresenta il disco di parità).

-La Tolleranza è pari a 1 disco quindi si tollera un solo disco guasto.

-Disco di parità: il blocco di parità in RAID 4 si trova facendo l'operazione di XOR tra gli strips della stessa stripe.

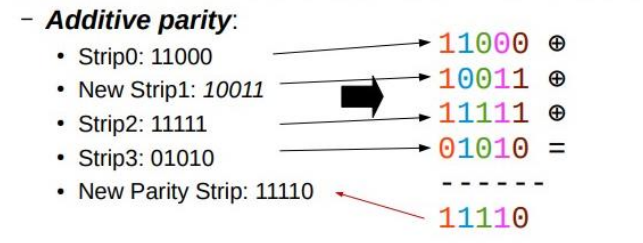


Vedi Tavola di verità XOR

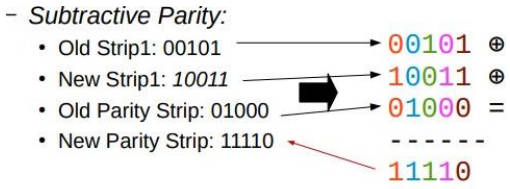
L'XOR da 0 quando c'è un numero pari di 1 mentre da 1 quando c'è un numero dispari di 1.

-Richieste di scrittura e di lettura: Le richieste di lettura vengono fatte solo da dischi dati mentre le richieste di scrittura vengono fatte sia dai dischi di dati sia dal disco di parità.
Per le richieste di scrittura ci sono **due approcci** quello di **parità additiva** e quello di **parità sottrattiva.**

- **Parità additiva:** Legge le altre strip della stripe dal data disk, calcola il blocco di parità e salva i nuovi dati e blocchi su disco. (In breve, è la somma tra tutte le strip) Con questo metodo si fanno 3 letture in parallelo e 2 scritture in parallelo.
- **Parità sottrattiva**: Legge vecchi dati e parità, computa il blocco di parità e scrive i nuovi dati e blocchi. (In breve, è la somma tra vecchia e nuova strip più il vecchio blocco di parità) Con questo metodo si fanno 2 letture in parallelo e 2 scritture in parallelo.



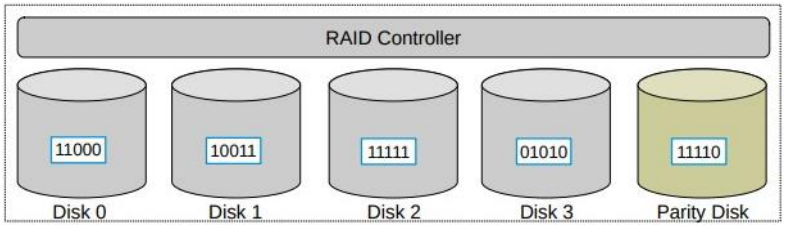
Additive parity method: 3 reads in parallel, 2 writes in parallel



Subtractive parity: 2 reads in parallel, 2 writes in parallel.
In this case, the subtractive parity method is more efficient than the additive parity method!

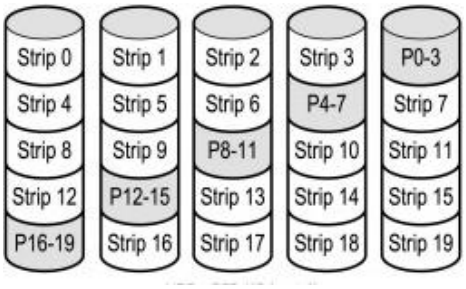
In caso di errore del disco i dati persi possono essere recuperati dal disco di parità.
In caso di crash in lettura, posso calcolare la strip mancante facendo lo XOR tra le altre strip e quella di parità (perciò usando il metodo di parità additiva).
Vantaggi RAID 4) Affidabile e buone performance di lettura e possono essere soddisfate grandi richieste di scrittura.
Svantaggi RAID 4) Costoso per scritture corte. **Small-write problem:** non posso svolgere in parallelo più operazioni che riguardano dischi diversi perché usano lo stesso disco di parità.

Esercizio: con n dischi RAID 4 (n-1 + parità) e B<N blocchi (strips) della stessa stripe da cambiare, quando usi la parità additiva e quando la sottrattiva?



1)Considera 2 scritture una nel primo disco da 11000-> a 00111 su strip 0 e una nel secondo disco da 10011-> a 01100 su strip 1.
Metodo sottrattivo: 3 letture e 3 scritture entrambe in parallelo (dischi 0 e 1 + il disco di parità)
Metodo additivo: N-3(=2) 2 letture e 3 scritture in parallelo.
In questo caso conviene utilizzare il metodo additivo.
2)Considera numero op. di entrambi i metodi per cambiare le B strips in N dischi raid 4 con B<N.
Trova B=f(N) tale che: letture sottrattive = letture additive.
Se numero scritture da fare = N/2 -1; se > -->additivo, se < --> sottrattivo.

-LIVELLO RAID 5: In questo livello RAID si utilizza lo stripping a livello di blocco più il blocco di parità distribuito (viene anche detto **block-interleaved distribuited parity organization).**



(Stripping a blocchi + blocchi di parità distribuita).

-La capacità richiesta a questo livello è di n+1 (con n blocchi di dati e 1 parità intervallati, in pratica e un disco di parità ma con i blocchi di parità distribuiti su tutti i dischi in mode round-robin).

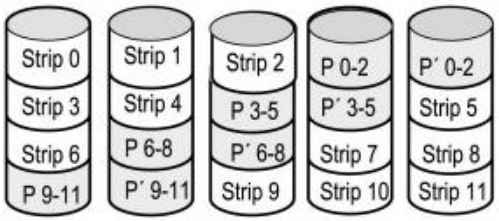
-La tolleranza fallimenti è di 1 disco.

Riassumendo in questo RAID non c'è un singolo disco di parità e quindi non essendoci nessun disco a parità singola non è presente nessun collo di bottiglia, inoltre posso soddisfare in parallelo più richieste piccole di scrittura su diverse stripes e dischi in parallelo. Accedo a più blocchi di parità in parallelo.

Vantaggi RAID 5) Affidabile e buone performance di lettura e possono essere soddisfate grandi richieste di scrittura. Inoltre, c'è un miglioramento delle richieste di lettura poiché ogni disco può partecipare per servire le richieste di lettura (a differenza di RAID 4). Infine, la velocità di richiesta di scrittura è migliorata poiché è possibile eseguire più scritture in parallelo.

Svantaggi RAID 5) Il **problema della Small write** è ancora esistente in alcuni casi.

-LIVELLO RAID 6: In questo livello si utilizza lo stripping a livello di blocco più il blocco a doppia parità distribuita. Lo schema di ridondanza è P+Q con, P blocchi di parità basati sull'XOR mentre i blocchi di parità Q sono basati su codici ECC di tipo reed-solomon. **(Striping + 2 blocchi di parità distribuiti)**. **Spesso si usano raid 5 e 6 insieme.**



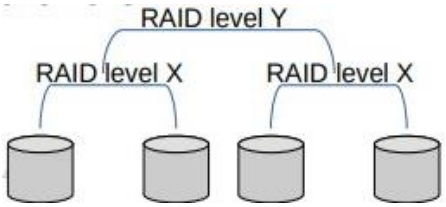
-**La capacità** richiesta è di n+2 (2 dischi parità)

-**La tolleranza** è di due dischi guasti.

Vantaggi RAID 6) Questo livello è affidabile per via della possibilità di avere 2 possibili dischi guasti e il sistema funziona ancora accompagnate con delle buone prestazioni ma non buone come quelle di RAID 5 poiché le operazioni di scrittura non sono comparabili.

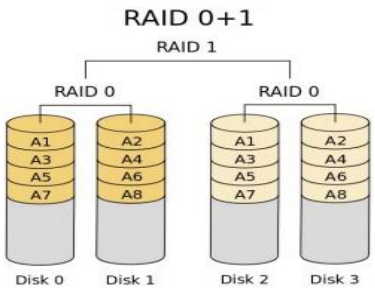
Svantaggi RAID 6) Scrittura peggiore del RAID 5 (poiché con la scrittura bisogna modificare 2 bit parità).

LIVELLO	FUNZIONAMENTO	CAPACITA'	TOLLERANZA	
1	Mirroring (copia di ogni disco)	n+n	N	Sicuro ma costoso
2	Divisione a livelo bit + ECC per correzione	n+m (disco parità)	Dipende ECC	
3	striping + bit di parità	n +1 disco parità	1	Rileva solo errori dispari e non sa dove
4	striping a blocchi + blocco di parità	n+1	1	Blocchi parità = xor tra 2 strips della stessa stripe
5	Striping a blocchi + blocchi di parità distribuita	n+1	1	I bit di parità sono distribuiti tra tutti i dischi
6	Striping + 2 blocchi di parità distribuiti.	N+2	2	2 bit parità distribuiti per doppia sicurezza

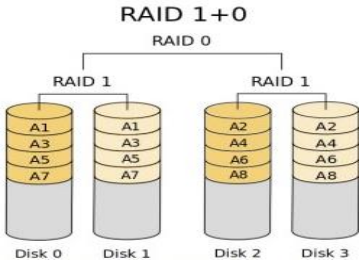


- **RAID annidati/ibridi**

Combinano uno o più livelli standard RAID in una gerarchia.
Numerazione composta (basso->alto).
Schema comune: X+Y.



❖ **RAID 0+1**: detto **mirror of stripes (copia di stripes).**
n dischi divisi in stripe, ognuna copiata in altri n dischi.
Capacità Dischi: n+n.
Tolleranza: 1->dopo 1 fallimento diventa un RAID 0 nell'immagine per rendere il sistema fuori uso devono guastarsi per esempio il disco 0 e il disco 2 oppure il disco 1 e il disco 3.
VANTAGGI) Buone performance in lettura e migliore del raid 0
SVANTAGGI) Costoso.



❖ **RAID 1+0**: detto **stripe of mirroring**
Capacità Dischi: n+n
Tolleranza: >=1 && <=n nell'immagine per rendere il sistema fuori uso devono guastarsi due dischi dello stesso raid oppure.
Quando uno crasha posso usare l'altro disco del set copiato.

IMPLEMENTAZIONE RAID
Controller: a livello HW è un circuito separato connesso (host bus adapter) che si collega nel pc tramite un cavo o un disco. Buone performance ma costoso.
A livello SW: è parte del S.O. che controlla la gestitone dei dischi attraverso i disk controller. Più economico ma più lento di un'implementazione a livello HW.
Hot-Spare disks: rimpiazzano i dischi guasti automaticamente SOLO in caso di fallimento. Ogni disco provvede ad un rimpiazzo senza dover intervenire.

ES: 5 dischi RAID 3.

1)come scrivo 11000111, 10000101, 01110111, 01010101? 40 scritture e 5 scritture possono essere fatte in parallelo. (5 poiché i dischi sono 5)

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
1	1	0	0	0
0	1	1	1	1
1	0	0	0	1
0	1	0	1	0
0	1	1	1	1
0	1	1	1	1
0	1	0	1	0
0	1	0	1	0

2)Come cambio il secondo byte in 01101101?

Faccio 8 scritture e 5 scritture sono fatte in parallelo.

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
1	1	0	0	0
0	1	1	1	1
0	1	1	0	0
1	1	0	1	1
0	1	1	1	1
0	1	1	1	1
0	1	0	1	0
0	1	0	1	0

3)Il terzo disco (disk 2) crasha e voglio leggere il secondo byte?

Controllo il bit di parità per la prima stripe è del secondo byte è 0 perciò usando il metodo del bit di parità pari siccome è presente un numero dispari di 1 e devo arrivare ad avere un numero pari di 1 poiché il bit di parità e 0 inserisco 1. Nella seconda stripe siccome il bit di parità è 1 e ho un numero dispari di 1 inserisco 0 per mantenere la disparità degli 1.

