Giuseppe Pelagatti

Programmazione e Struttura del sistema operativo Linux

Appunti del corso di
Architettura dei Calcolatori e Sistemi Operativi (AXO)

Parte IO: Input/Output e File System

cap. IO3 – I Gestori di Periferiche

IO3. I GESTORI DI PERIFERICHE

1. Introduzione

I gestori di periferiche (**device drivers**) sono dei componenti il cui funzionamento, a differenza di quello del filesystem, è fortemente intrecciato con quello del nucleo del sistema. Come abbiamo visto, il modo in cui è scritto un gestore di periferiche mette in luce e motiva buona parte degli aspetti di organizzazione del nucleo del sistema visti nei precedenti capitoli.
Tra tutti i componenti di un sistema operativo i gestori di periferiche sono quello che richiede più spesso di essere modificato o esteso per trattare nuovi tipi di periferiche. Pertanto, si tratta dell'area dei sistemi operativi sulla quale capita più spesso di dover lavorare. D'altra parte, la comprensione del funzionamento del nucleo è fondamentale per scrivere un buon gestore di periferica. Infatti, il gestore adatta una specifica periferica a funzionare nel contesto del sistema operativo.
Ogni volta che un nuovo tipo di periferica deve essere gestito da un sistema operativo è necessario che venga realizzato il relativo gestore. Un gestore di periferica è quindi un componente software relativo ad un specifica coppia <sistema oeprativo/ tipo di periferica>. Chi deve realizzare un gestore di periferica deve quindi conoscere sia le caratteristiche Hardware della periferica da gestire, sia il modo in cui il gestore deve interfacciarsi con il resto del sistema operativo considerato. In genere, per ogni sistema operativo esiste un manuale dal tipo "guida alla scrittura di un gestore di periferica", che spiega come affrontare quest'ultimo aspetto.
Nel seguito di questo capitolo ci occuperemo di questo aspetto relativamente al sistema operativo LINUX, cioè delle caratteristiche che deve possedere un generico gestore di periferica per LINUX.

2. Tipi di periferiche e classificazione dei gestori

In LINUX deve esistere uno specifico gestore per ogni tipo di periferica installata: un gestore del disco fisso, un gestore dei floppy disk, un gestore dei CD, un gestore dei terminali, ecc…
I tipi di periferiche sono divisi in due classi: le **periferiche a carattere** (**character devices**), come le tastiere e le stampanti, e le **periferiche a blocchi** (**block devices**), come i vari tipi di dischi.
Storicamente, le periferiche a carattere erano definite tali perché ricevevano e inviavano un carattere alla volta, mentre le periferiche a blocchi erano quelle che trasferiscono con una sola operazione interi blocchi di dati in un'area di memoria detta **buffer**. Oggi, molte periferiche come le stampanti trasferiscono interi blocchi di caratteri, ma rimangono periferiche a carattere, perché la definizione corretta delle due classi è la seguente:

- le periferiche a blocchi sono quelle con accesso a blocchi **casuale** (**random access**), cioè nelle quali qualsiasi blocco della periferica (LBA) può essere letto o scritto in qualsiasi momento. In pratica sono i dischi e le periferiche il cui funzionamento può essere assimilato a quello dei dischi.
- le periferiche a carattere sono quelle sequenziali, cioè quelle nelle quali, anche se molti caratteri vengono trasferiti in un'unica operazione, non ha senso il concetto di indirizzamento casuale dei dati. Ad esempio, una stampante riceve una sequenza di caratteri, e, anche se i caratteri vengono trasferiti a blocchi invece che uno alla volta, non ha nessun senso parlare dell'indirizzamento di tali blocchi. I blocchi infatti esistono all'atto del trasferimento, ma non sono individuabili prima o dopo il trasferimento stesso.

Per quanto riguarda le modalità di un trasferimento a blocchi, si riveda il paragrafo relativo al DMA del capitolo sulle funzionalità del calcolatore.
In corrispondenza di questa classificazione i gestori sono suddivisi in gestori a carattere (`character device driver`) e gestori a blocchi (`block device driver`).

