

# Input/output

Sistemi Operativi  
Lez. 32

# Ruolo del SO

- Le periferiche di I/O sono i dispositivi attraverso i quali un calcolatore scambia dati/interagisce con la realtà esterna
- Per ogni periferica collegata ad un calcolatore il SO deve essere in grado di
  - Inviare comandi alla periferica
  - Intercettare segnali di interrupt da questa generati
  - Gestire errori
- Per capire come possono essere realizzate queste funzionalità è necessario conoscere i principi di funzionamento di una periferica

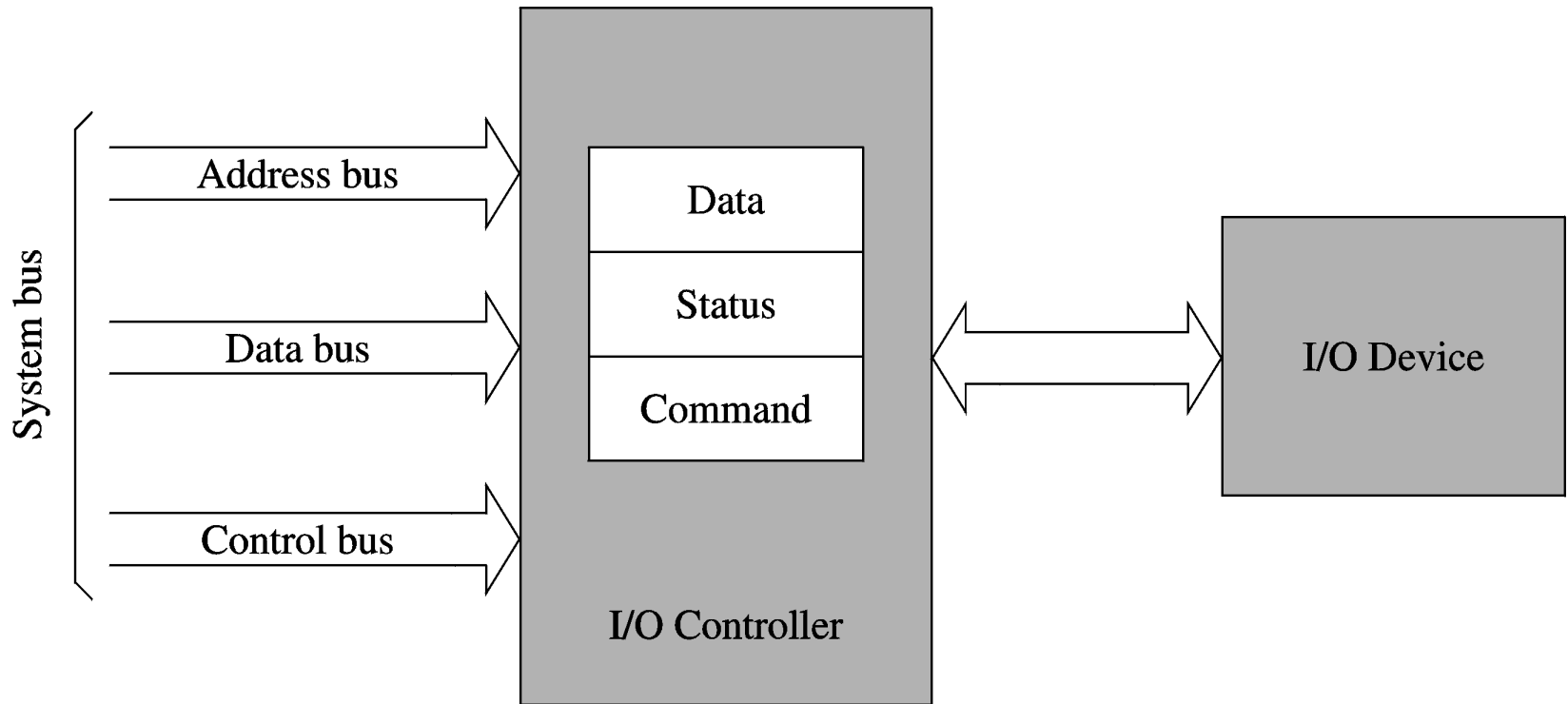
# Dispositivi di I/O

- I dispositivi di I/O hanno:
  - Una componente meccanica
  - Una componente elettronica
- La componente elettronica chiamata **device controller** impartisce comandi al dispositivo fisico (drive/device) e ne controlla la corretta esecuzione
- L'insieme dei comandi impartiti dal controller al device, il loro formato, i codici di errore riportati definiscono l'interfaccia tra il device ed il dispositivo
- Un controller può controllare più device purché abbiano la stessa interfaccia
- Interfacce famose: IDE (Integrated Device electronics, SCSI (Small Computer System Interface), USB (Universal Serial BUS)

# Tipi di Device

- I dispositivi possono essere suddivisi in due grosse categorie in funzione delle modalità con cui “manipolano” i dati su cui operano:
  - Dispositivi a blocchi: memorizzano e trasferiscono le informazioni in blocchi di dimensione fissa, e ciascun con un suo indirizzo. La dimensione del blocco varia da 512-32,768 byte
  - Dispositivi a carattere: memorizzano e trasferiscono stringhe di byte senza riferimento ad alcuna struttura di blocco
- Il clock è un dispositivo di I/O che non ricade in nessuna delle due predette categorie

# Device Controller



# Device Controller

- Interposti tra il SO e la periferica, offrono un'interfaccia di comunicazione tra i due “mondi”
