Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

Pietro Frasca

Lezione 16

Martedì 6-12-2016

Driver di un dispositivo

- Vediamo come è strutturato un driver di dispositivo, mostrando uno schema di funzione di lettura (la scrittura è simile) dell'interfaccia *e la funzione di risposta all'interruzione.*

  int _read(int disp, char *pbuf, int cont)

  - **disp** identifica il dispositivo
  - **pbuf** il puntatore al buffer di sistema in cui trasferire i dati letti
  - **cont** il numero di byte da leggere

- La funzione read ritorna il valore -1 nel caso si verifichi un'eccezione.

```
int _read(int disp, char *pbuf, int cont){
     //numero di dati da leggere
     descrittore[disp].contatore = cont;
     //indirizzo del buffer di sistema
     descrittore[disp].pbuffer = pbuf;
     // bit start=1; bit r=1 (lettura)
    descrittore[disp].controllo = <opcode_read>
     <ATTIVAZIONE DEL DISPOSITIVO>;
     // sospensione del processo
     wait(descrittore[disp].dato_pronto);

     // il processo torna in esecuzione
     if (descrittore[disp].stato == <codice errore>)
            return -1;
     else
        return cont-descrittore[disp].contatore;
}
```

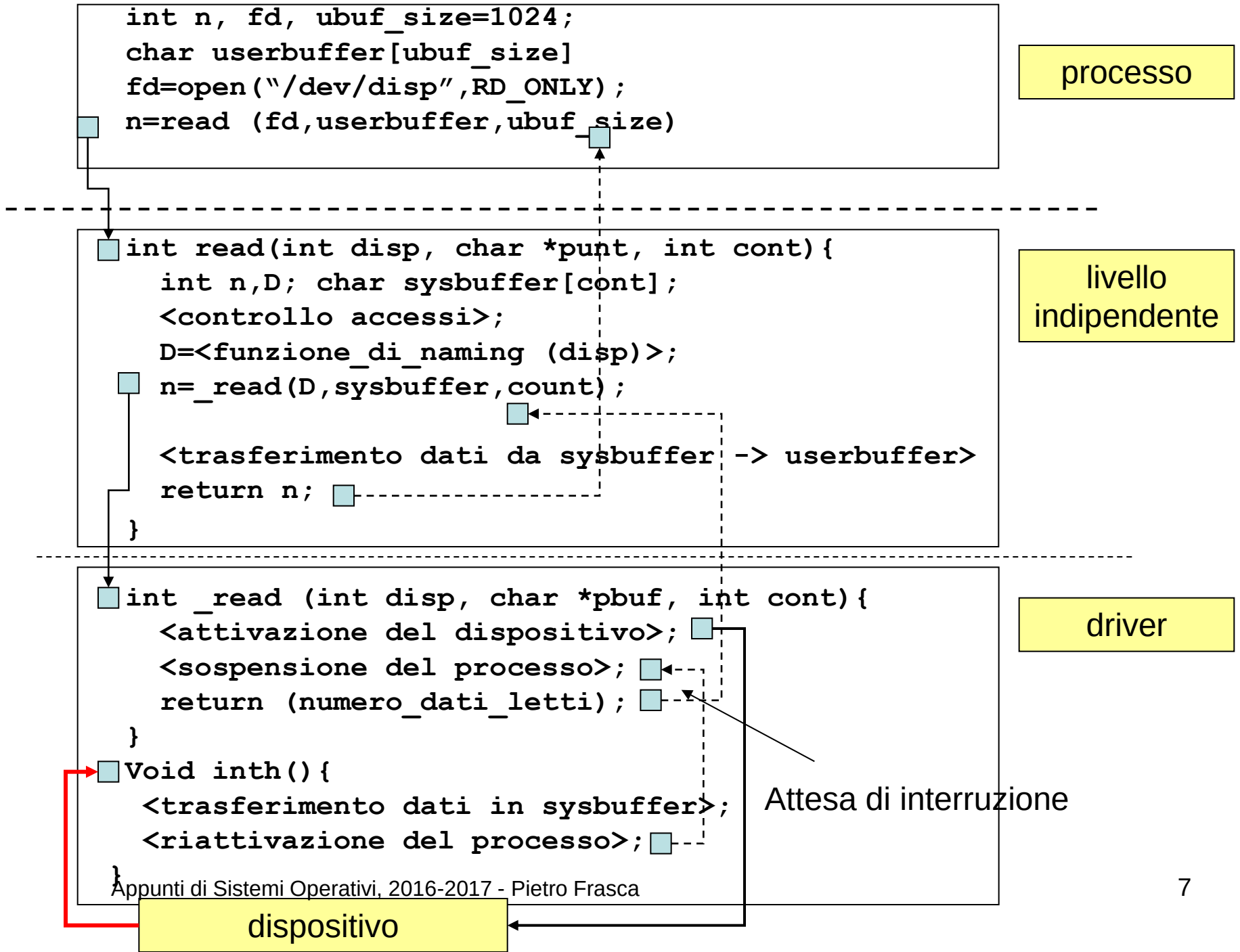
```
void inth() {
  //funzione di gestione delle interruzioni
   char dato_letto;
   if (descrittore[disp].stato != <codice errore>){
       // assenza di errori
       dato_letto = <VALORE DI REGISTRO_DATI>;
       // trasferimento di dato_letto in memoria
       *descrittore[disp].puntatore = dato_letto;
       descrittore[disp].puntatore++;
       descrittore[disp].contatore--;
       if (descrittore[disp].contatore !=0)
              <RIATTIVAZIONE DISPOSITIVO>;
       else {
               descrittore[disp].stato=<TERMINAZIONE OK>;
              <DISATTIVAZIONE DEL DISPOSITIVO>;
       }
```

```
    else {
        //presenza di errori
        <FUNZIONE GESTIONE ERRORE>;
        if (<errore_grave>)
              descrittore[disp].stato=<codice errore>

    }
    //riattivazione del processo
    signal(descrittore[disp].dato_pronto);
    return; // ritorno da interruzione
}
```

