

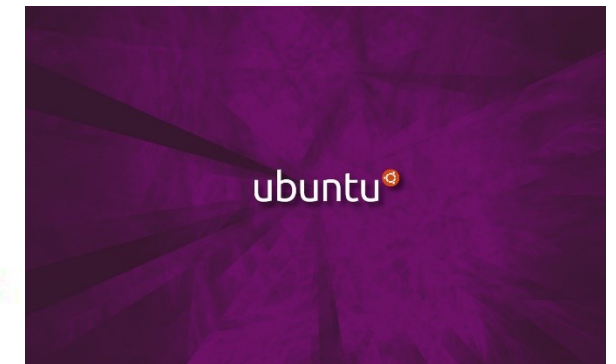
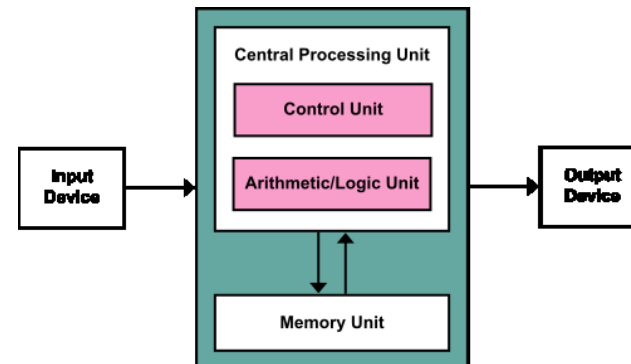
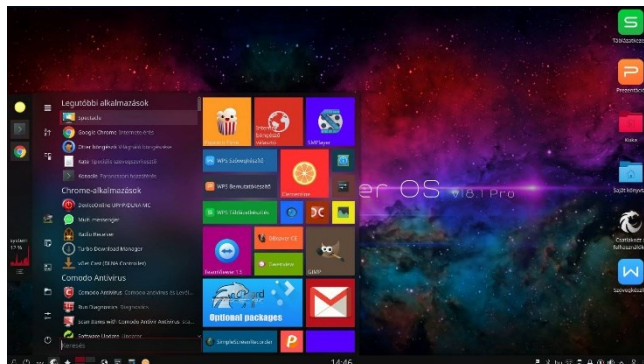
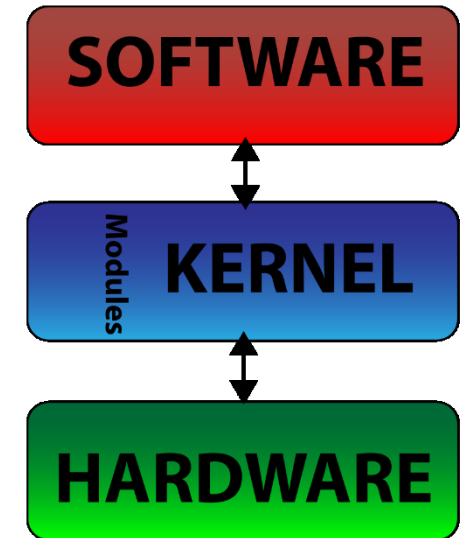
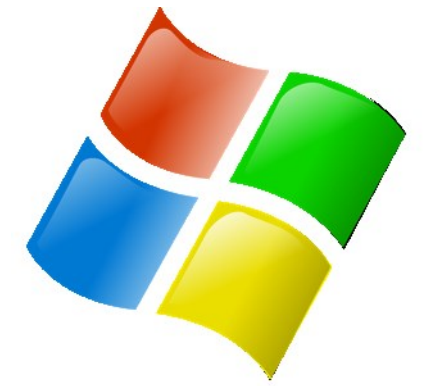


**UNIVERSITÀ DEGLI STUDI  
DELLA BASILICATA**

*Corso di Sistemi Operativi*

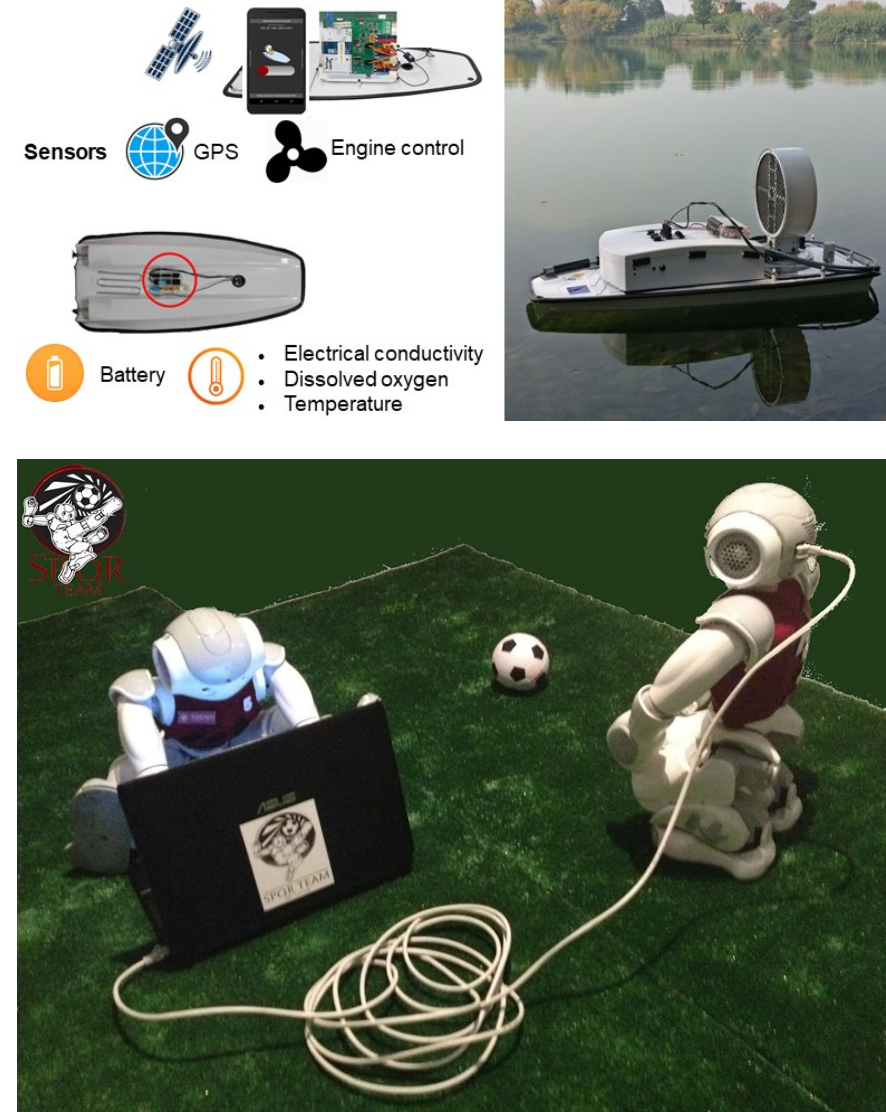
# Sistemi di I/O

Docente:  
**Domenico Daniele  
Bloisi**



# Domenico Daniele Bloisi

- Ricercatore RTD B  
Dipartimento di Matematica, Informatica  
ed Economia  
Università degli studi della Basilicata  
<http://web.unibas.it/bloisi>
- SPQR Robot Soccer Team  
Dipartimento di Informatica, Automatica  
e Gestionale Università degli studi di  
Roma “La Sapienza”  
<http://spqr.diag.uniroma1.it>



# Informazioni sul corso

---

- Home page del corso:  
<http://web.unibas.it/bloisi/corsi/sistemi-operativi.html>
- Docente: Domenico Daniele Bloisi
- Periodo: I semestre ottobre 2020 – febbraio 2021
  - Lunedì 15:00-17:00
  - Martedì 9:30-11:30



**Le lezioni saranno erogate in modalità esclusivamente on-line**

Codice corso Google Classroom:

<https://classroom.google.com/c/MTQ2ODE2NTk3ODIz?cjc=67646ik>

# Ricevimento

---

- Su appuntamento tramite Google Meet

Per prenotare un appuntamento inviare  
una email a

[domenico.bloisi@unibas.it](mailto:domenico.bloisi@unibas.it)



# Programma – Sistemi Operativi

---

- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

# Sistema operativo e I/O

---

Il ruolo di un sistema operativo nell'I/O è quello di **gestire e controllare** le operazioni e i dispositivi di I/O

# Sottosistema di I/O

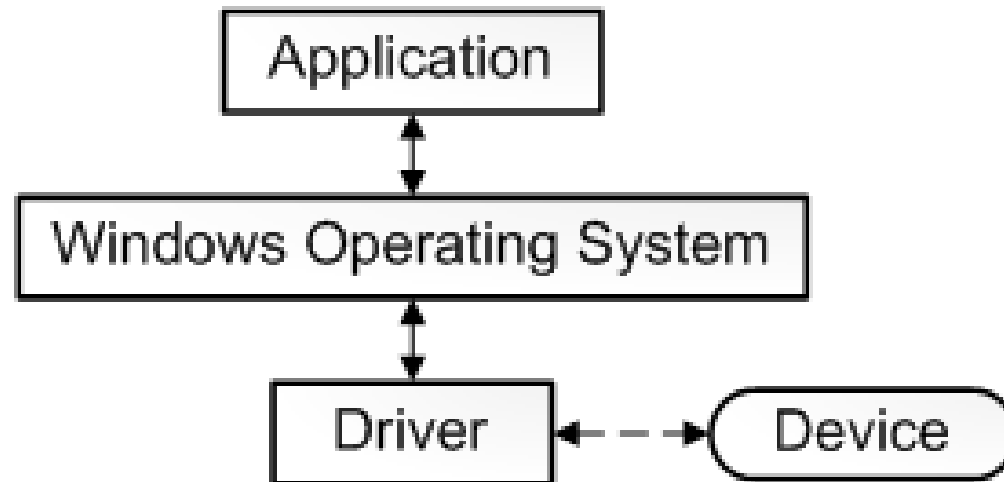
---

- I dispositivi di I/O possono essere molto diversi per funzioni e velocità → necessitano di diversi sistemi di controllo
- Il sottosistema di I/O del kernel separa il resto del kernel dalla complessità di gestione dei dispositivi di I/O

# Driver di dispositivo

---

I driver dei dispositivi offrono al sottosistema di I/O una interfaccia uniforme per l'accesso ai dispositivi di I/O





# Hardware di I/O

Se più dispositivi condividono un insieme di fili, la connessione è detta *bus*.

Un **bus** è un insieme di fili e un protocollo rigorosamente definito che specifica l'insieme dei messaggi che si possono inviare attraverso i fili.