- Possono essere programmati usando appositi registri ed in alcuni casi data buffer
- Caricando opportune sequenze di bit all'interno di questi registri, il sistema operativo può richiedere l'esecuzione di comandi il cui risultato può essere prelevato sempre attraverso questi registri
- L'insieme di comandi usati per comunicare con il controller, il loro formato, ed i codici di errore costituiscono l'interfaccia del controller verso il sistema operativo, questa interfaccia è device-dependent

# Comunicare con il controller

- Per comunicare con un controller dobbiamo affrontare due problemi:
  - Come accedere ai registri (data, status,...)
    - Questo accesso dipende dalle modalità di I/O mapping
      - » Memory-mapped I/O
      - » Isolated I/O
  - Un protocollo per comunicare
    - 3 tipi
      - Programmed I/O & Polling
      - Direct memory access (DMA)
      - Interrupt-driven I/O

# Memory mapped I/O

- I controller condividono lo spazio fisico della memoria, cioè ai registri dei controller sono assegnate determinate locazioni di memoria centrale
- Le operazioni di I/O sono le stesse operazioni usate per la gestione delle memoria (LW,SW)
- Non esiste alcuna istruzione particolare di I/O

# Isolated I/O

- Lo spazio degli indirizzi dei controller è diverso da quello della memoria centrale
- Si adottano istruzioni di I/O dedicate a manipolare i registri
- A livello hw è necessario prevedere degli opportuni meccanismi (I/O bus, linee dedicate) per discernere gli indirizzi di I/O da quelli relativi alla memoria

# Isolated I/O in INTEL

- I/O port: ad ogni registro è assegnato un numero di 8/16 bit, che specifica la porta di I/O corrispondente a cui ci si riferisce usando istruzioni di I/O
  - Es. `mov al, 0x20`  
`out 0x20, al`
- Sistema ibrido: Buffer dati mappati in memoria, registri di controllo come I/O port

