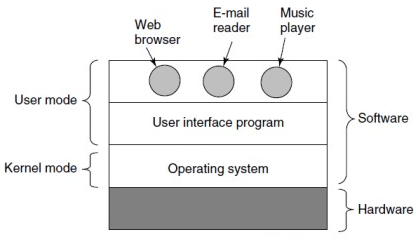
**SISTEMI OPERATIVI 2**

**INTRODUZIONE**

Dove si trova il S.O.?

Nella parte in modalità kernel del software.

L'esecuzione delle app utente non si affida solo sulla CPU ma anche su componenti HW.

**I/O devices**: componenti HW che forniscono input o direzionano l'output.

Il S.O. controlla i dispositivi e le operazioni I/O (ognuna a velocità diversa) ed è organizzato in livelli, ognuno con le sue componenti e funzioni.

* Lvl I/O: **sottosistema I/O**: usa un'interfaccia (uniforme ai livelli superiori)-> per accedere ai dispositivi e dei drivers->per gestirli.

|  |  |
| --- | --- |
| **Dispositivo a blocchi** | **Dispositivo a caratteri** |
| Lavora ad una granularità di blocchi di dati di dim fissa | Lavora su n byte |
| Unità d'indirizzamento minima: blocco | Non indirizzabile |
| Trasferisce blocchi interi | Operazioni tipiche: get e put |
| Accesso randomico/casuale (letto/scritto indipendentemente dagli altri) | Accesso seriale (dall'inizio, senza seek) una volta letto un byte non torno indietro |
| Es: Hard disk Drive, dvd drive | Es: mouse, tastiera, usb |

**Dispositivi network:** via di mezzo, lavorano su pacchetti di caratteri ma la granularità non è di dimensioni stabilite, supporta accesso casuale.

**I/O HARDWARE:**

**Device**: meccanico, comunica col pc tramite porta (cavo, usb...) Bus: set di cavi comuni a più devices (PCI Express)

**Controller**: elettronico, 1 o + chip controllano il device.

* Accetta comandi dal S.O. e li esegue su dispositivo
* Interfaccia semplice (nasconde le parti complesse)

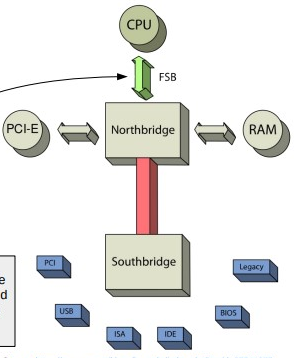
/\*\* ES: S.O. vuole leggere un settore del HD.

S.O. vede HD come sequenza di settori da 512B l'uno.

HD: ogni settore ha un indirizzo triplo (cilindro, testa e settore) e una struttura preambolo+dati+controllosomma

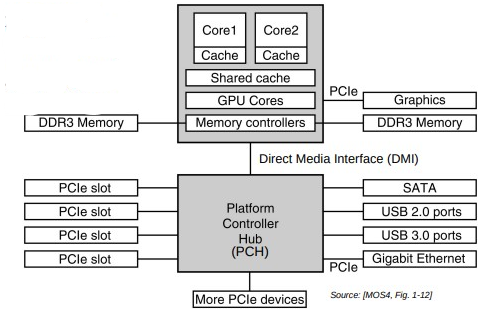
Settore 11206->conversione in indirizzo triplo->->->raggiunge posizione corretta->legge e salva in buffer ->error-checking ->ok? return dati \*\*/

* Cpu e controller comunicano tramite BUS (1 o +)

HW vecchi: 2 controller.

* **Northbridge:** Controller di memoria Hub, comunicazioni ad alta velocità per la memoria
* **Southerbridge**: Controller Hub, comunicazioni lente per I/O
* **Front Side Bus**: connessione tra Cpu e north.

La CPU con il tempo diventa più veloce e il Bus non le tiene testa limitandone le prestazioni.

Architetture moderne:

* **CPU**: comprende la maggior parte delle funzioni Northbridge
* **Platform Controller Hub**: performa funzioni del Southerbridge e le rimanenti del nord (PCIe)
* **DMI BUS**: trasferimenti tra le due parti.

Un controller può avere più forme:

* Singolo chip: chip nella schedamadre del PC che controlla segnali nei cavi di una porta a cui il device è connesso. Nelle ultime gen integrato nella CPU
* Un circuito che connette il PC tramite uno slot di estensione. Spesso chiamato adapter bus host. Utile per comunicazione protocollo-device complessa.
* Controllers build-in: controllers periferici del device, permettono all'host di eseguire comandi ad un livello più alto.

Come comunicano CPU E I/O??

**Modello di riferimento**: I devices sono tutti diversi tra loro per architetture HW, interfacce con cui interagire, velocità, funzionalità... Per semplicità noi usiamo modelli di riferimento.

* Modello di riferimento di sistema: CPU, memoria e disp I/O sono connessi da UN SINGOLO BUS e comunicano tramite esso.
* Modello di riferimento device: Interfaccia che definisce un protocollo per controllare le operazioni (data buffer) e struttura interna: responsabile dell'astrazione.

Registri di controllo: accessibili da CPU, per interazioni con dispositivi CPU. Registri dati, registri stato, registro dei comandi (scrive tramite cpu di controllo cosa fanno i dispositivi).

Data buffer: salvare dati da dispostivi che non accendono al S.O., usati per salvare dati che devono essere scritti sul dispositivo.

I REGISTRI DEVONO ESSERE ACCESSIBILI DA CPU:

* Approccio port-mapped I/O:

Numero di porta (integer) per ogni dispositivo

Port space: collezione di tutti i port number I/O (mod Kernel) !=mem number.

Per comunicare istruzioni speciali I/O: scelgo dispositivo->sposto bit da/a registro->input su registro o output su porta

* Approccio memory-mapped I/O:

Registri mappati nel memory space.

Per ogni reg: indirizzo mem senza memoria assegnata.

* Approccio ibrido: port per op semplici e memory per complesse/costose.

|  |  |
| --- | --- |
| **MEMORY-MAPPED** | **PORT-MAPPED** |
| + semplice sviluppo dei drivers (alto liv) | -port space !=mem space-> serve codice di assemblaggio ->poco portatile |
| + semplice escludere gli users (nascondo gli indirizzi con i registri dei dispositivi) | - richiede meccanismi appositi perché spazio port != mem |
| +progettazione CPU semplice/economica (istruz che riguardano mem funzionano per i registri) | +richiede nuove istruzioni nell'ISA perché gli spazi non coincidono |
| -devo disattivare il caching (se occupo un reg rischio di occupare il dispositivo) | +non consuma memoria fisica |
| - devo esaminare le referenze ai bus per capire a quale mi riferisco | +nessun problema con più bus |
| Comunicazione: read e write come in port-> MOV REG, ADDR (mem address->cpu register) ->MOV ADDR, REG. | Comunicazione: selezioni dispositivo -> sposto bit sul device register -> IN REG, PORT (port->reg), OUT PORT REG (reg->port). TUTTO IN KERNEL |

ES: leggere una parola all’indirizzo X: indirizzo X sul bus indirizzi->READ signal sul bus di controllo->

->a)Port-Mapped: segnale sul bus di control line per dire se serve spazio I/O o di memoria->risponde il device adatto

->b)Memory-Mapped: ogni modulo e dispositivo I/O compara la address line e gli indirizzi richiesti ->Se l’indirizzo è nel range risponde.

