Struttura interna del sistema operativo Linux

5. I device driver

A cura di:

Anna Antola Giuseppe Pozzi

DEI, Politecnico di Milano

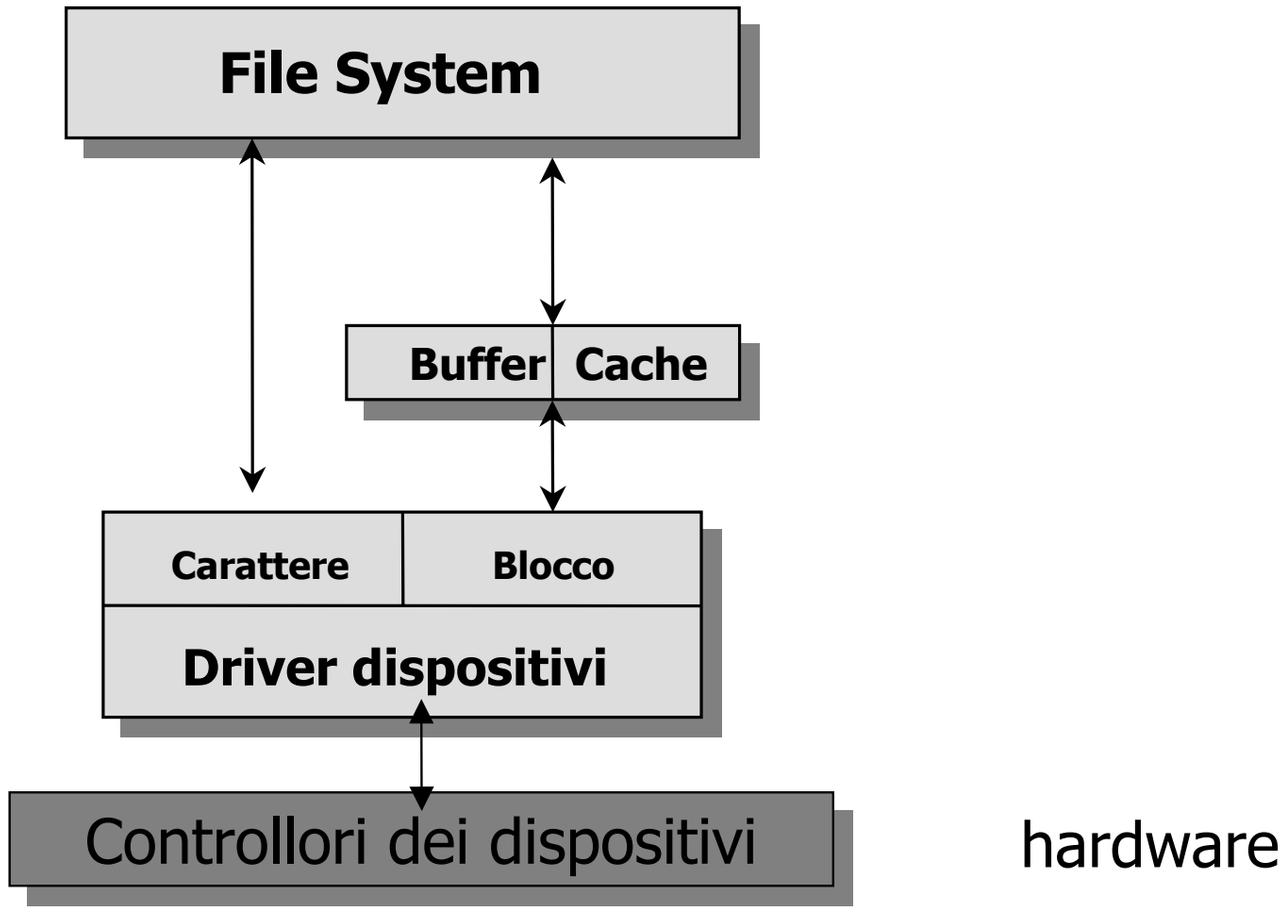
anna.antola/giuseppe.pozzi@polimi.it

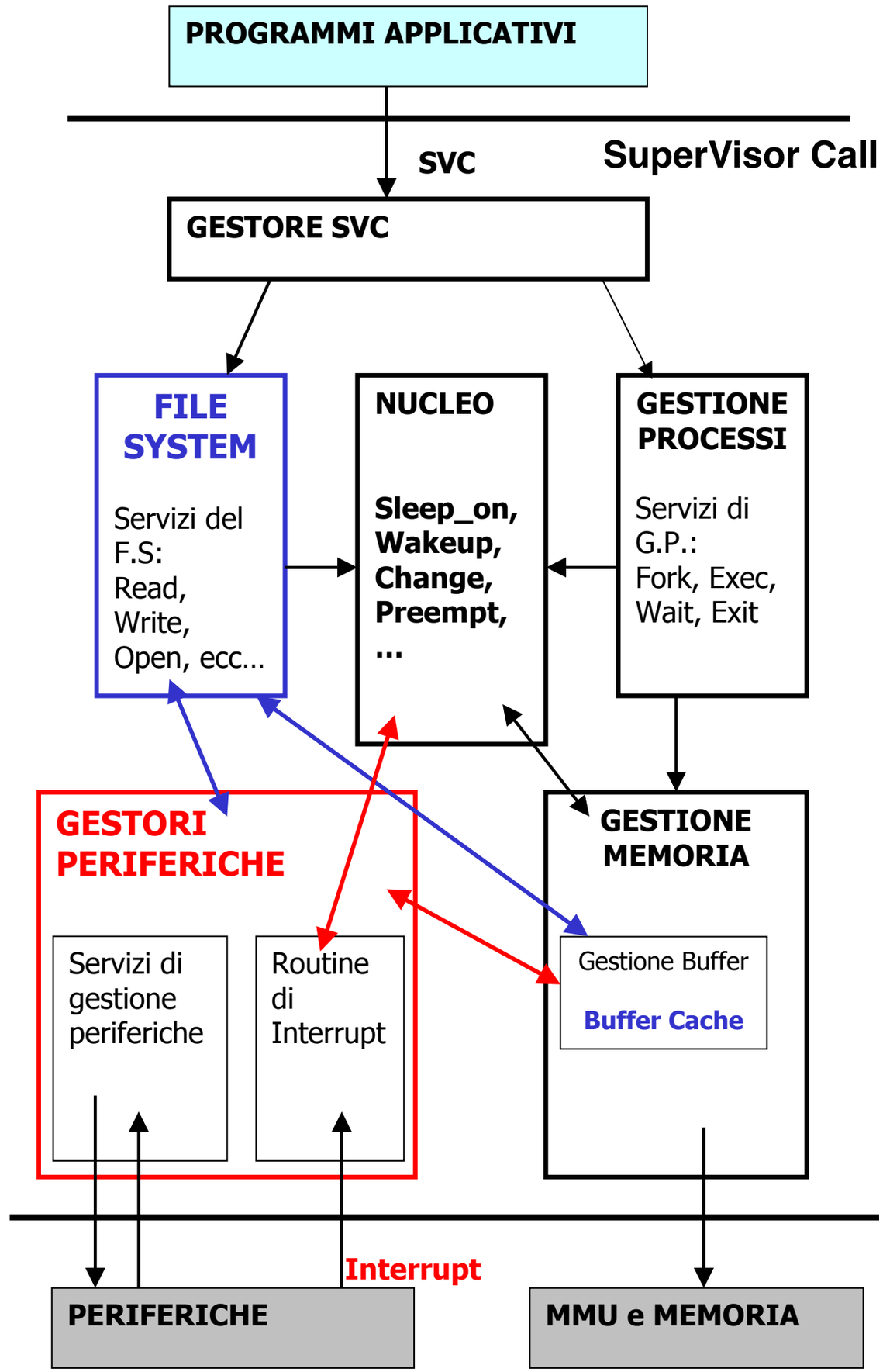
- versione del 30 marzo 2004 -

Device driver - gestori delle periferiche

- Interagiscono con il file system: tutte le periferiche vengono viste come file speciali
- Interagiscono con il nucleo del Sistema Operativo per gestire la sincronizzazione tra la periferica e il calcolatore e il trasferimento dati
- Sono la parte di Sistema Operativo che viene aggiornata con maggior frequenza