- **SOLUTION:**

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
1	1	X	0	0
0	1	X	1	1
0	1	X	0	0
1	1	X	1	1
0	1	X	1	1
0	1	X	1	1
0	1	X	1	0
0	1	X	1	0

01X00 → 01100
11X11 → 11011

- *Overall 8 reads*
- *Groups of 4 reads can be performed in parallel (to read an entire stripe, except the strip on disk 2)*

4)Crasha ANCHE il quarto disco (3) e voglio leggere il primo byte?

In questo caso non posso leggere il byte poiché RAID 3 tollera al massimo un disco guasto.

- **SOLUTION:**

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
1	1	X	X	0
0	1	X	X	1
0	1	X	X	0
1	1	X	X	1
0	1	X	X	1
0	1	X	X	1
0	1	X	X	0
0	1	X	X	0

Cannot recover from error!!

Vedi slide dalla 95 alla 98 per il secondo esercizio sui RAID slide I/Op04.

ESERCIZIO: RAID a 5 dischi livello 4 con strip da 1 byte. NOTA: Tutti i dischi sono vuoti; Ogni punto parte dallo stato lasciato dai punti precedenti.

– Come vengono scritti i seguenti byte: 11000111, 10000101, 01110111, 01010101 su ciascun disco? Vengono fatte cinque scritture in parallelo.

- **SOLUTION**:

Overall 5 writes in parallel

```
11000111 ⊕
10000101 ⊕
01110111 ⊕
01010101 =
----------
01100000
```

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
11000111	10000101	01110111	01010101	01100000

– Cosa succede se il sistema operativo vuole aggiornare il terzo byte a 10001000?

Vengono fatte in questo caso 2 scritture e 2 letture in parallelo e vien usata la parità sottrattiva poiché richiede una lettura in meno di quella additiva.

- **SOLUTION**:

- *2 parallel reads*
- *2 parallel writes*

```
01110111 ⊕
10001000 ⊕
01100000 =
----------
10011111
```

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
11000111	10000101	10001000	01010101	10011111

– Cosa succede se il secondo disco (disco 1) si arresta in modo anomalo e il sistema operativo vuole leggere il secondo byte?

Per leggere il secondo byte in questo caso si fanno quattro letture in parallelo e poi si fa l'XOR dei quattro byte letti e otteniamo così il primo byte che volevamo leggere.

- **SOLUTION**:

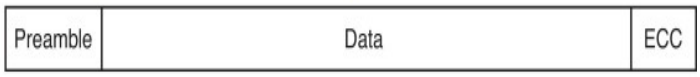
4 parallel reads

```
11000111 ⊕
10001000 ⊕
01010101 ⊕
10011111 =
----------
10000101
```

Disk 0	Disk 1	Disk 2	Disk 3	Parity Disk
11000111	X	10001000	01010101	10011111

FORMATTAZIONE A BASSO LIVELLO DEI DISCHI

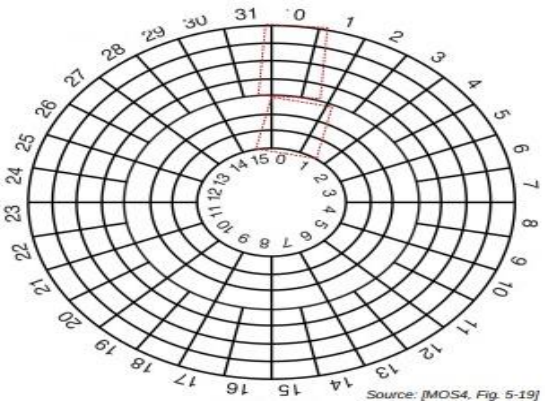
Prima che un disco venga usato per salvare dati deve essere formattato a basso livello.
Per farlo si definisce la geometria fisica del disco (serie concentrica di tracce ognuna con numero di settore e piccolo divario (gap) tra i settori).
Tale lavoro viene fatto dal produttore/SW
Ogni settore ha seguente il formato:



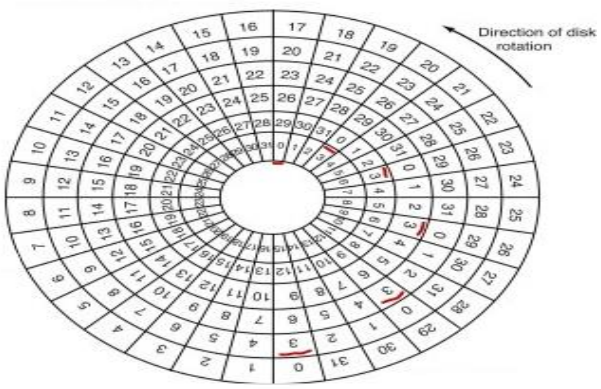
- **Preambolo:** È il campo che identifica l'inizio di un settore e contiene un pattern di bit che permette all'HW di riconoscere l'inizio di un settore e informazioni come numero di settore, cilindro, ecc.
- **Dati:** campo che contiene i veri dati del settore. Dimensione dipende dal livello di formattazione.
- **ECC**: salva le informazioni ridondanti che usa per trovare eventuali errori (di solito 16B)

-Scrittura di un settore: il controller aggiorna l'ECC con valori calcolati da tutti i byte nel campo.
-Lettura di un settore: il controller ricalcola l'ECC e lo confronta con il valore salvato, se è diverso significa il campo dati è stato corrotto, se l'errore è piccolo il controller lo identifica e lo corregge.

Come numerare i settori di ogni traccia?
Uso il settore radiale (colonne).
SVANTAGGIO) Per spostarmi da un settore all'altro (righe) devo muovere molto il braccio è quindi è molto il tempo sprecato. La soluzione è il **Cylinder skew**.
Cylinder skew: la posizione del settore 0 in ogni traccia è offset della traccia precedente.



DOPO:



La quantità di **Cylinder skew** varia in base alla geometria fisica. L'inclinazione del cilindro (ovvero il **Cylinder skew**) può essere calcolato come segue:

$$\textbf{CYLINDERSKEW = OFFSET} = \frac{\textbf{seek time tra le tracce (ovvero il tempo di ricerca tra le tracce)}}{\textbf{tempo di interarrivo al settore (tempo di passaggio al nuovo settore)}}$$

Il seek time tra le tracce lo si trova anche come track to track seek time.

$$\textbf{tempo di interarrivo al settore} = \frac{\textbf{maximum rotational latency}}{\textbf{numero di settori per traccia}}$$

Il tempo di interarrivo al settore si può anche scrivere sector interarrival time.

ESERCIZIO: Si supponga che un'unità disco (Disk drive) abbia le seguenti caratteristiche:
DATI: velocità di rotazione: 10.000RPM, 300 settori a traccia, Track-to-Track seek time: 800 µsec.
INCOGNITA: Calcola la Cylinder skew.
CALCOLI:
Una rotazione richiede: 1/10.000=6 msec.
Tempo per ogni settore: 6msec/300=20 µsec. → tempo di interarrivo al settore.
Settori che passo nel seek time: 800 µsec/20 µsec= 40 settori
In conclusione, l'inclinazione del cilindro dovrebbe essere di almeno 40 settori.

ESERCIZIO: Si supponga che un'unità disco (Disk drive) abbia le seguenti caratteristiche:
DATI: velocità di rotazione: 5400RPM, 300 settori a traccia, Track-to-Track seek time: 800 µsec.
INCOGNITA: Calcola la Cylinder skew.
CALCOLI:
Una rotazione completa: 1/5400=11.11msec;
rotazione di 1 settore 11.11msec/300=37.03 µsec;
Settori nel seek time: 800/37.03=21.60 settori
In conclusione, l'inclinazione del cilindro dovrebbe essere di almeno 22 settori.

Effetti della formattazione:
La formattazione riduce la capacità del disco di circa il 20%. Spesso alcuni produttori dicono la capacità prima della formattazione per far sembrare i dischi più grandi o cambiano le unità di misura per tale motivo è presente una notevole confusione sulle reali capacità dei dischi.
La formattazione fissa un limite max di velocità di trasferimento dati.
-Con Ns numero di settori per traccia,
-S size del settore,
-R velocità rotazione:
possiamo trovare la velocità dati massima (detta anche Dmax) ottenibile. In genere si esprime in byte/sec.
Velocità dati massima = Dmax = $\frac{\boldsymbol{dimensioni\ traccia\ (track\ size)}}{\boldsymbol{maximum\ rotational\ latency}}$
track size = Ns*S,

ESERCIZIO: LA FORMATTAZIONE INFLUISCE SULLE PRESTAZIONI
DATI: velocità di rotazione=10.000RPM, Ns=300 settori a traccia di dimensioni S=512B ciascuno.
INCOGNITA: Trovare la velocità dati massima.
CALCOLI:
track size=512byte*300=1536000byte;
maximum rotational latency=60/10.000=6msec;
Dmax=153600/6msec=25.600.000 B/sec.

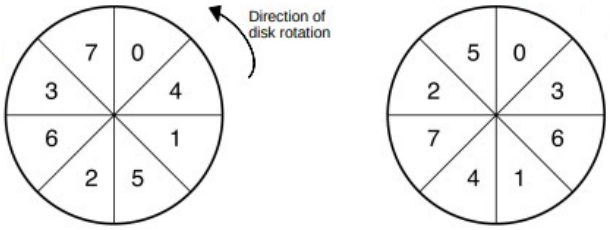
Per leggere settori consecutivi: devo leggere il primo, dopodiché, calcolare l'ECC e infine si avvia il trasferimento dei dati in memoria per far ciò bisogna completare una rotazione intera. È già a questo punto ci accorgiamo che non è molto veloce come metodo.
Come velocizziamo? Attraverso il **Sector interleaving**.

Sector interleaving

Il Sector interleaving facilita le letture di settori consecutivi, ma rallenta gli altri.

1)**Interleaving singolo**: i e i+1 sono a 1 di distanza. Vuol dire che è presente un settore che ne separa due consecutivi ad esempio 0 e 1 sono separati dal settore 4. (figura a sinistra)

2)**Interleaving doppio:** i e i+1 sono a 2 di distanza. Vuol dire che sono presenti due settori che separano due settori consecutivi ad esempio 0 e 1 sono separati dal settore 3 e 6. (figura a destra)



La quantità di interleaving da usare dipende dalla velocità di rotazione del disco e dalla velocità di trasferimento della memoria.

SECTOR INTERLEAVING = $\frac{in-memory\ sector\ transfer\ time}{sector\ interarrvial\ time}$

-in-memory sector transfer time→ è il tempo impiegato per trasferire un settore dal buffer del controller alla memoria o viceversa. Inoltre, include il tempo per controllare l'ECC.

-sector interarrival time→ è il tempo di passaggio di un nuovo settore sotto la testina del disco.

Il sector interleaving può influire sulle prestazioni perché, in generale, la lettura dei settori con il sector interleaving è più lenta che senza di esso. Un'atra soluzione è quella di usare un buffer largo che memorizza tracce intere.

ESERCIZIO: calcola il sector interleaving con i seguenti dati: R=10.000RPM, 1900 settori a traccia, tempo trasferimento in memoria = 10 µsec.

DATI:

-velocità di trasferimento = 10.000RPM,

-1900 settori a traccia,

-tempo trasferimento in memoria = 10 µsec.

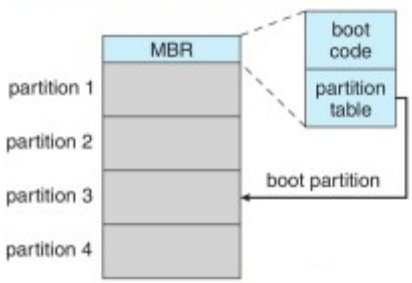
CALCOLI:

-maximum rotational latency = 60sec/10.000RPM = 6msec.

-Sector interarrival time = Tempo passaggio settore = 6msec/1900settori=3.2 µsec

-Sector interleaving = 10 µsec/3.2 = 3.125 settori.