# I/O port INTEL

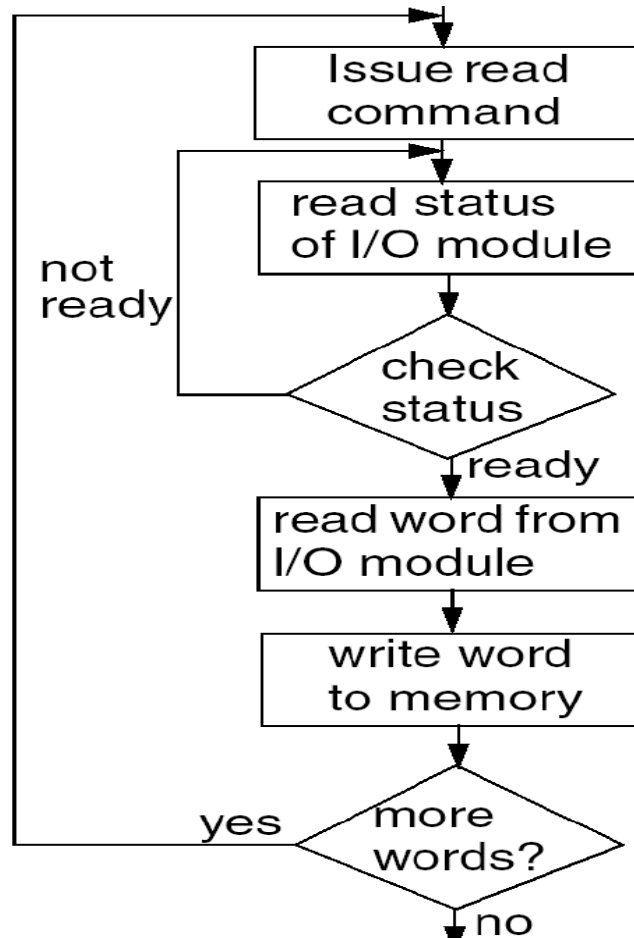
I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

# Modalità di comunicazione tra controller e driver

# Programmed I/O & Polling

- *Continuous polling*: all'interno del driver è presente un loop di busy waiting che continuamente verifica se il dispositivo è pronto o meno ad accettare dati o comandi, oppure a produrre un output
- *Periodic Polling*: in alcuni casi, il driver verifica periodicamente se un dispositivo è pronto ad operare, in caso affermativo procede altrimenti si blocca e ripete l'operazione ad intervalli di tempo prefissati
  - Es: un mouse USB deve essere "interrogato" con una frequenza di 100 Hz