Connessioni con il file system




PROGRAMMI APPLICATIVI
SVC
SuperVisor Call
GESTORE SVC
FILE SYSTEM
Servizi del F.S: Read, Write, Open, ecc…
NUCLEO
Sleep_on, Wakeup, Change, Preempt, …
GESTIONE PROCESSI
Servizi di G.P.: Fork, Exec, Wait, Exit
GESTORI PERIFERICHE
Servizi di gestione periferiche
Routine di Interrupt
GESTIONE MEMORIA
Gestione Buffer
Buffer Cache
Interrupt
PERIFERICHE
MMU e MEMORIA

Tipi di periferiche

- In LINUX esiste un driver (gestore) per ogni tipo di periferica
- I tipi di periferiche sono divise in due classi
  - Periferiche a blocchi (block devices)
  - Periferiche a carattere (charcter devices)
    - I dispositivi a blocchi vengono gestiti dal sistema come dispositivi di memorizzazione ad accesso casuale (es. dischi, nastri ....) grazie alla presenza dei buffer che regolano il trasferimento dei dati con il file system. Un blocco può essere indirizzato, indipendentemente dagli altri, per un trasferimento
    - I dispositivi a carattere sono tutti gli altri dispositivi e interagiscono direttamente con il file system, un carattere per volta (es. terminali ....). Il trasferimento dei dati può avvenire a blocchi, ma i blocchi sono significativi solo durante il trasferimento stesso

File speciali e driver (1)

- I file speciali delle periferiche sono memorizzati nel direttorio /dev
- La visualizzazione del direttorio /dev è del tipo

```
brw-------   1 root    system    11,  65   re1a
brw-------   1 root    system    11,  66   re1b
brw-------   1 root    system    11,  67   re1c
crw-rw-rw-   1 root    system    9, 5124   rmt0m
crw-rw-rw-   1 root    system    9, 5120   rmt0l
crw-------   1 root    system    44,   6   rre0g
```

File speciali e driver (2)

- Ogni dispositivo ha associato un file speciale (blocco o carattere) ed è identificato da una coppia di numeri <major, minor>
- I file speciali possono venire creati solo dall'amministratore di sistema (root) tramite la funzione mknod (pathname, type, major, minor)
- L'accesso alle periferiche è attuato tramite le chiamate di accesso ai file (open, close, read, write...)
- Tutte le periferiche dello stesso tipo, cioè gestite dallo stesso driver hanno lo stesso major e quindi condividono gli stessi servizi
- L'esecuzione del servizio richiesto è parametrizzata tramite il minor

Struttura del driver (1)

- Le principali funzioni di un driver di periferica sono
  - inizializzazione del dispositivo alla partenza del Sistema Operativo e gestione dello stato della periferica (in servizio/fuori servizio)
  - ricezione e/o trasmissione dati dalla periferica
  - gestione degli errori
  - gestione degli interrupt da periferica
- Ogni driver può essere visto come costituito da
  - una **routine di inizializzazione** che esegue delle operazioni di inizializzazione del driver
  - un **insieme di routine** che costituiscono i **servizi eseguibili** e implementati per quel tipo di periferica
  - la **routine di risposta all'interrupt** attivata dall'interrupt della periferica, il cui indirizzo viene inserito nel corrispondente vettore di interrupt

Struttura del driver (2)