Prima che un disco possa contenere informazioni deve essere fatto il partizionamento del disco e la formattazione alto livello.



-Partizionamento del disco

Partizioni: parti separate del disco, permetto a più SO di coesistere. Il settore di partenza di ogni partizione e la sua dimensione sono salvati nella **partition table** (**ovvero la tabella delle partizioni**) contenuta nella **Master boot record (MBR)** che si trova nel primo settore di un disco (prima della prima partizione) assieme al boot loader (codice per chiamare il vero caricatore del S.O.).
La Partition table supporta 4 principali partizioni tra cui 1 è la partizione estesa che contiene più partizioni logiche, ognuna descritta da un **Extended boot record** (**EBR**) (nella partizione estesa prima delle logiche, formano una lista).
GUID PARTITION TABLE (GPT): rimpiazza la MBR perché supporta 128 partizioni invece di 4 e usa 64 bits invece di 32 per rappresentare numeri di settore.
Formattazione ad alto livello di ogni parte: -imposta dei file-system iniziali sulle partizioni (salva la mappa degli spazi liberi ed allocati), -installa un boot sector, -scansiona dei difetti, -identifica il file system usato mettendo un numero nella partition table entry.

DISK ARM SCHEDULING (PROGRAMMAZIONE DEL BRACCIO DEL DISCO)
Il tempo per leggere/scrivere un disco dipende dal seek time ovvero il tempo di ricerca, dalla latenza di rotazione e dal tempo di trasferimento. Le performance vengono influenzate dalle parti meccaniche (seek+rotation).
Come riduco il seek time?
Quando arriva una richiesta I/O se il disk drive e il controller sono inattivi (non occupati) la accetto, altrimenti la accodo nel disk drive. (coda = tabella indicizzata da numero di cilindro e ogni entry ha una coda di richieste per ogni cilindro).
Come scelgo la richiesta tra quelle in coda? Esistono diversi algoritmi di pianificazione del braccio:
1-First come first served (FCFS): Con questo algoritmo le richieste sono soddisfate nell'ordine in cui sono arrivate.
-Esempio, ordine arrivo richieste: 11, 1, 36, 16, 34, 9, 12.
-Movimenti braccio: |11-11| + |11-1| + |1-36| + |36-16| + |16-34| + |34-9| + |9-12| = 111 cilindri.
2-Shortest seek first (SSF): Seleziona la richiesta con il minor tempo di ricerca dalla posizione attuale della testa.
-Esempio, ordine arrivo richieste: 11, 1, 36, 16, 34, 9, 12.
-Movimenti braccio: |11-11| +|11-12|+|12-9|+|9-16|+|16-1|+|1-34|+|34-36|= 61 cilindri.
È un algoritmo buono ma c'è il rischio che si verifichi la starvation (morte di fame ricorda S.O.1).
La starvation è quando un processo si trova in coda ma non verrà mai eseguito.
3-Algoritmo ascensore (look): il braccio continua a muoversi nella stessa direzione finché non finisce le richieste in quel senso e allora cambia direzione. Usa un bit UP/DOWN.
-Esempio, ordine arrivo richieste: 11, 1, 36, 16, 34, 9, 12.
-Movimenti braccio:|11-11|+|11-12|+|12-16|+|16-34|+|34-36| +|36-9| + |9-1| = 60 cilindri.
La lunghezza è buona e non c'è il rischio di starvation (aspetta max 2xNumero cilindri). Sono però sfavorite le richieste agli estremi.

4-Look circolare(C-look): muove la testa da una end del disco all'altra servendo richieste per strada, ma quando raggiunge l'ultima torna subito all'inizio. Più lento del look ma è più equo.
-Esempio, ordine arrivo richieste: 11, 1, 36, 16, 34, 9, 12.
-Movimenti braccio:|11-11| + |11-12| + |12-16| + |16-34| + |34-36| + |36-1| + |1-9] = 68

ESERCIZIO: richieste 10, 22, 20, 2, 40, 6, 38 con 50 cilindri, la posizione iniziale è il cilindro 20 e 2msec di seek tra tracce.
FCFS: 20->10->22->20->2->40->6->38 → =10+12+2+18+38+34+32=146*2msec=292msec
SSF: 20->20->22->10->6->2->38->40 →=60*2=120msec
Look: 20->22->38->40->10->6->2 →=58*2=116msec
C-look:20->22->38->40->3->6->10 →=66*2=132msec

Gestione degli errori

I produttori di dischi spingono sempre i limiti della tecnologia aumentando le densità di bit lineari (ovvero il numero di bit nell'unità di spazio) sui dischi. Ragion per cui, i difetti di fabbrica sono inevitabili.
La superfice magnetica di ogni piatto è divisa in piccole regioni magnetiche, più piccole in dimensione di micrometri, e ognuna rappresenta 1 bit, alta densità richiede regioni molto uniformi.
ESERCIZIO di esempio:
DATI
-densità lineare su disco 5.25", -traccia nel mezzo con circonferenza di 300mm,
-300 settori da 512B ciascuno.
-INCOGNITA: calcola la densità di registrazione lineare:
-CALCOLI:

Densità di registrazione lineare = $\frac{\boldsymbol{dimensione\ traccia\ (track\ size)}}{\boldsymbol{lunghezza\ traccia\ (track\ lenght)}}$

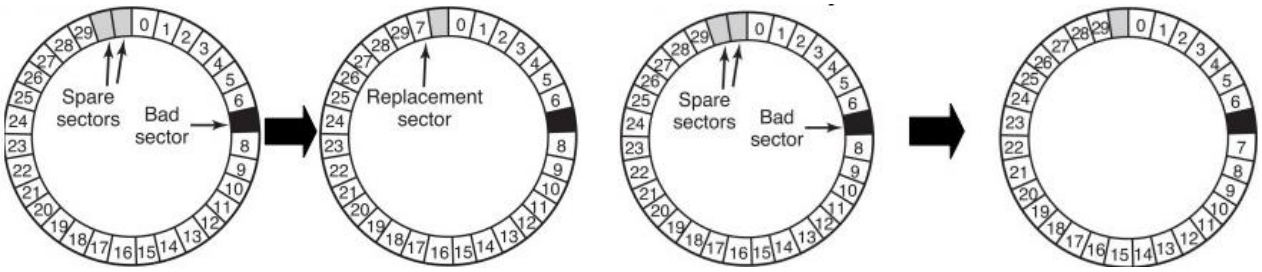
Track size = 300*512*8bit = 1228800bit

Densità di registrazione lineare = track size/track length = (1228800 bit/300 mm) = 4096 bit/mm
Che sale a circa a 5000 bit/mm se consideriamo anche il preambolo e l'ECC.

I difetti di fabbricazione introducono i settori danneggiati.
-Bad sectors (settori danneggiati)
Sono settori che non rileggono correttamente i valori appena scritti.
→Se **il difetto è piccolo** controllo e correggo con l'ECC.
→Se **il difetto è grande** devo gestirlo in un altro modo.
Ci sono due approcci generali per affrontare i settori danneggiati se il difetto è grande:
- A livello di controllore
- A livello di S.O.

Gestione errori grandi:

- **<u>Approccio a livello di controller</u>**: tutti gli HD hanno dei spare sectors ovvero dei settori di riserva che possono essere usati dal controller del disco per rimpiazzare i bad sectors. Questo approccio è del tutto trasparente al S.O.
  Esistono due modi per sostituire i settori danneggiati:
  -Sostituzione con <u>sector forwarding</u>: Il controller rimappa il singolo settore che non sarà più sequenziale. Può aumentare i futuri tempi di accesso. (figura di sinistra)
  -Sostituzione con <u>sector slipping:</u> tutti i settori scivolano di uno. (figura di destra)



  Per segnare la nuova posizione o uso una tabella interna per ogni traccia o riscrivo il preambolo per rimappare i numeri di settore.
- **<u>Approccio a livello software</u>:** usato se un controller non sa mappare o ho finito i settori di scorta. Creo una lista di bad sectors, scansiono il disco e quando trovo il settore danneggiato o i settori danneggiati rimappo la tabella.
  Il S.O. crea anche un file di sistema segreto composto da tutti settori danneggiati, per assicurarsi che i settori danneggiati non siano allocati a nessun file utente.

ERRORI DEL BRACCIO (MECCANICI)

Il controller tiene traccia della posizione del braccio, per il seek, ordina al braccio di muoversi e all'arrivo legge il preambolo della posizione, se sbagliata c'è un errore di seek.
-Vecchi controllers: mettono un bit di errore e il driver ricalibra il comando di muovere il braccio all'inizio e lo resetta. I controller moderni lo fanno automaticamente.
Se non basta ricalibrare, si usa un pin di reset che forza il controller a bloccarsi e resettare, se non basta viene stampato un messaggio di errore. In questo caso è necessario riparare o sostituire l'intera unità.

INTERFACCIA UTENTE

Permette alle persone di interagire con il computer, include SW di input (tastiera, mouse, touch screen) e di Output (monitor, stampante, touch screen).

1-Tastiera: la tastiera contiene un microprocessore,

- comunica con una porta seriale PS/2 o una USB con controller nella scheda madre.
- **Interrupt:** esso viene generato ogni volta che viene premuto un tasto o viene rilasciata una chiave. Un driver estrae le informazioni riguardo ciò che succede nella porta I/O associata al tasto.
- **Scan-code:** numero id del tasto premuto, 1 byte (7 bits x tasto+1 premuto/rilasciato). Viene convertito in un charcter code del character set usato (ACII o unicode)
- **Driver di tastiera:** tiene traccia dello stato di ogni tasto-> posso distinguere maiuscole da minuscole e fare combinazioni con Shift, alt, ctrl... Deve interpretare tutte le combinazioni con gli stessi risultati es: premo shift, premo A, lascio A, lascio shift e premo shift, premo a, lascio shift, lascio A. Interfaccia semplice ma flessibile

  I driver della tastiera possono lavorare in modo non canonico (raw) orientato a caratteri e non modificato oppure in modo canonico tenendo conto solo del risultato.

  ES: cosp <- <- <- iao invio = ciao (in canonico) o tutti gli 11 tasti in non canonico.

  Molte sequenze in modalità canonica hanno significati particolari (es ctrl+...).

  Posso cambiare la modalità.
- **Buffering:** i buffer sono necessari in entrambe le modalità, il programma può non avere ancora richiesto l'input e quindi i buffer devono contenere anche i tasti "digitati ma non consumati".

2-Mouse: sono più semplici delle tastiere, poiché, hanno meno pulsanti, ruote o sensori touch...

- **Interrupt:** ogni 0.1 mm (mickey) e max 40 msg/sec. Indicano posizioni relative all'ultimo messaggio.
- Messaggi contengono: Δx e Δy per posizioni e i pulsanti

ESERCIZIO

DATI: -mouse con movimento max rate=2cm/sec e ogni messaggio è di 0.1mm e 3 bytes.

INCOGNITA: -Calcola il max data rate (b/sec) se ogni mickey venisse considerato separato.

CALCOLI:

-Velocità massima di movimento in mm/sec = 2cm/sec = 20mm/sec messaggi in 1 sec.

-Il numero massimo di messaggi in un secondo = 20mm/sec*0.1mm = 200messaggi/sec.

-La velocità dati massima = 200messaggi/sec*3B/messaggi = 600B/sec

Consumo energetico

Per ridurre l'impatto ambientale, costi e aumentare batteria dei dispositivi.

Consumo di un computer moderno: 200W *10^8 per un singolo computer. Se consideriamo tutti i computer nel mondo l'energia prodotta è pari a quella di 20 impianti nucleari.

Componenti che consumano di più: display, CPU, HD.

COME RIDURRE I CONSUMI?

- Livello hardware: ridurre sprechi e spegnere. Progressi lenti.
- Livello software: spegnere parti non usate (o che sono in standby) andando a ridurre le prestazioni.

