05b - Gestione dell'I/O

Unità a Disco

Basic File System

Fanno parte del Basic File System le funzioni:

- **Caching** : memoria veloce per tenere una copia dei dati che si stanno utilizzando (la cache contiene sempre una copia, il buffer non è detto).
- **Buffering** : memorizzazione transitoria dei dati in memoria mentre essi sono trasferiti fra due dispositivi o tra un'applicazione ed un dispositivo.

- **Spooling** : è una coda contenente dati per un dispositivo che non può accettare flussi di dati intercalati (i dati provenienti da una singola applicazione si registrano in uno specifico file su disco, quando il flusso è terminato, si inserisce il file nello spool).
- **Prenotazione dei dispositivi** : sistema di prenotazione ed accodamento dei processi che richiedono l'uso di dispositivi non condivisibili.
- **Gestione degli errori** : dovuti a motivi contingenti o permanenti. Gli errori spesso vengono riportati in un error log.

  - Errori contingenti → il SO ritenta.
  - Errori permanenti → viene restituito un codice di errore.
- **Scheduling** : stabilire un ordine di esecuzione efficace ad un insieme di richieste di I/O. Si applicano criteri simili a quelli visti per la CPU.

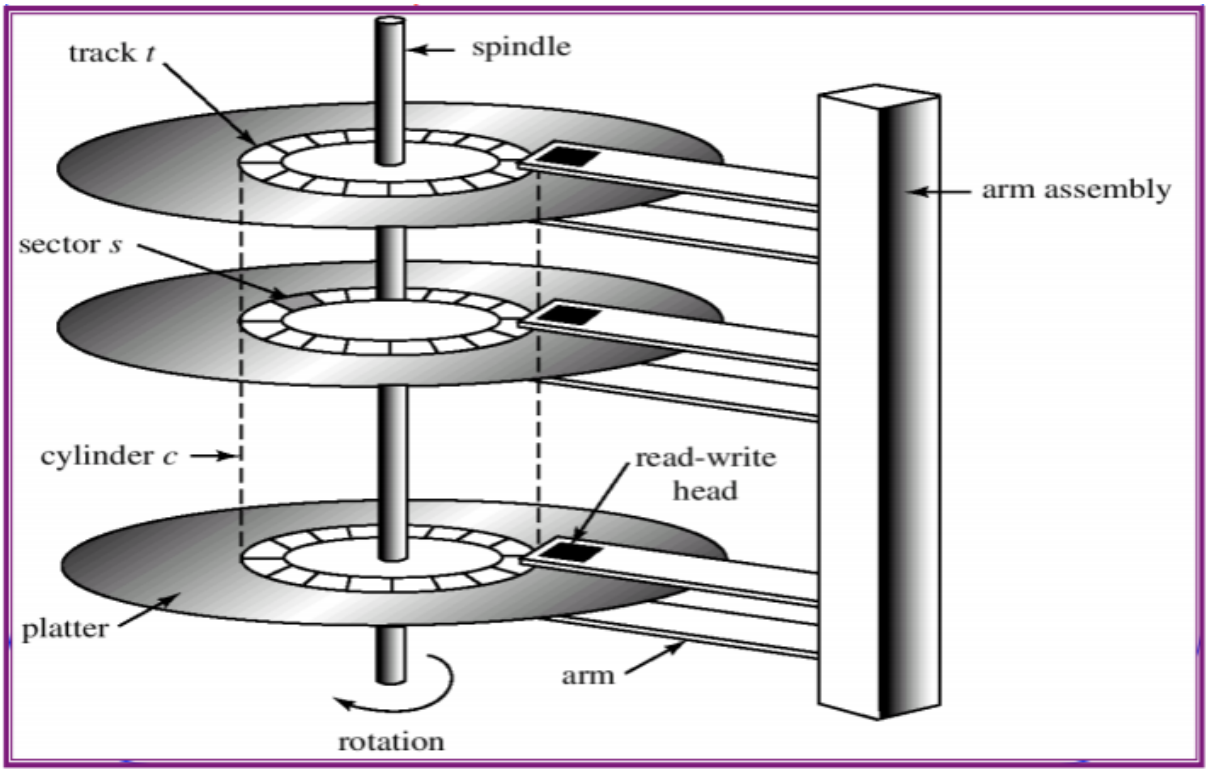
Gestione degli errori

Errori di programmazione : il programmatore chiede qualcosa di impossibile/inconsistente (scrivere su un CD-ROM, tentare una seek su una seriale, accedere ad un dispositivo non installato, ...).
- È il sottosistema di I/O che si accorge dell'errore.
- Azione : abortire la chiamata, segnalando l'errore al chiamante.

Errori del dispositivo : dipende dal dispositivo.
- È il controller che si accorge dell'errore, inviando un'interruzione hw.
- Se transitori : cercare di ripetere le operazioni fino a che l'errore viene superato (rete congestionata).
- Abortire la chiamata : adatto per situazioni non interattive, o per errori non recuperabili.
- Far intervenire l'utente/operatore : adatto per situazioni riparabili da intervento esterno.