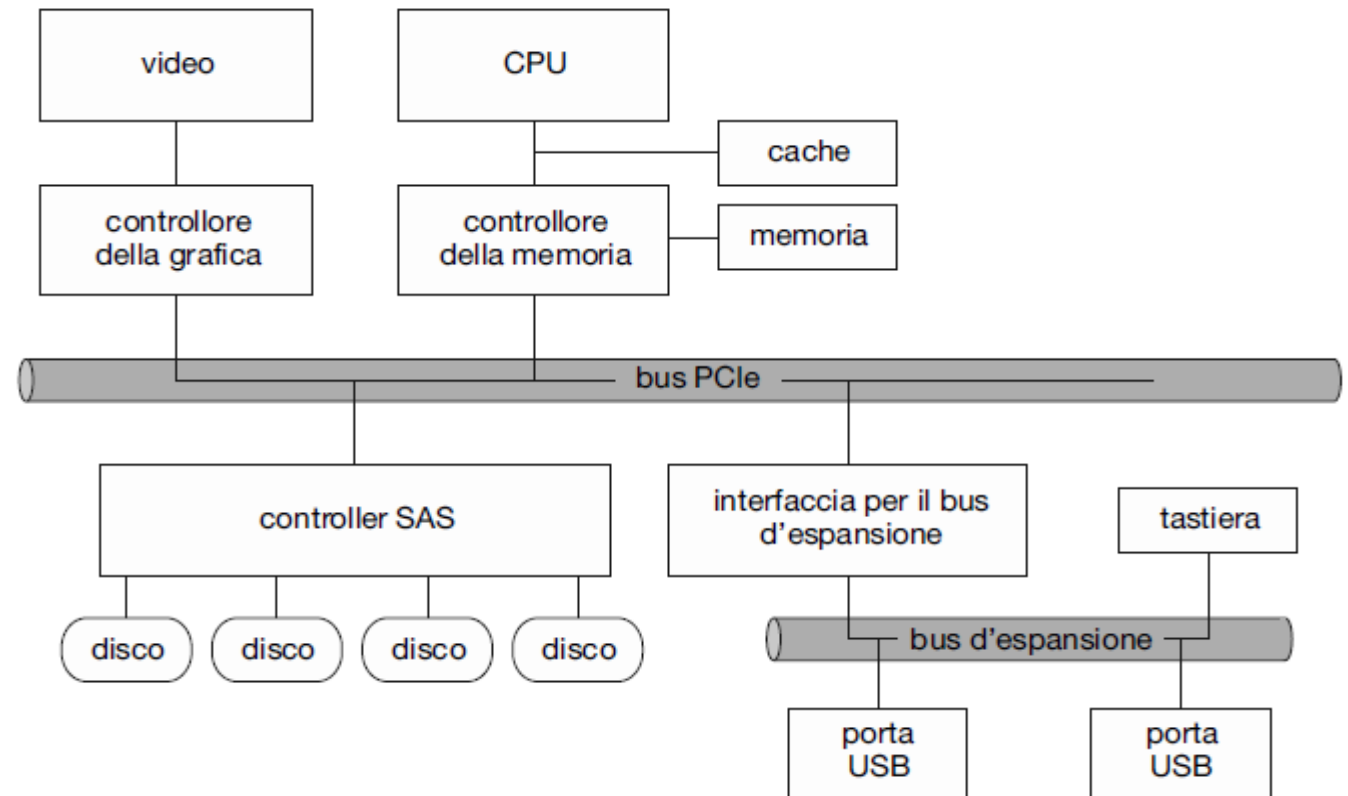


Figura 12.1 Tipica struttura del bus di un PC.

# Memory mapped I/O

---

Il controllore di dispositivo può supportare l'**I/O memory mapped** (*I/O mappato in memoria*).

- I registri di controllo del dispositivo sono mappati in un sottoinsieme dello spazio di indirizzi della CPU
- La CPU esegue le richieste di I/O leggendo e scrivendo i registri di controllo del dispositivo alle locazioni di memoria fisica a cui sono mappati

# Memory mapped I/O

In passato, i PC usavano spesso istruzioni di I/O per controllare alcuni dispositivi e l'I/O memory mapped per controllarne altri.

indirizzi per l'I/O (in esadecimale)	dispositivo
000-00F	controllore DMA
020-021	controllore delle interruzioni
040-043	timer
200-20F	controllore dei giochi
2F8-2FF	porta seriale (secondaria)
320-32F	controllore del disco
378-37F	porta parallela
3D0-3DF	controllore della grafica
3F0-3F7	controllore dell'unità a dischetti
3F8-3FF	porta seriale (principale)

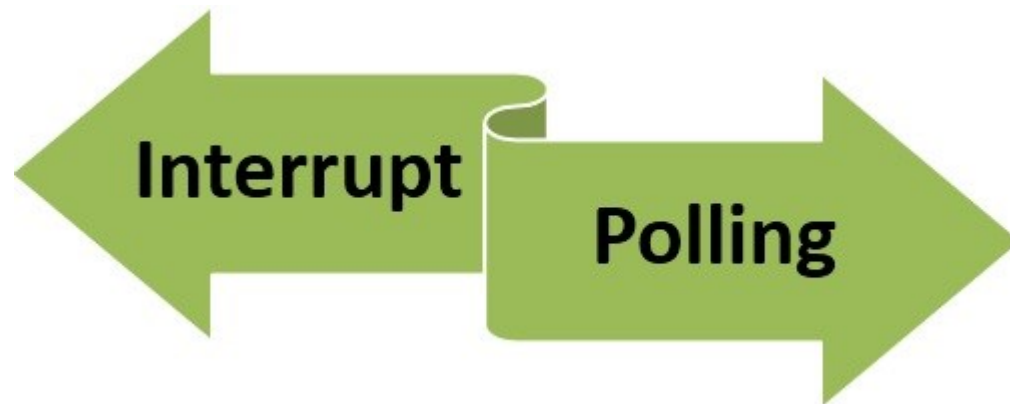
**Figura 12.2** Indirizzi delle porte dei dispositivi di I/O nei PC (elenco parziale).

# Polling vs. interrupt

---

Interrupt e polling sono le due modalità con cui gli eventi generati dai dispositivi connessi al PC possono essere gestiti dalla CPU

- Nella gestione con **polling**, la CPU tiene traccia delle comunicazioni dei dispositivi di I/O a intervalli regolari
- Nella gestione con **interrupt**, il dispositivo di I/O interrompe la CPU comunicando ad essa che ha bisogno di andare in esecuzione



# Interruzioni

Le **interruzioni** sono usate diffusamente dai sistemi operativi moderni per gestire **eventi asincroni** e per eseguire procedure in **modalità supervisore** nel kernel.

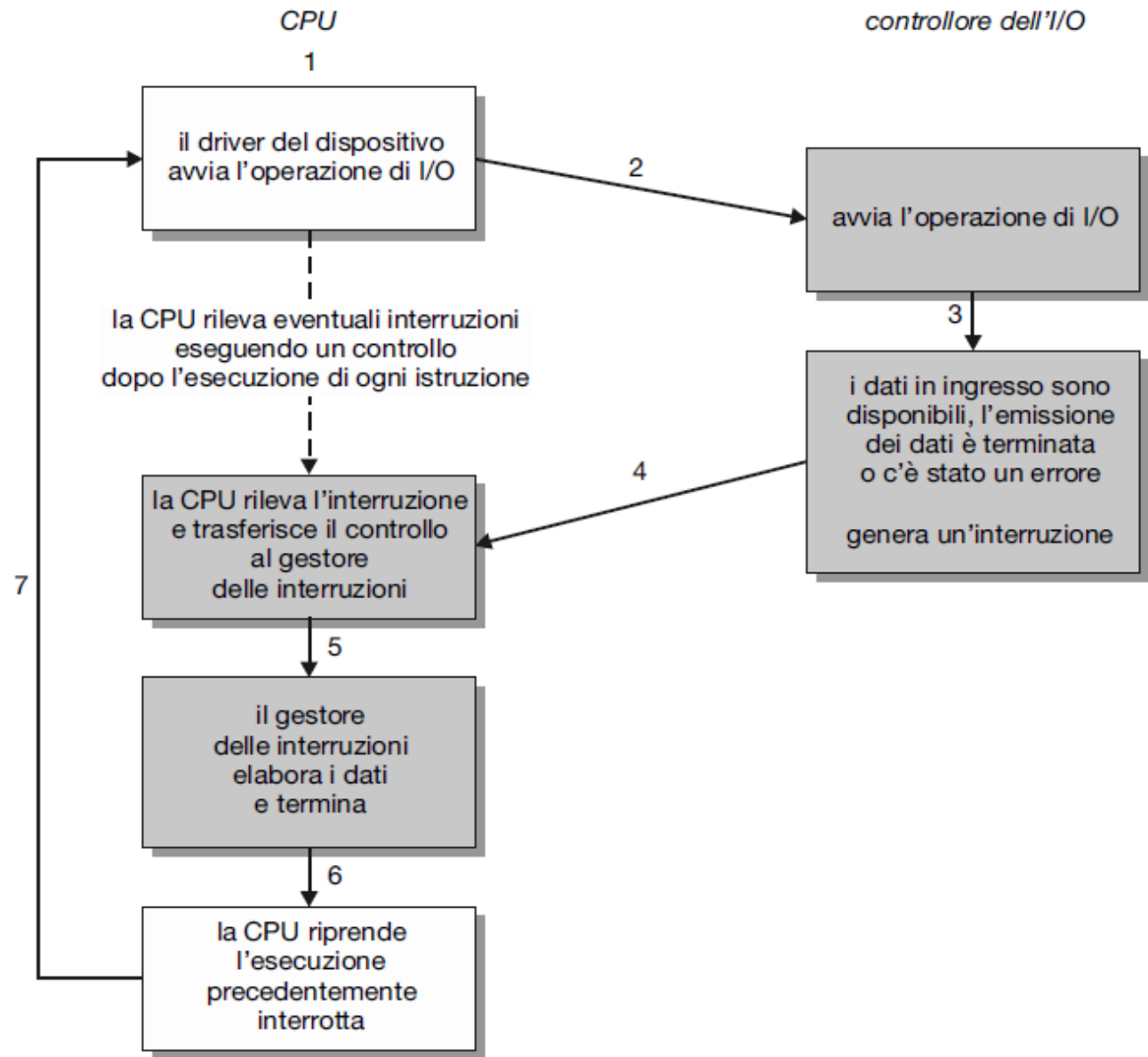


Figura 12.3 Ciclo di I/O basato sulle interruzioni.