Stati di un dispositivo: il cambiamento di stato richiede tempo ed energia.

- On: (in uso).
- Off: (senza energia, non in uso).
- Sleeping: riduco i consumi perché non lo devo usare a breve.
- Ibernazione: grande risparmio ma ci vuole tempo a riattivarlo (buono se non lo uso per tanto)

La transizione tra due stati di alimentazione richiede tempo e consuma energia. Spetta al S.O. gestire le transizioni dello stato di alimentazione al momento giusto.

ESEMPIO: Spegnimento di un dispositivo: Quale spengo? Quando? Spegnere per poco tempo può causare una transizione che costa più del risparmio. Se si aspetta troppo a lungo per spegnere un dispositivo inattivo, l'energia viene sprecata per niente.

Sono necessari algoritmi che consentono al S.O. di prendere decisioni: quantitative (quanta energia salvo rispetto a quella che uso) e soggettive (quanto ritardo per accensione è tollerabile).

SOLUZIONI PER RISPARMIO ENERGETICO

1-Schermi LCD illuminati da dietro, posso spegnere parti del display quando non le uso per qualche minuto e riattivarle istantaneamente.

2-Hard Disk: far girare il disco magnetico consuma energia, perciò, lo fermo quando non serve da notare che la ripartenza è lenta e quindi anche il riavvio consuma molta energia. Bisogna studiare quando conviene spegnerlo.

3-Break-even point (punto di pareggio): pagina dopo

<u>3-Break-even point (punto di pareggio)</u>: È il periodo per cui l'energia per rendere il disco low-power e farlo ripartire poi è la STESSA di tenerlo attivo.

Esd + Ps*(Td-Tsd-Twu) + Ewu = Pw*Td. Pertanto, il punto di pareggio è dato da:

$$\textbf{Td} = \frac{Esd + Ewu - Ps*(Tsd+Twu)}{Pw-Ps}$$

-Esd: È l'energia spesa per mettere il disco in risparmio energetico.

-Ps: È il consumo del disco in modalità energetico istantaneo.

-Pw: È il consumo del disco in modalità attiva istantaneo.

-(Td-Tsd-Twu): È il tempo speso in modalità energetico.

-Ewu: È il consumo energetico per mettere il disco in modalità attiva.

-se **t < td** vuol dire che ci vuole meno energia per far girare il disco piuttosto che farlo girare e poi farlo girare di nuovo.

-se **t > td** vuol dire risparmio più energia a far girare il disco verso il basso è poi rialzarlo più avanti.

4-L'hardware può fare un prefetch ovvero dei dischi in una cache larga, se un blocco richiesto è in cache, il disco "fermo" non deve ripartire.

5-Il SO può rimandare delle operazioni che richiedono il disco a quando sarà attivo è molto complesso da implementare.

GESTIONE DEL'ALIMENTAZIONE DELLA CPU (CPU POWER MANAGEMENT)

Consumo della CPU: P = Pdinamico + Pstatico.

Pdinamico = è il consumo della CPU quando sta eseguendo calcoli.

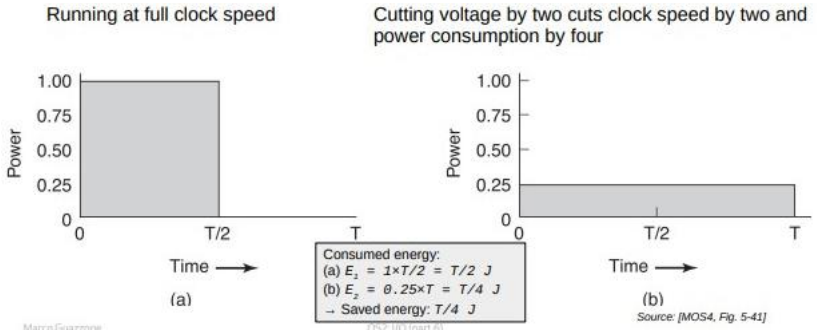
Pstatico = è il consumo della CPU quando non sta eseguendo calcoli.

Proporzionale al quadrato del voltaggio(tensione) e alla frequenza di clock**: P = K* V^2 *F +Pstatico**

V = Tensione, K = costante dipendente da fattori di progettazione, F = frequenza.

La riduzione della tensione riduce anche la frequenza di clock.

Voltaggio dinamico e scaling di frequenza (DVFS): 1/2 voltaggio = 1/2 velocità MA 1/4 consumo.



Problemi: la relazione tra tempo esecuzione e frequenza di clock non è sempre lineare, non posso sempre ridurre la frequenza di clock quanto voglio, riduco le prestazioni e spesso maggior parte del consumo è statico non dinamico.

P-state (performance state): usati per ridurre il consumo di energia delle CPU cambiando voltaggio e frequenze durante le operazioni. P0=max frequenza e consumo ->se aumenta la P riduco frequenza e consumi (max P16).
C-states (power states): usati per ridurre i consumi spegnendo alcuni componenti.
C0= stato operativo, C1: stato di halt (non lavora ma si sveglia velocemente), C2... spengono delle componenti.

ESERCIZIO: programma decomprime e mostra a schermo un frame ogni 40msec.
DATI
-Pstatico = 0,
-Piena potenza: consuma x J in 40 msec e svolge il suo lavoro in 20msec -> consuma x/2 J.
#Strategia 1: esegui a piena potenza per 20 msec e poi spento:
E (energia consumata) = x/2J
#Strategia 2: metà potenza per 40 msec:
E (energia consumata) = x/4 J => È la MIGLIORE.

ESERCIZIO:
-Un utente digita 1 char/sec = 1000 char/msec e 100msec per processarlo.
-Si supponga che la tensione massima V venga ridotta a V/n in modo che il consumo di energia P scende a P/n^2.
-Trovare il valore ottimo di n?
-Qual è la corrispondente percentuale di risparmio energetico rispetto al mancato taglio della tensione?
CALCOLI
Il valore ottimale di n è 10, poiché, bisogna fare la divisione fra il tempo di arrivo del prossimo carattere fratto il tempo che impiega la CPU per elaborare un carattere, quindi:
→ n = 10 poiché (1000 char/msec)/(100msec) = 10
-L'energia consumata in un secondo quando la tensione non viene interrotta
E_{nocut} = Pd * Tproc (tempo per processare un char) + Ps * (Ttot -Tproc).
E_{nocut} = Energia senza tagli= P*0.1sec + 0*0.9sec=0.1*P J
-L'energia consumata in 1 sec quando la tensione viene interrotta di n=10 è:
$E_{cut} = Pd/n^2 * Ttot + Ps *(Ttot - Tot).$
E_{cut} = Energia con tagli: (P/100) *1sec = 0.01*P J
-**Il risparmio energetico è quindi di:**
$\Delta E = E_{nocut} - E_{cut}$
-**Ciò corrisponde a una percentuale di risparmio energetico di:**
$\Delta E\% = \frac{100 * \Delta E}{Enocut}$, $\Delta E\%$ è del 90%

SOLUZIONE ESERCIZIO CONSUMO
n=frequenza operazione/durata del lavoro
E senza tagli per durata lavoro: E=1*energia consumata/n+0*energia non consumata* tempo in cui è spento=(1P* 1/n)+(0P*n-1)=1P in 1sec
E con tagli: energia consumata/n^2 in 1sec
Se c'è il P statico, devo sommarlo ad entrambe per n/n secondi
E tagliata: E non tagliata= x: 100 per % energia consumata->calcolo con differenza il risparmio}

Approcci per risparmiare energia:

- Metto la cache in sleep state: spengo cache L2 o L3, la mem principale rimane accesa, la cache può essere sempre riaccesa senza perdere info in modo dinamico e veloce
- Metto la main memory in hibernate state:iberno la memoria, scrivo il contenuto sul disco, spengo la memoria->Più risparmio per lunghi periodi, serve spazio su disco.

Dispositivi di risparmio energetico wireless:
Ricevitori: sono sempre attivi per captare i messaggi, per spegnerli senza perdere informazioni devo usare degli SMART POINTS: allo spegnimento inviano un messaggio all'access point che registra i messaggi ricevuti e li manda al ricevitore quando si attiva.
Radio trasmettitori: si spengono, il buffer riceve i messaggi e quando è pieno sveglia il radio trasmettitore. Quando spengo? guidato da user, app o timer. Quando accendo? User, app, timer, periodicamente o a buffer pieno.

THERMAL MANAGEMENT: I PC si surriscaldano -> ventole per raffreddamento (sempre accese nei portatili) -> consumo energia e riduce batteria. Spengo e accendo solo quando molto caldo.

ADVANCE CONFIG. AND POWER INTERFACE: specifiche per risparmio energia, configurazione, monitoring... Comunica tutte le informazioni riguardanti calore, batteria...

Alcune APPs hanno la modalità "risparmio energetico" per consumare meno limitando le funzionalità.

FILE SYSTEM

Il File System è quel componente che si occupa dei file per la loro memorizzazione. Ogni applicazione deve poter salvare e recuperare a lungo termine le informazioni.
I requisiti per l'archiviazione a lungo termine delle informazioni sono:

- Ci deve essere una grande capacità di memorizzazione.
- Sopravvivenza delle informazioni dopo la terminazione del processo.
- Accesso in contemporanea da parte di più processi. CANONICI

La memoria principale non rispetta tali requisiti, perciò, si usa una memoria secondaria (es: dischi).
→**Come salvo le informazioni? Dove? Come gestisco i permessi? Come so quale blocco è libero?**
→**Uso un file**: un dispositivo ne può contenere tanti, leggibili, duplicabili e creabili, persistenti...

Per studiare un file system ci sono **due punti di vista**:
1-Utente: da questo punto di vista il file system è il servizio fornito dal S.O. per il salvataggio e il recupero delle informazioni salvate. I tipici problemi risolti da un file system sono la struttura, la nomina, la protezione e le operazioni su un file, si ignorano i dettagli a basso livello.
2-System designer (progettista del sistema): da questo punto di vista il file system è un insieme di strutture dati e meccanismi usati per gestire in modo efficiente un dispositivo di memorizzazione per salvare o recuperare informazioni. Si studiano la larghezza dei blocchi logici, l'implementazione e la mappatura degli spazi liberi.

Dal punto di vista dell'utente
FILE: Un file è l'unità di archiviazione logica dei dati.

1-Nome (file naming): Ogni S.O. ha le sue regole. Una certa lunghezza, numeri e caratteri speciali ammessi (non tutti), case-sensitive, case-persevering (lascia maiuscolo o minuscolo come scritto, non mette tutto in un modo), supporto o no del ".", estensione (non sempre obbligatoria, a volte più di una).

Extension	Meaning
.bak	Backup file
.c	C source program
.gif	Compuserve Graphical Interchange Format image
.hlp	Help file
.html	World Wide Web HyperText Markup Language document
.jpg	Still picture encoded with the JPEG standard
.mp3	Music encoded in MPEG layer 3 audio format
.mpg	Movie encoded with the MPEG standard
.o	Object file (compiler output, not yet linked)
.pdf	Portable Document Format file
.ps	PostScript file
.tex	Input for the TEX formatting program
.txt	General text file
.zip	Compressed archive

2-File structure: esistono tre modi per strutturare un file e sono:
1-sequenza di byte: In questo caso un file è una sequenza non strutturata di byte e il S.O., non conosce il contenuto dei file. Si fornisce il massimo della flessibilità ovvero non c'è alcun vincolo da parte del S.O. È l'approccio usato da Windows e Unix.
2-sequenza di record: In questo caso un file è una sequenza di record di lunghezza fissa. È poco usato. (unità di misura = record di lunghezza fissa)
3-Albero di record: In questo caso un file è un albero di record in cui, ogni record ha un campo chiave. L'albero è ordinato in base alla chiave. Questa struttura ad albero viene gestita dal S.O.