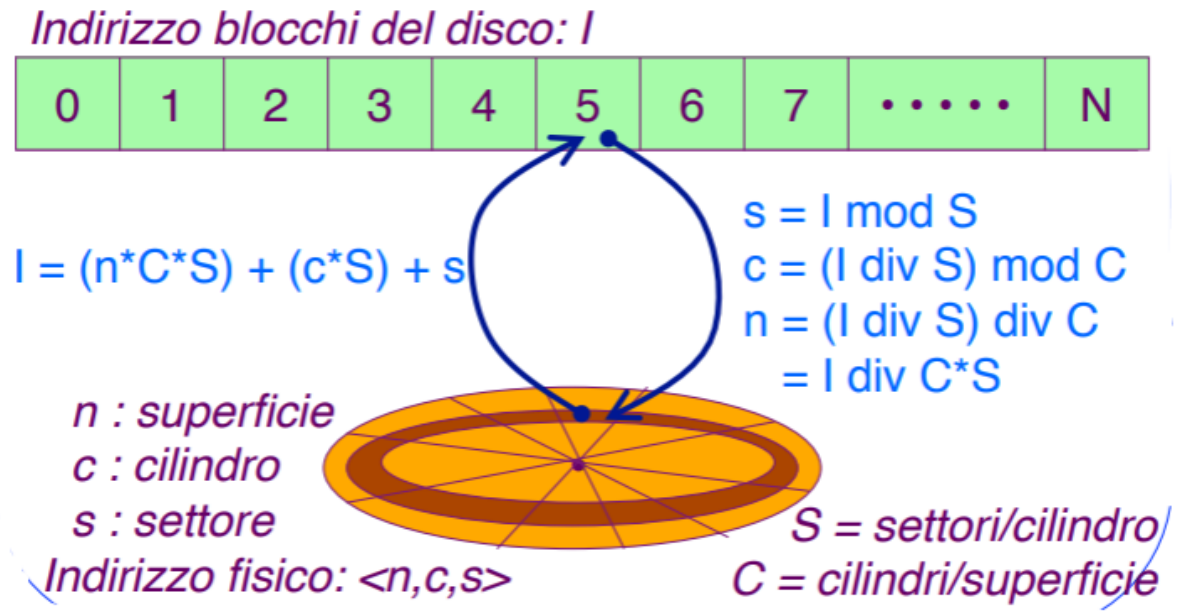
Struttura del disco

A basso livello, le unità a disco sono suddivise in settori individuati dalla seguente tripletta : $< superficie, cilindro, settore >$



Le unità a disco sono indirizzate come un unico grande array mono-dimensionale di blocchi logici, dove i blocchi logici sono l'unità di trasferimento più piccola.

I blocchi logici dell'array mono-dimensionale vengono associati ai **settori** del disco in modo sequenziale.



Scheduling del disco

Al fine di garantire un uso efficiente dell'unità a disco significa avere tempo di accesso contenuti e ampiezza di banda elevata.

Il **tempo di accesso** è composto da :

- *Tempo di ricerca* (seek time) : il tempo necessario alle testine per posizionarsi nel cilindro contenente il settore desiderato.
- *Latenza di rotazione* : il tempo necessario affinché il disco ruoti fino a quando il settore desiderato si trova sotto la testina, non gestibile.

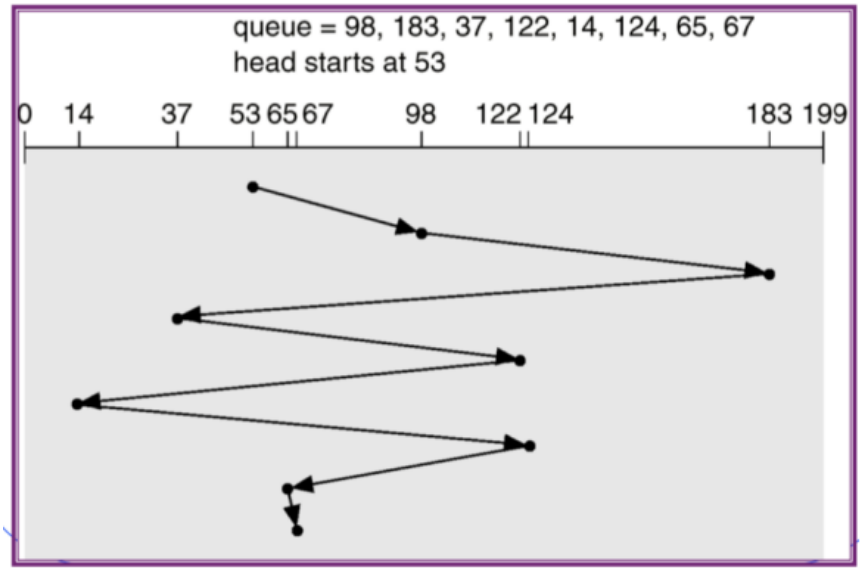
Per ridurre questo tempo di accesso possiamo agire solo sul tempo di ricerca ed in particolare sulla distanza di ricerca; il nostro obiettivo è quello di ridurre il numero di settori visitati, in quanto il tempo è inversamente proporzionale alla distanza percorsa.

L'**ampiezza di banda** (bandwidth) è il numero totale di byte trasferiti diviso il tempo totale intercorso fra la prima richiesta e il completamento dell'ultimo trasferimento.

Per mezzo della gestione dell'ordine delle richieste di I/O relative al disco si possono migliorare sia il tempo d'accesso sia l'ampiezza di banda

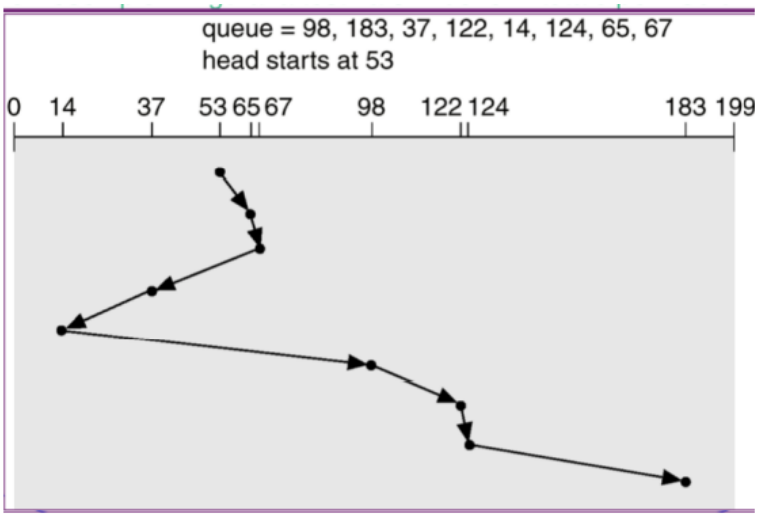
I diversi algoritmi di scheduling vengono illustrati prendendo in considerazione i cilindri che contengono i blocchi richiesti. Questo è dovuto alla scelta di considerare solo il tempo di ricerca.