# Interruzioni

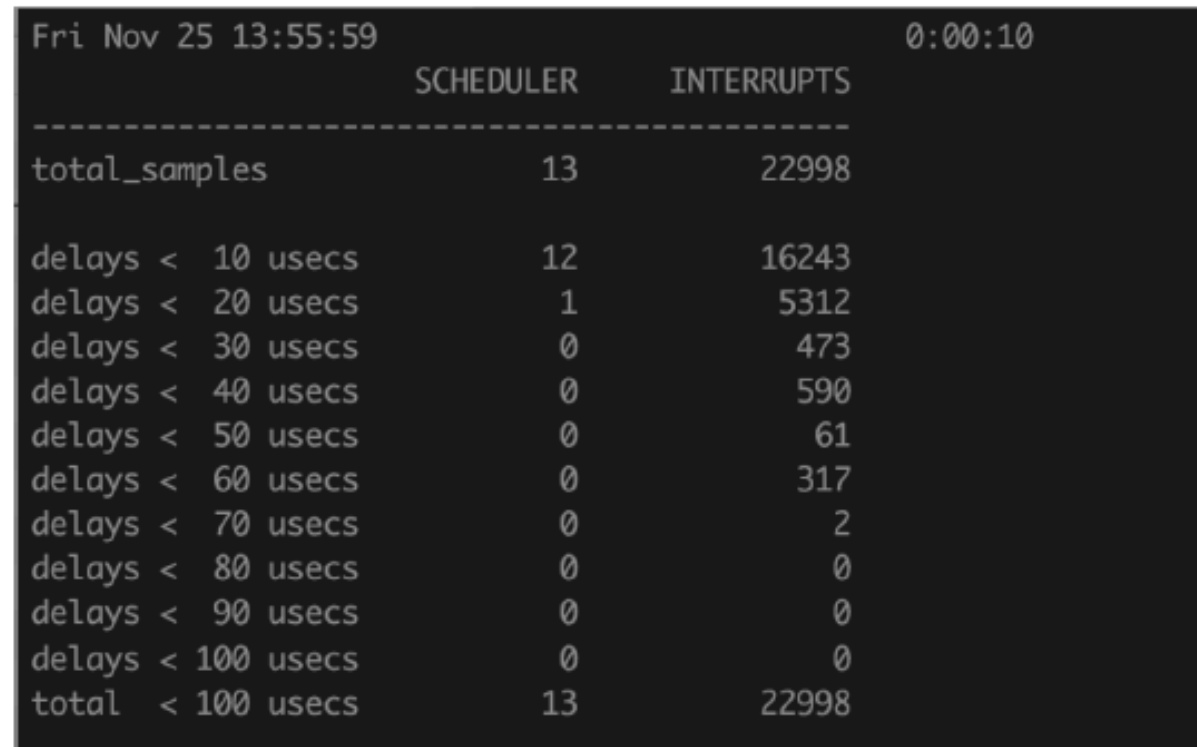
---

- Per far sì che i compiti più urgenti siano portati a termine per primi, i calcolatori moderni usano un **sistema di priorità delle interruzioni**.
- I controllori dei dispositivi, i guasti hardware e le chiamate di sistema generano **interruzioni** al fine di innescare l'esecuzione di procedure del kernel.
- Poiché le **interruzioni** sono usate in modo massiccio per affrontare situazioni in cui il tempo è un fattore critico, è necessario avere un'efficiente gestione delle interruzioni per ottenere buone prestazioni del sistema.

# Interruzioni

Anche i moderni sistemi monoutente gestiscono **centinaia di interruzioni al secondo** e i server ne gestiscono persino **centinaia di migliaia al secondo**

La schermata mostra l'output del comando **latency** su **macOS**, rivelando che in dieci secondi un computer desktop senza particolari carichi di lavoro ha eseguito quasi **23.000 interrupt**.



```
Fri Nov 25 13:55:59                                0:00:10
                                SCHEDULER      INTERRUPTS
-----
total_samples                    13             22998
delays < 10 usecs                12             16243
delays < 20 usecs                 1              5312
delays < 30 usecs                 0               473
delays < 40 usecs                 0               590
delays < 50 usecs                 0                61
delays < 60 usecs                 0               317
delays < 70 usecs                 0                 2
delays < 80 usecs                 0                 0
delays < 90 usecs                 0                 0
delays < 100 usecs                0                 0
total < 100 usecs                13             22998
```

	SCHEDULER	INTERRUPTS
total_samples	13	22998
delays < 10 usecs	12	16243
delays < 20 usecs	1	5312
delays < 30 usecs	0	473
delays < 40 usecs	0	590
delays < 50 usecs	0	61
delays < 60 usecs	0	317
delays < 70 usecs	0	2
delays < 80 usecs	0	0
delays < 90 usecs	0	0
delays < 100 usecs	0	0
total < 100 usecs	13	22998

Figura 12.4 Il comando **latency** di Mac OS X.

# Interruzioni mascherabili e non mascherabili

Gli eventi da **0 a 31**, non mascherabili, si usano per segnalare varie condizioni d'errore; quelli **dal 32 al 255**, mascherabili, si usano, per esempio, per le interruzioni generate dai dispositivi → **livelli di priorità delle interruzioni**

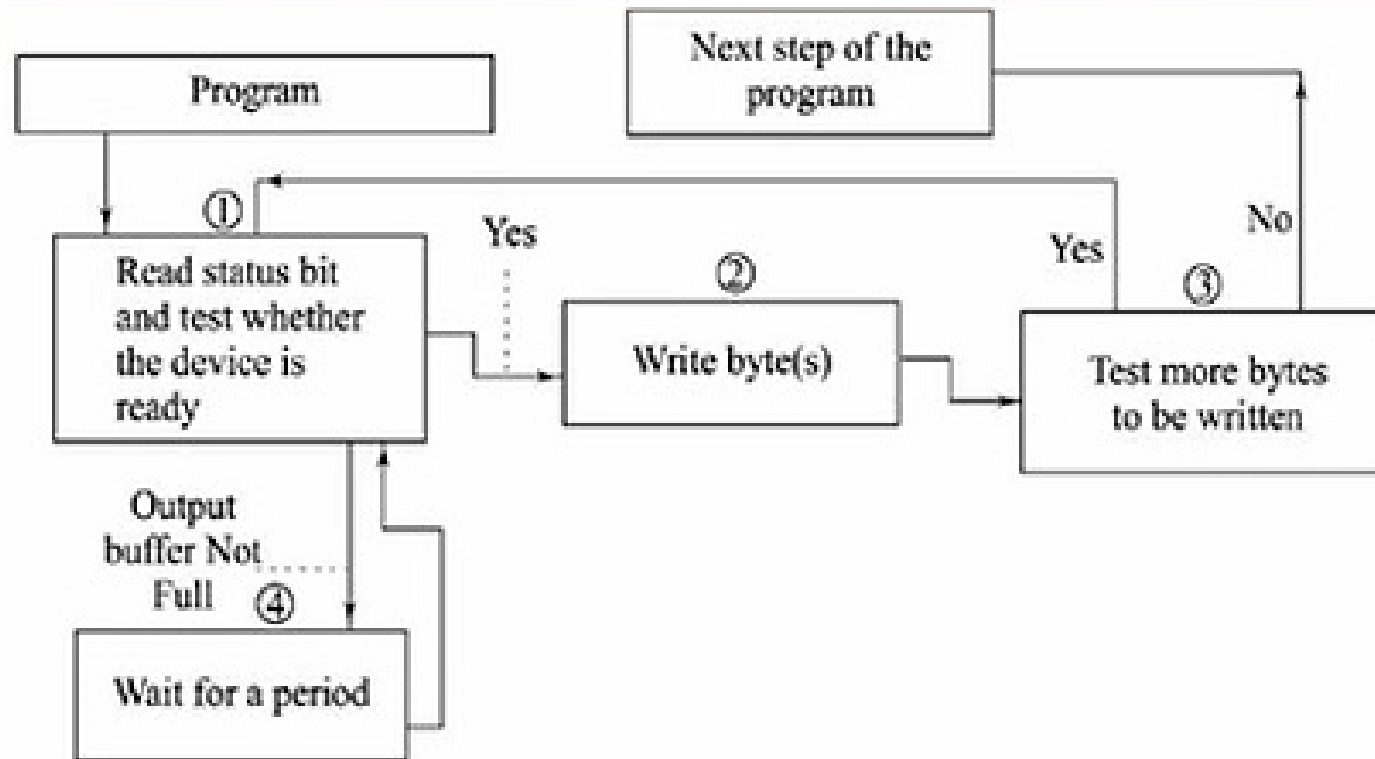
indice del vettore	descrizione
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19-31	(Intel reserved, do not use)
32-255	maskable interrupts

Figura 12.5 Vettore delle interruzioni della CPU Intel Pentium.



# Programmed I/O (PIO)

Nell'I/O programmato, la **CPU scrive i dati** nel registro del controllore di dispositivo un byte alla volta