- Ogni driver ha associata una "tabella", realizzata tramite la *struct file_operations*, che contiene i puntatori alle routine di servizio del driver stesso

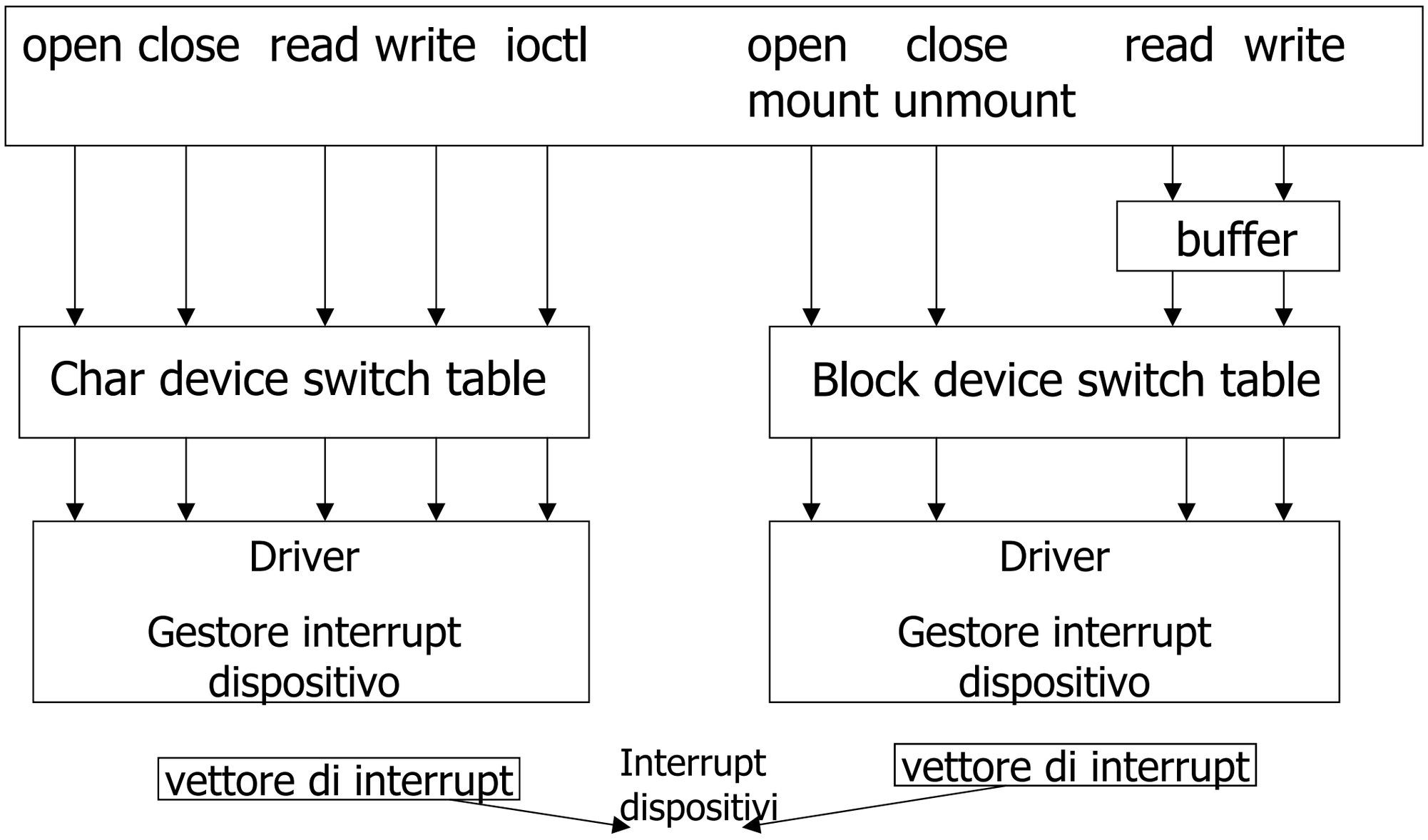
```
struct file_operations {
        int (*lseek) ( );
        int (*read) ( );
        int (*write) ( );
        ...
        int (*ioctl) ( );
        ...
        int (*open) ( );
        void (*release) ( );
        ...}
```