FCFS (First Come First Served)



Andiamo a selezionare la richiesta secondo l'ordine di arrivo, nell'esempio in figura la testina si muove in totale per 640 cilindri. Vado a sommarli in modulo. Osserviamo come però il disco essendo un dispositivo elettromeccanico, la distanza percorsa potrebbe andare a influenzare sul corretto funzionamento e causando nel lungo termine possibili danneggiamenti.

SSTF (Shortest Seek Time First)



Stesso ragionamento fatto nel caso della CPU con SJF, ovvero diamo priorità alle richieste più brevi, quindi con la minor distanza. Ovvero si seleziona la richiesta con il minimo tempo di ricerca rispetto alla posizione corrente della testina.

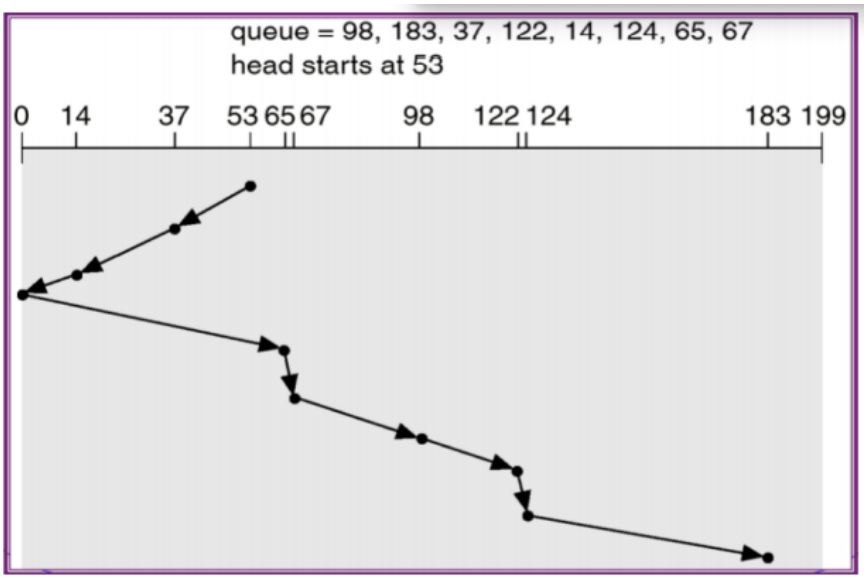
Nota come però potrebbe verificarsi STARVATION. Osserviamo inoltre come però questo algoritmo riduce il numero spostamenti e cambi di direzione. NON E' L'ALGORITMO OTTIMO, e nemmeno in media si avvicina all'algoritmo migliore.
Motivazione intuitiva: la distanza più breve cambia ad ogni step, quindi non sono in grado di trovare una sequenza ottima. Quindi se modifico la sequenza potrei avere un comportamento migliore.

SCAN e C-SCAN

SCAN :

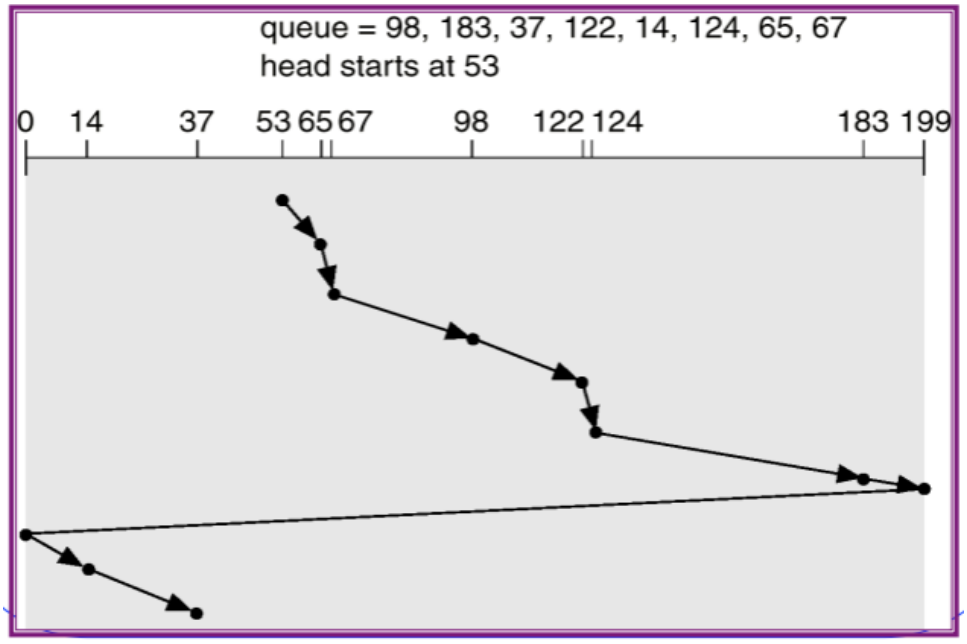
Il braccio del disco parte da un estremo del disco e si muove verso l'altro estremo per poi tornare indietro, servendo le richieste a mano a mano che le incontra (*algoritmo dell'ascensore*).

NON È L'ALGORITMO OTTIMO, percorro un tratto senza la necessità di quella operazione, ci converrebbe fermare all'ultima richiesta in quella determinata direzione.