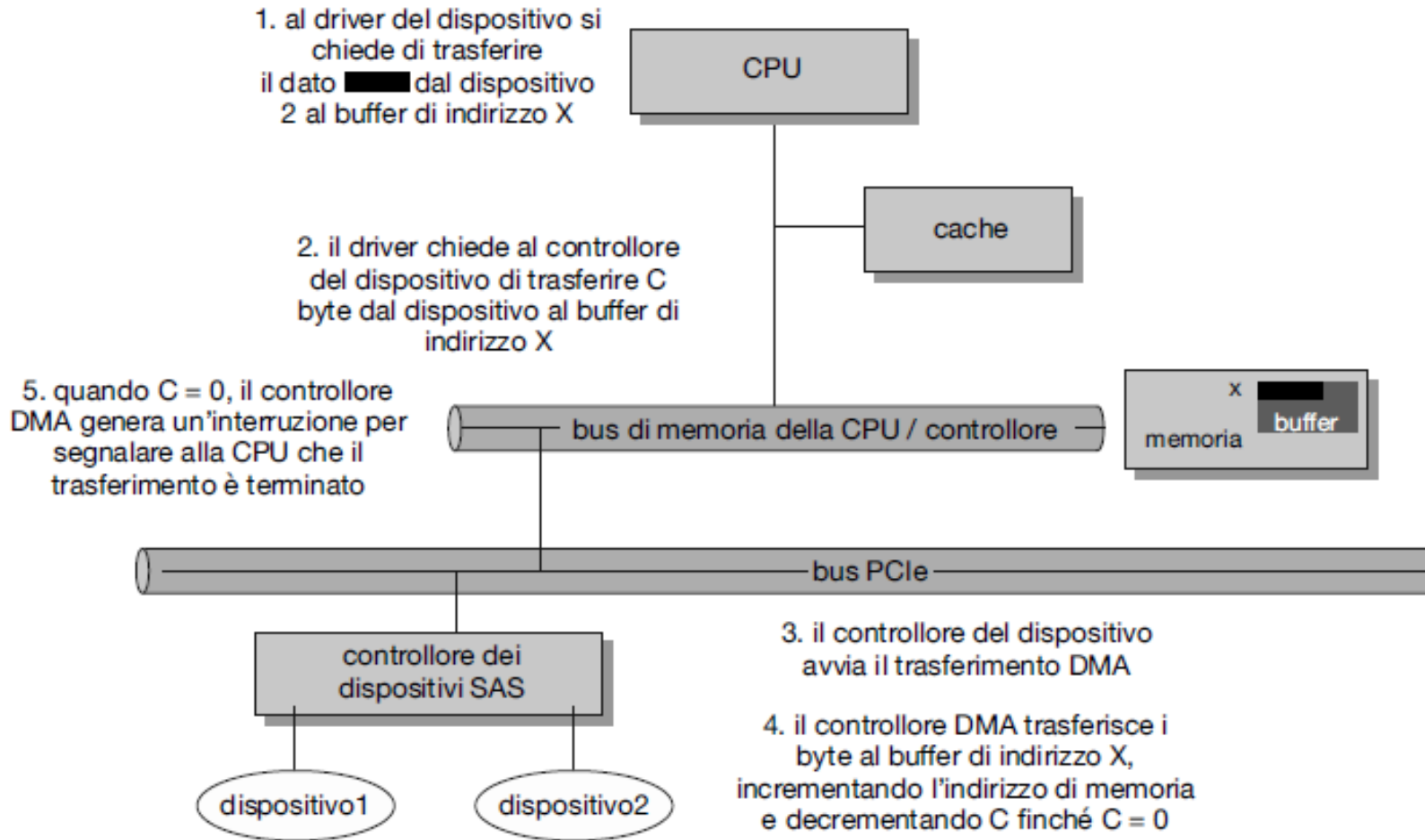


# Svantaggi del PIO

---

- Per il trasferimento di grandi quantità di dati tale tecnica risulta inefficiente poiché **può sovraccaricare la CPU**
- Per evitare di sovraccaricare la CPU, si assegnano i compiti di trasferimento dati a un processore specializzato, detto controllore dell'accesso diretto in memoria

# Direct memory access (DMA)



**Figura 12.6** Passi di un trasferimento DMA.

# Riassumendo

---

bus

controllore

porta di I/O e suoi  
registri

procedura di  
handshaking tra la  
CPU e il controllore  
di un dispositivo

esecuzione  
dell'handshaking per  
mezzo del polling o  
delle interruzioni

delega dell'I/O a un  
controllore DMA nel  
caso di trasferimenti  
di grandi quantità di  
dati.

# Interfaccia di I/O delle applicazioni

La figura a lato  
illustra la **divisione in  
strati software** di  
quelle parti del kernel  
che riguardano la  
gestione dell'I/O

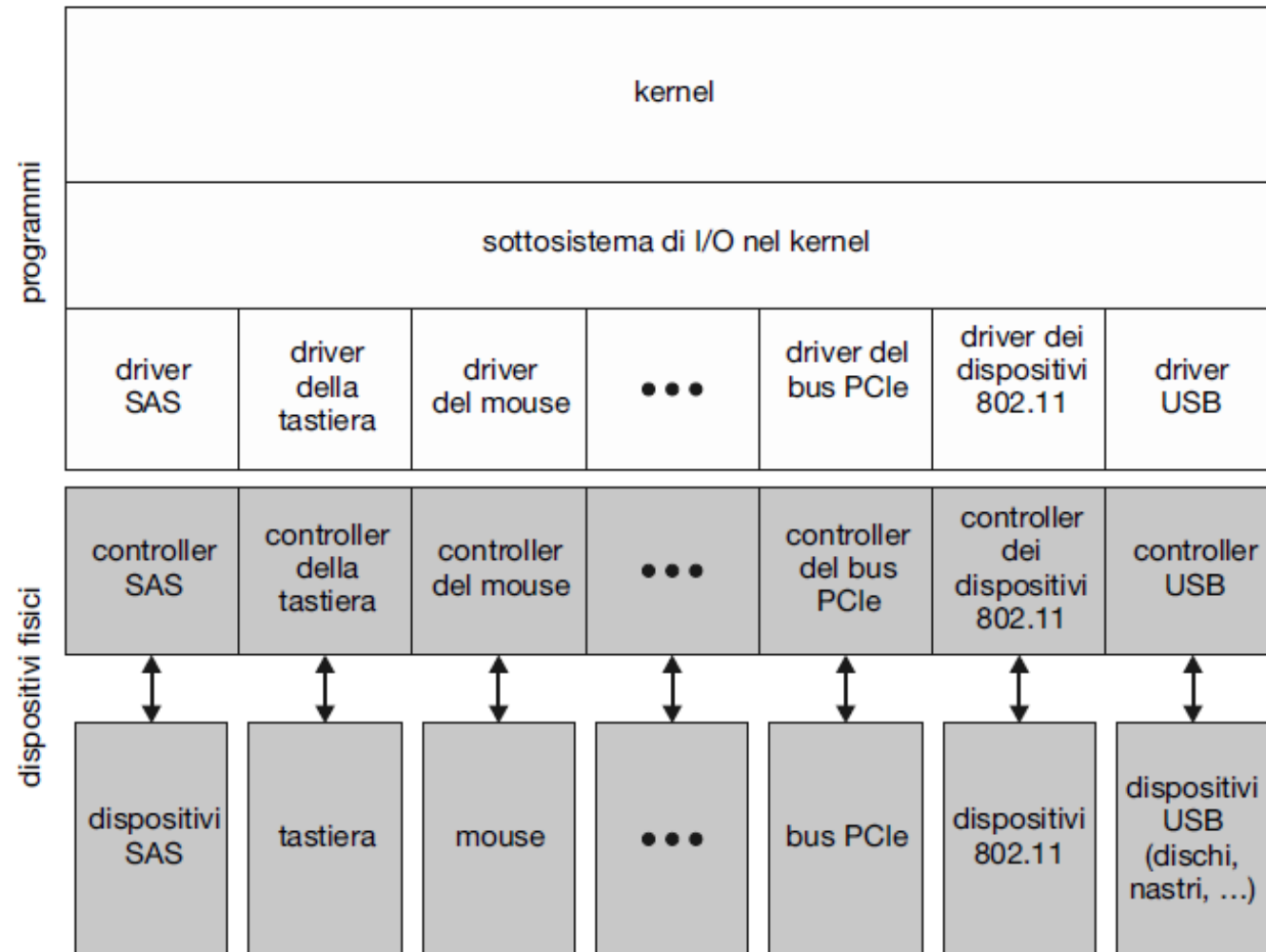


Figura 12.7 Struttura relativa all'I/O nel kernel.

# Trasferimento a flusso di caratteri

---

Trasferimento a  
flusso di  
caratteri o a  
blocchi

Chiamate di sistema:

`get ()` → acquisisce un carattere

`put ()` → invia un carattere

La tastiera è un esempio di dispositivo al quale si accede tramite una interfaccia a flusso di caratteri.

Altri esempi sono stampanti e schede audio

# Dispositivo sincrono

---

Dispositivi  
sincroni o  
asincroni

Un dispositivo sincrono trasferisce dati con un tempo di risposta prevedibile, in maniera coordinata rispetto al resto del sistema

Esempi di comunicazione sincrona:

- videoconferenza
- telefonata

# Dispositivo asincrono

---

Dispositivi  
sincroni o  
asincroni

Un dispositivo asincrono ha tempi di risposta irregolari o non prevedibili, non coordinati con altri eventi del computer

Esempi di comunicazione asincrona:

- email
- chat



# Dispositivi sequenziali

---

Dispositivi  
sequenziali o ad  
accesso diretto

Un dispositivo sequenziale trasferisce dati secondo un ordine fisso dipendente dal dispositivo

Esempio di dispositivo sequenziale:  
CPU → esegue una sequenza di operazioni,  
una alla volta, in successione.  
La CPU è anche un dispositivo di tipo  
sincrono (usa un clock per gestire la  
sincronizzazione)

# Dispositivi ad accesso diretto

---

Dispositivi  
sequenziali o ad  
accesso diretto

L'utente di un dispositivo ad accesso diretto può richiedere l'accesso a una qualunque delle possibili locazioni di memorizzazione

Esempi di dispositivi ad accesso diretto:

- CD
- HDD
- USB flash drive

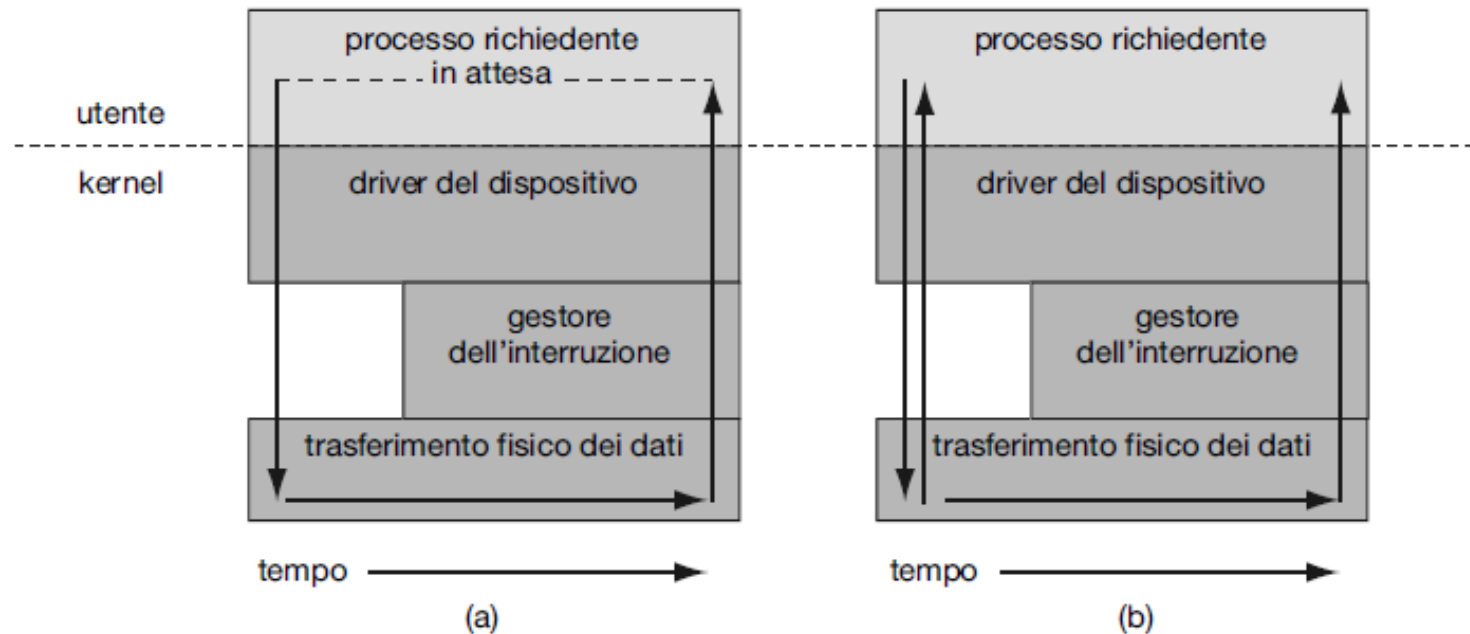
# Interfaccia di I/O delle applicazioni

aspetto	variazione	esempio
modalità di trasferimento dei dati	a caratteri a blocchi	terminale unità a disco
modalità d'accesso	sequenziale casuale	modem lettore di CD-ROM
prevedibilità dell'I/O	sincrono asincrono	unità a nastro tastiera
condivisione	dedicato condiviso	unità a nastro tastiera
velocità	latenza tempo di ricerca velocità di trasferimento attesa fra le operazioni	
direzione dell'I/O	solo lettura solo scrittura lettura e scrittura	lettore di CD-ROM controllore della grafica unità a disco

**Figura 12.8** Caratteristiche dei dispositivi per l'I/O.

# I/O sincrono/asincrono

Una possibile alternativa alle **chiamate di sistema non bloccanti** è costituita dalle **chiamate di sistema asincrone**. Esse restituiscono immediatamente il controllo al chiamante, senza attendere che l'I/O sia stato completato.