**COME FUNZIONA L'I/O?**

**I/O programmato (polling):** CPU manda richiesta I/O e continua a controllare (busy waiting) lo stato/flag per vedere se è libero. Semplice ma attesa attiva.

**Interrupt-driven I/O:** la cpu manda la richiesta e il dispositivo se ne occupa mentre la cpu fa altro, ad operazione finita il dispositivo manda un INT alla cpu e, se non ci sono altre operazioni con priorità maggiore, se ne occupa l'int handler.

Per gestire un int:

1)Controller: mette il numero del dispositivo sul bus indirizzi e va in Kernell.

2)Cpu: carica il PC (program counter) dal vettore,

3)Handler: inizia l'esecuzione

4)Cpu: avvisa che è pronta per il prossimo int con un'istruzione "return from int".

Oggi la cpu funziona a pipeline: 3 stati fetch-decode-execute.

CPU superscalare: ha unità di esecuzione specializzate e le istruzioni vengono divise in micro-operazioni, ognuna con una unità.

Interrupt preciso: lascia la macchia in uno stato ben definito: 1)Salvataggio su luogo conosciuto 2)Istruzione prima di quella del PC completate 3)Nessuna istruzione completata dopo il PC 4) conosco lo stato del pc

Interrupt impreciso: viola 1 o + proprietà-> lento e spreca

**I/O usante DMA:** il DMA controller svolge parte del lavoro della cpu, così nel frattempo la cpu può fare altro. Trasporta dati in modalità PIO e manda un int quando ha finito.

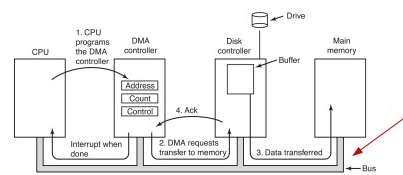
+)Non spreca dati -)Complesso, richiede un dispositivo in più e nuove istruzioni

Il DMA controller contiene diversi registri che possono essere scritti e letti dalla CPU:

* Memory address register: dove scrivere o leggere dati
* Byte counter register: contatore per numero di byte trasferibili o ricevuti
* Control register: I/O register con una direzione del trasferimento, trasferisce gruppi di byte o parole

**MODALITA' FLY-BY:** lenta ma efficace

LETTURA DA DISCO CON DMA

1. La CPU programma i registri DMA (cosa trasferire e dove) e manda un comando al disk controller
2. il DMA trasferisce dati attraverso il BUS: il buffer contiene dati validi->piazzamento indirizzo sul BUS->richiesta di lettura sul BUS per il disk controller.
3. Le informazioni vengono prese dal buffer e caricate in memoria: fetch di una parola e scrittura in memoria
4. Controller comunica al DMA di aver concluso il trasferimento mandando un segnale di acknowledgment alla DMA attraverso il BUS
5. byte counter = byte counter-1
6. ****if(byte counter>0): leggo altri; else: INT alla Cpu->quando la CPU serve l'interrupt i dati sono già disponibili

**MODALITA' FLY-THROUGH**: alternativa al fly-by più flessibile ma meno efficiente

Salvataggio dati nel buffer->scrittura su memoria o dispositivo.

DMA controller e la CPU contendono l'accesso in memoria in un modo che dipende da come lavora la DMA.

* **Cycle-stealing**: per ogni parola trasferita, il controller DMA deve acquisire il BUS e rilasciarlo a fine trasferimento. DMA priorità>Bus (dispositivi I/O tollerano meno ritardi). +)La CPU lavora tanto e non ozia. **1 TRASFERIMENTO PER VOLTA**

-)Inefficiente (trasferimento lento) e la DMA può rubare cicli alla CPU

* **Burst mode**: il DMA controller dice al dispositivo di acquisire il bus-> **+ TRASFERIMENTI**   
  -> rilascio BUS. +)Efficiente (1 acquisizione x + trasferimenti)

-)Può bloccare la CPU e altri dispositivi in caso di burst lungo o I/O lento.

I DMA reali riescono a gestire diversi trasferimenti di dati allo stesso tempo:

* ogni trasferimento richiede un device controller diverso.
* Un set di registri, uno per canale
* La CPU inizia a trasferire scrivendo nei set legati al canale
* I trasferimenti vengono gestiti secondo l'ordine deciso dalla DMA
* Possono esserci più richieste pendenti contemporaneamente

**SOTTOSISTEMA KERNEL I/O**: parte del SO che si separa dal resto perché gestire i dispositivi I/O è più complesso. Interfaccia uniforme con i livelli superiori, comunica con ogni dispositivo.

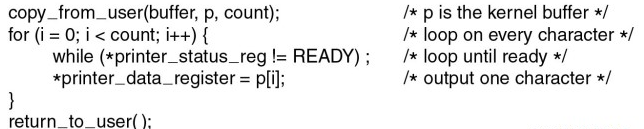
**SOFTWARE I/O NELLO SPAZIO-UTENTE**: sotfware nello spazio utente con un'interfaccia ad alto livello per il sottosistema Kernell I/O

Come vengono gestite le richieste I/O?

ES: stampa stringa: copia su buffer kernell->registro di stampa e queries->loop di stampa char

* I/O PROGRAMMATO

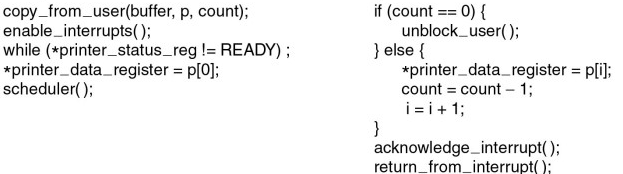
Lettura/scrittura da devices->Polling (busy waiting) per vedere quando può ricevere altri dati ->ripeti ciclo.



ES: velcoità = 100 char/sec =1 char per 10msec->CPU occupata per 10msec->tanto tempo occupato

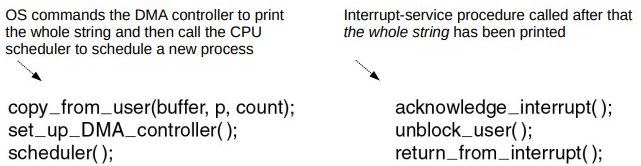
* I/O GUIDATO DA INTERRUPT

Ricezione richiesta->processo bloccato per x operazioni-> SO fa altro->INTERRUPT completamento o fallimento -> ripeto per completare richiesta ->fine: processi sbloccati



+)Niente attesa attiva -)Gli interrupt richiedono tempo->spreco

* DMA: molto simile al PIO ma la DMA si prende i carico della CPU-> CPU libera.