La funzione di inizializzazione di ogni driver di periferica, al termine dell'operazione, restituisce al S.O. (nucleo) un puntatore alla propria tabella delle operazioni

Strutture dati del nucleo per i driver

- Interfaccia tra sistema operativo e driver è descritta da due tabelle:
  - Block device switch table – tabella driver per i dispositivi a blocchi
  - Character device switch table – tabella driver per i dispositivi a carattere
- Ogni tipo di dispositivo ha una riga, nella tabella appropriata, che indirizza al driver corrispondente (contiene il puntatore della tabella delle operazioni passato al termine dell'inizializzazione)

Driver di periferica



ioctl = input/output control

Driver di periferica

DESCRIPTION top

The **ioctl**() function manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with **ioctl**() requests. The argument *d* must be an open file descriptor.

The second argument is a device-dependent request code. The third argument is an untyped pointer to memory. It's traditionally **char** **argp* (from the days before **void** * was valid C), and will be so named for this discussion.

An **ioctl**() *request* has encoded in it whether the argument is an *in* parameter or *out* parameter, and the size of the argument *argp* in bytes. Macros and defines used in specifying an **ioctl**() *request* are located in the file *<sys/ioctl.h>*.

ioctl = input/output control

Chiamate di sistema e driver

- Le chiamate di sistema fanno riferimento a un descrittore di file (o al nome) per il quale, attraverso la tabelle dei file aperti si identifica il corrispondente i-node.
- L'i-node identifica il tipo di file speciale e indirizza alla riga della tabella dei driver per dispositivi a blocchi o alla tabella per dispositivi a carattere identificandone la riga attraverso un numero contenuto nell'i-node (major number)
- Il servizio richiesto identifica la colonna della tabella che contiene l'indirizzo della routine del driver che svolge questo servizio a cui viene passato come parametro il numero di identificazione univoca del dispositivo (minor number), anch'esso contenuto nell'i-node

Indirizzamento di una routine di servizio di un driver

A livello di processo: invocazione di open (/dev/tty1, …)
che si traduce in una **SVC**

A livello di S.O.:
- attivazione di open ( ) nel File System
- ricerca tramite il nome del file dell'i-node corrispondente

i-node del file speciale /dev/tty1
major=4, minor=1

Tabella dei driver a carattere

0	
1	
2	
3	
4	
5	
…	
…	

Tabella delle operazioni del driver 4

0	1	2	…	…	…
lseek	read	write	…	open	

Funzione open del driver 4

- lancio in esecuzione della routine open con il minor =1

Principi di funzionamento per driver a carattere

Scrittura e lettura

- nel caso di periferiche gestite a interrupt, l'interruzione si verifica nel contesto di un processo diverso da quello che ha invocato il servizio della periferica
- le routine del driver possono memorizzare temporaneamente i dati che devono inviare (o devono ricevere) alla periferica in un buffer del driver allocato appositamente