3-Tipo di file:

1-File normali: di testo (sequenze di caratteri stampabili) o binari (non ascii, non leggibili)

2-Cartelle: file di sistema per mantenere la struttura di un file system.

3-Devices files: a caratteri speciali o a blocchi speciali.

Alcuni S.O. hanno un file system fortemente tipizzato ovvero che riconoscono automaticamente in un range molto ampio, user-friendly ma poco flessibile un'ampia gamma di tipi di file.

4-Accesso: Ci sono due tipi di accessi:

1-Sequenziale: I byte o i record sono letti/scritti in ordine a partire dall'inizio verso la fine, e non posso andare fuori ordine ed inoltre, per leggere una posizione devo aver letto le precedenti.

2-Casuale/Diretto: I byte e i record possono essere letti/scritti in qualsiasi ordine, senza avere accesso alle posizioni precedenti. Per far ciò Usano il la funzione seek (che incrementa/decrementa di poco il puntatore alla posizione corrente ovvero l'offset) o un indirizzo di posizione.

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

5-Attributi (metadati): proprietà associate ad ogni file: data e ora di creazione, ultima modifica, permessi...

6-Operazioni: set di comandi e funzioni per interagire con i file.

1-Create: crea un file vuoto (aggiunge anche attributi default),

2-delete: rimuove un file esistente (libera spazio),

3-open: apre un file prima di usarlo (preleva le informazioni dal disco alla memoria principale, +1 processi aperti),

4-close: chiude un file aperto (se sta ancora scrivendo salva le info e distrugge strutture dati dalla memoria principale ma non quelle su disco, -1 processi aperti),

5-read: legge i dati dal file dalla posizione corrente

6-write: scrive i dati nel file dalla posizione corrente (può aumentare la size, sovrascrivere...), 7-7-7-

7-rename: cambia il nome di un file esistente**,**

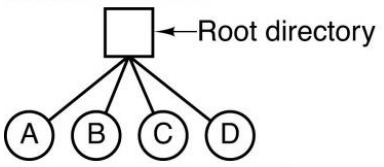
8-seek: cambia posizione al puntatore per il file ad accesso casuale,

9-append: aggiunge i dati alla fine del file,

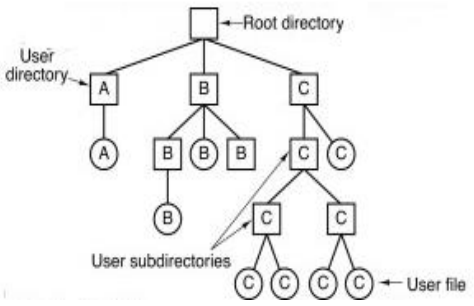
10-getAttributes: legge i metadati di un file,

11-setAttributes: cambia i metadati di un file

DIRECTORY: sono un modo per organizzare collezioni di file nel file system, ogni file system contiene una root directory e ogni file system ha almeno una directory chiamata directory radice. In modo informale, una directory è un file che contiene un elenco di nomi di file con informazioni associate.
Possono avere diverse organizzazioni:



- **Livello singolo:** la root contiene tutto, è facile da implementare ma è limitata poiché, non organizzabile e non va bene per i sistemi multiutente.
- **Gerarchia di cartelle:** strutturata come un albero con la root come radice, le cartelle possono avere sottocartelle, possono esserci cartelle private nei sistemi multiutente.



1-Nome del percorso: il nome del percorso è il modo per specificare il nome di un file in una struttura di directory. Sequenza di nomi separata da un path separator "/". "." = posizione corrente, ".." = parent della posizione corrente.
-Nome del percorso assoluto: percorso dalla radice al file, inizia sempre con la radice.
-Nome del percorso canonico: semplice e diretto (senza . e senza ..)
-Nome del percorso relativo: relativo alla posizione corrente.
2-Operazioni: creazione, cancellazione, readdir (leggi la prossima entry in una cartella aperta), rename, opendir, closedir (aprono e chiudono una directory).

3-Crea collegamento: Il linking è una tecnica che consente a un file di farsi trovare in più di una directory. Le varie operazioni del linking sono:
Lynk: crea un Hard link: crea collegamento reale a un file (o directory) che punta alla stessa struttura interna usata dal S.O. per tenere traccia del file (o directory) collegato.
Symlink: crea un Soft link: crea un collegamento software a un file (o directory), il link è memorizzato in un altro file di tipo link la cui struttura interna contiene il percorso del file (link simbolico).
Unlink: distrugge un link a un file esistente rimuovendo la entry associata. se hard decrementa il contatore dei link.

```c
#include <dirent.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int mkdir(const char *pathname, mode_t mode); /* Create */
int rmdir(const char *pathname); /* Delete */
DIR *opendir(const char *name); /* Opendir */
int closedir(DIR *dirp); /* Closedir */
struct dirent *readdir(DIR *dirp); /* Readdir */
int rename(const char *oldpath, const char *newpath); /* Rename */
int link(const char *oldpath, const char *newpath); /* Link (hard link) */
int symlink(const char *target, const char *linkpath); /* Link (symbolic link) */
int unlink(const char pathname); /* Unlink */
…

struct dirent {
  ino_t          d_ino;       /* Inode number */
  off_t          d_off;       /* Position in the directory (non POSIX) */
  unsigned short d_reclen;    /* Length of this record (non POSIX) */
  unsigned char  d_type;      /* Type of file (non POSIX) */
  char           d_name[];    /* Null-terminated filename (max NAME_MAX bytes) */
};
```

File dal punto di vista del progettista

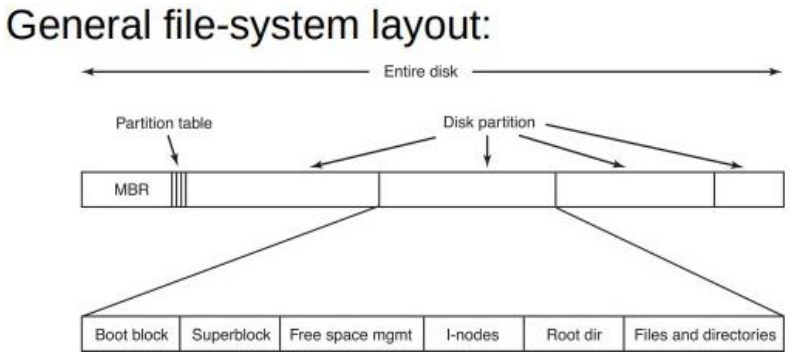
FILE

Il file è una collezione di blocchi contigui (blocco: gruppo di settori di disco consecutivi). **Per ogni file c'è un file-control block** che permette di trovare i blocchi associati al file.

Cartella (directory)

Un file di directory è formato da entries e ognuna associa il nome di un file nella file-control block.

Salvataggio e ritrovamento: I file systems sono salvati in un disco e quelli in una partizione possono essere diversi da quelli nelle altre partizioni. In figura il layout generale di un file system.



1-Boot block (boot sector): Contiene il boot loader del S.O. installato nella sua partizione. La partizione contiene sempre un boot block anche senza un S.O. da avviare. Se non contiene alcun S.O. è vuoto.

2-Superblock (volume control block): È un blocco speciale che contiene i parametri chiave di un file system (File system type ID, num di blocchi, dim blocchi...) viene portato in memoria la prima volta che uso il file system.

3-Free space management: Gestisce le informazioni riguardo i blocchi liberi. Ha una struttura che tiene traccia dei blocchi liberi.

4-I-nodes (file control blocks): informazioni riguardo ogni file. Struttura dati che tiene traccia dei metadati e blocchi associati ad ogni file.

5-Root dir: cartella radice, top del file system tree.

6-File e **Cartelle:** blocco di dati per file e directory.

Montaggio del file system: il file system è salvato su disco ma le strutture dati sono caricate e create nella memoria principale quando sono usate. Utile sia per la gestione del file-system sia per il miglioramento delle performance.

Smontaggio del file system: le strutture dati in memoria sono aggiornate durante le operazioni e anche distrutte al tempo di smontaggio. Tutti i cambiamenti devono essere scritti su disco.

Le tipiche strutture dati in memoria (dipendenti dal SO) sono:

- **Tabella di montaggio**: contiene informazioni riguardo ogni file system montato
- **Cache delle cartelle**: contiene informazioni sulle cartelle a cui si è fatto accesso di recente.
- **Tabella dei file aperti a livello sistema**: contiene una copia dei file control block di ogni file aperto e le loro informazioni. Ogni entry ha un contatore per sapere quanti sono i processi aperti che usano questo file.
- **Tabella file aperti per processo**: contiene file aperti da quel processo.
- **cache dei blocchi**: tiene i blocchi del disco recentemente usati.

OPERAZIONI

1-Creazione di un file(create): crea un nuovo file control block, aggiorna la struttura delle cartelle che lo contengono aggiungendo le entry, se oltre a creare il file lo si apre si svolgono anche altre operazioni.

2-Apertura di un file(open): Il S.O. cerca nella struttura delle cartelle il nome del file dato, quando la trova lo cerca nella tabella dei file aperti a livello di sistema per cercare una voce (entry) associata al file.

Entry trovata: file già aperto

1. nuova entry nei file aperti per programma
2. setta il ptr a una entry della tabella dei file aperti a livello di sistema in una entry della tabella dei file aperti per processo
3. incrementa il contatore dei file aperti di +1.

Entry non trovata: file non aperto (bisogna creare il file ancora)

1. nuova entry tab programmi aperti livello di sistema leggendo il relativo file control block su disco
2. Setta a 1 il counter nella entry della tab file aperti livello di sistema
3. Crea una entry nella tabella file aperti per processo
4. Setta ptr della entry della tab dei file aperti a liv sistema in una entry della tabella dei file aperti per processo.

La open ritorna un puntatore alla entry nella tabella dei file aperti per processo.

3-Chiusura di un file: Rimuovo la entry dalla tabella del file aperti per processo, perciò, decremento il contatore di -1 e se il contatore è a 0 rimuovo la entry della tabella del file aperto a livello di sistema può essere rimossa.

Per le altre operazioni: uso il ptr alla entry della tab per processo.

IMPLEMENTAZIONE FILE

Come assegno i blocchi a un file? Come tengo traccia di quali blocchi sono su quale file? Esistono diversi schemi di allocazione file.

CARATTERIZZAZIONE QUANTITATIVA DI UNO SCHEMA DI ALLOCAZIONE DEI FILE

F = dimensione file in bytes;

B = dimensione blocco in bytes;

<alloc-data> = numero di bytes allocati per salvare dati;

<mgmt> = numero bytes usati per la gestione dell'allocazione dei file;

<alloc-mgmt> = bytes allocati per mgmt;

<allocated> = bytes allocati per un file.

→**È il numero di blocchi allocati(N)**: N >= $\frac{F}{B}$

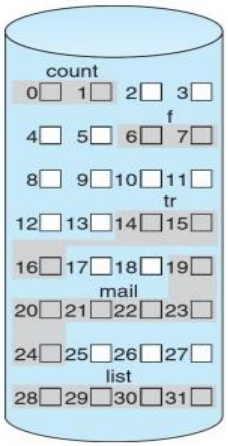
→**Overhead**: ovvero spazio complessivo destinato alla gestione (**<alloc-mgmt>**).