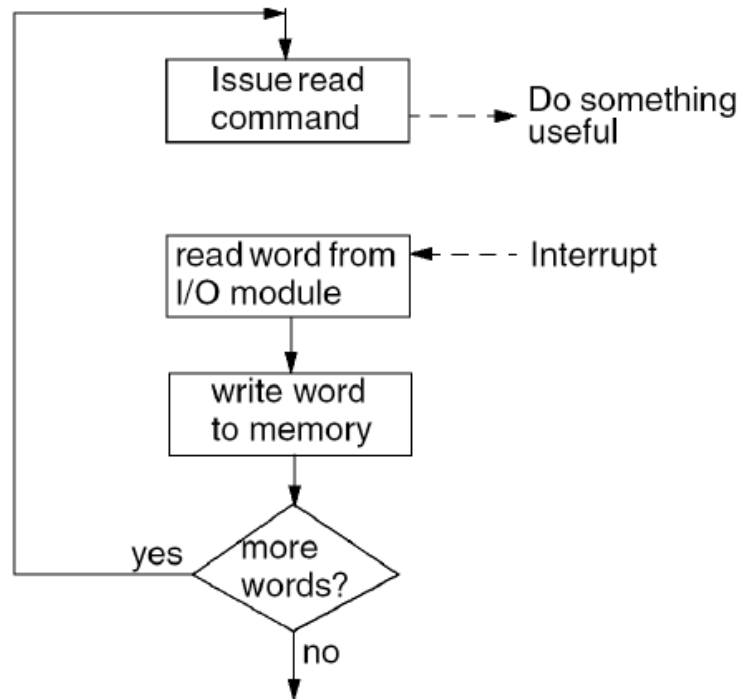
# I/O Controllo Programma



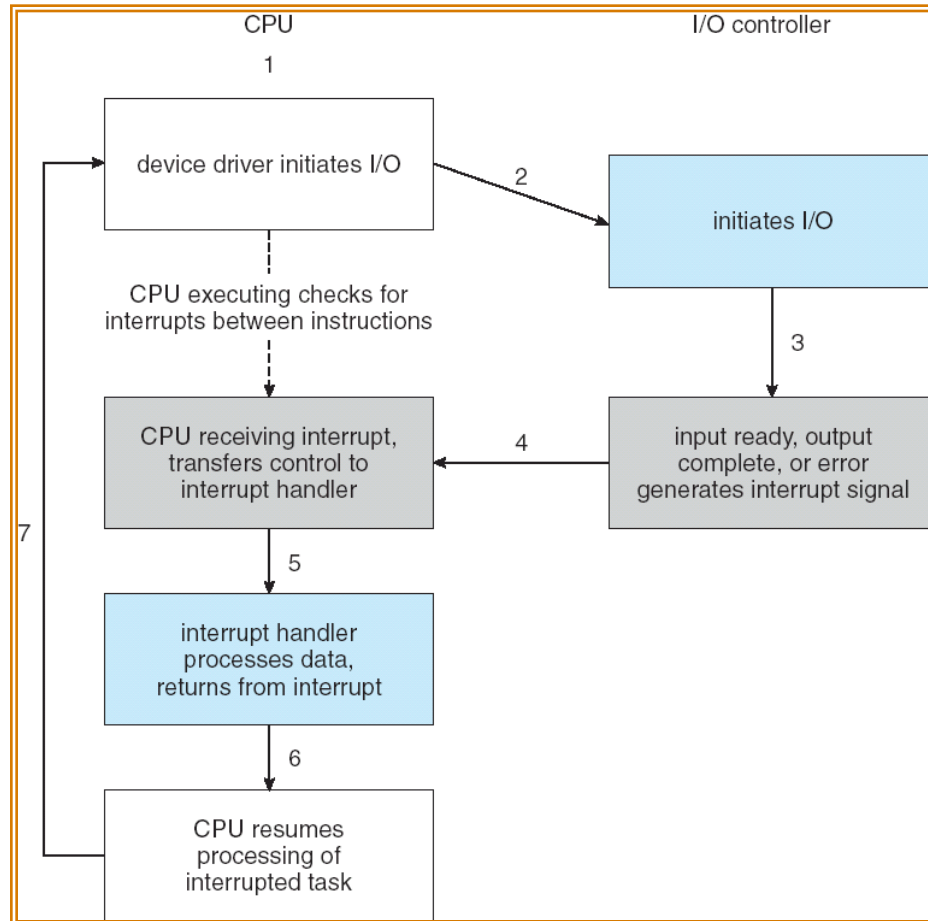
# Interrupt Driven I/O

- I registri di controllo di un device controller posseggono almeno uno status bit che indica se un'operazione di output è stata completata oppure se c'è un nuovo dato da consumare
- Nelle periferiche più sofisticate questo insieme a questi bit viene asserita una linea di IRQ sul bus di sistema
- Quindi ogni volta che si inserisce un nuovo controller su un sistema va indicato l'IRQ di riferimento, per evitare conflitti questa scelta viene spesso demandata al BIOS in fase di boot