+)- sprechi CPU +)- int -)richiede la DMA (complessp)

**Obiettivi della progettazione di softwares I/O:**

* Indipendenza dai dispositivi: programmi generali che funzionano con tutti i dispositivi
* Definizione uniforme dei nomi: il nome di un file non deve dipendere dal dispositivo, dev'essere univoco
* Gestione trasparente degli errori: possibilmente vicino al livello HW, segnalati al SO solo se gravi o frequenti
* Supporto di diverse modalità di trasferimento: deve esserci la scelta tra asincrona (con int mentre la CPU fa altro) o sincrona (bloccante ma semplice)
* Gestione buffering: spesso i dati dei device vengono bufferizzati prima di esser salvati a destinazione. Devo evitare che il buffer sia pieno o vuoto gestendo le letture e scritture
* Gestione diversa per dispositivi I/O condivisi

Dispositivi dedicati: 1 processo per volta (es: stampante)

Dispositivi condivisi: anche + processi (es: schermo)

Software I/O in 4 livelli: I/O sw livello utente > sw SO indipendente da dispositivo > driver di device > int Handler> HW

Int Handler: livello più basso, nasconde complessità ai livelli superiori.

Gestisce gli INT in parte nel SW e in parte in HW

Gestione nel software:

1. Salvataggio contesto del processo corrente (registri, tabella pagine...)
2. Prepara contesto e stack x gestire l'int
3. manda un acknowledge al controller int avvisandolo della presa in carico
4. gestione int
5. Scelta del prossimo processo da runnare
6. Preparazione prossimo contesto ed esecuzione

**DRIVERS DISPOSITIVO**: nasconde i dettagli per controllare i dispositivi a livelli superiori.

Accetta richieste di alto livello e le trasforma in comandi di basso livello.

Ognuno gestisce uno o un gruppo di dispositivi.

Eseguiti in modalità Kernel perché necessitano accesso ai registri di controllo.

Modalità installazione:

* STATICA: ricompilo l'intero SO ad ogni nuovo driver->costoso. SO=file binario. Ok e cambia raramente
* DINAMICA: caricato a run-time, SO richiede un reboot per rendere i cambi effettivi

Installazione MOD USER: +)Stabile e affidabile +)Semplice manutenzione e sviluppo

-)Come consegno gli int? (o uso il PIO o creo un piccolo spazio Kernel)

-)Passaggio user->kernel lento

STRUTTURA GENERALE DI UN DRIVER DI DISPOSITIVO:

Come fa un driver le operazioni I/O?

1. Accetta le richieste di astrazione I/O dai livelli superiori
2. Controllo parametri->se validi traduco in livello più basso
3. Controlla se il device è occupato->Accoda la richiesta o la esegue direttamente
4. Invio una serie di comandi low-level 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. Comando rilasciato: if(PIO): aspetta fine dell'operazione; if (int driven): aspetto interrupt
6. Controllo errori: se non ci sono->output
7. Return informazioni di stato (es: true/false, 0/-1)
8. Scelta della prossima richiesta

POSSIBILI PROBLEMI:

* 2 interrupt in contemporanea
* Interruzione dell'interrupt nel mezzo dell'esecuzione e richiamato (re-entered) prima del corretto completamento della sua precedente invocazione
* sistemi hot-pluggabili: rimuovo senza rimozione sicura->non devo perdere dati

**SOFTWARE DEVICE-INDIPENDENTI**: contengono tutte le funzioni comuni a tutti i dispositivi e provvedono a una interfaccia uniforme ai livelli maggiori e minori

Funzioni tipiche:

* interfacciamento uniforme per i driver dei dispositivi: fornisce un'interfaccia uniforme tra drivers e dispositivi

+)Semplifica connessione e sviluppo di nuovi dispositivi

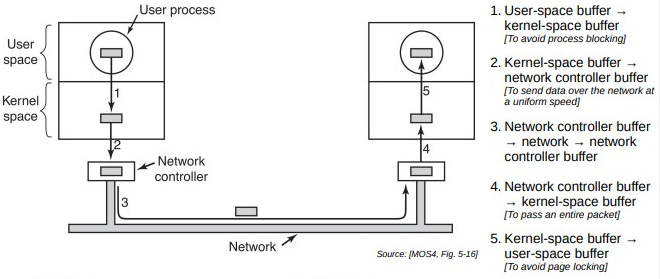
Viene realizzata usando API (gestisce dispositivi a blocchi o a caratteri) e Uniform naming scheme (mappa il nome dei dispositivi)

* Buffering: gestione di scrittura/lettura dei dispositivi a blocchi/caratteri.

//ES: leggere dati da modem ADSL:

* No buffering: 1 carattere x volta e int->troppi int
* Buffering: più caratteri (fino a riempire buffer)->int->Problemi con Paging
* Buffering in kernel: 1 blocco di caratteri->int-> buffer pieno = copia in buffer utente (op- atomica) ->e se: kernel buff pieno + pagina in caricamento?
* Double buffering o Circular buffering (2 o + buffer in kernel)

Troppo buffer = minori prestazioni.



* Report di errori: bisogna gestire al meglio gli errori, anche quelli specifici del dispositivo

Errori: di programmazione (del chiamante->report), di I/O (di solito se ne occupa il driver), di interazione, batch, CRITICI (termina il programma)

* Allocazione e rilascio dei dispositivi dedicati:

Dispositivi dedicati= utilizzabili da 1 processo per volta (es: stampante).

Devo esaminare e approvare/rifiutare le richieste

1)Provo ad aprirli e vedo se riesco->rischio deadlock (mai rilasciato)

2)Uso meccanismi apposta per rilasciarlo o occuparlo->accodamento richieste e NO deadlock

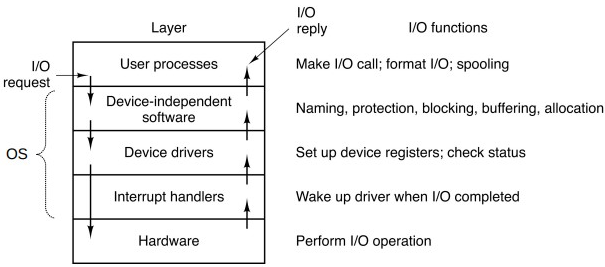
* Provvedere a una dimensione di blocco indipendente dal device: dispositivi blocchi possono avere dimensioni diverse, ma devo passare ai livelli superiori una dimensione unica

**Spazio utente I/O**: la maggior parte dei softwares I/O funzionano in kernel, ma una piccola parte sono sistemi di librerie linkate insieme con programmi utente o programmi interi in user-space.

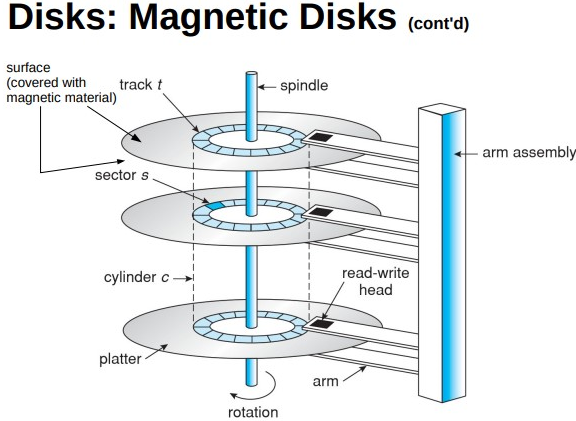
* Librerie di sistema collegate con programmi user: le funzioni di libreria invocano le system call e a volte settano variabili di stato per segnalare errori.
* Programmi interi in user-space: solitamente sono daemons/servizi che runnano continuamente in background, spesso sono associati con il sistema 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.