Overhead = ($\frac{<\text{alloc}-\text{mgmt}>}{<\text{allocated}>}$) %

→**Wasted**: ovvero lo spazio sprecato

Wasted = ($1 - \frac{<\text{mgmt}>+\text{F}}{<\text{allocated}>}$)%

1-Schema: *ALLOCAZIONE CONTIGUA*



Salva i file come una sequenza contigua di disk blocks (blocchi del disco). (vedi figura a destra)
Considero 20KiB file:
-1Kib disk block->File di 50 blocchi;
-2Kib disk block->File di 25 blocchi.
Le entry delle directory contengono oltre al nome del file: indirizzo disco del primo blocco e il numero di blocchi. (vedi figura in basso)
Ogni file inizia al blocco seguente la fine del precedente.
VANTAGGI) È molto semplice da implementare e inoltre, supporta sia l'accesso sequenziale e sia quello diretto. Il numero di ricerche (Seek) è minimo alte prestazioni.
SVANTAGGI) Ci sono due svantaggi:
1-**Frammentazione esterna:** se cancello un file rimane un buco sul disco non facile da riempire. Una possibile soluzione è la **deframmentazione (compatto i dati)** ma è costosa.
2-**Devo conoscere quanto spazio serve per ogni file**: difficile da sapere, in questo caso il S.O. può richiederlo all'utente oppure può stimarlo ma sono comunque informazioni difficili da stimare.

directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Descrizione formule per caratterizzazione dell'allocazione contigua
-F' = dimensione stimata;
-F = dimensione attuale del file;
-B = dimensione blocco del disco;
-numero di blocchi N → $N = \left(\frac{F'}{B}\right)$; -overhead% = 0; -wasted% = $\left(1 - \frac{F}{N*B}\right)\%$
ESERCIZIO
DATI: F' = 8Kib; F = 4Kib; B =1Kib
INCOGNITA: caratterizzazione dell'allocazione contigua?
CALCOLI:
-N=8/1=8 blocchi di disco allocati per questo file; -Overhead%=0; -Wasted%= $\left(1 - \frac{4\text{Kib}}{8*1\text{Kib}}\right)\% = 50\%$
2-Schema *ALLOCAZIONE ESTENDIBILE*:
È uno schema di allocazione contigua migliorato, infatti, ogni file consiste in una o più regioni non contigue di blocchi contigui (detti **extents**).
All'inizio, quando viene creato un file, viene allocata una sequenza contigua di blocchi liberi, quindi, se le dimensioni del file aumentano, viene aggiunto un altro blocco di spazio contiguo. Nota che gli extents hanno una lunghezza variabile o fissa. Ogni file ha una tabella degli extents per tener traccia degli extents allocati.
VANTAGGI) Non devo conoscere la dimensione in anticipo, inoltre, non è presente la frammentazione esterna per la lunghezza fissa infatti riuso un buco lasciato dalla rimozione del file estendendolo quindi, uno spazio vuoto verrà riusato in una nuova misura.
→Entry per la lunghezza variabile: <indirizzo primo blocco, numero blocchi>
→Entry per la lunghezza fissa: <indirizzo primo blocco>

SVANTAGGI) Costoso, inoltre, maggiore Overhead per tenere traccia delle estensioni associate a ciascun file. Le estensioni (extents) a lunghezza fissa causano frammentazione interna, mentre le estensioni (extents) a lunghezza variabile causano frammentazione esterna.

Descrizione formule per caratterizzazione estensioni di lunghezza fissa

-F: dimensione file;

-B: dimensione del blocco;

-P: dimensione del puntatore blocco del disco;

-E: numero di blocchi per estensioni (extents);

-numero di extents allocati (M): $M = (\frac{F}{E*B})$

-numero di blocchi necessari per salvare la tabella degli extents (T): $T = (\frac{M*P}{B})$

-numero di blocchi allocati (N): $N = (M*E) + T$

-overhead%: (spazio necessario per salvare la tabella delle estensioni / spazio allocato)

overhead% = $(\frac{T*B}{N*B})\% = (\frac{T}{N})\%$

-wasted%: (1 - (spazio utile per salvare la tabella delle estensioni + dimensione file/ spazio allocato))

wasted% = $(1 - \frac{(M*P)+F}{N*B})\%$

ESERCIZIO:

DATI: F = 4Kib, B = 1Kib, P = 4 byte, E = 4 blocchi

INCOGNITA: caratterizzazione estensioni di lunghezza fissa?

CALCOLI:

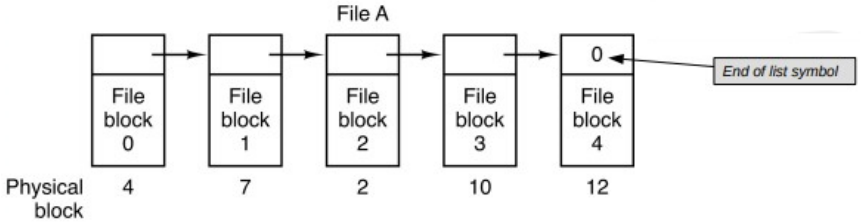
M = 4/4*1=1 numero extent allocati,

T= 1*4B/1Kib = 1, N=1*4+1 = 5, -Overhead% = 1/5 = 20%, -Wasted% = 19.92%

3-Schema: ***Allocazione in LINKED-LIST***

I blocchi di un file sono organizzati come una linked-list, in cui, ogni blocco contiene alcuni dati e un puntatore al blocco successivo.

Le entry delle cartelle contengono solo l'indirizzo del primo blocco (head).



VANTAGGI) I file possono crescere senza liniti ed inoltre, l'accesso sequenziale è semplice da implementare. Non esiste nessuna frammentazione esterna: perciò non è necessario fare la deframmentazione del disco.

SVANTAGGI) Le prestazioni non sono molto buone poiché l'accesso sequenziale può richiedere diverse operazioni di ricerca. Inoltre, c'è un alto overhead poiché ogni blocco deve contenere le informazioni per il puntatore al blocco successivo. Infine, non affidabile poiché, la perdita o il guasto di puntatore al blocco successivo fa perdere il resto della lista che era collegato ad esso.

Descrizione formule per caratterizzazione dell'allocazione alle liste

F = dimensione del file

B = dimensione del blocco del disco

P = dimensione del puntatore del blocco del disco

-numero di blocchi allocati (N): N= $\frac{F}{B-P}$

-overhead%: (spazio utile per memorizzare il blocco del puntatore / spazio allocato)

Overhead% = $\left(\frac{N*P}{N*B}\right)\% = \frac{P}{B}\%$

-**wasted%:**1 – ((spazio utile per salvare il blocco del puntatore + dimensione del file)/ spazio allocato)

Wasted% = $1\text{-}(\frac{N*P+F}{N*B})\%$

ESERCIZIO

DATI: F=4Kib; B=1Kib; P=4B

INCOGNITA: caratterizzazione dell'allocazione alle liste?

CALCOLI:

-N = F/ (B – P) = 4kIB/(1KIB-4b) = 5; -Overhead% = (P/B) % = 0.39%

-Wasted% = 1 - ((N*P+F)/(N*B)) % = 19.61%

4-Schema: ***allocazione su "Cluster":*** estende lo schema a linked-list

È una collezione di blocchi contigui in "cluster" di lunghezza fissa invece che blocchi. Ogni cluster contiene un puntatore al cluster successivo. È simile all'allocazione basta sull'estensione, ma non usa una tabella separata per archiviare i puntatori ai cluster.

Un cluster è composto da quattro blocchi e un puntatore al cluster successivo.

VANTAGGI) È presente un basso sovraccarico poiché è presente un puntatore per ogni cluster invece di uno per ogni blocco. Inoltre, ha delle performance migliori delle liste poiché: nell'accesso sequenziale fa meno ricerche (seek) perché i blocchi nel cluster sono contigui mentre nell'accesso casuale (random) perché scansiono solo i cluster e accedo direttamente ai blocchi all'interno di un cluster.

SVANTAGGI) È presente la frammentazione interna nei cluster. Infine, non affidabile poiché, la perdita o il guasto di puntatore al blocco successivo fa perdere il resto della lista che era collegato ad esso.

Descrizione formule per caratterizzazione dell'allocazione basata su cluster

F = dimensione del file

B = dimensione del blocco del disco

P = dimensione del puntatore del blocco del disco

C = numero di blocchi per cluster (lunghezza del cluster)

-numero di cluster allocati (M): M = $\frac{F}{(C*B)-P}$ **; -numero di blocchi allocati (N): N = M*C**

-overhead%: (spazio utile per memorizzare i blocchi del puntatore / spazio allocato)

Overhead% = $\left(\frac{M*P}{N*B}\right)\% = \frac{P}{B}\%$

-**wasted%:**1 – ((spazio utile per salvare i blocchi del puntatore + dimensione del file)/ spazio allocato)

Wasted% = $1\text{-}(\frac{N*P+F}{N*B})\%$

ESERCIZIO:
DATI: F=4Kib; B=1Kib; P=4B; Cluster lenght C=4.
INCOGNITA: caratteristiche dell'allocazione basata su cluster?
CALCOLI:
-M = 4Kib/(4*1Kib-4B) = 2; -N=4*2=8; -Overhead = (4B/4*1Kib) % = 0,10%.
-Wasted= 1-((2*4B+4Kib)/(8*1Kib)) % = 49,90 %.
5-Schema: *linked-list +FAT:* estende lo schema di allocazione delle liste
In questo schema si inseriscono i puntatori ai blocchi successivi in una tabella di allocazione dei file (FAT). Il FAT (file allocation table) è una tabella dell'allocazione dei file, contiene un'entry per ogni blocco del disco, e ogni entry contiene il puntatore al blocco successivo o un simbolo speciale se è l'ultimo blocco. Il FAT viene salvato su disco e per questioni di efficienza sarebbe meglio portarlo nella memoria principale.
VANTAGGI) Lo stesso schema di assegnazione delle linked-list, inoltre non vi è nessun sovraccarico all'interno dei blocchi del disco ed infine l'accesso casuale è molto più veloce rispetto allo schema delle liste.
SVANTAGGI) Quando FAT viene portato nella memoria principale, può richiedere molta RAM. Infine, l'affidabilità è ancora un problema.
Descrizione formule per caratterizzazione allocazione di liste con FAT
F = dimensione del file
B = dimensione del blocco del disco
P = dimensione del puntatore del blocco del disco
D = dimensione partizione del disco
-numero di voci FAT (lunghezza FAT) (M): M = $\frac{D}{B}$;
-dimensione FAT (S): (lunghezza FAT) * (dimensione puntatore del blocco disco): S = M*P;
-numero di blocchi allocati per file (non considerare FAT): N = $\frac{F}{B}$;
-overhead % = dimensione del FAT / dimensione della partizione del disco; -overhead% = $\frac{S}{D}$ %;
-wasted % = (1 – ($\frac{F}{N*B}$))% ;
ESERCIZIO
DATI: F=4kiB; B=1KiB; P=4B; disk partition size: D=256GiB
INCOGNITA: caratterizzazione allocazione di liste con FAT
CALCOLI:
-numero di void nel FAT: M = D/B = 256GiB/1KiB = 2^{28} B
-dimensione FAT: S = M*P = 2^{28} B*4B = CIRCA 1GiB
-Blocchi allocati: N=F/B=4KiB/1KiB = 4;
-Overhead = 0,39%;
-Wasted = (1-4Kib/(4*1Kib))% = 0%

6-Schema: *allocazione indicizzata*

Ogni file ha la propria tabella delle allocazioni (detta index block) salvata in uno o più blocchi contenenti gli indirizzi del blocco.

La entry-iesima punta al blocco-iesimo, per trovare il blocco "i-esimo" uso il puntatore all'iesima entry. L'index block viene caricato in memoria per motivi di efficienza ma, solo quando il file correlato è aperto, e viene scaricato quando il file è chiuso.

VANTAGGI) Occupa meno spazio della FAT. Lo stesso dello schema di allocazione delle liste. Non è presente overhead all'interno dei blocchi di dati poiché, i puntatori ai blocchi dati vengono memorizzati nel blocco dell'indice. Accesso casuale rapido, efficiente e facile da implementare.

SVANTAGGI) L'overhead e lo spazio sprecato dell'allocazione indicizzata è maggiore rispetto a quello delle linked-list. La dimensione dei file è limitata dalla dimensione dell'index block.