# Interrupt driven I/O



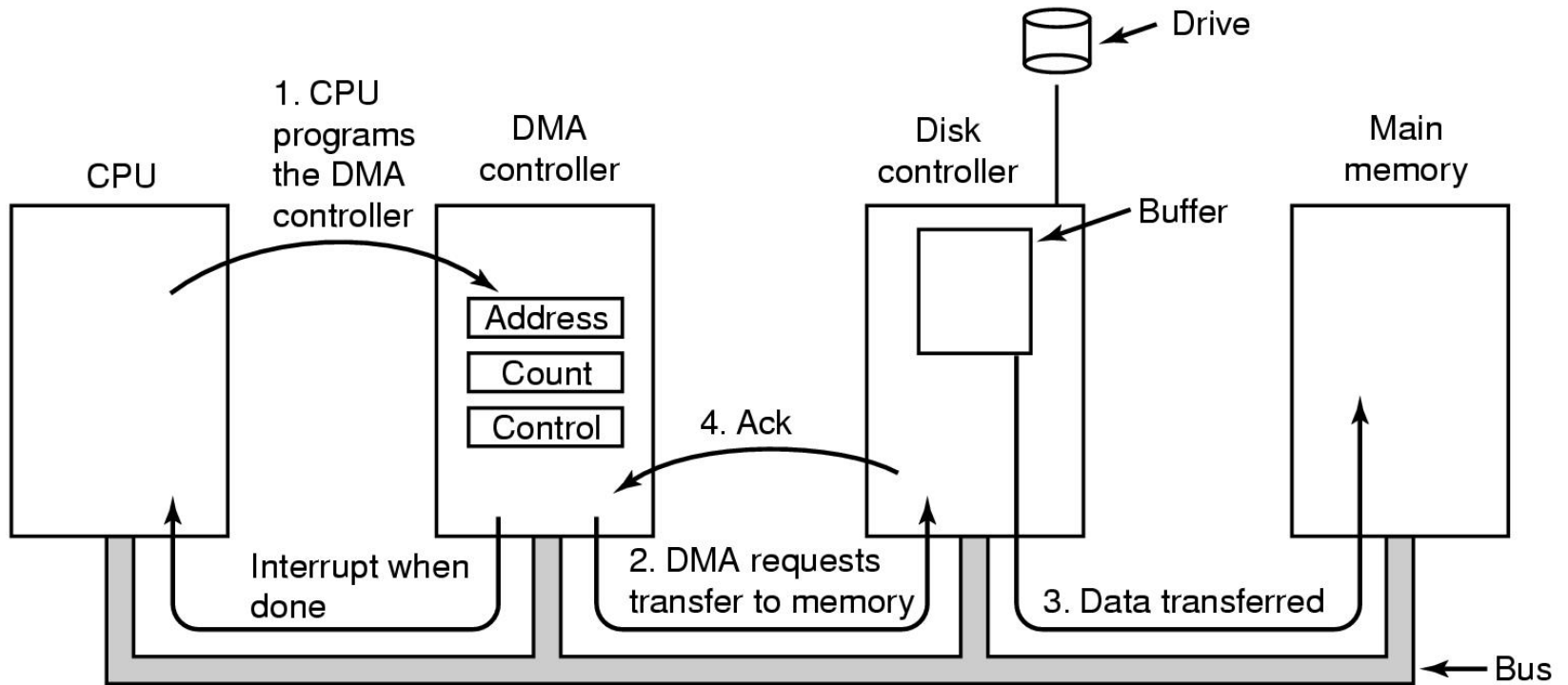
# Interrupt-driven I/O



# DMA-CPU

- La CPU (il driver) programma il DMA controller, predisponendone i registri con i seguenti dati:
  - indirizzo di memoria da dove prelevare o depositare i dati;
  - Numero di byte da acquisire/prelevare
  - Informazione di controllo (read/write)
- Dopodiché riprende la normale operatività
- Il DMA controller prende il controllo del memory bus directly e impartisce i comandi all' I/O controller per effettuare lo spostamento di dati richiesto dalla CPU
- Il Disk controller trasferisce i dati in memoria principale
- Terminato il trasferimento il disk controller invia un ACK al DMA controller
- Il DMA controller invia un interrupt al processore per avvertirlo del completamento dell'operazione