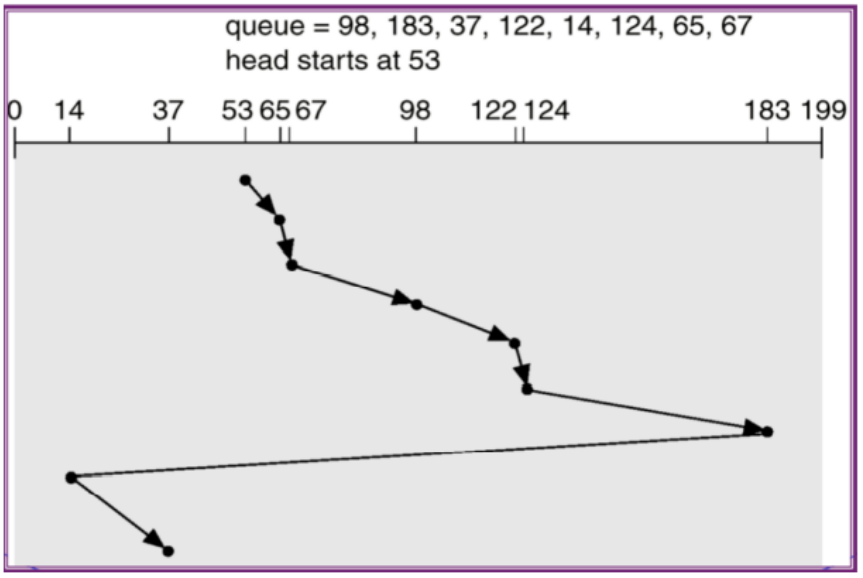


C-SCAN :

Come SCAN, solo che arrivato ad un estremo del disco, riparte verso l'altro estermo senza servire le richieste durante il viaggio di ritorno. Tratta i cilindri come una lista circolare.



Look & C-Look



Esempio di C-Look

Varianti di SCAN e C-Scan.

Il braccio si sposta verso una estremità solo fino a quando non si sono altre richieste da servire in quella direzione.
Lista concatenata, ma permette di andare in una direzione crescente, una volta torno al cilindro 0 e poi posso ripartire.

Gestione dell'Unità a disco

- **Formattazione a basso livello** (**formattazione fisica**) : suddivisione del disco in settori che il controller del disco può leggere e scrivere.
- **Partizioni** : suddivisione del disco in uno o più gruppi di cilindri. Ogni partizione viene gestita dal SO come una unità a disco a se stante.
- **Formattazione logica** : creazione del file system.
- **Blocco di avviamento** (**boot block**) : contiene il programma di bootstrap.
- **Area di swap** : è l'area del disco utilizzata dal SO per la memoria virtuale. Può essere collocata :
  - all'interno del normale file system
  - in una partizione a se stante

Strutture RAID

I dischi RAID (*Redundant Array of Inexpensive Disks*) sono batterie ridondanti di dischi.

- Miglioramento delle prestazioni tramite il parallelismo:
  -
  Sezionamento dei dati (*data striping*) : consiste nel distribuire i bit di ciascun byte su più dischi.
- Miglioramento dell'affidabilità tramite la ridondanza:
  -
  Copiatura speculare (*mirroring* o *shadowing*) : ogni disco logico consiste di due dischi fisici e ogni scrittura si effettua su entrambi i dischi. Se uno dei dischi si guasta, si possono utilizzare i dati dell'altro disco.
  -
  Sezionamento + bit di parità (o + codici di correzione degli errori) : per diminuire il grado di ridondanza richiesto si può pensare di combinare il sezionamento con l'uso dei codici di correzione degli errori (o con l'uso dei bit di parità). Questi codici associano ad ogni byte uno o più bit il valore dei quali dipende dai valori dei bit contenuti nel byte stesso.