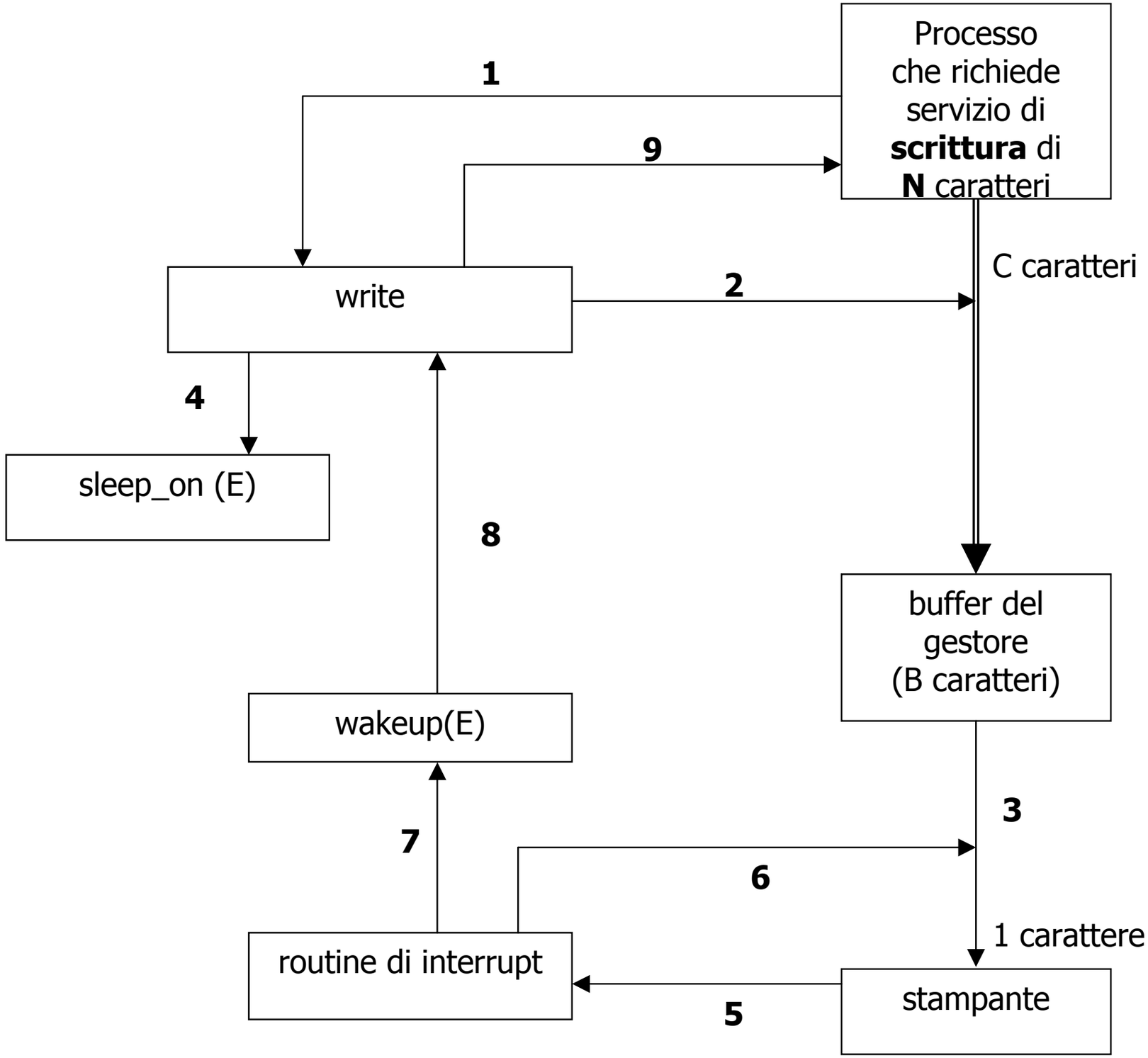
Esempio di scrittura (1)

- Il processo richiede un servizio di scrittura di *N* caratteri tramite una *write ( ).* Viene attivato il S.O. che attraverso il *major*, associato al dispositivo stampante, attiva la ***routine write*** del driver specificando il *minor*
- La routine write del driver copia i dati dallo spazio utente nel buffer del driver. I dati copiabili sono pari alla dimensione B del buffer
- La routine write esegue la OUT del primo carattere da buffer a stampante
- La routine write invoca la Sleep_on per sospendere il processo in attesa che la stampante sia pronta a ricevere il prossimo carattere da stampare. Un nuovo processo viene mandato in esecuzione
- La stampante genera un interrupt, segnalando che è pronta a ricevere un nuovo dato
- Viene attivata la routine di risposta all'interrupt associata al driver della stampante che, se nel buffer esistono altri caratteri da stampare, esegue una nuova OUT e termina. La routine di risposta all'interrupt viene attivata tante volte quanti sono i caratteri presenti nel buffer da stampare