Flusso di controllo durante un trasferimento

- Vediamo un esempio di come è strutturato il flusso di controllo durante l'esecuzione di una chiamata di sistema relativa ad un trasferimento di dati.
- L'esempio mostra l'esecuzione di una chiamata di sistema che un processo P esegue per leggere, **in modalità sincrona,** un blocco di byte da un dispositivo.


int n, fd, ubuf_size=1024;
char userbuffer[ubuf_size]
fd=open("/dev/disp",RD_ONLY);
n=read (fd,userbuffer,ubuf_size)
processo
int read(int disp, char *punt, int cont){
int n,D; char sysbuffer[cont];
<controllo accessi>;
D=<funzione_di_naming (disp)>;
n=_read(D,sysbuffer,count);
<trasferimento dati da sysbuffer -> userbuffer>
return n;
}
livello indipendente
int _read (int disp, char *pbuf, int cont){
<attivazione del dispositivo>;
<sospensione del processo>;
return (numero_dati_letti);
}
Void inth(){
<trasferimento dati in sysbuffer>;
<riattivazione del processo>;
}
driver
Attesa di interruzione
dispositivo

Gestione del temporizzatore

- Un dispositivo molto importante è il timer che generalmente svolge i seguenti compiti:
  - In un sistema time-sharing con politica round-roubin genera un segnale di interruzione ad intervalli regolari di tempo (**quanto di tempo**);
  - E' usato nelle applicazioni per la gestione della data e dell'ora e per la generazione di interruzioni software dipendenti dal tempo;
  - Nei sistemi real-time è usato per fornire ai processi applicativi servizi che consentono di stabilire attese programmate e ricevere segnali di time-out;
- A livello HW, il controllore di un timer contiene:
  - Registri di controllo e stato;
  - Un registro contatore programmabile per stabilire un intervallo di tempo, trascorso il quale il timer genera un segnale di interruzione.
- Anche il timer ha il suo driver, in cui viene implementata una funzione **delay** disponibile per i processi applicativi, un descrittore e la funzione di risposta all'interruzione.