La differenza principale tra queste due classi di gestori risiede nel fatto che nei gestori a blocchi le singole operazioni di lettura e scrittura non sono direttamente derivate da corrispondenti richieste di servizi nel programma utente. Come abbiamo visto nel capitolo sul File System, la necessità di leggere un blocco fisicamente sul disco viene determinata dal Gestore dei Buffer in base alle richieste del Filesystem stesso, che a loro volta derivano da trasformazioni dei servizi di `read` e `write` richiesti dall'utente (Figura 1). Inoltre, il tentativo di tenere il più a lungo possibile in memoria i dati già letti dal disco, ottimizzando le prestazioni del sistema, rende ancora più indiretto il legame tra i servizi richiesti dai processi e le operazioni fisiche effettuate sulle periferiche a blocchi.

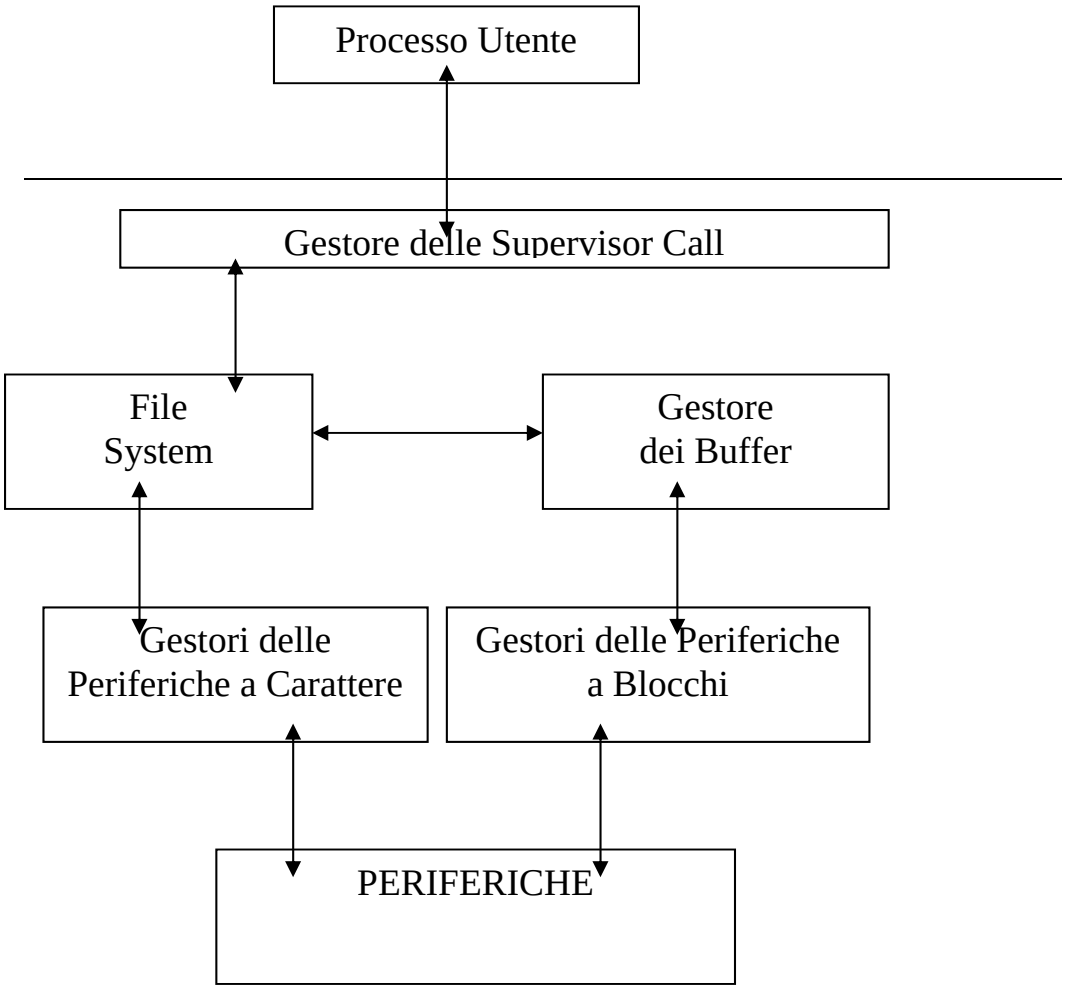

Figura 1 – Gestori delle periferiche a blocchi e a carattere

2. File speciali e identificazione dei gestori

Per accedere una periferica si devono utilizzare i servizi del file system con riferimento al file speciale della periferica. Pertanto, per ogni periferica installata nel sistema deve esistere un corrispondente file speciale, normalmente nel direttorio /dev.
Ogni periferica installata in un sistema è identificata da una coppia di numeri, detti **major** e **minor**; tutte le periferiche dello stesso tipo, cioè gestite dallo stesso gestore, condividono lo stesso major, mentre il minor serve a distinguere le diverse periferiche appartenenti allo stesso tipo.
Se elenchiamo il contenuto del direttorio /dev otteniamo un risultato tipo il seguente:

```
systty      4, 0
tty1        4, 1
tty2        4, 2
```

nel quale la prima colonna contiene il nome del file speciale e la colonna normalmente utilizzata per indicare le dimensioni (size) del file contiene una coppia di numeri che sono il major e il minor.
I file speciali non possono essere creati con la funzione creat( ), ma devono essere creati con una funzione che può essere utilizzata solo dall'amministratore del sistema (root); tale funzione è

```
mknod(pathname, type, major, minor),
```

dove `pathname` è il nome del file speciale, `type` indica che tipo di file si vuole creare (c=character special file, b=block special file) e `major` e `minor` sono i numeri che si vogliono assegnare alla periferica. Normalmente, la funzione `mknod` è utilizzata tramite un corrispondente comando di shell, ad esempio:

```
>mknod /dev/tty4 c 4 5
```