Esempio di scrittura (2)

- Quando il buffer è vuoto, la routine di risposta all'interrupt invoca la funzione wakeup che risveglia il processo nella routine write. Se esistono altri caratteri da stampare (N>B), la routine write copia i rimanenti caratteri nel buffer del driver ed esegue la prima OUT e pone il attesa il processo tramite una nuova Sleep_on
- I caratteri vengono trasferiti dal buffer alla stampante tramite l'attivazione della routine di risposta all'interrupt, come visto prima. Quando il buffer è vuoto, la routine di risposta all'interrupt invoca la funzione wakeup che risveglia il processo nella routine write. Se i caratteri da stampare sono terminati, la routine write esegue il ritorno al modo U del processo

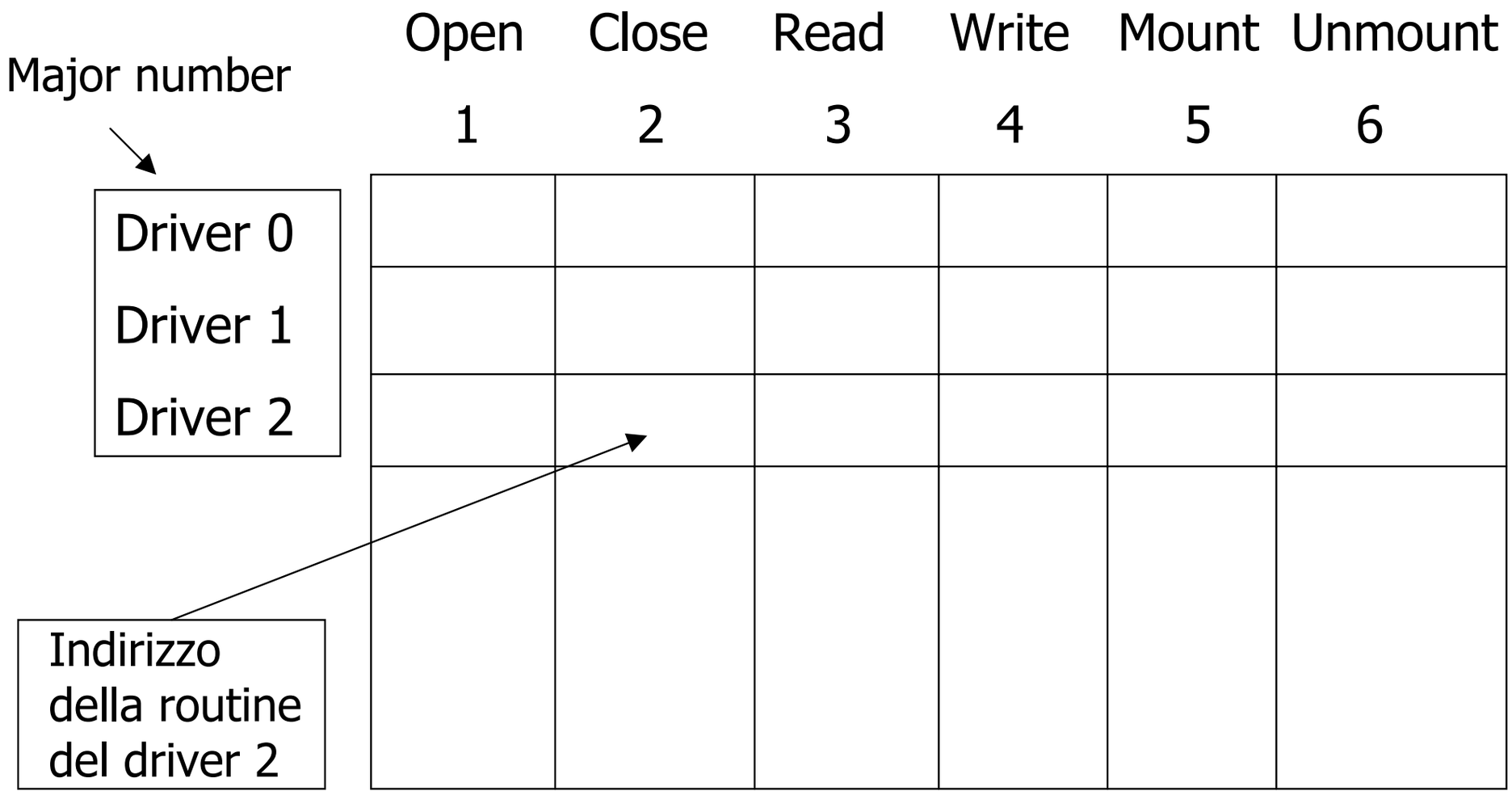
Scrittura


Processo che richiede servizio di **scrittura** di **N** caratteri
1
9
write
2
C caratteri
4
sleep_on (E)
8
buffer del gestore (B caratteri)
wakeup(E)
3
7
6
routine di interrupt
1 carattere
5
stampante

Driver dei dispositivi a blocchi

- Nei driver dei dispositivi a blocchi, la lettura da disco di un blocco viene invocata dal gestore dei buffer di sistema (*buffer cache*) in funzione delle richieste del file system
- Il gestore dei buffer di sistema richiede al driver del disco la lettura o scrittura di un certo numero di settori del disco (corrispondenti ad un blocco). Il gestore dei buffer deve quindi fornire al driver del disco:
  - l'indirizzo del settore iniziale su disco
  - il numero di settori da trasferire
  - l'indirizzo del buffer di sistema per l'operazione
- Tali parametri costituiscono l'inizializzazione del DMA per il trasferimento di quel particolare blocco. Il gestore dei buffer accoda le richieste e procede nell'elaborazione
- L'interrupt di fine DMA viene utilizzato dal gestore dei buffer per considerare conclusa un'operazione di trasferimento di un blocco