I/O control

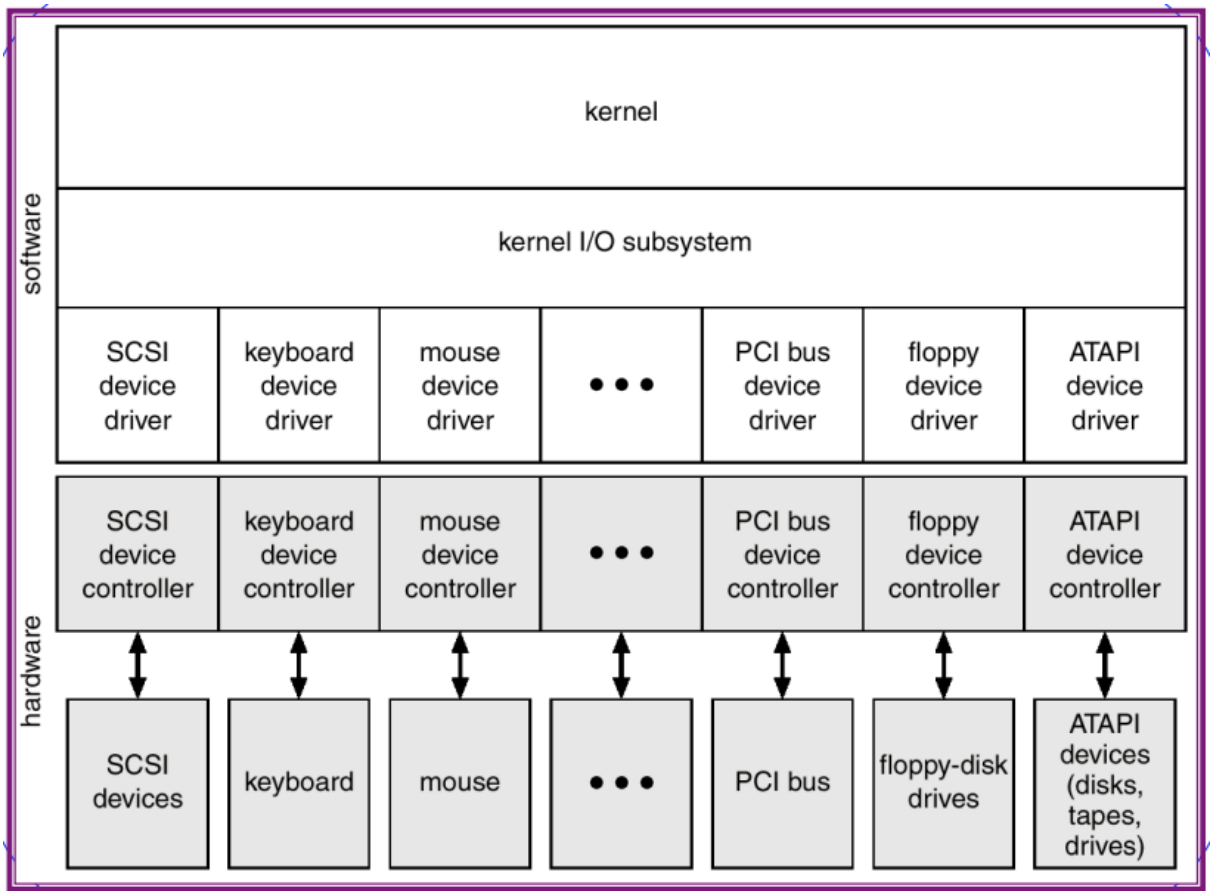
Driver

Un driver è l'insieme di procedure che permette ad un SO di pilotare un dispositivo hardware. Esso permette al sistema operativo di utilizzare l'hardware senza sapere come esso funzioni, ma dialogandoci attraverso un'interfaccia standard (registri del controller della periferica) che astrae dall'implementazione dell'hardware e che ne considera solo il funzionamento logico.

Un driver è un software (spesso di terze parti) che accede al controller dei device. I driver:

- Hanno la vera conoscenza di come far funzionare il dispositivo
- Implementano le funzionalità standardizzate, secondo poche classi (ad es.: carattere/blocchi)
- Vengono eseguiti in modalità kernel
- Per includere un driver, può essere necessario ricompilare o rilinkare il kernel.
- Attualmente si usa un meccanismo di caricamento run-time

Struttura di un driver



Il kernel attraverso il driver invia una richiesta al controller del dispositivo; un modulo del driver traduce tale richiesta nell'operazione opportuna per il controller.

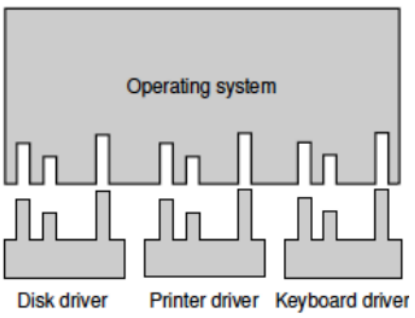
I driver dei dispositivi eseguono i seguenti passi:
1) Controllare i parametri passati all'invocazione del driver: serve per capire come eseguire l'operazione richiesta.
2) Accodare le richieste in una coda di operazioni (soggette a scheduling!).
3) Eseguire le operazioni, accedendo al controller.
4) Far passare il processo allo stato *waiting* (I/O guidato da interrupt) o attendere la fine dell'operazione in busy-waiting.
5) Controllare lo stato dell'operazione nel controller. Deve sapere quando ha finito.
6) Restituire il risultato.

Questo significa che il driver deve interagire con il controller. Per non tenere in attesa attiva il processo (e quindi poter assegnare la CPU ad un altro processo) è il dispositivo che invia interruzioni hardware per segnalare che ha finito. Tale interruzione provoca l'esecuzione di una routine per gestire questa interruzione, questa routine sarà parte del driver.

Quindi il driver sarà composto da due differenti funzioni:
• Quelle invocate mediante interruzioni software dagli strati superiori del kernel.
• Quelle invocate mediante interruzioni hardware inviate dal dispositivo.

Tra queste interruzioni possono esserci quelle dedicate all'inizializzazione del dispositivo (quando lo colleghiamo).

- I driver devono essere rientranti : cioè a metà di una esecuzione può essere lanciata una nuova esecuzione.
- I driver non possono eseguire system call, ma possono accedere ad alcune funzionalità del kernel (es: allocazione memoria per buffer di I/O)
- Nel caso di dispositivi "hot plug": gestire l'inserimento/disinserimento a caldo.



- I driver devono avere una interfaccia uniforme
    - Gli sviluppatori dei driver si devono attenere ad una specifica di cosa devono implementare
    - Deve offrire anche un modo di denominazione uniforme, flessibile e generale

Esempio: driver di una tastiera

Assunzioni semplificative:

- La periferica è un dispositivo di input di tipo carattere.
    - I registri comando e stato si riducono ad un unico registro flag (flag = 0 equivale a richiesta di operazione e flag = 1 equivale a operazione completata)
- La sola operazione possibile è il trasferimento di un carattere in ingresso attraverso il registro di interfaccia dati.
- Il driver non esegue controlli su eventuali malfunzionamenti.
- La procedura `leggi_linea` restituisce in uscita il numero dei caratteri effettivamente trasferiti.
- L'operazione è considerata conclusa se giunge un carattere di fine linea o se il buffer è pieno.
- La lunghezza della linea di caratteri e il carattere da interpretare come fine linea possono essere selezionati attraverso la procedura inizializza.
- La mutua esclusione nell'esecuzione delle procedure viene realizzata attraverso il meccanismo di disabilitazione delle interruzioni.

```
#define BUFFER_SIZE 10
/* Device Control Block */
typedef struct {
 /* buffer di memorizzazione dei caratteri */
 char buffer[BUFFER_SIZE];
```

```
 /* numero caratteri trasferiti */
 int count;
 /* carattere di fine linea */
 char eol;
 /* lunghezza massima di una linea */
 int lungh_linea;
 /* coda di attesa per processo richiedente */
 PCB *trasf_avvenuto;
 /* operazione in corso */
 boolean occupato;
} t_DCB;


/* registri hardware del controller */
typedef struct {
 /* registro di lettura e scrittura dati */
 char dati;
 /* registro di comando e di stato */
 boolean flag;
} t_registri;
t_DCB DCB;
t_registri registri;


/* Procedura di richiesta operazione –
richiamato dalla system call */
int leggi_linea(PCB p; char *local_buffer){
    if (DCB.occupato)
        return –1;
    else {
        DCB.occupato = 1; /* inizializzazione */
        DCB.count = 0;
        registri.flag = 0; /*comando iniz. controller */
        /* attesa del completamento operazione */
        add p to DCB.trasf_avvenuto;
        local_buffer = DCB.buffer /*trasferimento dati*/
        DCB.occupato = 0;
        return DCB.count;
```

```
    }
}


/* Interrupt handler: richiamato dall'interrupt HW */
void risposta_interruzione(){
 /* acquisizione di un carattere */
 DCB.count++;
 DCB.buffer[DCB.count] = registri.dati;
 if (DCB.count == 80) ||
 (DCB.buffer[DCB.count] == DCB.eol)
 /* fine dell'operazione */
 resume p from DCB.traf_avvenuto;
 else
 /* nuovo comando al controller */
 registri.flag = 0;
}


/* Procedura di inizializzazione */
void inizializza(int lunghezza; char eol){
 DCB.lungh_linea = lunghezza;
 DCB.eol = eol;
 DCB.occupato = 0;
}
```