Descrizione formule per caratterizzazione allocazione indicizzata

F = dimensione del file

B = dimensione del blocco del disco

P = dimensione del puntatore del blocco del disco

I = numero di blocchi del disco allocati al blocco dell'indice

-dimensione del blocco dell'indice: S = I*B ; -numero di blocchi allocati sole per i dati: M = $\frac{F}{B}$;

-numero di blocchi allocati per il file: N = I+M;

-overhead% = (dimensione del blocco indice/spazio allocato); -overhead% = $(\frac{S}{N*B})\% = \left(\frac{I}{N}\right)\%$

-wasted% = (1 - $\frac{P*M+F}{N*B}$**) ;**

ESERCIZIO

DATI: block Ptrs size **P**:4B; disk block size **B**:1Kib; index block size **I**: 1 disk block, 4 disk blocks

INCOGNITA: Trovare la dimensione massima del file?

CALCOLI:

Con 1Kib di index block abbiamo: $1Kib/2^2B = 2^{10}B/2^2B=2^8$ entries * 1Kib=256Kib

Con 4Kib di index block abbiamo: = $4Kib/2^2B = 2^{12}B /2^2B =2^{10}B$ entries*1Kib=1Mib

ESERCIZIO

DATI: F = 4KiB; B = 1KiB; P = 4B; I = 1;

INCOGNITA: caratterizzazione dell'allocazione indicizzata?

CALCOLI:

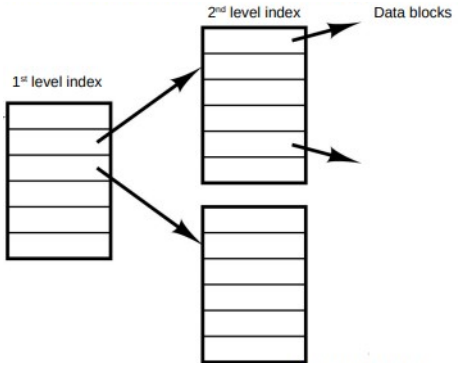
-M = F/B = 4KiB/1KiB = 4; -N = I+M = 1+4 = 5; -overhead% = (I/N) % = (1/5) %= 20%;

-wasted% = (1 - $\frac{4B*4+4KiB}{5*1KiB}$)% = 19,69%**;**

Quanto dev'essere grande l'index block?

Deve essere piccola per ridurre overhead e sprechi ma non troppo, perché, se è troppo piccolo, poi potrebbe non essere in grado di contenere abbastanza puntatori per un file di grandi dimensioni.
Esistono diverse estensioni di implementazione dei file con allocazioni indicizzate:

- **1.linked scheme**: un file può avere diversi index block organizzati come una linked-list in cui, ogni blocco ha una entry per salvare il puntatore al prossimo blocco. Con questo schema non c'è nessun limite di dimensioni per i file.
- **2.multilevel scheme**: ogni livello di blocchi punta a un set del livello inferiore fino all'ultimo livello che punta al blocco del file. La dimensione massima di un file è determinata dal numero di livelli.



ESEMPIO

DATI:

B (dimensione del blocco) = 4Kib;
P (dimensione del puntatore del blocco) = 4B;
index block size: 4Kib (ovvero, il blocco dell'indice richiede 1 blocco del disco).

CALCOLI:

-indice a livello singolo

Dimensione massima del file = 4KiB*$\frac{4096B}{4B}$ = 4MiB

-indice a due livelli

Dimensione massima del file = 4KiB*$\frac{4096B}{4B}$ *$\frac{4096B}{4B}$ = 4GiB

- **3.combined scheme (index-node** o **i-node):** in questo schema c'è un blocco di indice principale: index-node (o i-node).
  Le prime n entries del nodo sono puntatori diretti al blocco, che contengono gli indirizzi ai blocchi dei file, le altre entries contengono puntatori di blocco indiretti che contengono indirizzi di indici multilivello.

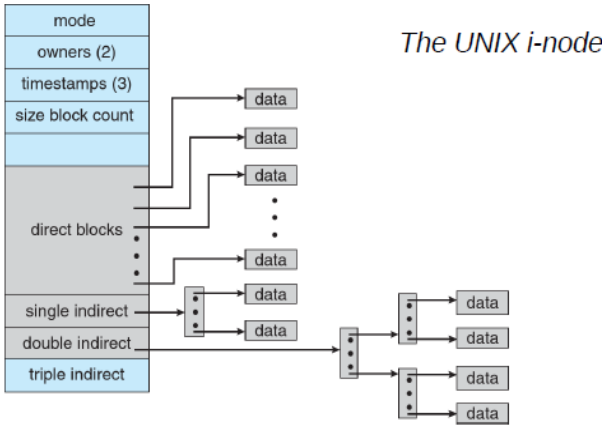
ESERCIZIO:

DATI: 12 puntatori a blocchi diretti + 3 puntatori di blocco indiretti singoli, doppi o tripli.

-puntatore a blocco indiretto singolo: punta a un blocco di indice a livello singolo.

-puntatore a doppio blocco indiretto: punta a un blocco di indice a due livelli.

-puntatore a triplo blocco indiretto: punta a un blocco di indice a tre livelli.



La dimensione e la struttura del nodo indice determinano la dimensione massima del file:

-N puntatori diretti; -M puntatori indiretti; -Ogni blocco di indice è lungo L voci.

→L'i-esimo puntatore indiretto con i>=1 punta a un blocco di indice di livello i.

Per N puntatori diretti e M puntatori indiretti, ogni index block ha L entries.

-Numero di data block (blocchi dati) indirizzabili: $A = N + \sum_{i=1}^{M} L^i$

-La dimensione massima del file: S=A*B (con B bytes per blocco)

ESERCIZIO

DATI: N=10, M=3, L=256, B=1Kib

INCOGNITA: Qual è la dimensione massima del file?

CALCOLI:

-$A = 10 + 256^1 + 256^2 + 256^3 = 16843018$ numero max di blocchi di dati indirizzabili

-S = A*B = 16843018*1Kib = 16.06 GiB

IMPLEMENTAZIONE DI DIRECTORY

Directory: Una directory è una raccolta di entries (una per ogni file contenuto nella cartella) che mappano (associano) un nome di un file alle informazioni necessarie per trovare i blocchi del disco di quel file e tali informazioni variano in base al modo in cui i file sono implementati.

Una cartella può contenere gli attributi di un file o possono essere nel file control block.

Come si gestiscono i nomi dei file all'interno delle entries delle directory? Qui di seguito vediamo quattro opzioni:

1. Non si accettano nomi di file lunghi. Ma non è adatto per l'uso moderno.
2. Uso un campo di lunghezza fissa e setto la sua dimensione molto grande. Questo metodo è semplice ma spreco molto spazio nella directory.
3. Si usano delle entry di lunghezza variabile: ogni entry di directory inizia con un header contenente la lunghezza della entry e degli attributi del file (info su proprietario, creazione...) poi dopo c'è un campo di lunghezza variabile che termina con un carattere speciale.
   SVANTAGGIO di questa opzione) C'è la possibilità che si verifichi la frammentazione esterna per risolvere tale problema deframmento.
4. Si usano delle entry con lunghezza fissa e conservo i nomi del file in un'altra area: ogni entry di directory contiene un puntatore all'heap in cui è possibile trovare il nome del file.
   VANTAGGIO di questa opzione) Nessuna frammentazione esterna e inoltre, non è necessario il riempimento per allinearsi ai limiti delle parole di memoria.
   SVANTAGGIO di questa opzione) C'è il problema della frammentazione interna, infatti quando un file viene rimosso, nell'heap viene introdotto uno spazio di dimensioni variabili in cui il nome del file successivo da inserire potrebbe non rientrare. Per risolvere si usa la compattazione dei nomi.

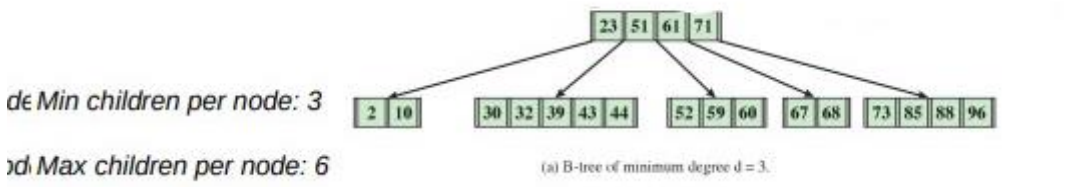
Operazioni chiavi nelle cartelle: **ricerca file** in base al nome, **creazione e cancellazione**.

Implementazione cartelle:

1-lista lineare: In questo caso le voci di directory sono organizzate come un elenco lineare e le operazioni di ricerca per nome di file hanno una scansione lineare ovvero partono sempre dall'inizio dell'elenco per andare verso la fine. È un metodo semplice ma inefficiente.

2-hashtable: Ogni cartella ha una hashtable che mappa il nome di un file in un puntatore alla entry per il file. Le entries sono organizzate come una lista. L'operazione di ricerca viene svolta attraverso il nome del file e si calcola il valore hash per il nome del file e si scansiona le possibili collisioni fino a trovare la corrispondenza esatta. Le operazioni di ricerca sono molto più rapide rispetto alla lista lineare però, è più complesso da gestire rispetto alla lista lineare e inoltre richiede spazio aggiuntivo per archiviare le hashtable.

3-B-tree: È un albero di ricerca auto-bilanciato che ha in ogni nodo un numero variabile di chiavi e figli (>=d, <=2d). Le chiavi sono in ordine crescente. Figlio sx minore e dx maggiore.



(a) B-tree of minimum degree d = 3.

Le operazioni che esistono sono: ricerca, inserimento e cancellazione della chiave.
La complessità massima è uguale a O(log n). Da notare che B-tree è progettato specificamente per l'archiviazione secondaria.

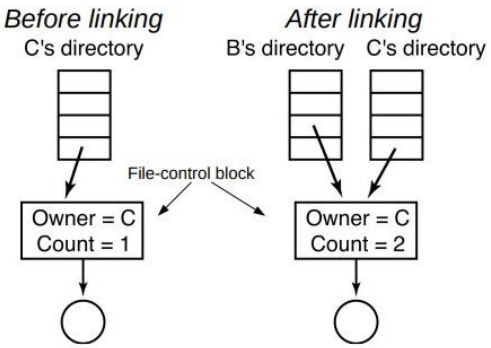
4-Approccio ibrido: Con questo metodo quando ci sono pochi file si usa l'approccio a lista lineare mentre, se ci sono tanti file si usa un approccio a B-tree.

5-Caching (memorizzazione nella cache): mappo quindi associo il percorso al relativo puntatore per una entry della directory. Prima di cercare un nome controllo se è in cache, se è presente lo prendo se no lo cerco e se lo trovo poi lo salvo in cache.

FILE CONDIVISI

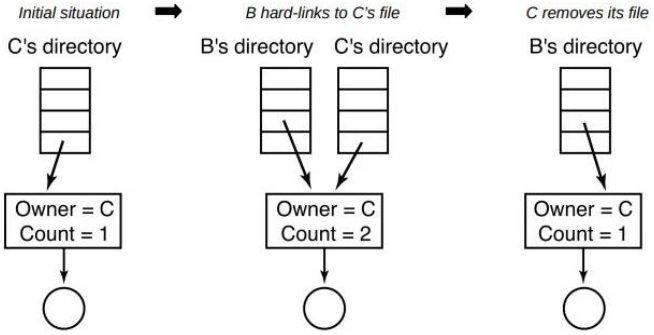
I file condivisi sono file che appaiono contemporaneamente in più cartelle di diversi users.
Le cartelle coinvolte contengono dei link al file condiviso che cambiano l'organizzazione del file system da un albero ad un DAG (grafo aciclico).