**Dischi**

* Magnetici: lettura e scrittura alla stessa velocità, usati come memorie secondarie
* Solid-state disk (SSD): vel lettura>vel scrittura, usati come memoria secondaria, costosi
* Dischi ottici: lettura + volte ma 1 scrittura. Per distribuzione di programmi, film... (es: DVD, blue-ray...)

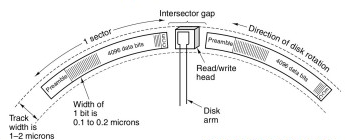
**DISCHI MAGNETICI**

Ogni superficie di un piatto è logicamente divisa in tracce circolari, divise in settori.

1 settore = 512B

I settori sono separati 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->lavorare su una sequenza di settori della stessa traccia è molto più veloce di lavorare su settori di tracce diverse.

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

* **Host controller**: controller alla fine del bus
* **Disk controller**: microcontroller presente in ogni disk drive

Old disks: lavoro host controlller >> disk controller (consegnano uno stream di bit)

Modern disk: lavoro disk controller >> host controller (si occupa di altri comandi di alto livello

Svolgimento operazioni di disco I/O:

* SO piazza un comando nel registro dei comandi dell'host controller
* host controller manda un comando al disk controller
* disk controller opera il disk-drive HW per portare via i comandi
* Disk controller ha una cache bult-in: trasferimenti al disk drive succedono tra cache e superfice disco, trasferimenti all'host tra cache e host controller

I controller moderni permettono i **Seek sovrapposti**: mentre il controller aspetta il completamento del seek su un drive, inizia il seek di un altro drive. Alcuni controller supportano il trasferimento sovrapposto di dati tra un disco e il suo controller (su un altro disco).

Il trasferimento dati tra CONTROLLER e MEMORIA PRINCIPALE non può essere overlappato.

**GEOMETRIA DEL DISCO:**

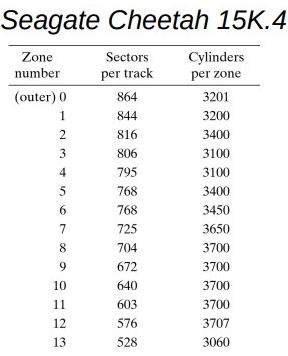
Vecchi dischi:il numero di settori per traccia era lo stesso per tutti i cilindri (stessa geometria)

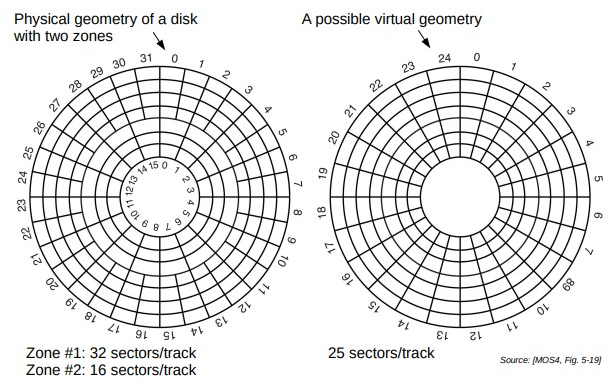
Dischi moderni: geometria fisica ≠ geometria virtuale.

**Zone bit recording** (ZBR): chiamato anche multiple zone recording.

La lunghezza della traccia fisica (circonferenza) incrementa allontanandoci dal centro.

I cd sono divisi in *recording zone* :stessa zona = stesso num settori.

Geometria virtuale: SO richiede un settore usando un indirizzo virtuale e il controller lo traduce in un indirizzo reale/fisico CHS (cilindro, testa e settore).

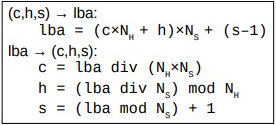
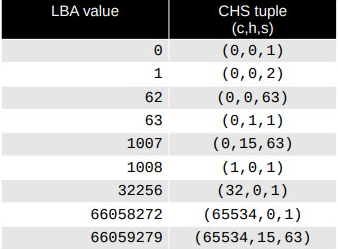


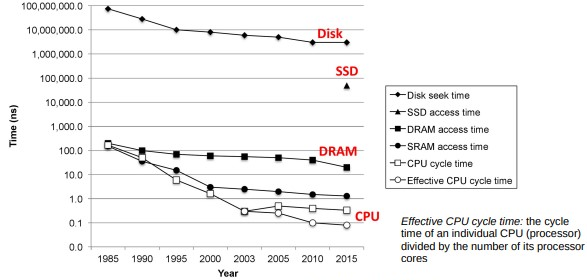
MODALITA' DI INDIRIZZAMENTO REALE:

* **Cylinder-Head-Sector (CHS)**: il SO pensa che ci siano NC cylinders, NH heads/cylinder, and NS sectors/track. I settori vengono numerati da 1 in poi, gli altri 0. Il controller traduce indirizzo virtuale in fisico.
* **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-... |
|  |  |

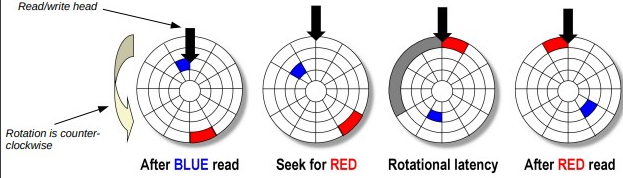
Conversione:

PERFORMANCE DEL DISCO:

Per richiesta I/O:

1. Attiva il motore del braccio (assembly arm) e posiziona su head. (seek Ts)
2. Aspetta il settore giusto (Rot. lat. Tr)
3. Lettura/scrittura appena il settore si muove sotto la testa e trasferisce dati da/a buffer. (transf. Tt)



**SEEK TIME**: secondi che ci mette il braccio del disco a posizionarsi sul cilindro desiderato.

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

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

**LATENZA/RITARDO DI ROTAZIONE**: secondi per rotare alla head del disco. Velocità di rotazione.

Massima: tempo x rotazione completa. Media: mezza rotazione.

**THROUGHPUT**: Data transfer time. Flusso tra superficie disco e controller quando il settore target è sotto la head. Tempo di trasferimento.int

Medio:trasferire un settore tra superficie e controller. Avg Tt=max Rt/Avg sector x track

TEMPO POSIZIONAMENTO: seek time + rotational latency

TEMPO DI ACCESSO PER 1 SETTORE: posizionamento +tempo di trasferimento. Ta=Ts+Tr+Tt

Avg Ta=Avg Ts + Avg Tr + Avg Tt

ES: rot speed R:7200RPM; Avg Ts:9msec; Avg sect/track N:400

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

ES: performance tipiche.

Ts = sistemi highend 3msec, desktops 9msec, laptops 12/15msec

Tr = speed: 3600-15000RPM; palmari: speed 3600 e latenza 8.3msec; highend=2msec

**AFFIDABILITA' DEL DISCO**

Fallimento completo: il disco non è più capace di read/write per crash pesanti/potenza/smagnetizzazione --->Rimpiazzo il disco.