**Figura 12.9** Due metodi per l'I/O; (a) sincrono e (b) asincrono.

# Callback

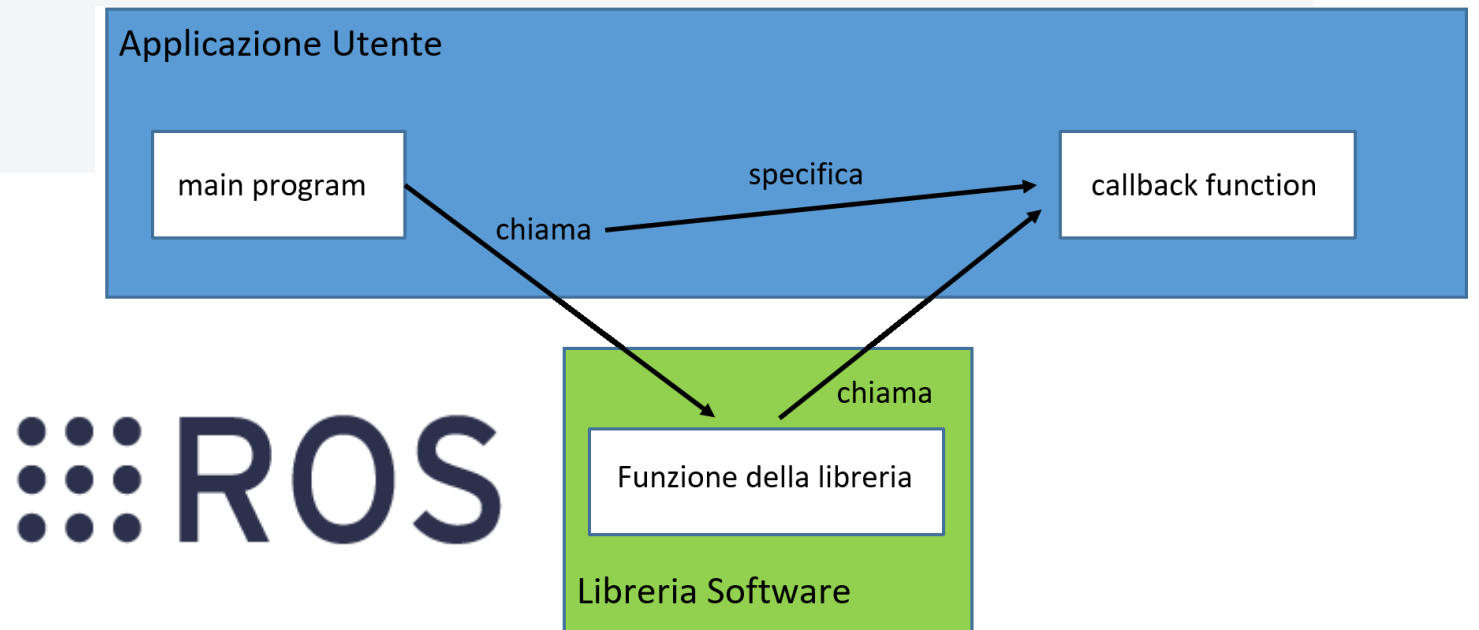
---

Nelle **chiamate di sistema asincrone** l'applicazione continua ad essere eseguita e il completamento dell'I/O è successivamente comunicato all'applicazione:

- Per mezzo dell'impostazione del valore di una variabile nello spazio di indirizzi dell'applicazione
- Tramite un interrupt software
- Tramite una callback eseguita fuori del normale flusso lineare di elaborazione dell'applicazione

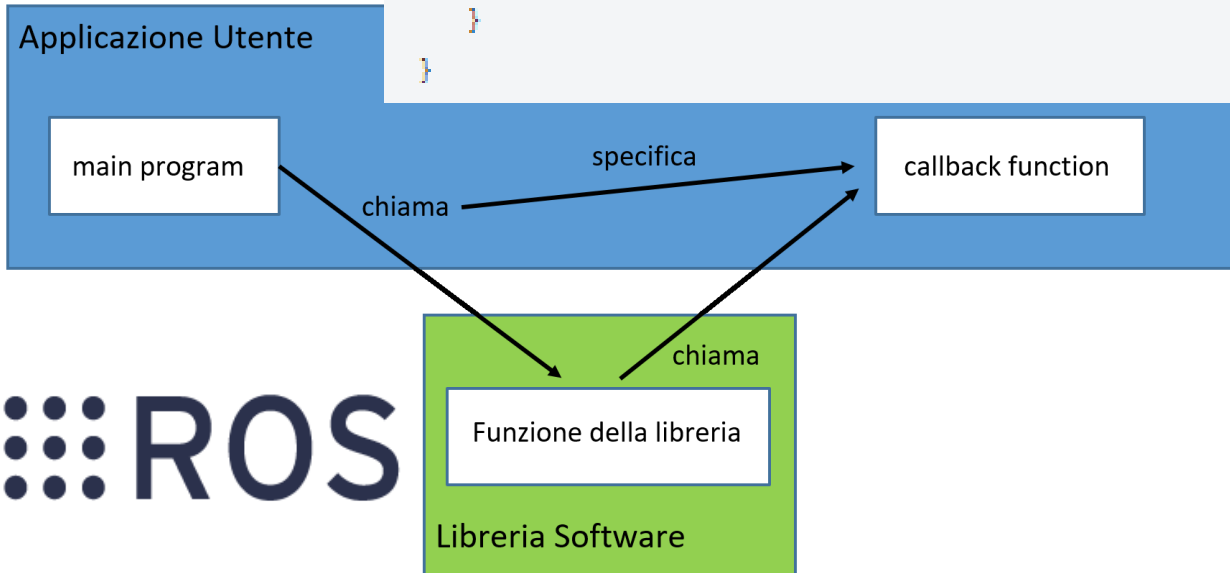
# Callback: esempio

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "image_listener");
    ros::NodeHandle nh;
    cv::namedWindow("view");
    cv::startWindowThread();
    image_transport::ImageTransport it(nh);
    image_transport::Subscriber sub = it.subscribe("camera/image", 1, imageCallback);
    ros::spin();
    cv::destroyWindow("view");
}
```



# Callback: esempio