Devices

Dispositivi a blocchi e a caratteri

- **Dispositivi a blocchi**

  I dispositivi a blocchi comprendono i dischi.
  - Sono a disposizione istruzioni quali
  read, *write, seek*.
  - Le applicazioni comunicano attraverso l'interfaccia del file system oppure

attraverso l'I/O di basso livello.
- É possibile l'accesso memory-mapped.

- **Dispositivi a caratteri**

  I dispositivi a carattere comprendono tastiere, mouse, porte seriali.
  - Sono a disposizione istruzioni quali
  get, put.

  - Esistono librerie che permettono operazioni di editing (quali l'accesso riga per riga o il backspace).

- **Dispositivi di rete**
  I dispositivi di rete sono profondamente diversi dagli altri, tanto da avere una interfaccia propria.

Controller dei Dispositivi di I/O

Ogni dispositivo è collegato al bus di sistema tramite un'interfaccia dove risiedono alcuni registri (di controllo, per permettere la sincronizzazione CPU-Dispositivo, buffer per contenere i dati in ingresso/uscita). Questa interfaccia è chiamata **controller**.

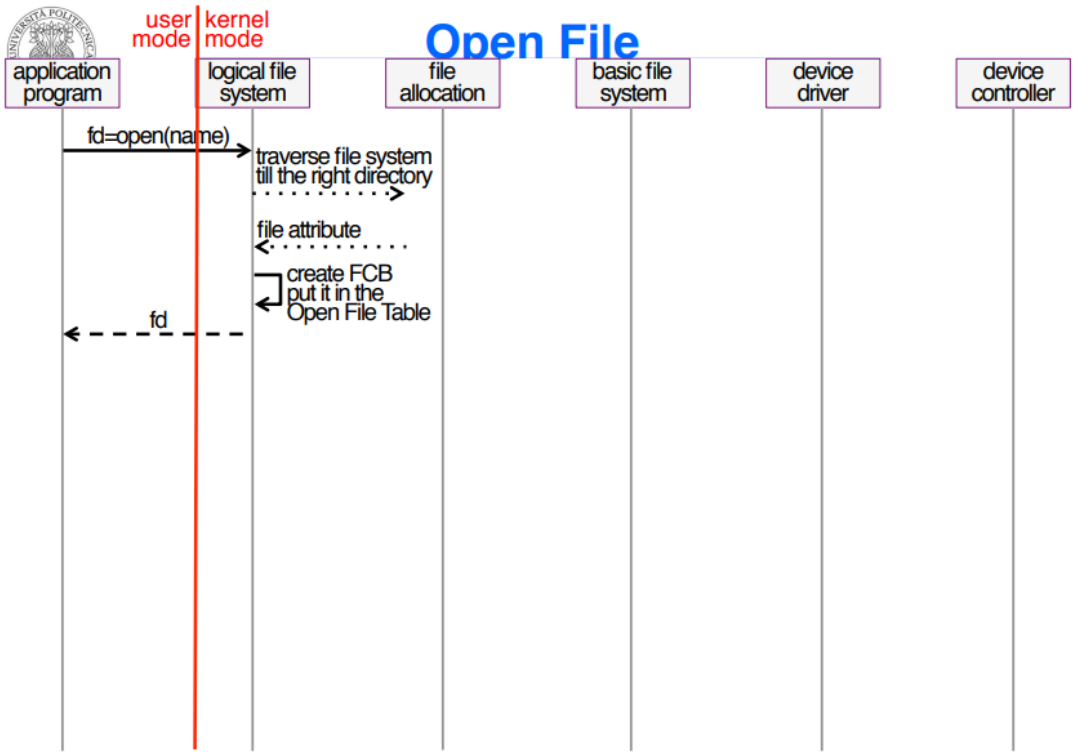
La CPU colloquia con il controller scrivendo/leggendo i suoi registri:

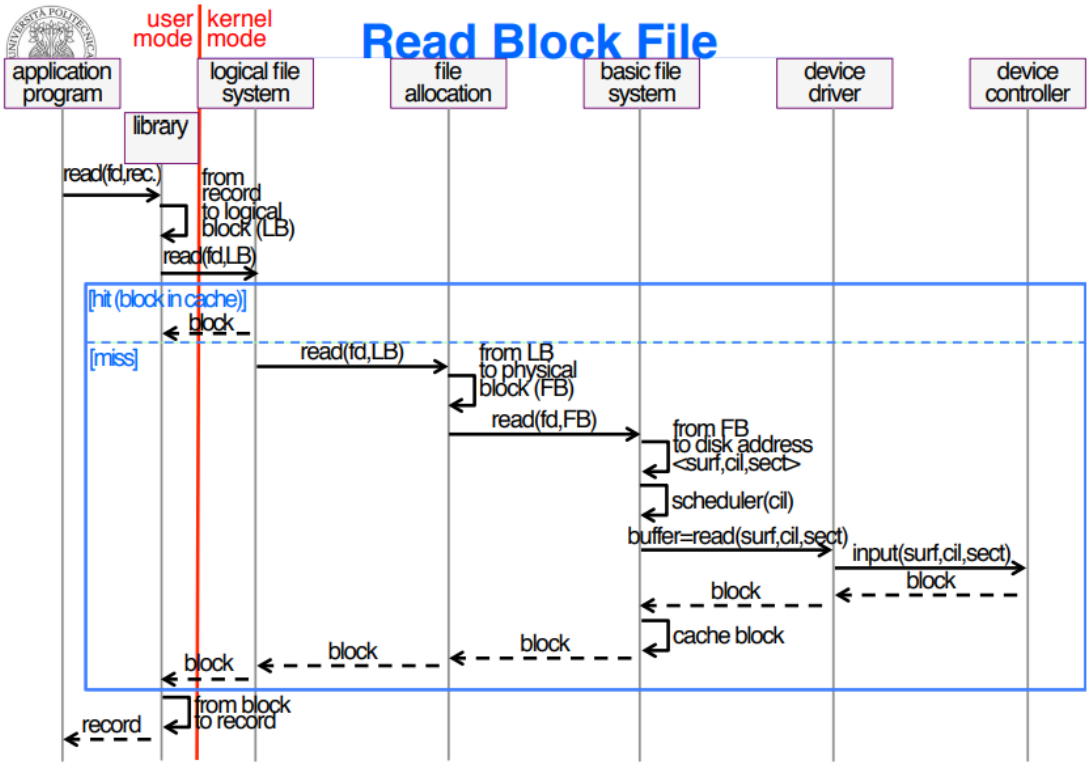
- utilizzando apposite istruzioni (IN / OUT) che avranno come parametro un "indirizzo" che identifica un particolare controller
- utilizzando normali istruzioni LOAD/STORE a indirizzi associati ai registri del controller: questa tecnica si chiama Memory Mapped I/O

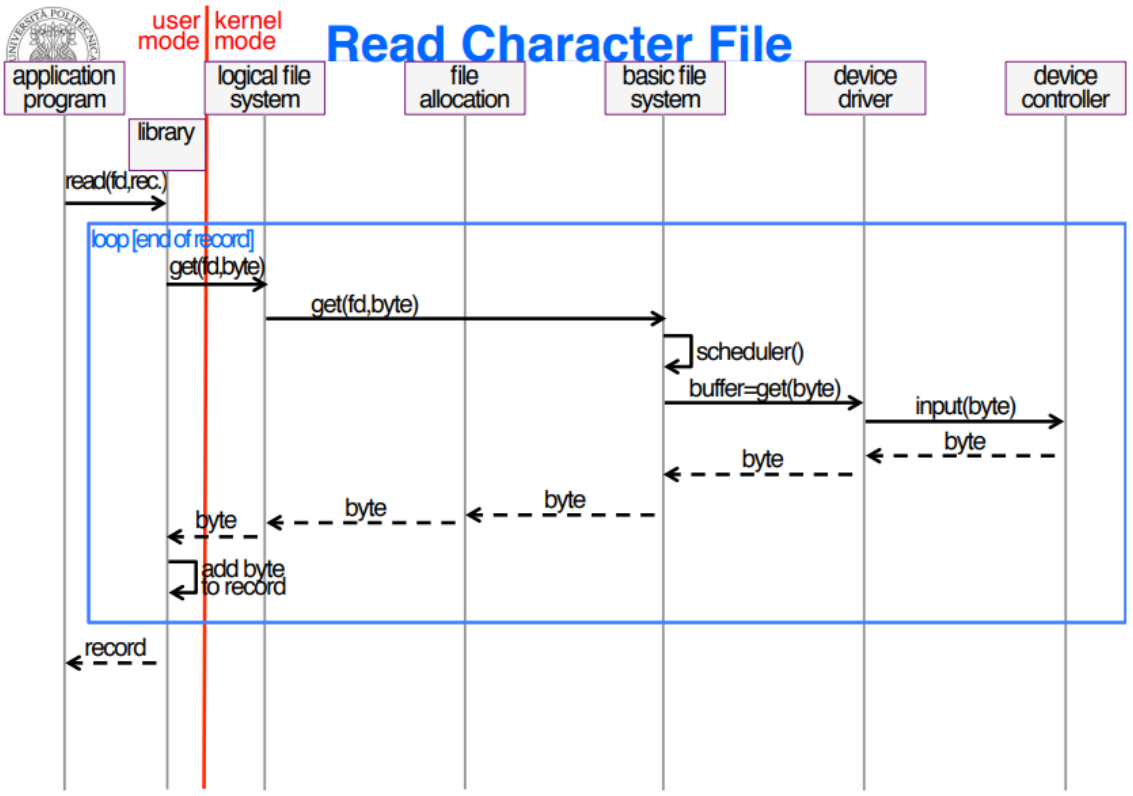
Nelle architetture più semplici si trovano normalmente tre tipi di schemi per eseguire le azioni di I/O:

- I/O programmato con Busy-Waiting
- I/O guidato dalle interruzioni (Interrupt)
- I/O con accesso diretto alla memoria (DMA)

Ciclo di vita di una richiesta di I/O


Open File
user mode
kernel mode
application program
logical file system
file allocation
basic file system
device driver
device controller
fd=open(name)
traverse file system till the right directory
file attribute
create FCB put it in the Open File Table
fd