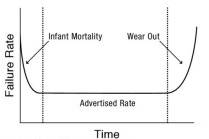
Failure rate: numero di fallimenti che mi aspetto in un'unità di tempo. Annuale (AFR) o orario (HFR).

Mean time to failure: tempo medio prima che fallisca. MTTFa=1/AFR, MTTFh=1/HFR.

Se posso riparare un disco: Mean Time To Repair e Mean Time Between Failures.

Esercizio: dato un disco con AFR λA = 0.005 failure/year calcola MTTFh

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 alta (a causa di difetti)->col tempo migliora.

Advertised rate: maggior parte della vita = pochi fallimenti randomici.

Wear out: dopo molto tempo aumento fallimenti->morte.

**Failure distribution**: probabilità del prossimo fallimento (ratio costante di fallimento. Può avere una distribuzione esponenziale negativa con λ>0.

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

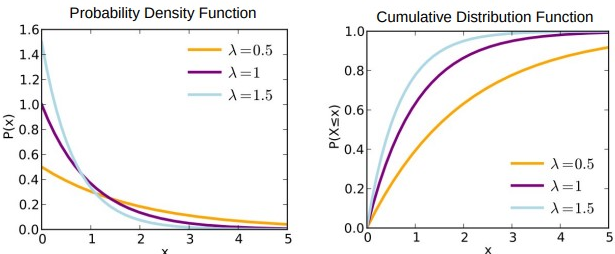
F(t)=Pr{X ≤ t}= funzione di distribuzione cumulativa, prob che il t per il prox fallimento sia <=t

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 distrib. esponenz. 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.

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

AFR di 100 dischi: 100\*0.00585=0.585 fallimenti annuali.

MTTFh=(24\*365)/0.585=1.5\*10^4=2 anni.

Tasso fallimento λ100disks > λdisk, MTTF100disks < MTTFdisk.

**DISCHI RAID**

Redundant Arrays of Independent Disk = una collezione di tecniche di organizzazione dischi per aumentare le prestazioni e 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.

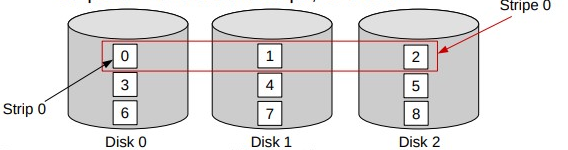
Esternamente per il SO un RAID appare come un singolo disco logico: si presenta al SO come array lineare di settori e i controller traducono ogni richiesta logica in richiesta fisica.

Perfezionamento tramite **parallelismo**:

* uso tanti dischi e gli accedo in parallelo
* data striping: 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 torvano 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**.

* **bit-level striping:** ogni bit in un disco diverso
* **Byte-level striping:** ogni byte in un disco diverso
* **Block-level striping:** ogni blocco di byte contigui in un indirizzo 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

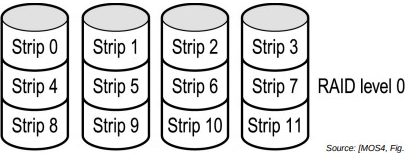
-)Richieste riguardanti strips sullo stesso disco devono essere svolte in sequenza

-)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

Perfezionamento tramite **ridondanza:**

**Mirroring**: ogni disco è complicato->semplice ma costoso. Rischio se perdo anche il clone (Mean time to repair). Duplica richieste di scrittura/lettura.

RAID+Mirroring x capacità desiderata Cdisk x n.

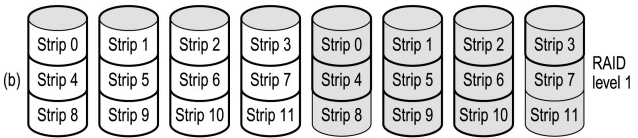
* **RAID lvl 0:** strping a blocchi, richiede disk, tolleranza errori=0

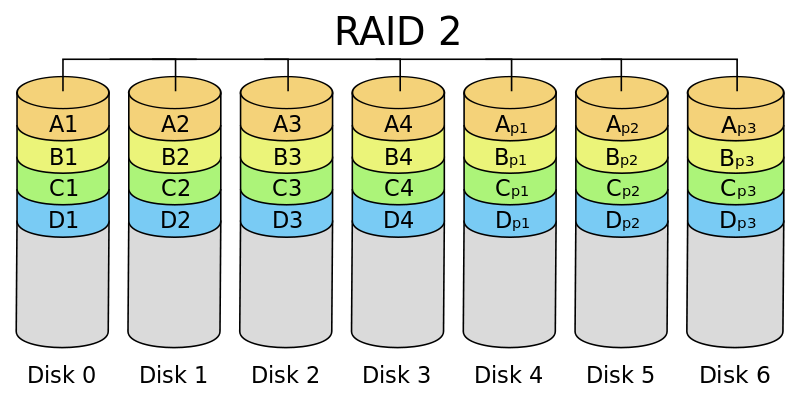
+)Performance e capacità -)0 Ridondanza e pessimo per piccole richieste

* **RAID lvl 1:** Mirroring (dati striped e con copia), capacità Cdisk×n: n+n, tolleranza errrori fino a n.

+)Tolleranza, recupero semplice, 2xVelocità di gestione richieste.

-)Costoso, poco miglioramento, tanto costo extra per salvataggio.



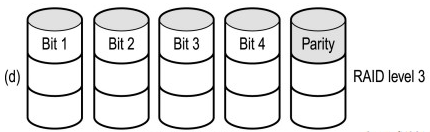
* **RAID lvl 2**: divide i dati a livello di bit (non blocco) e usa un codice ECC (error correction code) per correggere gli errori sui singoli bit e rilevare errori doppi.

Capacità Cdisk x n: n+m <2xn (m = disco di parità)

Tolleranza: dipende dal ECC (>=1).

Le richieste di lettura riguardano solo il disco dati, quelle di scrittura anche quello di parità.

-)1 richiesta I/O per volta, costoso e complesso

* **RAID lvl 3:** bit-interleaved parity organization

striping + bit di parità (versione semplificata dei liv. 2)

Capacità: Cdisk \* n dischi: n +1 disco di parità.

Bit di parità: bit P aggiunto alla fine di una stringa binaria, 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, se != Errore.