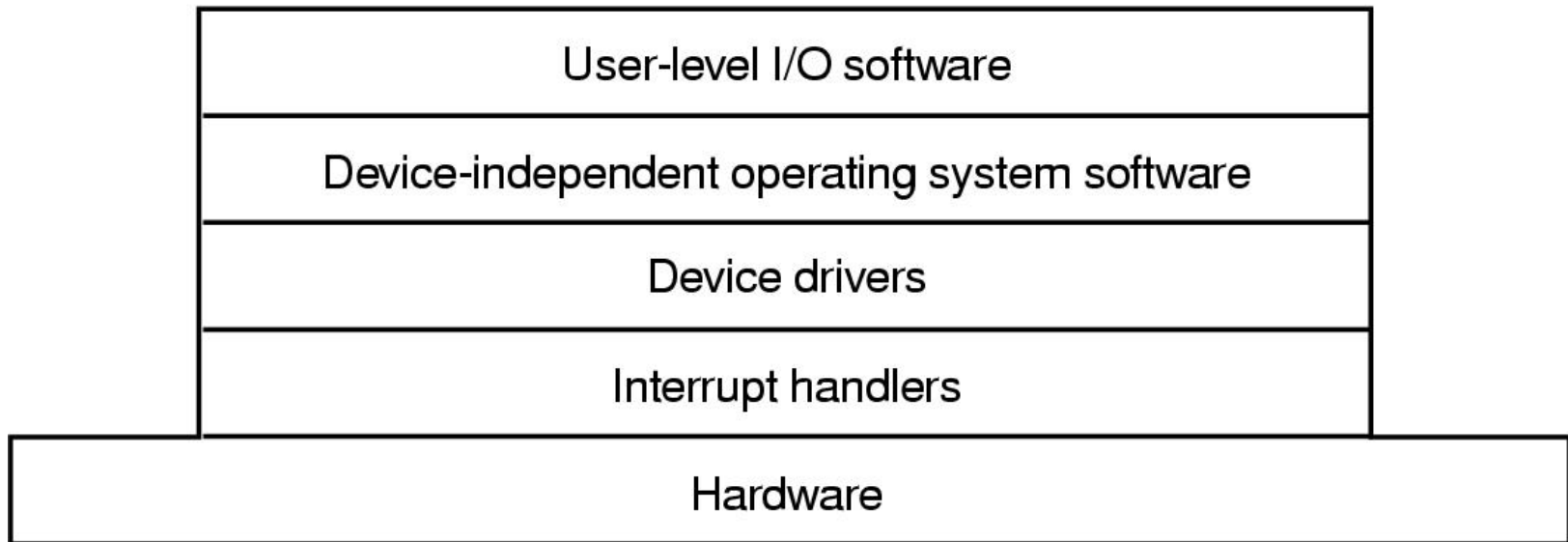
# DMA



# DMA

- PRO
  - Alleggerisce la CPU dall'overhead imposto dalla gestione degli interrupt nel caso di periferiche veloci
- CONS
  - Il DMA è in genere più lento della CPU, in molti sistemi la CPU (più lenta) si trova spesso in attesa del DMA
  - Il DMA aumenta i costi del sistema

# I/O sw system



- Il sottosistema per la gestione dell'I/O viene solitamente strutturato in 5 livelli, ciascuno formato da diverse componenti

# Interrupt Handler

- Gli Interrupt handler sono la parte più nascosta dell'architettura e sono caratterizzati da stretti vincoli di tempo, eseguono quindi lo stretto necessario per
  - Acquisire i dati dell'ultima operazione di I/O
  - Predisporre le condizioni per garantire l'esecuzione della prossima operazione di I/O
  - Risvegliare il driver della periferica che stava attendendo il completamento dell'operazione di I/O

# Device Driver

- Ogni periferica è caratterizzata da un proprio insieme di registri e di comandi, affinché la periferica funzioni correttamente è necessario che registri e comandi siano correttamente predisposti e impartiti
- Il device driver è un programma che svolge esattamente queste funzioni
- Esiste un device driver per ogni periferica solitamente realizzato dal costruttore della periferica stessa
- Solitamente concepito come parte del kernel, per consentire l'accesso rapido alle periferiche, in Minix i driver operano in user mode

# Device driver

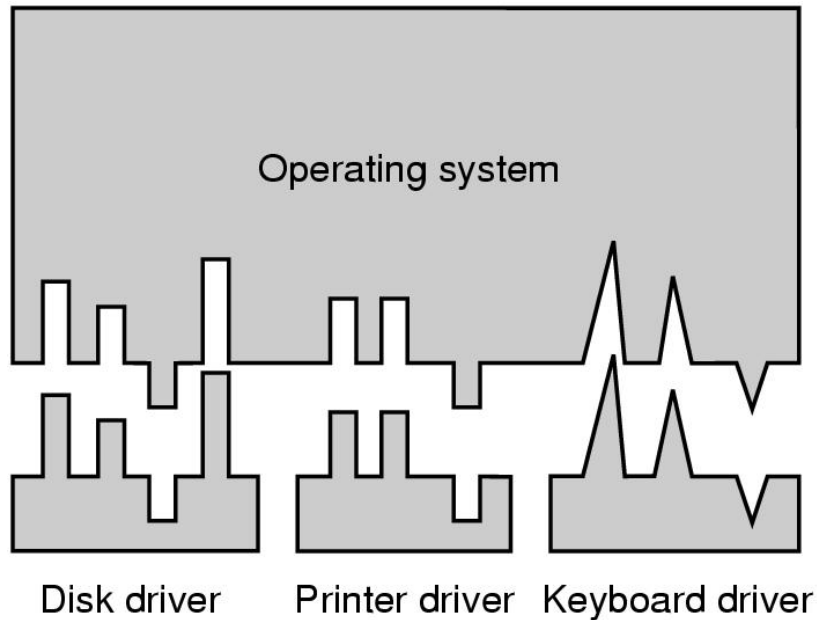
- Il driver costituisce quindi l'interfaccia tra il SO e il controller della periferica
- Riceve dal device independent sw la richiesta per l'esecuzione di comandi d'alto livello, che traduce in una sequenza di comandi per il controller
- Avvia il controller e resta in attesa di essere risvegliato da un interrupt, tramite l'interrupt handler