potrebbe essere utilizzato da root per creare un file speciale di tipo carattere di nome tty4, contenuto nel direttorio /dev e associato ai numeri major=4 e minor=5.
I valori major e minor sono posti nel i-node del file speciale, che ovviamente non contiene puntatori ai dati.

3. Le routine del gestore di periferica

Le principali funzioni svolte da un gestore di periferica in LINUX sono le seguenti:

- inizializzare la periferica alla partenza del sistema operativo
- porre la periferica in servizio o fuori servizio
- ricevere dati dalla periferica
- inviare dati alla periferica
- scoprire e gestire gli errori
- gestire gli interrupt della periferica

Il gestore è sostanzialmente costituito da un insieme di routine che vengono attivate dal nucleo al momento opportuno, e dalla routine di interrupt che viene attivata dagli interrupt della periferica.
In ogni gestore esiste una tabella, che chiameremo **tabella delle operazioni**, che contiene puntatori alle funzioni del gestore stesso; il tipo di tale tabella è *`struct file operations`*. Tale tipo, la cui definizione è mostrata parzialmente in figura 2, contiene molti elementi, dei quali solo alcuni sono utilizzati dalla maggior parte gestori. Alcune funzioni saranno discusse più avanti.

```
struct file_operations{
       int (*lseek) ( );
       int (*read) ( );
       int (*write) ( );
       …
       int (*ioctl) ( );
       …
       int (*open) ( );
       void (*release) ( );
       …
}
```

Figura 2 – La struttura file_operation (parziale)

Alla partenza del sistema operativo viene attivata una funzione di inizializzazione per ogni gestore di periferica installato. Tale funzione di inizializzazione, dopo aver svolto le operazioni necessarie di inizializzazione, restituisce al nucleo un puntatore alla propria tabella delle operazioni.
Il nucleo possiede al proprio interno due tabelle, che chiameremo **tabella dei gestori a carattere** e **tabella dei gestori a blocchi**.
Nella Tabella dei gestori a carattere il nucleo inserisce l'indirizzo della tabella delle operazioni di un gestore in posizione corrispondente al major del tipo di periferica gestito dal driver. In questo modo, dopo l'inizializzazione, il nucleo possiede una struttura dati come quella mostrata in figura 3, che permette, dato un major, di trovare l'indirizzo di una qualsiasi funzione del corrispondente driver.
Il meccanismo per l'utilizzazione dei gestori a blocchi è più complesso, perché passa attraverso il Gestore dei Buffer e mira a ottimizzare l'ordine delle operazioni di accesso al dispositivo, e verrà descritto più avanti.