Parità pari: 2k, else:0. Parità dispari: 2k+1, else: 1

-)RILEVA SOLO QUANTITA' DISPARI DI ERRORI

-)NON SA DOVE SIANO GLI ERRORI

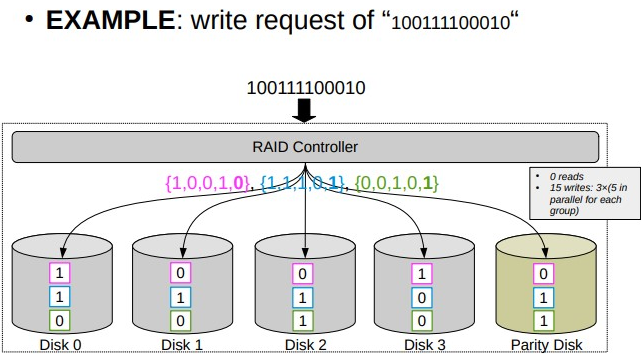
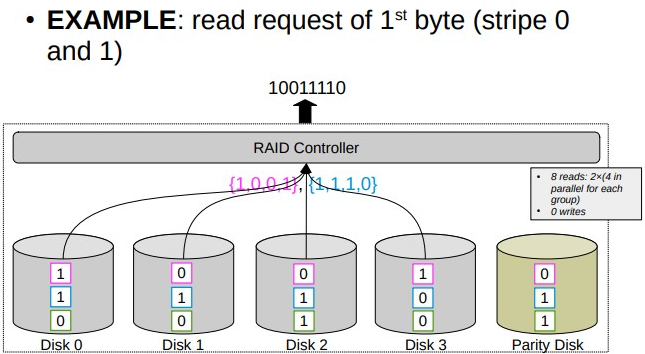
+)CRASH DI UN DISCO: so quale crasha, conosco il bit errato e posso risolvere il problema.

Tolleranza fallimenti disco: 1, se crashano 2 dischi o 1 disco e quello di parità, non posso più ricalcolare l'errore.

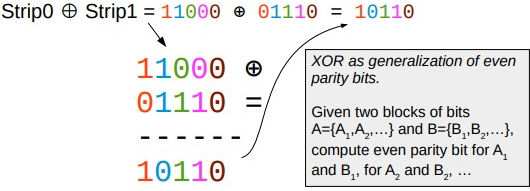
La scrittura tocca il bit di parità, la lettura no.

+)Affidabile, performance del raid 2 con meno costo

-)Complesso e solo 1 I/O per volta



* **RAID lvl 4:** block-intreleaved parity organization = striping a blocchi + blocco di parità

Capacità Cdisk \* n: n+1 (disco parità)

Tolleranza fallimenti: 1.

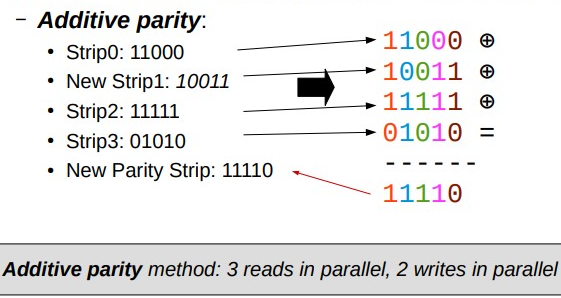
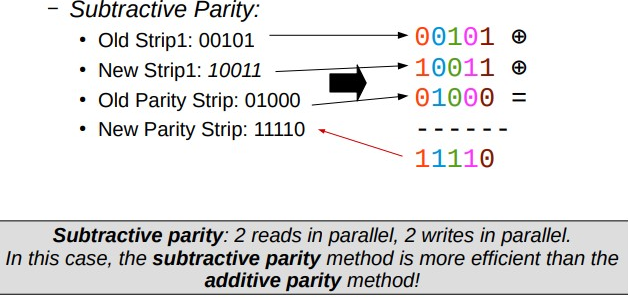
Disco di parità: contiene blocchi di parità = XOR tra due strips della stessa stripe.

Lo XOR in più operazioni restituisce 1 se lo faccio tra una quantità di 1 dispari, 0 altrimenti.

Blocco parità 1: P1=B1,1 xor B1,2 xor B1,3... se la strip è lunga k, anche la strip di parità è lunga k.

Le richieste di scrittura usano anche i dischi di parità, devo ricalcolarli. Due possibili approcci:

* + - Parità additiva: legge le altre strips della stripe dal data disk, calcola il blocco di parità e salva i nuovi dati e blocchi su disco. (somma tra tutte le strips)
    - Parità sottrattiva: legge vecchi dati e parità, computa il blocco di parità e scrive i nuovi dati e blocchi. (somma tra vecchia e nuova strip + parità)

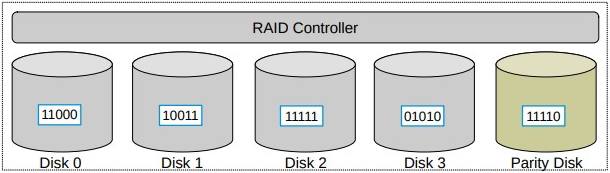


In caso di crash in lettura, posso calcolare la strip mancante facendo lo xor tra le altre strips e quella di parità (metodo additivo).

+)Affidabile e buone performance di lettura e lunga scrittura

-)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) dello stesso stripe da cambiare, quando usi la parità additiva e quando la sottrattiva?



1)Considera 2 scritture 11000->00111 su strip 0 e 10011->01100 su strip 1

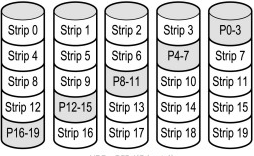
Metodo sottrattivo: 3 letture e 3 scritture (dischi 0 e 1 + parità)

Metodo additivo: N-3(=2) letture e 3 scritture.

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.

* **RAID lvl 5**: block-interleaved distribuited parity organization.

Striping a blocchi + blocchi di parità distribuita.

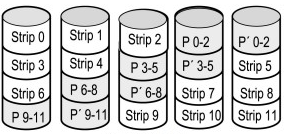
Capacità Cdisk \* n: n+1 (blocchi di dati e parità intervallati.

Tolleranza falimenti: 1.

Non c'è un singolo disco di parità->no collo di bottiglia -> posso fare + small write su diverse stripes e dischi in parallelo. Accedo a più blocchi di parità in parallelo.

-)Small write ancora esistente e causa anche costi.

+)vantaggi del raid 4 + request rate di read e write.



* **RAID lvl 6**: schema ridondanza P+Q

Striping + 2 blocchi di parità distribuiti.

Capacità: Cdisk \* n: n+2 (2 dischi parità)

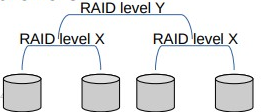
Tolleranza: 2.

+)Affidabile

-)Scrittura peggiore del RAID 5 (scrittura = modifica 2 bit parità)

Spesso si usano raid 5 e 6 insieme.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

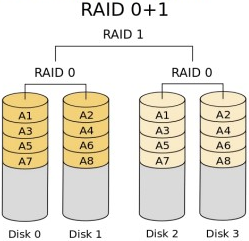


* **RAID annidati/ibridi**

Combinano uno o + livelli standard RAID in una gerarchia.

Numerazione composta (basso->alto).

Schema comune: X+Y.

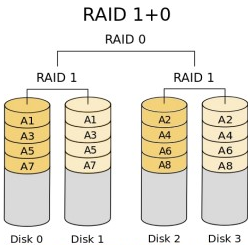
* **RAID 0+1**: mirror of stripes (copia di stripes).

n dischi divisi in stripe, ognuna copiata in altri n dischi.

Dischi: n+n.

Tolleranza: 1->dopo 1 fallimento diventa un RAID 0.

+)Buone performance in lettura e migliore del raid 0

-)Costoso.