# Device-Independent I/O Software

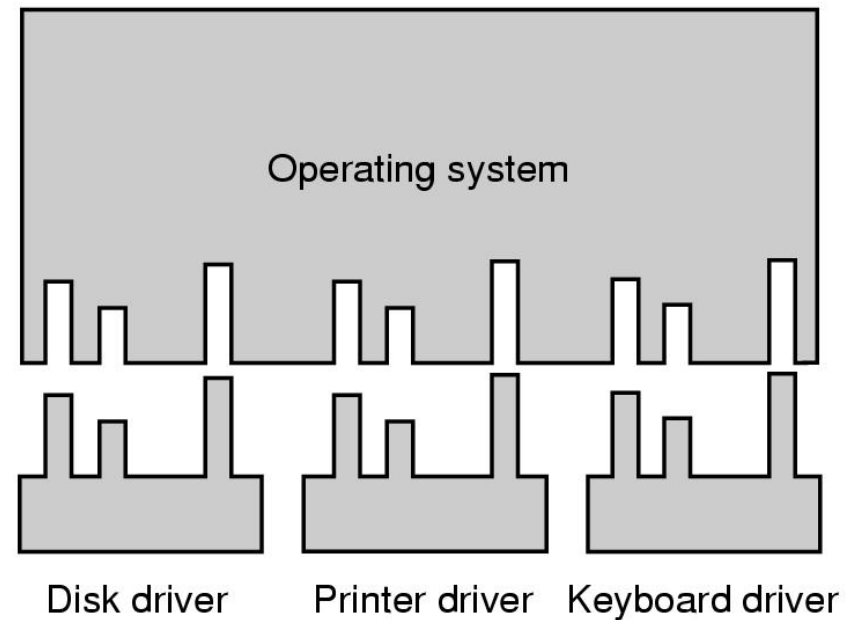
- Scopi:
  - Fornire un'interfaccia standard alle applicazioni per la gestione dell'I/O
  - fornire le funzionalità comuni a tutti i dispositivi di I/O

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicate devices
Providing a device-independent block size

# Interfaccia verso i driver



(a)



(b)

- **Interfaccia Driver-SO:** le funzioni che il driver deve mettere a disposizione e le funzioni di kernel che può chiamare
- **Un interfaccia uniforme** facilita la stesura di driver, e rende possibile l'aggiunta di driver senza ricorrere alla modifica del kernel

# Device-Independent I/O Software

- Uniform naming: si preoccupa di mappare il nome simbolico di un dispositivo nel corrispondente driver
- Buffering: si preoccupa di gestire la diversità tra i dati acquisiti da un dispositivo e i dati richiesti da un'applicazione
- Error Reporting: si preoccupa di fornire un'interfaccia comune verso i diversi messaggi di errore che possono derivare dai driver, a seguito di errori nei dispositivi

# Device-Independent I/O Software

- Allocating/Releasing : si preoccupa di allocare deallocare l'uso di risorse ai processi utente, funzionalità particolarmente critica per risorse non condivisibili, a causa del deadlock
- Device independent block size: nasconde al livello superiore alcuni dettagli implementativi come ad esempio la diversa dimensione dei blocchi

# Riassumendo