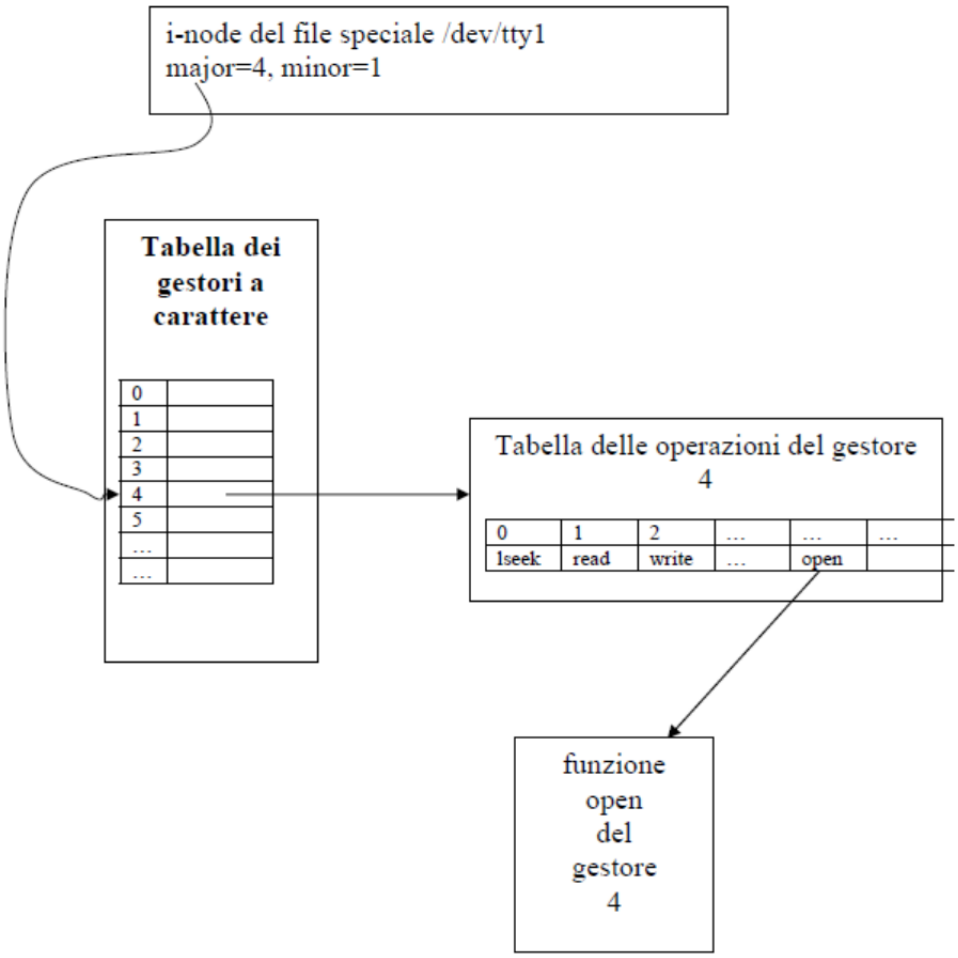

Figura 3 – Tabelle utilizzate dal sistema per indirizzare una funzione di un gestore

Come esempio di uso del meccanismo consideriamo un processo che invoca la funzione `open(/dev/tty1).` Il filesystem accede al file `/dev/tty1` tramite i propri meccanismi e scopre che si tratta di un file speciale a carattere contenente i numeri `major`=4 e `minor`=1. Il filesystem allora accede alla tabella dei gestori a carattere in posizione 4 e vi trova l'indirizzo della tabella delle operazioni di quel gestore. A questo punto il filesystem può attivare la funzione `open` di quel gestore, passandole come parametro il `minor` che aveva precedentemente trovato, in modo che la funzione `open` sappia quale è il terminale da aprire.

4. Principi di funzionamento di un gestore di periferica

Abbiamo visto come il sistema operativo possa indirizzare le funzioni di un gestore. Vediamo ora alcuni aspetti fondamentali del funzionamento di un gestore sotto LINUX.
Normalmente, dopo l'inizializzazione del sistema, un gestore di periferica viene attivato solamente quando un processo richiede qualche servizio relativo alla periferica. In questo caso la funzione del gestore che viene attivata opera nel contesto del processo che ne ha richiesto il servizio, e quindi l'eventuale trasferimento di dati tra il gestore e il processo non pone particolari problemi. Tuttavia, dato che le periferiche come abbiamo visto operano normalmente tramite interrupt, quando un processo P richiede un servizio a una periferica X, se X non è pronta il processo P viene posto in stato di attesa, e un diverso processo Q viene posto in esecuzione. Sarà l'interrupt della periferica a segnalare il verificarsi dell'evento che porterà il processo P dallo stato di attesa allo stato di pronto. Quindi, *quando*

si verificherà l'interrupt della periferica X il processo in esecuzione sarà Q (o un altro processo subentrato a Q), ma non P, cioè non il processo in attesa dell'interrupt stesso.