* **RAID 1+0**: stripe of mirroring

Dischi: n+n

Tolleranza: >=1 && <=n

Quando uno crasha posso usare l'altro disco

del set copiato.

IMPLEMENTAZIONE RAID:

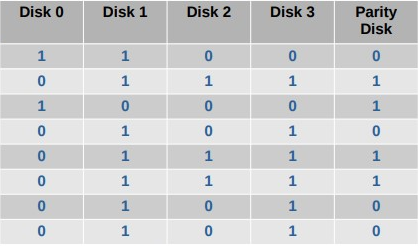
Controller: a livello HW è un circuito separato connesso (host bus adapter) che si collega nel pc tramite un cavo o un disco. Buone performance ma costoso.

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.

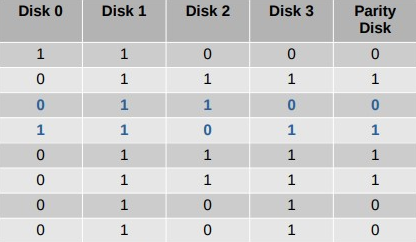
Hot-Spare disks: rimpiazzano i dischi guasti automaticamente SOLO in caso di fallimento. Ogni disco provvede ad un rimpiazzo senza dover intervenire.

ES: 5 dischi RAID 3.

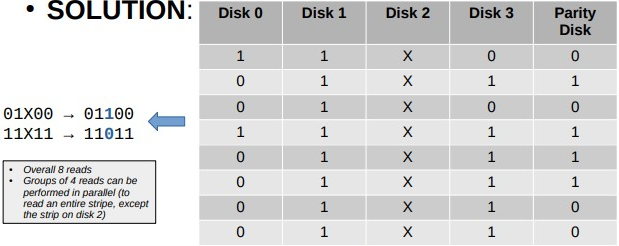
1)come scrivo 11000111, 10000101, 01110111, 01010101 ? 40 scritture e 5 scritture per volta



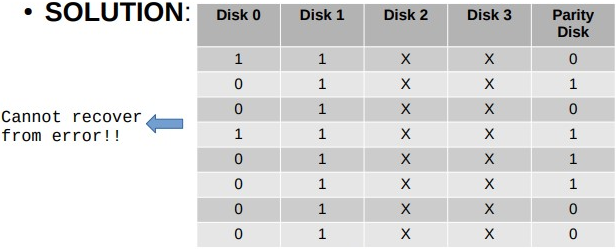
2)Come cambio il secondo byte in 01101101?



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



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



**FORMATTAZIONE A BASSO LIVELLO DEI DISCHI**

Prima che un disco venga usato per salvare dati deve essere formattato a basso livello.

Definisce la geometria fisica del disco (serie concentrica di tracce ognuna con numero di settore e piccolo gap tra i settori).

Fatto dal produttore/SW

Ogni settore ha il formato:

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

Scrittura di un settore: il controller aggiorna l'ECC con valori calcolati da tutti i byte nel campo.

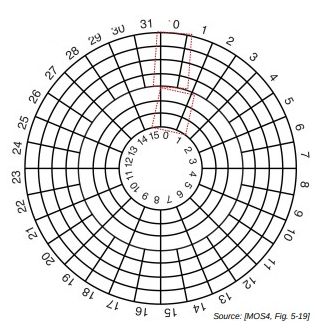
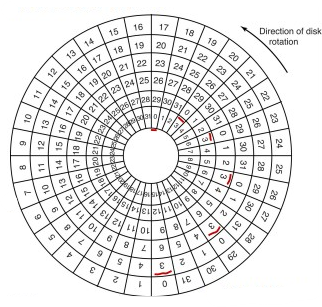
Lettura di un settore: il controller ricalcola ECC e compare con il valore salvato, se != il campo dati è stato corrotto, se l'errore è piccolo il controller lo identifica e corregge.

Come numero i settori in ogni traccia?

Uso il settore radiale (colonne)

-)Per spostarmi da un settore all'altro (righe) devo

muovere molto il braccio-> tempo sprecato.

**Cylinder skew**: la posizione del settore 0 in ogni traccia è offset della precedente.

DOPO:

Offset = seek time tra tracce / tempo passaggio a nuovo settore.

ES: Disk drive con Rt speed: 10.000RPM, 300 settori a traccia, T-to-T seek time: 800 μsec.

Calcola la Cylinder skew.

1 rotazione: 1/10.000=6 msec.

Tempo per ogni settore: 6msec/300=20 μsec.

Settori che passo nel seek time: 800 μsec/20 μsec=40

ES: Rt speed: 5400RPM, 300 settori a traccia, T-to-T seek: 800 μsec.

1 rotaz 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

Velocità minore = meno settori.

Effetti della formattazione:

La formattazione riduce la capacità del disco di circa 20% , spesso dicono la capacità prima della formattazione per far sembrare i dischi più grandi o cambiano le unità di misura.

La formattazione fissa un limite max di velocità di trasferimento dati .

Con Ns numero di settori per traccia, S size del settore e R velocità rotazione:

Max data rate Dmax = track size/max latenza rotazione = (Ns \* S)/(60sec/R)

ES: R=10.000RPM, Ns=300 settori a traccia di dim S=512B.

Dim traccia=512B\*300=1536000; Max rotaz=60sec/10.000=6msec; Dmax=153600/6msec=25.600.000 B/sec.

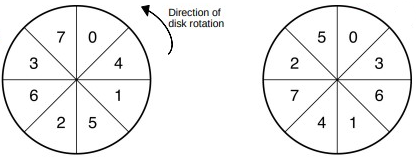
Lettura 2 settori consecutivi: leggo il primo->calcolo ECC->trasferimento dati->rotazione intera.

Come velocizzo?

**Sector interleaving**: facilita letture di settori consecutivi, ma rallenta gli altri.

1)Interleaving singolo: i e i+1 sono a 1 di distanza.

2)Interleaving doppio: i e i+1 sono a 2 di distanza.



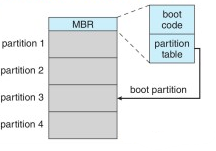
Altra soluzione: buffer largo che memorizza tracce intere.

ES: calcola sector interleaving con R=10.000RPM, 1900 settori a traccia, tempo trasferimento in mem = 10 μsec.

Max latenza=60sec/10.000RPM=6msec.

Tempo passaggio settore=6msec/1900settori=3.2 μsec

Sector interleaving: 10 μsec/3.2=3.125 settori.

Prima che un disco possa contenere informazioni dev'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*  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 la vera OS loader nel boot sector.

La Partition table supporta 4 principali partizioni tra cui 1 è la partizione estesa che contiene + partizioni logiche, ognuna descritta da un Extended boot record (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: set up dei file-system iniziali in partizioni (salva la mappa degli spazi liberi ed allocati), istalla u boot sector, scan dei difetti, identifica il fyle system usato mettendo un numero nella partition table entry.

**DISK ARM SCHEDULING**

Il tempo per leggere/scrivere un disco dipende da seek, latenza di rotazione e tempo di trasferimento. Le performance dipendono 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?

* **First come first served** (FCFS): ordine di arrivo, lento e ingiusto.