```
void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
    try
    {
        cv::imshow("view", cv_bridge::toCvShare(msg, "bgr8")->image);
        cv::waitKey(30);
    }
    catch (cv_bridge::Exception& e)
    {
        ROS_ERROR("Could not convert from '%s' to 'bgr8'.", msg->encoding.c_str());
    }
}
```



# Callback: esempio

---

- Lato Applicazione Utente viene effettuata una chiamata di funzione al metodo “subscribe” che avvia un thread di ascolto sul topic prescelto `it.subscribe("camera/image", 1, imageCallback);`
- Poi viene eseguita la funzione “spin” che fa entrare l’Applicazione Utente in uno stato di attesa indefinito, fino al richiamo della callback `ros::spin();`
- Lato Libreria Software verrà invocata la callback nel momento in cui arriverà un nuovo messaggio sul topic.



# Sottosistema di I/O del kernel

---

Il kernel fornisce molti **servizi riguardanti l'I/O**; i seguenti servizi sono offerti dal **sottosistema di I/O del kernel** e sono realizzati a partire dai dispositivi e dai relativi driver.

scheduling

gestione del  
buffer

gestione delle  
cache

gestione delle  
code di spooling

riservazione dei  
dispositivi

gestione degli  
errori →  
protezione  
dell'I/O

# Tabella dello stato dei dispositivi

Gli elementi della **tabella dello stato dei dispositivi** – uno per ogni dispositivo di I/O – indicano il *tipo*, l'*indirizzo* e lo *stato del dispositivo*: non funzionante, inattivo o occupato.

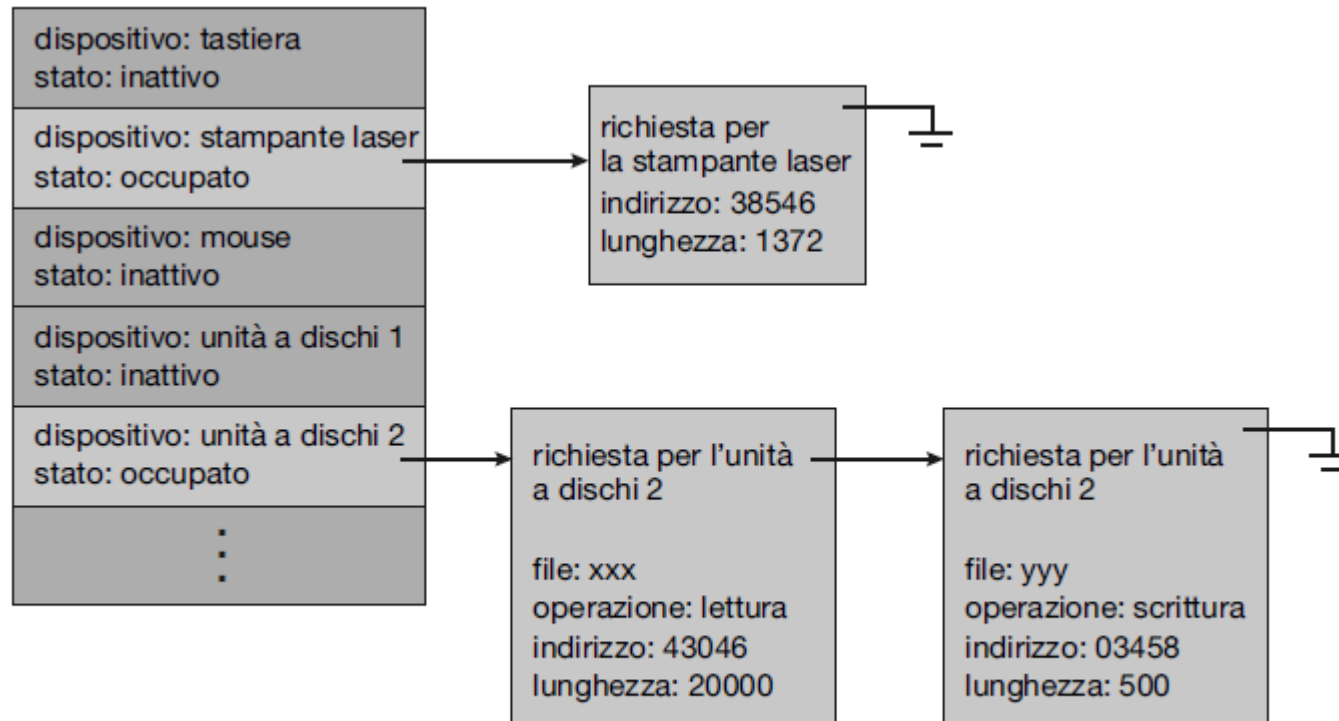


Figura 12.10 Tabella dello stato dei dispositivi.

# Velocità dell'interfaccia

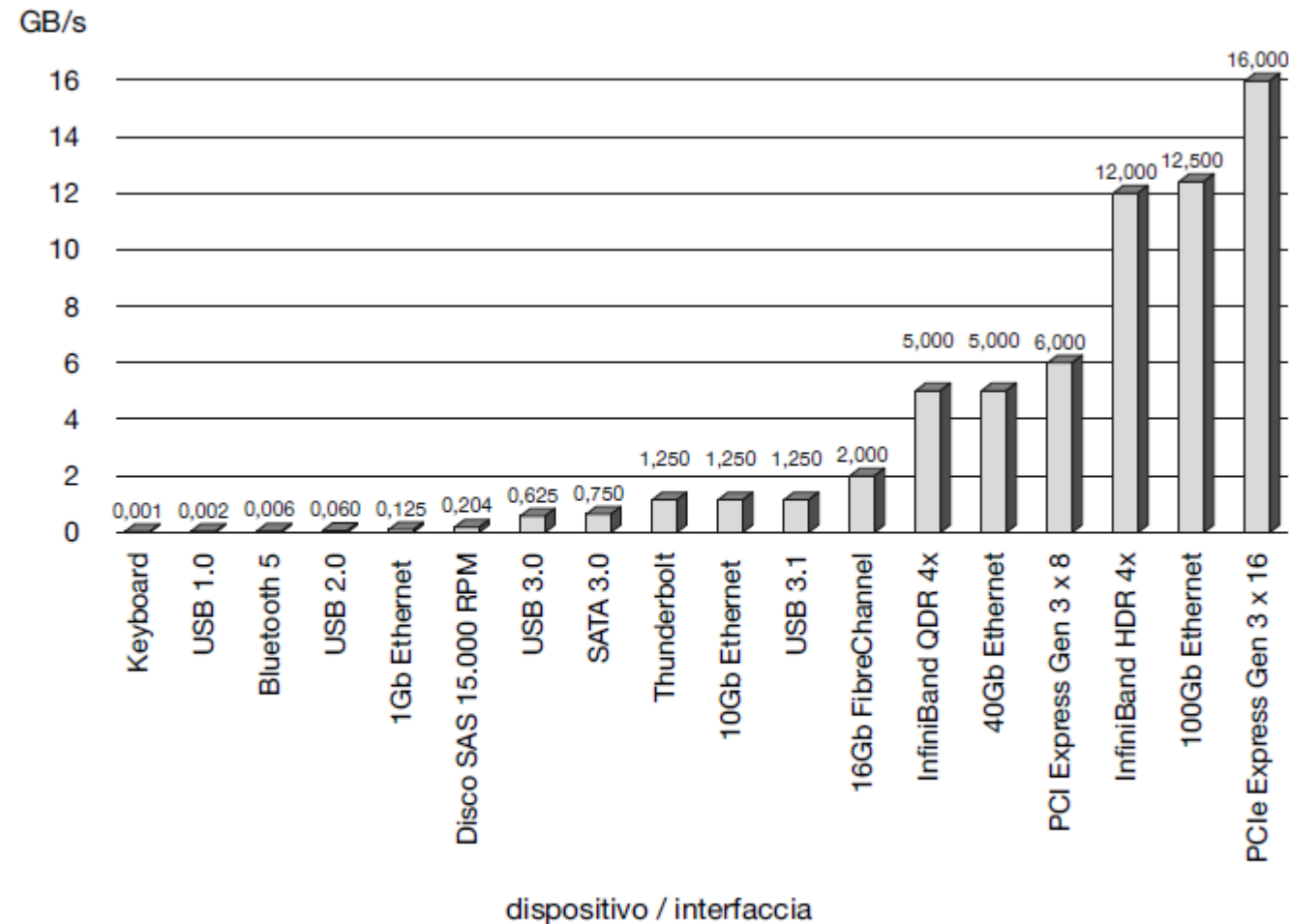


Figura 12.11 Dispositivi di I/O utilizzati in PC e data center e velocità dell'interfaccia.

# Protezione dell'I/O

Un programma utente, per eseguire l'I/O, invoca una chiamata di sistema per chiedere al sistema operativo di svolgere una data operazione nel suo interesse