- La funzione **delay**, generalmente ha un parametro che specifica il tempo di attesa prima che si verifichi un'interruzione (ad esempio, delay(1000))

Descrittore del timer

- Nel descrittore è presente un array di **N semafori**, uno per ogni processo e **tutti inizializzati a 0**. Ogni semaforo viene usato per sospendere il corrispondente processo che chiama la primitiva delay.
- L'array di interi **ritardo[]** memorizza l'intervallo di tempo di attesa programmato scaduto il quale il processo viene riavviato.
- Quando il timer genera il segnale di interruzione viene eseguita la funzione **inth** che scandisce l'intero array **ritardo**. Se l'elemento **i** dell'array ha un valore diverso da **0** vuol dire che il processo **Pi** deve attendere ancora. Il valore dell'elemento **i** viene quindi decrementato e se raggiunge il valore **0** il processo **Pi** viene riattivato.

```
void delay (int n) {
   int proc;
   proc=<indice del processo in esecuzione>;
   descrittore.ritardo[proc]=n;
   //sospensione del processo;
   wait(descrittore.fine_attesa[proc]);
}

void inth(){
   for (int i=0;i<N;i++){
      if (descrittore.ritardo[i]> 0)
         descrittore.ritardo[i]--;
       else if (descrittore.ritardo[i]==0)
            signal(descrittore.fine_attesa[i]);
    }
}
```

Gestione dei dischi

- Dispositivi di grande importanza
- forniscono il supporto per la memoria virtuale
- supporto alla memorizzazione dei file
- L'efficienza e l'affidabilità dei dischi si riflette sull'intero sistema
- Probabilmente nell'immediato futuro gli SSD sostituiranno completamente gli HD i quali tuttavia sono ancora molto diffusi