Come rendo le modifiche valide/attive per entrambi gli users?
Al solito ci sono diverse soluzioni:

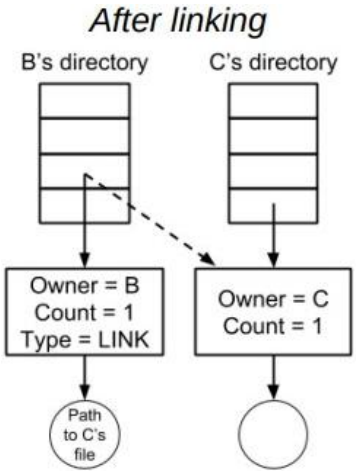


1-Hard linking: I disk block non sono elencati nelle entrys delle directory. I disk block sono nel file-control block del file condiviso. **Gli Hard links sono entries di cartella che puntano al file-control block.**

PROBLEMA) Se il proprietario cancella il file, l'altro utente avrà una entry che punta ad un file non valido o diverso ed è quindi difficile rimuovere tutte le entries. Una possibile soluzione a questo problema è **link-count** ovvero un contatore dei collegamenti a quel file. Se il contatore è a 0 significa che posso rimuovere il file-control block perché nessuna entry punta a quel file, altrimenti se una directory "C" rimuove il file ma link-count è diverso da 0 allora non si può rimuovere il file-control block ma si decrementa solo il valore di link-count.



C può contattare l'amministrazione per cedere il file che non usa più.



2-Symbolic linking: Viene creato un file di tipo link (link simbolico/soft) e viene messo nella cartella di un altro user. Il file contiene solo il path al file originale e non influenza il link-count.

Cosa succede se rimuovo il file originale?
A differenza degli Hard links, solo il vero proprietario di un file ha il puntatore al file-control block del file collegato.
Il link diventa inutile, infatti, quando il proprietario del file rimuove il file-control block corrispondente al file perché il numero di collegamenti è zero. Qualsiasi tentativo di usare il file tramite un linking simbolico avrà un esito negativo poiché il S.O. non è più in grado di individuare il file. Se invece è un utente diverso dal proprietario a rimuovere il collegamento nella sua directory di quel file il file collegato nella directory del proprietario non ne risente.

VANTAGGI) Symbolic links possono puntare alle cartelle. Non sono file veri e propri e quindi sono più facili da trovare. Alcuni S.O. permettono hard-linking solo agli users. Possono collegarsi a diverse partizioni, file systems o pc.
SVANTAGGI) Se il file collegato viene spostato o rimosso, tutti i file link diventano non validi. C'è un overhead aggiuntivo quello per accedere al link e quello per salvarlo.

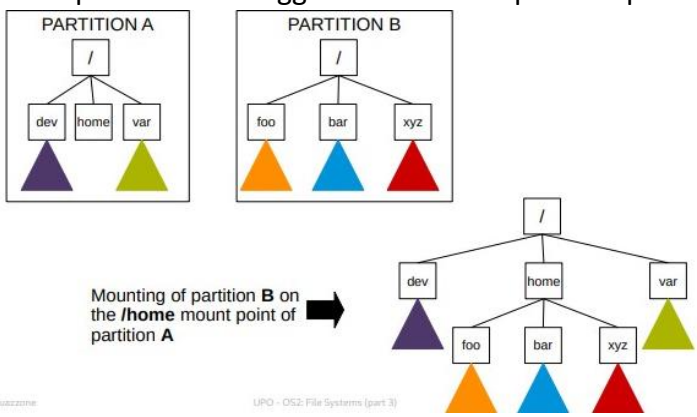
Qualunque sia il tipo di collegamento usato c'è un ulteriore problema da risolvere: ovvero quando i collegamenti sono consentiti, i file possono avere due o più percorsi. I programmi che trovano o leggono tutti i file in una determinata directory e le sue sottodirectory individuerebbero un file collegato più volte. **Come devono essere gestiti i collegamenti?**

a) Alcuni programmi non riconoscono i collegamenti possono finire in un loop;
b) Alcuni programmi invece forniscono un supporto limitato: zip dei link;
c) Alcuni programmi forniscono un supporto completo: tar;

Link	Posso rimuovere file originale?	Posso spostare file originale?	Influenza link counter?	Istanzia nuovo i-node quando lo crei	Posso spostare il file link?
Hard	No	Sì	Sì	No	Sì
Soft	No	No	No	Sì	Sì

MONTAGGIO DI UN FILE SYSTEM

Prima di poter accedere ai file sul file system, è necessario montare il file system. Il montaggio aggancia il file system ad un punto di montaggio e lo rende disponibile per il sistema.



SMONTAGGIO DI UN FILE SYSTEM

Quando un file system viene smontato esso viene staccato dal punto di montaggio e per riusarlo occorre prima rimontarlo.

Punto di montaggio: In genere un punto di montaggio è una cartella vuota dell'host o punto di accesso indipendente. Alcuni S.O. consentono il montaggio su una cartella non vuota e per farlo il S.O. fa queste tre fasi:

1)Oscuro i file esistenti della directory.

2)Monta il file system.

3)Rende visibili di nuovo i file precedentemente oscurati.

Unix: root file system "/" è sempre montato all'avvio (boot time) e posso attaccarci o staccarci altri file system.

Prima di fare una operazione di montaggio il S.O. controlla la presenza di un file system: legge il superblocco e controlla che il formato sia valido (se non valido controllo la consistenza). Infine, se il file system è valido, il S.O. annotta nella sua tabella di montaggio (mount table) in memoria che il file system sia montato e sia del tipo giusto.

ES: <u>montaggio in Unix</u>:

1) Il S.O. controlla la consistenza del nuovo file system.

2) Il S.O. crea strutture dati in memoria per tenere i metadati del nuovo file system.

3) Il S.O. aggiorna la mount table in memoria aggiungendo una entry che contiene un puntatore alla copia del superblocco del nuovo file system.

4) Il S.O. collega il nuovo file system al punto di montaggio settando 2 campi della copia in memoria del nodo: 1. Un flag che indica che la directory è un punto di montaggio, 2. Un campo che punta alla entry precedente della mount table.

ES: <u>attraversamento del punto di montaggio</u>:

1)Riconosco il punto di montaggio per il flag

2)Il S.O. legge le entry della tabella e analizza il superblocco

3)Il S.O. può iniziare ad attraversare le cartelle secondo il formato del file system

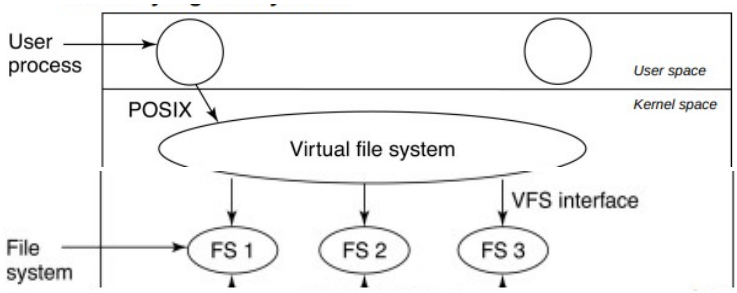
ES: <u>montaggio in windows</u>: permesso con l'introduzione del file system NTFS v.3.0 (New Technology File System)
Prima di NTFS v.3.0: schema a 2-liv: namespace + path name; file system montati in un namespace separato denotato da una lettera e colonna, es: X:\path\file.
Con NTFS v.3.0: SO può montare un file system in ogni punto (volume mount point) con la struttura esistente di NTFS, il FS è supportato da ogni FS di Windows e il mount point ha un riferimento al root dir.

VIRTUAL FILE-SYSTEM (VFS)

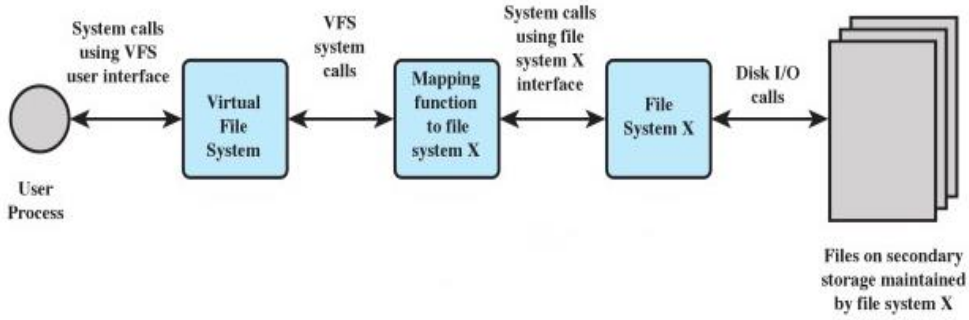
Il S.O. moderno deve integrare più tipi di file system in una struttura singola ed omogenea, i suoi file system possono essere remoti o in partizioni di dispositivi locali. Dal punto di vista dell'utente, sarebbe più comodo vedere tutti i file system montati come un'unica struttura di file system. Ed è da qui che nasce l'idea di un file system virtuale (VFS).

Un file system virtuale integra più file system in un'unica struttura omogenea. I file system sotto il VFS potrebbero essere: 1. File system archiviati su dischi installati sullo stesso computer su cui è in esecuzione il S.O. oppure, 2. File system remoti in questo caso si usa il protocollo NFS (network file system).

L'idea chiave dei file system è l'astrazione: isola aspetti comuni ad ogni file system in un separato layer (livello) del software, il layer (livello) è come un ponte: esibisce un'interfaccia comune astratta a livelli superiori e chiama le funzioni dei file system dei livelli inferiori. VFS ha un'interfaccia superiore al processo utente e una inferiore al file system concreto, ogni file system concreto ha le funzioni richieste dal VFS.



• The VFS concept



Astrazioni usate dall'interfaccia VFS

1-superblocchi: descrivono un file system montato

2-v-node: descrivono un file-control block associato ad un file specifico

3-file: descrive un file aperto associato al processo

4-dir entry: descrive una entry di una cartella

Ognuno ha un puntatore a una tabella di puntatori a funzione con tutte le possibili operazioni di quel tipo.

VFS ha delle strutture dati interne, tra cui:

1- mount table (tabella di montaggio): Tiene traccia dei file system montati. Ogni entry contiene un superblocco.

2- Tabella dei file aperti nel sistema: Tiene traccia di tutti i file aperti di tutti i processi nel sistema.

3- tab file aperti per processo: Tiene traccia di tutti i file aperti per un processo

Il VFS non salva su disco, infatti, crea e gestisce delle strutture in memoria, è il software associato con un file system concreto che fa il fetch e salva i metadati e dati del file system dal/sul disco.

Aggiungere supporto per un concreto file system in VFS:

- FS nuovo: scrivo funzioni con la stessa firma richiesta da interfaccia VFS.
- FS esistente: scrivo funzione "wrapper", la firma è la stessa delle funzioni dell'interfaccia VFS. L'implementazione consiste nel creare 1 o più chiamate al file system concreto.

Quando viene montato un file system concreto (incluso il file system di root) deve essere registrato con VFS. Il file system concreto fornisce puntatori alle funzioni che implementano l'interfaccia VFS. Dopo la registrazione, il file system concreto è pronto per essere usato.

ES: montaggio file system

1) Il file system concreto si registra con VFS fornendo puntatori alle funzioni che implementano l'interfaccia VFS.

2)VFS crea e aggiorna le strutture dati (creo superblocco, metto una entry con un suo puntatore nella mount table e aggiorno i v-node).

3)Ora si può usare il file system.

ES: process calls fd = open (“/usr/include/unistd.h”, O_RDONLY)

1. Analisi /usr ->FS già montato
2. Mount table: VFS trova il superblocco
3. Superblocco: VFS trova la cartella root del FS montato
4. Cartella root: segue il path include/unistd.h
5. Trova il file, crea nuovo v-node in mem che contiene un ptr
6. VFS chiama la funzione open e mette il risultato nel v-node
7. Creazione nuova entry nella v-nodes table in cui savare i v-node
8. VFS crea nuovo file che punta alla entry associata al v-node (crea salvataggio del return)
9. crea entry del processo corrente nel fd table in cui salvare il file creato
10. Return del f dal chiamante, così può usarlo per altre operazioni e vedere file/v-node/ptr associati.