Tabella driver dispositivi a blocchi



Sintassi mknod

- Per creare un file speciale per il dispositivo /dev/ra2 a blocchi, con il major uguale a 1 ed il minor uguale a 2:
  - mknod /dev/ra2 b 1 2

    b=block device, c=character device, p=pipe file
- Per creare un file speciale per il dispositivo /dev/ttya2 a caratteri, con il major uguale a 2 ed il minor uguale a 1:
  - mknod /dev/ttya2 c 2 1

    b=block device, c=character device, p=pipe file

komodo - Como

```
NAME
  mknod - Creates a special file

SYNOPSIS

  /usr/sbin/mknod special file [ b major device# minor device# |  c
  major device# minor device#]

  /usr/sbin/mknod filename p

DESCRIPTION

  The mknod command makes a directory entry.  The first argument is the name
  of the special device file.  Select a name that is descriptive of the dev-
  ice.

  The mknod command has two forms.  In the first form, the second argument is
  the b or c flag.  The last two arguments are numbers specifying the
  major device, which helps the operating system find the device driver code,
  and the minor device, the unit drive, or line number, which may be either
  decimal or octal.

  The assignment of major device numbers is specific to each system.  You can
  determine the device numbers by examining the conf.c system source file.
  If you change the contents of the conf.c file to add a device driver, you
  must rebuild the kernel.

  In the second form of mknod, you use the p flag to create named pipes
  (FIFOs).

  Only the superuser can create a character or device special file.

FLAGS

  b         Indicates that the special file corresponds to a block-oriented
            device (disk or tape)

  c         Indicates that the special file corresponds to a character-
            oriented device

  p         Creates named pipes (FIFOs)

/usr1/users/docenti/pozzi>
```