Organizzazione fisica

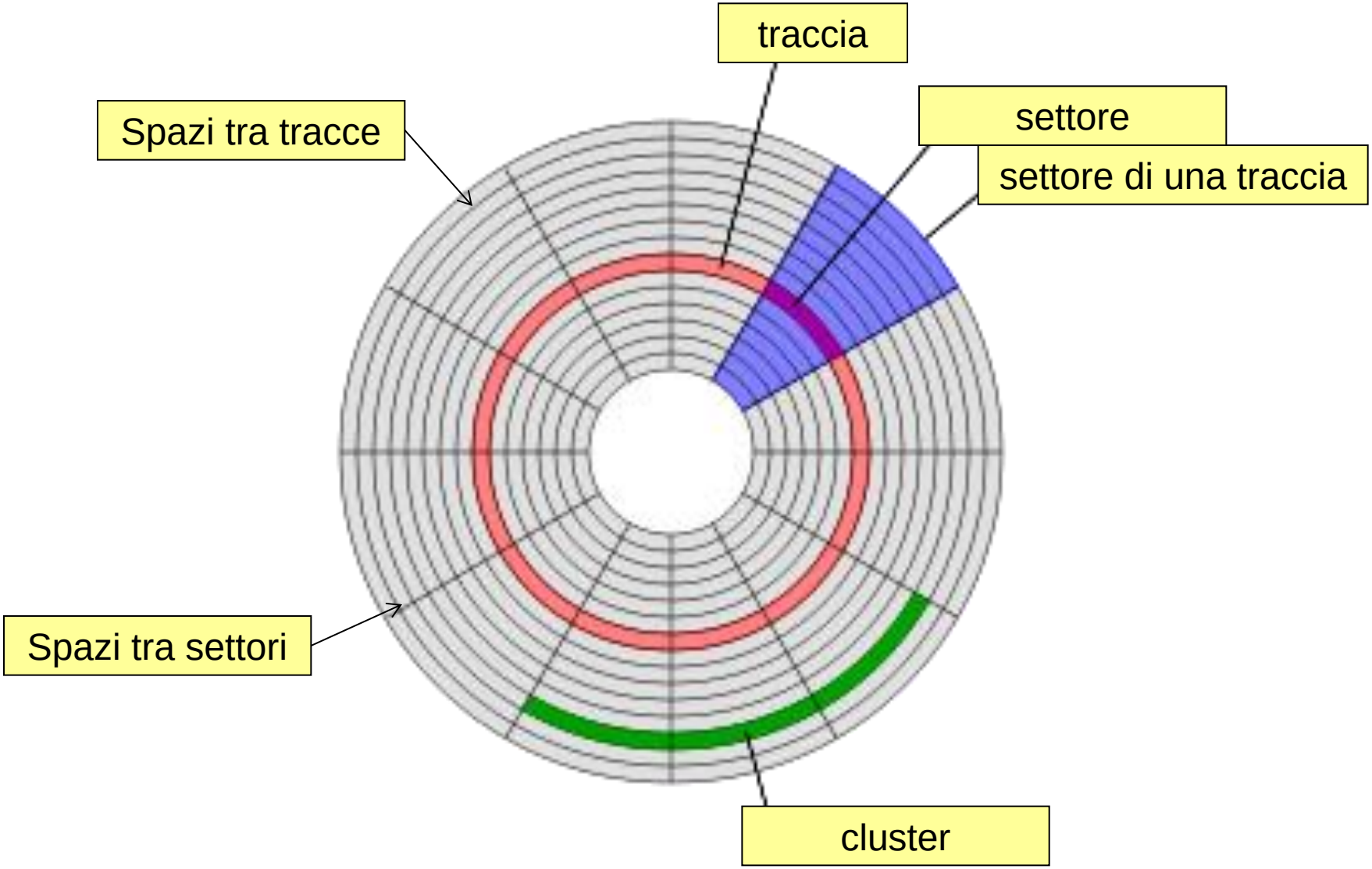
- Un disco è formato da vari dischetti tipicamente di alluminio o plastica, ricoperti di materiale magnetico.
- le testine possono essere fisse (una per ogni traccia) o mobili (una per ogni faccia del disco)
- Le testine mobili si muovono radialmente
- Sia due tracce adiacenti che due settori adiacenti sono separati da un piccolo spazio, privo di materiale magnetico (intertrack gap e intersector gap)