Il sistema, passando alla modalità privilegiata, verifica che la richiesta sia valida e, in tal caso, esegue l'operazione; esso trasferisce quindi il controllo all'utente.

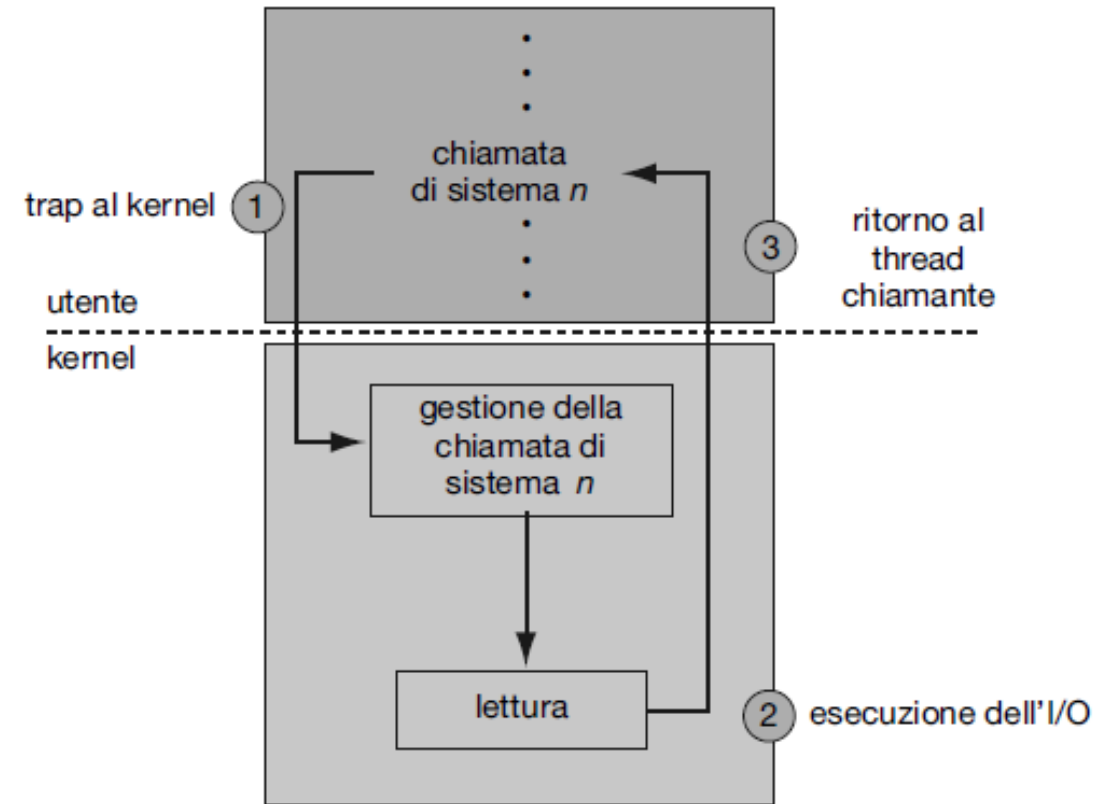


Figura 12.12 Uso delle chiamate di sistema per eseguire I/O.

# Strutture dati del kernel

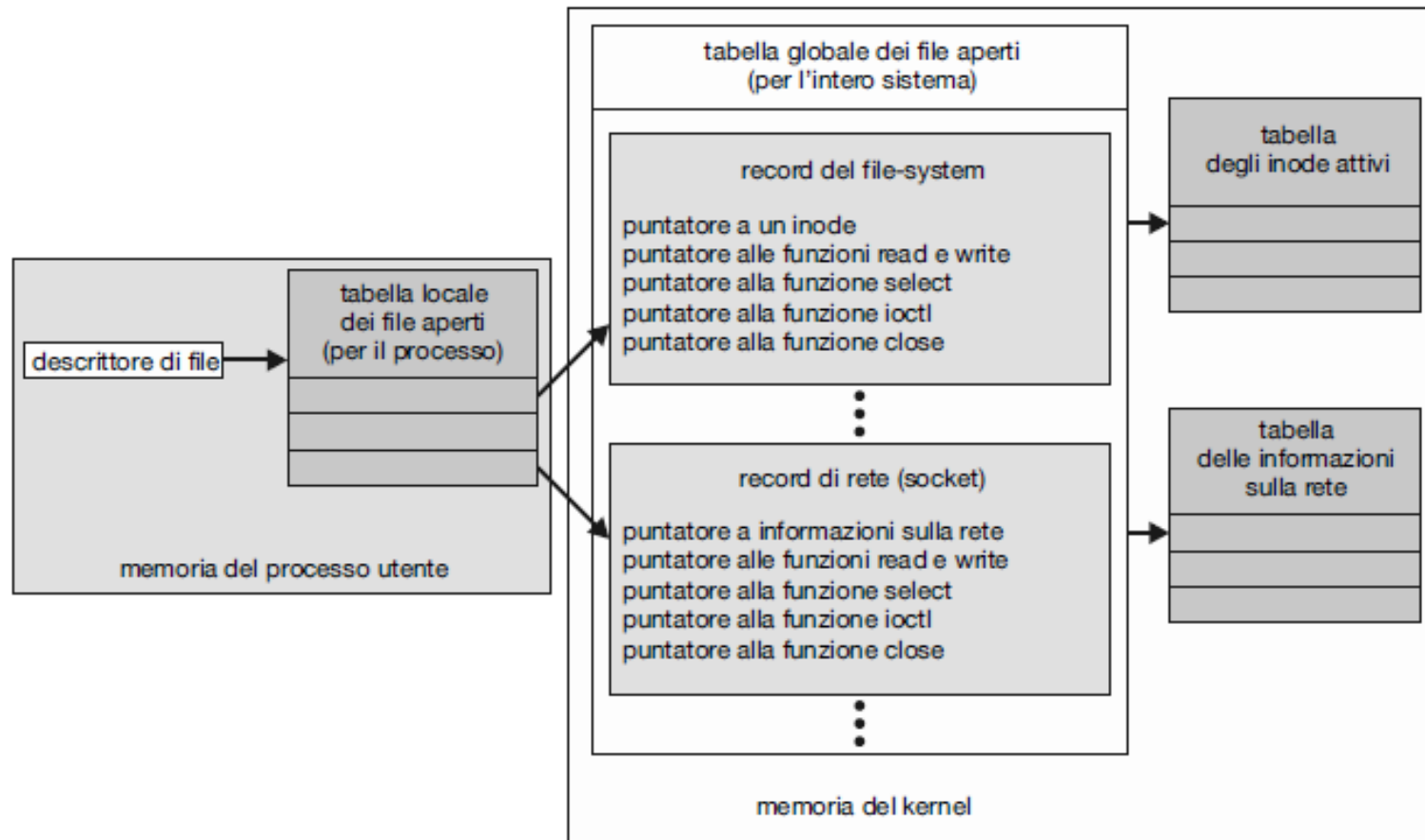


Figura 12.13 Struttura dell'I/O nel kernel di UNIX.

# Riassumendo

---

Il sistema per l'I/O coordina un'ampia raccolta di servizi disponibili per le applicazioni e per altre parti del kernel:

gestione dello spazio  
dei nomi per file e  
dispositivi

controllo  
dell'accesso ai file e  
ai dispositivi

controllo delle  
operazioni

allocazione dello  
spazio per il file  
system

allocazione dei  
dispositivi

gestione dei buffer,  
delle cache e delle  
code di spooling

scheduling dell'I/O

controllo dello stato  
dei dispositivi,  
gestione degli errori  
e procedure di  
ripristino

configurazione e  
inizializzazione dei  
driver dei dispositivi

gestione energetica  
dei dispositivi dell'I/O

# Esecuzione di una richiesta di I/O

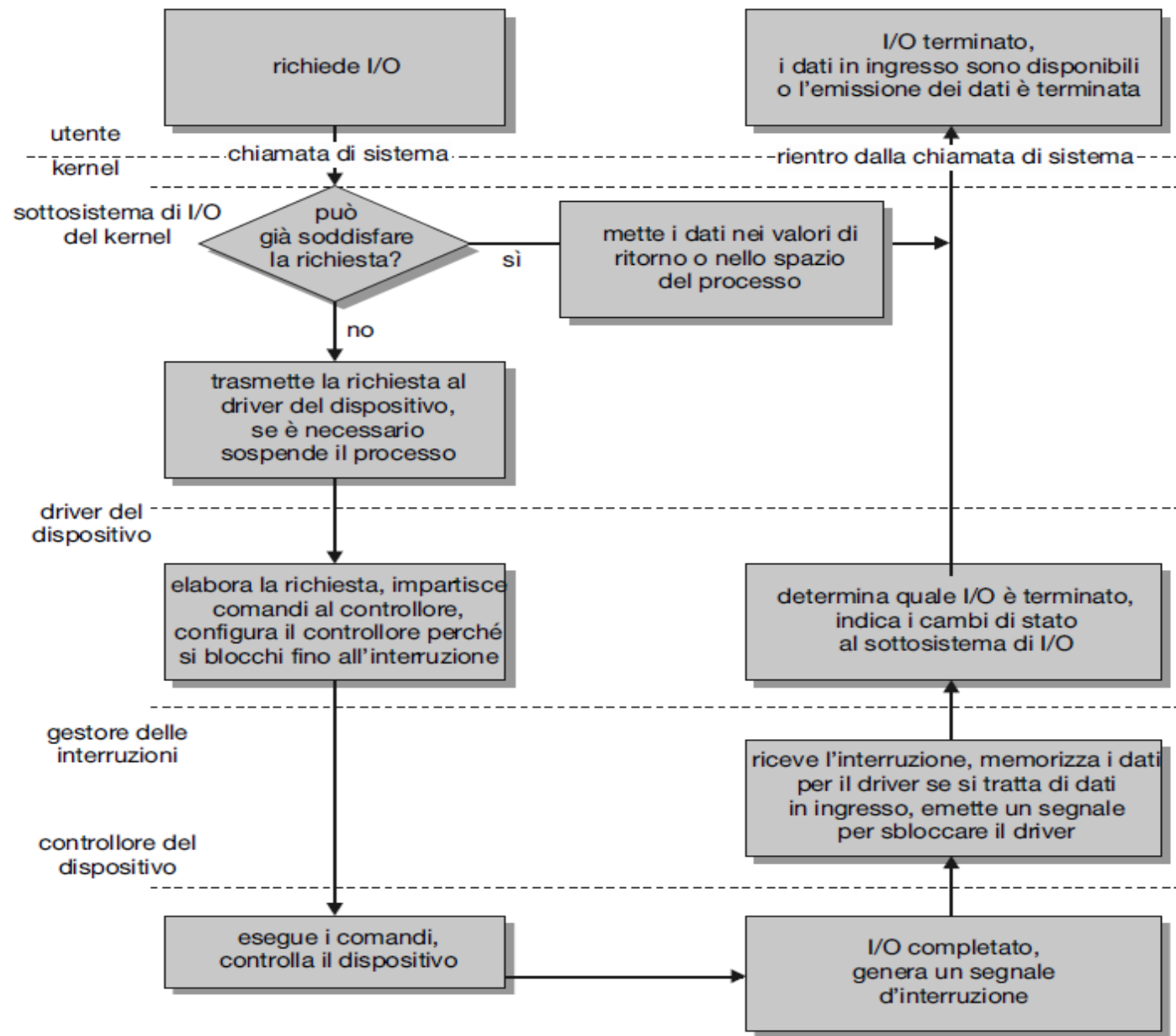


Figura 12.14 Schema d'esecuzione di una richiesta di I/O.

# Streams

**STREAMS** è una realizzazione e una metodologia che permettono di sviluppare in modo modulare e incrementale i driver e i protocolli di rete.

Utilizzando gli *stream*, i driver possono essere organizzati in una catena, attraverso cui passano i dati in maniera sequenziale e bidirezionale per l'elaborazione

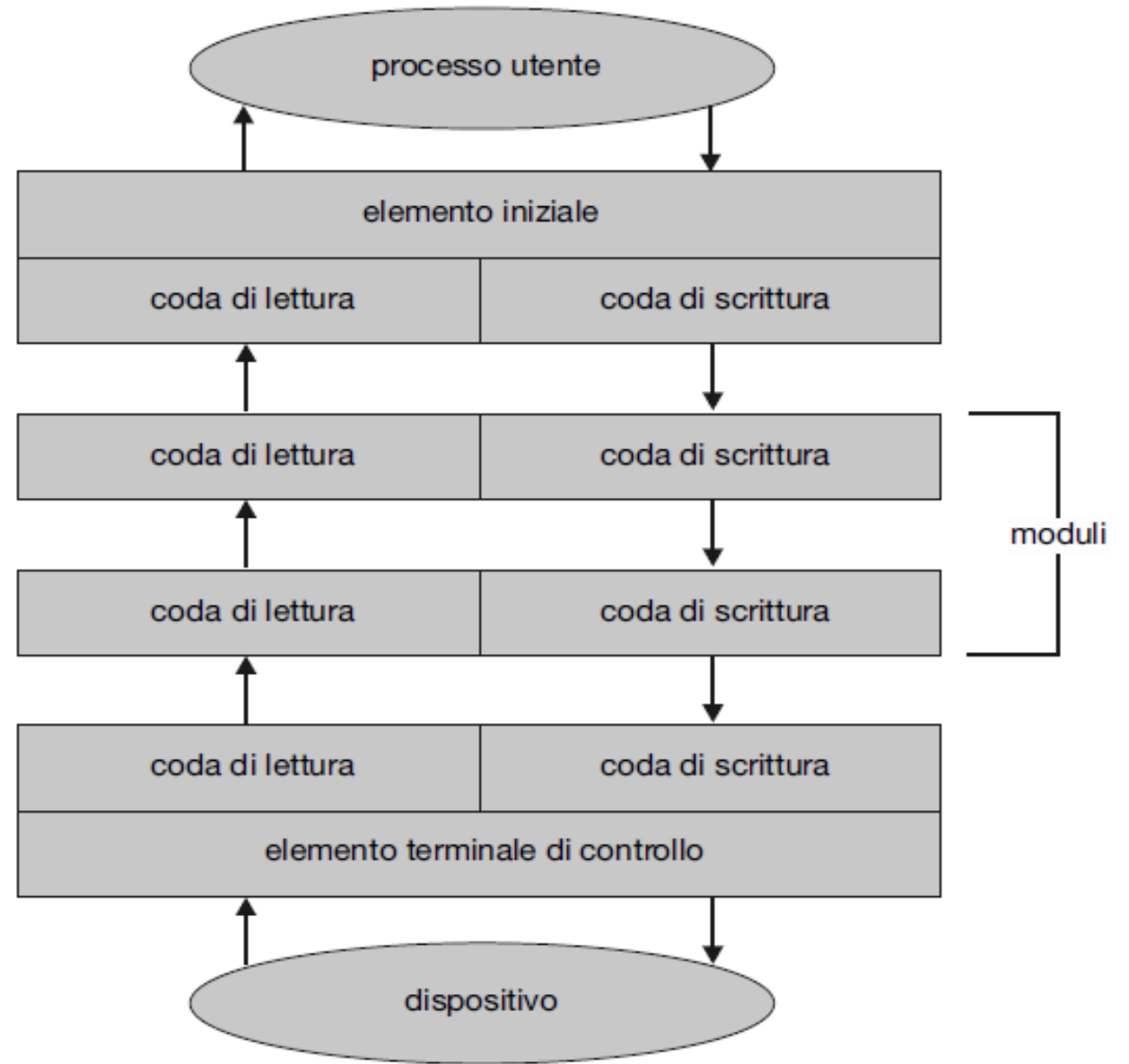


Figura 12.15 Struttura di STREAMS.



# `ioctl`

---

La system call `ioctl` permette di interagire con il driver di un dispositivo generico, per esempio una webcam.