Read Block File
user mode
kernel mode
application program
library
logical file system
file allocation
basic file system
device driver
device controller
read(fd,rec.)
from record to logical block (LB)
read(fd,LB)
[hit (block in cache)]
block
[miss]
read(fd,LB)
from LB to physical block (FB)
read(fd,FB)
from FB to disk address <surf,cil,sect>
scheduler(cil)
buffer=read(surf,cil,sect)
input(surf,cil,sect)
block
block
cache block
block
block
block
from block to record
record



Gestione dell'I/O in Linux

Linux classifica tutti i dispoisitivi in tre categorie:

- Dispositivi a blocchi : permettono l'accesso diretto a blocchi di dati di dimensione fissa.
- Dispositivi a caratteri : comprendono gran parte dei dispositivi, non sono in grado di fornire le funzionalità dei file ordinari.
- Dispositivi di rete : sono interfacciati con il sottosistema di gestione delle reti del kernel.

Dispositivi a Blocchi

È la principale interfaccia a tutti i dischi del sistema.

Linux accede ai dischi attraverso due cache:

- I dati sono memorizzati nella page cache, che è unita alla memoria virtuale del sistema;
- I metadati sono memorizzati nella buffer cache, una cache separata indicizzata dal blocco fisico del disco.

Il *block buffer* cache serve principalmente a due cose:

- come pool di buffer per I/O attivo,
- come cache per l'I/O completato.

Il request manager gestisce la scrittura e la lettura del buffer con dati provenienti da/diretti verso un driver di un dispositivo a blocchi.

Dispositivi a caratteri

Un driver di un dispositivo a caratteri deve avere un insieme di funzioni che realizzano le varie operazioni sui file.

Il kernel non esegue quasi nessuna operazione alla richiesta di leggere o scrivere su un dispositivo a caratteri, semplicemente passa la richiesta al dispositivo.

La sola eccezione è rappresentata dai terminali, nei confronti dei quali il kernel mantiene una interfaccia standard.

Dispositivi di rete

I dispositivi di rete in Linux permettono di gestire:

- Il protocollo standard di Internet (TCP/IP) per la comunicazione tra macchine Unix,
- Protocolli per macchine con SO non Unix, come Appletalk e IPX.

È realizzato con tre strati di software:
1) L'interfaccia delle socket (permette la gestione della rete come se fosse un file).
2) I driver dei protocolli.
3) I driver dei dispositivi di rete.

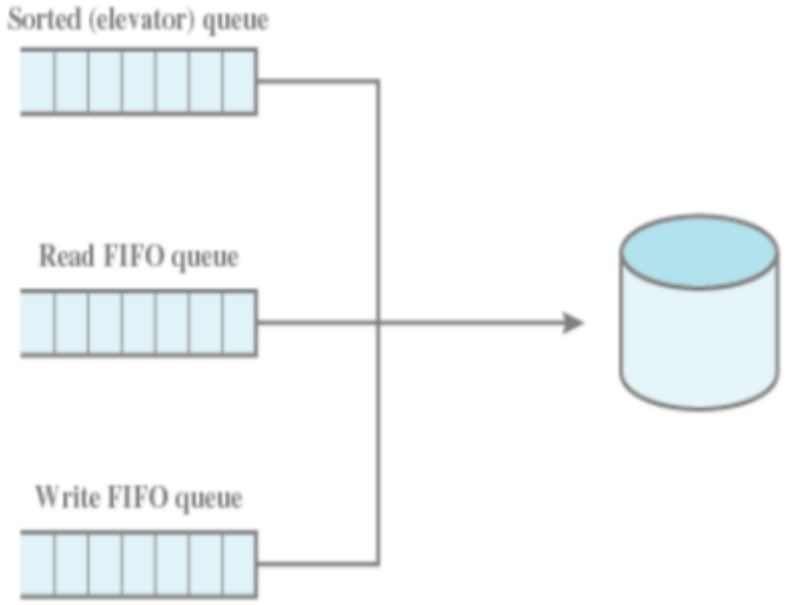
Scheduling del Disco

Fino al kernel 2.4 (Linus Torvalds) vi era il ***Linus Elevator Scheduler***, il quale utilizzava una coda sia per le richieste di lettura che scrittura, ordinava quest'ultime per numero di blocco e si muoveva in unica direzione (potremmo paragonarlo allo SCAN).

Dal kernel 2.6 (Jens Axboe) si adotta il
Deadline Scheduler che utilizza 3 code :

1. **INCOMING** : Coda dove vengono inserite tutte le richieste ordinate per numero di blocco
2. **READ** : Coda FIFO dove vengono replicate e inserite le richieste di lettura presenti nella incoming request
3. **WRITE** : Coda FIFO dove vengono replicate inserite le richieste di scrittura presenti nella incoming request.

Le richieste sono rimosse dalle code R/W se sono state servite dal Linus Elevator Scheduler; se allo scadere della richiesta questa non è stata ancora servita, si passa allo scheduler FCFS (FIFO) per evitare STARVATION, si soddisfa la richiesta e si ritorna al Linus Elevator Scheduler.



Fino al kernel 2.6.33 ci fu anche l
'***Anticipatory I/O scheduler***, i programmi utilizzano spesso settori consecutivi. Dopo aver soddisfatto una richiesta aspetta per un breve periodo (fino a 6ms) per vedere se arriva una richiesta di un settore vicino.

Dal kernel 2.6.33 si passò allo ***CFQ Scheduler (Completely Fair Queueing)*** che sostituisce il l'Anticipatory I/ O scheduler.

- Per le richieste di I/O sincrone: mantiene una coda FIFO di richieste per ogni processo,

- per le richieste di I/O asincrone: mantiene una coda FIFO di richieste per priorità.

Infine attua Round Robin tra le diverse code : il quanto di tempo è in funzione della priorità del processo e di altri parametri (è abbastanza complesso).