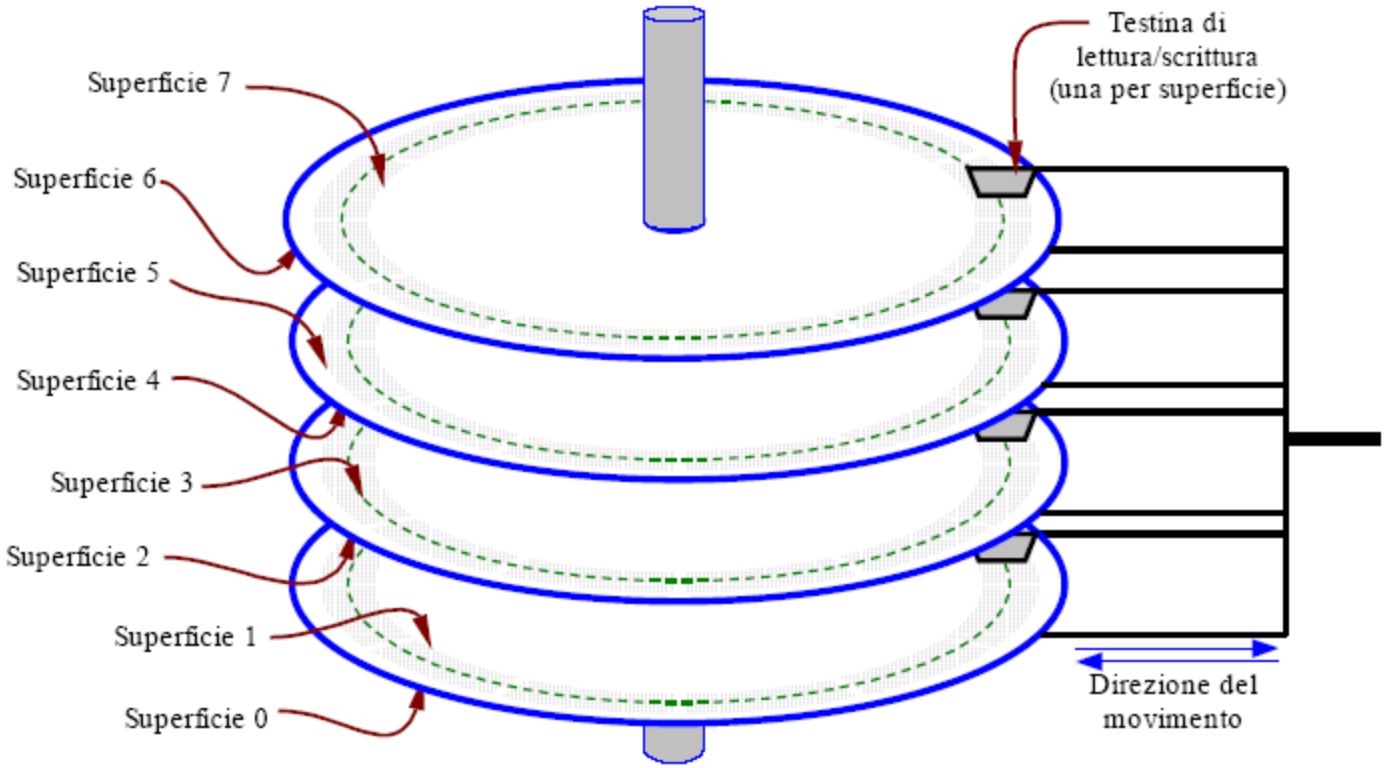
- Generalmente, il numero di dati memorizzabili su ogni traccia è costante, quindi la densità è maggiore per le tracce più interne. Questa caratteristica consente di semplificare la logica di controllo del disco.
- Interfacce standard di dischi molto diffuse sono ATA (Advanced Technology Attachment) conosciuta anche come IDE, che può essere Parallel ATA (PATA, meno recente) o Serial ATA (SATA, più recente) e SCSI (small computer system interface).
- Le dimensioni fisiche del disco fisso sono 3,5" o 2,5". Nei PC desktop i dischi sono praticamente tutti in formato 3,5", con l'attacco PATA sempre meno utilizzato a favore del più veloce SATA, standard giunto alla seconda versione.
- La maggior parte dei dischi fissi per desktop hanno una velocità di 7.200 rpm (120 rps), ma esistono modelli particolarmente veloci da 10.000 rpm.

- Le capacità attualmente più diffuse sono comprese tra i 500 GB a qualche TB.
- Nei notebook, si usano prevalentemente dischi da 2,5", che nella maggior parte dei casi raggiungono la velocità massima di 5.400 rpm, con pochissimi modelli da 7.200 rpm. Anche le capacità sono inferiori, e sono comprese tra 160 e 1 TB.

- La grande novità nel settore è costituita dai Solid State Drive (SSD), che man mano sostituiranno gli HD magnetici.
- I principali vantaggi di queste unità sono: minor peso e dimensioni, nessuna rumorosità, consumi e tempi di accesso ridotti. La velocità di trasferimento è intorno ai 500 MB/sec.
- Unico limite, al momento, il costo per GB di molto superiore ai classici hard disk.

Interno di un disco rigido


traccia
Spazi tra tracce
settore
settore di una traccia
Spazi tra settori
cluster



Le tracce in grigio formano un "cilindro"

- il trasferimento di dati tra disco e memoria avviene a gruppi di dimensione multipla di quella del settore (cluster).
- Il disco prima dell'uso deve essere formattato: sono memorizzati nei settori alcuni dati di controllo che consentono al controllore di identificare le tracce e i settori.
- Tipicamente i dischi hanno due facce per ciascun piatto.
- Il **cilindro** è l'insieme di tutte le tracce concentriche che formano l'HD.
- Il blocco è l'insieme di tutti i settori che occupano la stessa posizione nelle diverse tracce.