Tramite la `ioctl` sarà possibile ricavare e settare i parametri di tale dispositivo, per esempio ricavare la risoluzione della webcam o settarne la tipologia di acquisizione dati.

Per configurare dispositivi seriali a flusso di caratteri, per esempio un terminale, è possibile usare le API incluse nella interfaccia `termios`.

Tramite questa, avremo accesso a tutte le informazioni relative al dispositivo, per esempio baudrate, echo, etc...

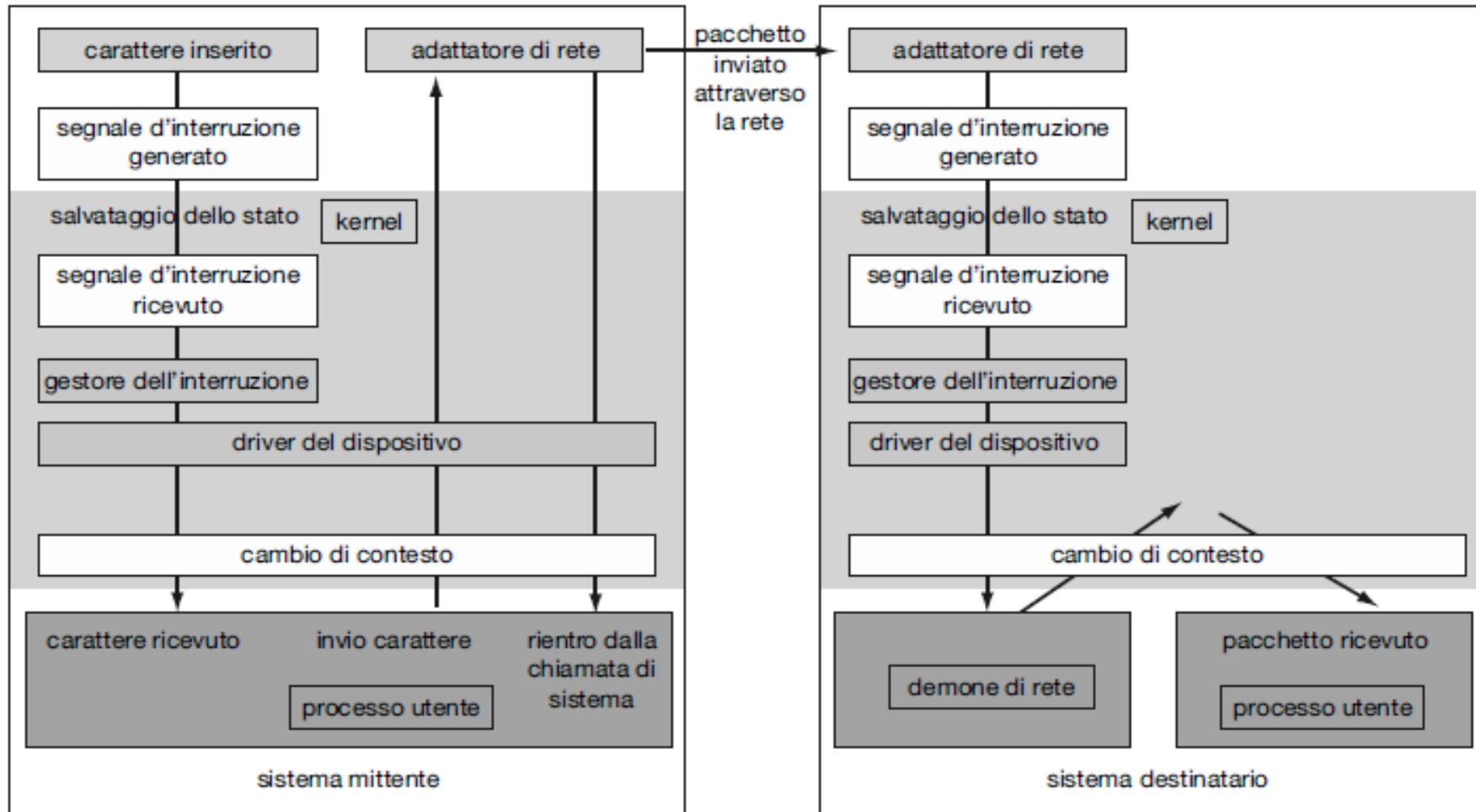
# Prestazioni

---

A causa dei molti strati di software presenti fra un dispositivo fisico e l'applicazione, le **chiamate di sistema per l'I/O** sono **onerose** in termini di utilizzazione della CPU.

Anche il **traffico di una rete** può portare a un **alto numero di cambi di contesto**; si consideri, per esempio, il login remoto da un calcolatore a un altro.

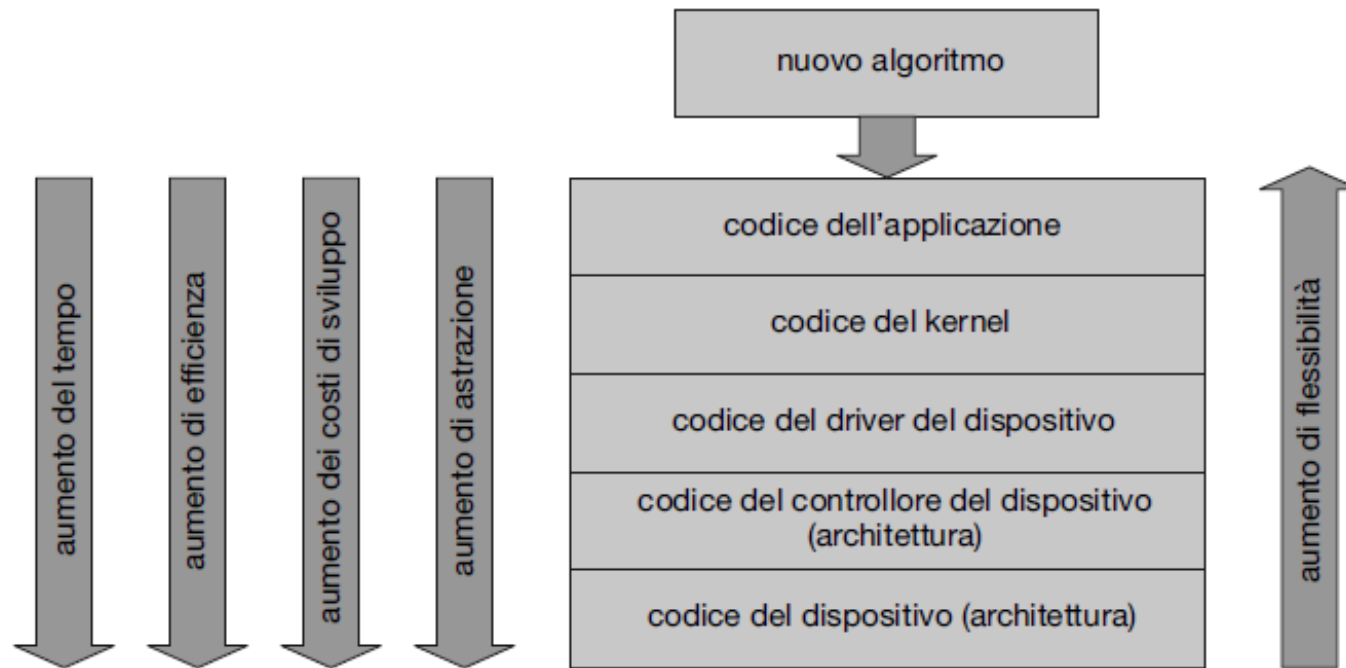
# Prestazioni



**Figura 12.16** Comunicazione tra calcolatori.

# Implementazione dei servizi di I/O

Ci si può chiedere se i **servizi di I/O** si debbano implementare nei dispositivi hardware, nei loro driver, o nelle applicazioni. Talvolta si può osservare (Figura 12.17) la seguente successione.



**Figura 12.17** Successione delle funzionalità dei servizi di I/O.

# Capacità e latenza

La Figura 12.18 mostra CPU e dispositivi di memoria in un grafico dove le due dimensioni rappresentano la capacità e la latenza delle operazioni di I/O. Inoltre, la figura mostra una rappresentazione della latenza di rete, utile per rivelare il tributo aggiuntivo imposto dal networking in termini di prestazioni.

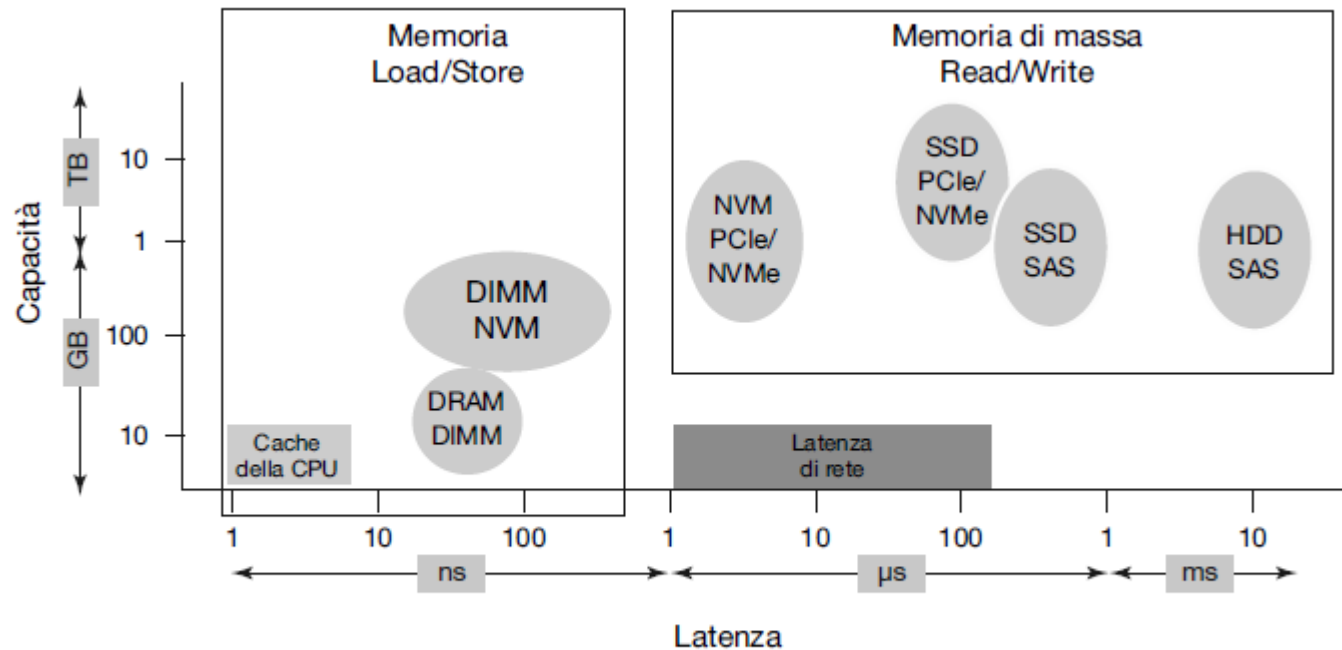


Figura 12.18 Prestazioni di I/O dei dispositivi di memorizzazione (e latenza di rete).



**UNIVERSITÀ DEGLI STUDI  
DELLA BASILICATA**

*Corso di Sistemi Operativi*

# Sistemi di I/O

Docente:  
**Domenico Daniele  
Bloisi**