Questo aspetto è fondamentale nella scrittura di un gestore di periferica, perché implica che, al verificarsi di un interrupt, i dati che la periferica deve leggere o scrivere non possono essere trasferiti al processo interessato, che non è quello corrente, ma devono essere conservati nel gestore stesso.

Si richiama ora l'esempio già visto trattando il Nucleo del Sistema Operativo, in particolare le funzioni di `wait_event` e `wakeup`: un processo esegue un'operazione di scrittura di N caratteri su una stampante.

Tale scrittura presuppone che il file speciale relativo alla stampante sia stato aperto e che il descrittore di tale file sia già noto al processo.
L'esecuzione dell'operazione è descritta nel seguito con riferimento allo schema di figura 4. Il buffer che compare in tale figura è una zona di memoria appartenente al gestore stesso, non è un buffer generale di sistema. Supponiamo che tale buffer abbia una dimensione di B caratteri. Le operazioni svolte saranno le seguenti:

1. Il processo richiede il servizio tramite una funzione *write( )* e il sistema attiva la routine write del gestore attraverso il major, la tabella dei gestori a caratteri, e la tabella delle operazioni del gestore;
2. la routine write del gestore copia dallo spazio del processo nel buffer del gestore un certo numero C di caratteri; C sarà il minimo tra la dimensione B del buffer e il numero N di caratteri dei quali è richiesta la scrittura;
3. la routine write manda il primo carattere alla periferica con un'istruzione di OUT;
4. a questo punto la routine write non può proseguire, ma deve attendere che la stampante abbia stampato il carattere e sia pronta a ricevere il prossimo; per non bloccare tutto il sistema write pone il processo in attesa invocando la routine sleep_on e passandole un numero di evento E;
5. quando la periferica ha terminato l'operazione, genera un interrupt;
6. la routine di interrupt del gestore viene automaticamente messa in esecuzione; se nel buffer esistono altri caratteri da stampare, la routine di interrupt esegue un'istruzione OUT per inviare il prossimo carattere alla stampante e termina (IRET); altrimenti, il buffer è vuoto e si passa al prossimo punto;
7. la routine di interrupt, dato che il buffer è vuoto, risveglia il processo invocando la funzione wake_up(E), cioè passandole lo stesso identificatore di evento che era stato passato precedentemente alla sleep_on;
8. wake_up ha risvegliato il processo ponendolo in stato di pronto; prima o poi il processo verrà posto in esecuzione e riprenderà dalla routine write che si era sospesa tramite sleep_on; se esistono altri caratteri che devono essere scritti, cioè se N è maggiore del numero di caratteri già trasferiti nel buffer, write copia nuovi caratteri e torna al passo 2, ponendosi in attesa, altrimenti procede al passo 9;
9. prima o poi si deve arrivare a questo passo, cioè i caratteri già trasferiti raggiungono il valore N e quindi il servizio richiesto è stato completamente eseguito; la routine write del gestore esegue il ritorno al processo che la aveva invocata, cioè al modo U.

Lo schema presentato è rudimentale e richiederebbe in realtà dei miglioramenti; ad esempio, per mantenere costantemente in funzione la periferica è opportuno che la routine di interrupt risvegli il processo un po' prima che il buffer sia completamente vuoto.