- Un settore di una traccia costituisce l'unità minima di memorizzazione dei dati. Il suo indirizzo **Is** è una funzione dei tre parametri: **f**, (faccia), **t** (traccia o cilindro) e **s** (settore):

$$Is = f(f,t,s);$$
$$Is = NT*f + NS*t + s$$

NT: numero di tracce per faccia (o testina)
NS: numero di settori per traccia

- In base a tale funzione un disco è visto logicamente come un array di blocchi contigui:

Parametri caratteristici di un disco

Numero di cilindri	13614
Tracce per cilindro	8
Settori per traccia	320
Byte per settore	512
Capacità	18.3 GB
Tempo min. di seek	0.6 ms
Tempo medio di seek	5.2 ms
Tempo di rotazione	6 ms
Tempo di trasf. di un settore	19 us

Criteri di ordinamento dei dati su disco e politiche di scheduling

- Per valutare le prestazioni di un disco si ricorre spesso al **tempo medio di trasferimento** (**Tmt**), che indica il tempo medio necessario per effettuare la scrittura o la lettura di una certa quantità di byte.

- Il **Tmt** dipende da due parametri:
  - 1) **il tempo medio di accesso** (**Tma**) costituito dal tempo che la testina impiega per posizionarsi in corrispondenza del settore desiderato, e
  - 2) il **tempo di trasferimento** (**Tts**) vero e proprio necessario per trasferire i dati del settore.

$$Tmt = Tma + Tts$$

- Il tempo medio di accesso **Tma** a sua volta, dipende da due fattori
  - 1) **tempo medio di seek (Tseek)** che è il tempo necessario per spostare la testina in corrispondenza della traccia contenente il settore desiderato;
  - 2) **Latenza di rotazione (Rotational Latency, Trl)** che è il tempo di rotazione che il disco impiega per trovarsi sul settore.

  Tma = Tseek + Trl

  La relazione precedente diventa:

  Tma = Tseek + Trl + Tts

- Il tempo **Tts** può essere approssimato, trascurando gli spazi tra settori (intersector gap), al valore **Trot/ns** dove **ns** indica il numero di settori per traccia e **Trot è** il tempo di rotazione che indica il tempo necessario per compiere un giro del disco. Per il disco dell'esempio si ha che:

  Tts=6 ms /320 = 0,01875 ms che è circa 19 μS

- E’ evidente che il tempo medio di trasferimento dipende fondamentalmente dal tempo medio di accesso **Tma** e quindi da **Tseek** e **Trl**.
- Per ridurre il **Tmt** è necessario agire su due aspetti:
  - Criteri di memorizzazione dei dati su disco;
  - Politiche di scheduling per l’accesso al disco da parte dei vari processi.

- Per mostrare l’importanza del modo di memorizzare i dati su disco consideriamo il caso di memorizzazione di un file di 320 KB:
  - 1) Memorizzazione su due tracce contigue.
  - 2) Memorizzazione su tracce e settori sparsi. (esempio di alta frammentazione del disco)

Memorizzazione su due tracce contigue

- Per il caso 1 il tempo **Tmt** si calcola:

  ns=file_size/sec_size=320*1024/512=640 settori

  se il file è allocato in due tracce adiacenti si ha che il tempo necessario per leggere le tracce è dato:

  prima traccia

  Tmt1=Tseek+Trot/2+320*Tts=5.2+3+0.019*320=14.28

  seconda traccia

  Tmt2=Tseek_min+Trot/2+320*Tts=0.6+3+0.019*320=9.68

  Tmt=Tmt1+Tmt2=14.28+9.68=23.96 ms

Memorizzazione su tracce e settori sparsi

- Nel caso 2 (settori sparsi) si ha:

  Tmt=(Tseek+Trot/2+Tts)*640 =
  (5.2 +3+0.019)*640=5260.16 ms

quindi il tempo aumenta di un fattore

  f=5260.16/23.96=219.54