ES: 11, 1, 36, 16, 34, 9, 12. Movimenti braccio: |11-11| + |11-1| + |1-36| + |36-16| + |16-34| + |34-9| + |9-12| = 111 cylinders

* **Shortest seek first (SSF)**: quella con meno seek dalla posizione corrente. Pochi movimenti.

ES: 11, 1, 36, 16, 34, 9, 12. Movimenti braccio: |11-11| +|11-12|+|12-9|+|9-16|+|16-1|+|1-34|+|34-36|= 61 cylinders. Veloce ma rischi starvation.

* **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.

ES: 11, 1, 36, 16, 34, 9, 12. Movimenti braccio:|11-11|+|11-12|+|12-16|+|16-34|+|34-36| +|36-9| + |9-1| = 60. La lunghezza buona e non starvation (aspetta max 2xNumero cilindri). Sfavorite le richieste agli estremi.

* **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 più equi.

ES: |11-11| + |11-12| + |12-16| + |16-34| + |34-36| + |36-1| + |1-9] = 68

ES: richieste 10, 22, 20, 2, 40, 6, 38 con 50 cilindri, pos iniziale 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=292

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**: la superfice magnetica di ogni piatto è divisa in piccole regioni magnetiche, ognuna rappresenta 1 bit, alta densità richiede regioni molto uniformi.

ES: densità lineare 5.25", traccia nel mezzo con circonferenza di 300mm, 512B a settore.

**Densità lineare=track size/ track length** = 300\*512\*8bit/300=4096 bit/mm. circa 5k con preambolo e ECC.

**Bad sectors**: settori che non leggono correttamente i valori appena scritti.

Difetto piccolo->controllo e correggo con ECC.

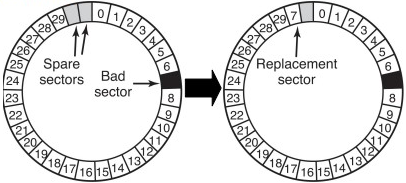
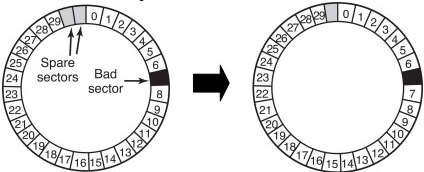
Difetto grande->devo gestirlo in un altro modo.

Gestione errori grandi:

* Approccio a lvl controller: tutti gli HD hanno dei spare sectors che possono essere usati dal disk controller per rimpiazzare i bad sectors. Trasparente al SO

Sostituzione con sector forwarding: il controller remappa il singolo settore che non sarà più sequenziale. Aumenta i futuri tempi di accesso

Sostituzione con sector slipping: tutti i settori scivolano di uno.



Per segnare la nuova posizione o uso una tabella interna per ogni traccia o riscrivo il preambolo per remappare i numeri di settore.

* Approccio a lv SW: 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 bad sectors remappo la tabella.

Crea anche un file di sistema segreto in caso nei settori corrotti ci siano files 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 recalibra il comando di muovere il braccio all'inizio e lo resetta. I controller moderni lo fanno automaticamente.

Se non basta recalibrare, si usa un pin di reset che forza il controller a bloccarsi e resettare, se non basta stampa un messaggio di errore.

**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).

**Tastiera**: contiene un microprocessore,

* comunica con una porta seriale PS/2 o una USB con controller nella scheda madre.
* Interrupt: a pressione e rilascio tasto. 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

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 buffers 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"

**Mouse:** più semplici delle tastiere, 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

ES: mouse con movimento max rate=2cm/sec, ogni messaggio è di 0.1mm e 3 bytes. Calcola il max data rate (b/sec) se ogni mickey venisse considerato separato.

max rate=2mc/sec=20mm/sec. messaggi in 1 sec: 20mm/sec\*0.1mm=200messaggi/sec.

max data rate 200msg/sec\*3B/msg=600B/msg

**Consumo energetico**: per ridurre impatto ambientale, costi e aumentare batteria dei dispositivi.

Consumo di un PC moderno: 200W \*10^8 PC nel mondo.->20 impianti nucleari.

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

COME RIDURRE I CONSUMI?

* Livello hw: ridurre sprechi e spegnere. Progressi lenti.
* Livello SW: spegnere parti non usate (stanby) e riducendo prestazioni.

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

* On (in uso) o Off (senza energia)
* 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)

Il SO decide quando a chi dare l'energia, quando e in quale stato mettere i dispositivi.

ES: Spegnere un dispositivo: Quale spengo? Quando? Spegnere per poco tempo può causare un transizione che costa più del risparmio.

Gli algoritmi devono prendere decisioni: quantitative (quanta energia salvo vs uso) e soggettive (quanto ritardo per accensione è tollerabile).

SOLUZIONI PER RISPARMIO ENERGETICO:

* Schermi LCD illuminati da dietro, posso spegnere parti del display quando non le uso per qualche minuto e riattivarle istantaneamente.
* HardDisk: far girare il disco magnetico consuma energia-> lo fermo quando non serve -> ripartenza lenta -> studio quando conviene spegnerlo.

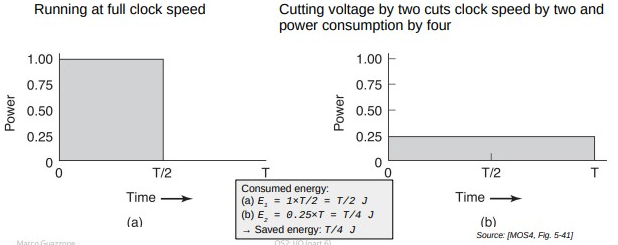
Break-even point: periodo per cui l'energia per rendere il disco low-power e farlo ripartire poi è la STESSA di tenerlo attivo.

Esd + Ps\*(Td-Tsd-Twu)+Ewu = Pw\*Td.

> B-E P =conviene spegnerlo. <B-E P non conviene spegnerlo.

* L'HD può fare un prefetch dei dischi in una cache larga, se un blocco richiesto è in cache, il disco "fermo" non deve ripartire.
* Il SO può rimandare delle operazioni che richiedono il disco a quando sarà attivo ->complesso.
* Consumo della CPU: P=P dinamico + P statico. Proporzionale al quadrato del voltaggio e alla frequenza di clock: P=k\*V^2\*f +P statico.

Riduco V e F->**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 freq 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.

ES: programma decomprime e mostra a schermo un frame ogni 40msec.

P static=0, piena potenza: consuma x J in 40 msec e svolge il suo lavoro in 20msec ->consuma x/2 J.

# Strategia 1: piena potenza per 20 msec e poi spento: E=x/2J

# Strategia 2: metà potenza per 40 msec: E= x/4 J =>MIGLIORE.

ES: digito 1 char/sec e 100msec per processarlo. Max voltaggio V->V/n => Potenza P->P/n^2

Valore ottimo di n?

n=10->100msec\*10=1sec ==> Energia senza tagli=P\*0.1sec + 0\*0.9sec=0.1\*P J

Energia con tagli: (P/100)\*1sec = 0.01\*P J= E senza tagli/10 risparmio 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.