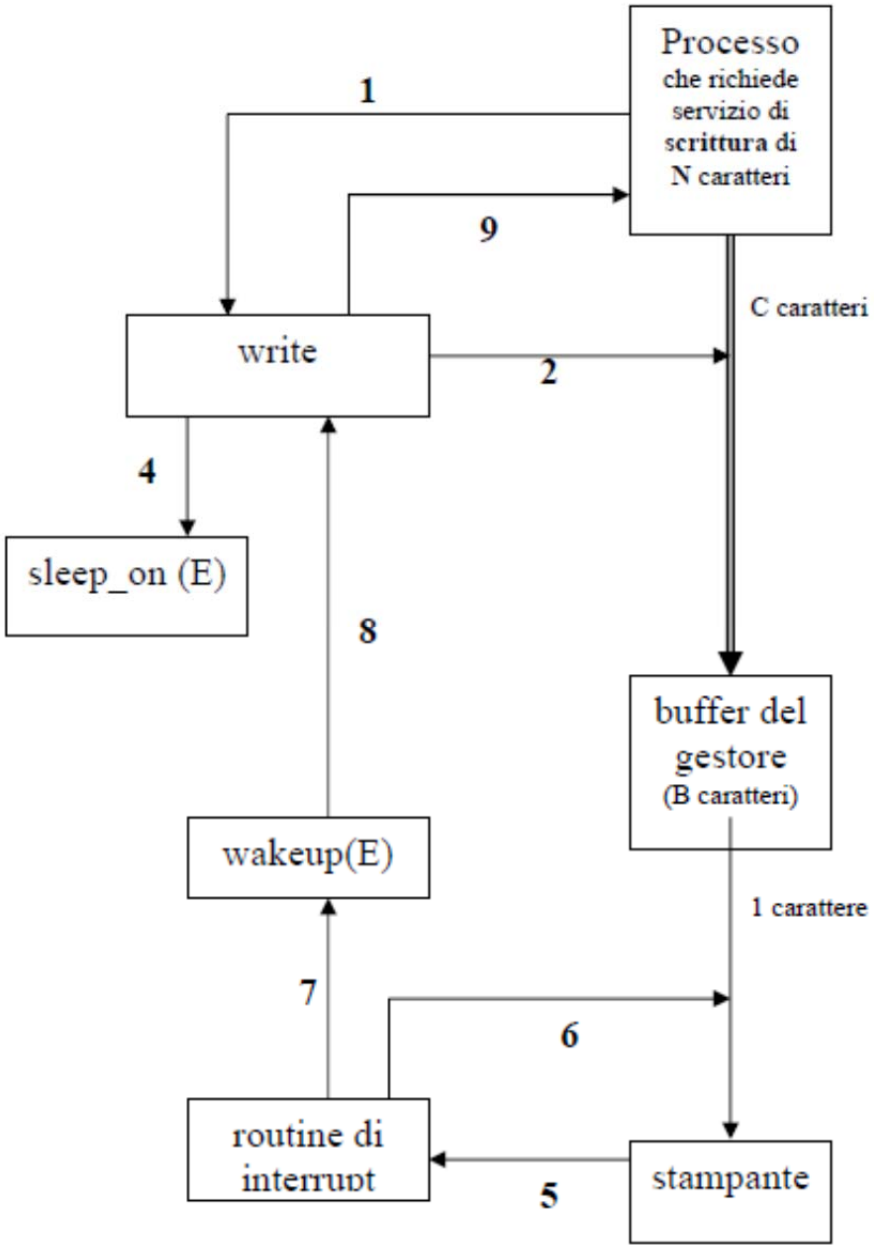

Figura 4 – Esecuzione di un servizio di scrittura

5. Le funzioni di un gestore a carattere

Le principali funzioni di un gestore a carattere sono `open( )`, `release( )`, `read( )`, `write( )` e `ioctl( )`. Vediamo brevemente cosa fanno.

`open`

Le tipiche funzioni di questa routine del gestore, che viene ovviamente invocata quando un processo richiede un servizio `open` relativo ad un file speciale, sono le seguenti:

verificare che la periferica esiste ed è attiva (on line);

se la periferica deve essere utilizzata da un solo processo alla volta, `open` dovrebbe controllare una apposita variabile del gestore: se la variabile indica “periferica aperta”, `open` restituirà un opportuno codice di errore, altrimenti `open` porrà la variabile al valore “periferica aperta”;

`release`

Se la periferica deve essere utilizzata da un solo processo alla volta, la funzione release viene chiamata quando questo esegue una `close` del file speciale, altrimenti solamente quando l’ultimo processo che ha aperto la periferica esegue la `close`. La principale funzione di questa routine è la verifica della terminazione effettiva delle operazioni, cioè che ad esempio non ci siano ancora caratteri da stampare in un buffer; in tal caso tipicamente la `release` pone il processo in attesa di tale terminazione.

`read`

La routine `read`, invocata in conseguenza della richiesta di un servizio `read` sul file speciale, che deve essere già aperto, opera in maniera simmetrica a quanto è stato descritto relativamente alla `write` nell’esempio trattato sopra: `read` deve leggere un carattere dalla periferica se disponibile e poi porre il processo in attesa su un evento opportuno; al risveglio `read` trasferirà i caratteri al processo ed eventualmente, se i caratteri trasferiti non sono sufficienti, tornerà in attesa.
Esercizio: disegnare uno schema simile a quello di figura 4 per la funzione `read` di N caratteri dalla tastiera.

`write`

Il funzionamento della routine `write` è stato già analizzato sopra.

`ioctl`

Questa routine deve permettere di svolgere operazioni speciali su una periferica; le operazioni speciali sono operazioni che hanno senso solamente per quel particolare tipo di periferica, ma non appartengono, a differenza delle operazioni viste sopra, alle operazioni standard svolte su tutte le periferiche (ad esempio, cambiare la velocità di trasmissione di una scheda di rete, oppure riavvolgere un nastro magnetico, ecc…)
La routine `ioctl` di un gestore viene attivata quando un processo invoca il servizio ioctl( ) relativamente al suo file speciale. Il servizio `ioctl` non è stato trattato nella programmazione di sistema, perché molto specificamente rivolto alla gestione delle periferiche. Il suo formato generale è

```
int ioctl(int fd, int comando, int  param)
```

Per quanto riguarda la specifica dei comandi e parametri esistono molte varianti, che dipendono sia dagli standard, sia dalle periferiche. Si rimanda quindi al manuale per i dettagli.
La routine `ioctl` di un gestore riceve dal corrispondente servizio il comando e i parametri. In pratica, è il progettista del gestore a indicare la specifica dei servizi ottenibili tramite il servizio `ioctl`; questo servizio è quindi uno strumento per permettere ad un processo di invocare, attraverso il file system, una routine del gestore passandole dei parametri.

6. Cenni ai gestori a blocchi

Le operazioni fondamentali che il Gestore dei Buffer può richiedere al Gestore della periferica sono la lettura e la scrittura di un certo numero di settori del disco; i parametri fondamentali di tali operazioni sono quindi:

- l’indirizzo del settore iniziale sul disco
- il numero di settori da leggere o scrivere
- l’indirizzo del buffer di memoria per l’operazione

Il meccanismo di richiesta di queste operazioni è meno diretto di quello adottato dai gestori a carattere, perché passa attraverso il Gestore dei Buffer.
Il gestore dei buffer invece di invocare direttamente una operazione dal gestore della periferica *aggiunge una richiesta di operazione ad un’opportuna coda di richieste di operazioni*, e sospende il processo.
La richiesta accodata è una struttura dati che contiene tutti i parametri necessari a svolgere l’operazione:

- l’indirizzo del blocco iniziale su disco

- il numero di byte da trasferire
- l'indirizzo del buffer di sistema da utilizzare
- la direzione del trasferimento

Quando il driver del disco processerà la richiesta utilizzerà questi parametri per inizializzare il DMA opportunamente.
Alla fine dell'operazione l'interrupt del DMA causerà il risveglio del processo.

Un aspetto importante di questo meccanismo indiretto consiste nel fatto che *il gestore della periferica può riorganizzare la coda delle richieste in modo da ottimizzare la sequenza di accesso ai settori del disco*.
L'ordine degli accessi ai blocchi eseguiti dal DMA non coincide quindi con la sequenza di accodamento delle richieste. La modifica dell'ordinamento delle richieste mira ad ottenere la riduzione delle latenze rotazionali necessarie (vedi esempio del capitolo IO1).
Un algoritmo di ottimizzazione adottato spesso può essere assimilato a quello degli ascensori negli edifici molto alti: l'ascensore si muove in una direzione costante, ma si ferma ogni volta che passa da un piano dove c'è una richiesta, anche se tale richiesta è stata attivata dopo altre richieste più lontane.