LIVELLI DEL SOFTWARE DI I/O

Danilo Croce

Dicembre 2023

OVERVIEW

- Principi dell'hardware di I/O
- **Livelli del software di I/O**

OBIETTIVI DEL SOFTWARE DI I/O

- **Indipendenza dal Dispositivo**:
  - Il software di I/O dovrebbe permettere l'accesso a diversi dispositivi senza specificare il tipo di dispositivo in anticipo.
  - **Esempio**: un programma che legge un file dovrebbe funzionare indifferentemente con dischi fissi, SSD o penne USB.
- **Denominazione Uniforme**:
  - I nomi di file o dispositivi dovrebbero essere stringhe o numeri indipendenti dal dispositivo.
  - **Esempio**: in UNIX, l'integrazione dei dispositivi nella gerarchia del file system consente un indirizzamento uniforme tramite nomi di percorso.
    - Non vogliamo digitare ST6NM04 per indirizzare il primo disco rigido.
      - `/dev/sda` **è meglio**
      - `/mnt/movies` **ancora meglio**

OBIETTIVI DEL SOFTWARE DI I/O (2)

- **Gestione degli Errori**:
  - Gli errori **vanno gestiti il più vicino possibile all'hardware**, idealmente dal controller stesso o dal driver del dispositivo.
  - **Errori transitori** (come quelli di lettura) spesso **scompaiono ripetendo l'operazione**.
- **Trasferimenti Sincroni vs Asincroni**:
  - La maggior parte dell'I/O fisico è asincrono, ma per semplicità, molti programmi utente trattano l'I/O come se fosse sincrono (bloccante).
  - Il sistema operativo rende operazioni asincrone sembranti bloccanti, ma fornisce anche l'accesso all'I/O asincrono per applicazioni ad alte prestazioni.
  - Il sistema operativo deve gestire DMA

OBIETTIVI DEL SOFTWARE DI I/O (3)

- **Buffering**:
  - Spesso i dati da un dispositivo non vanno direttamente alla destinazione finale, richiedendo un buffer temporaneo.
    - Un pacchetto che arriva su un'interfaccia di rete deve essere ricevuto e analizzato prima di capire quale applicazione (esempio browser) può usarlo
    - Un segnale audio deve essere posizionato preventivamente in un buffer per evitare interruzioni
  - L'uso di buffer può influenzare le prestazioni, soprattutto per dispositivi con vincoli real-time.
- **Dispositivi Condivisibili vs Dedicati**:
  - Dispositivi come dischi e SSD possono essere condivisi da più utenti, mentre altri come stampanti e scanner sono tipicamente dedicati.
  - Il sistema operativo deve gestire entrambe le categorie per evitare problemi come i deadlock.

TIPOLOGIE DI SW PER I/O

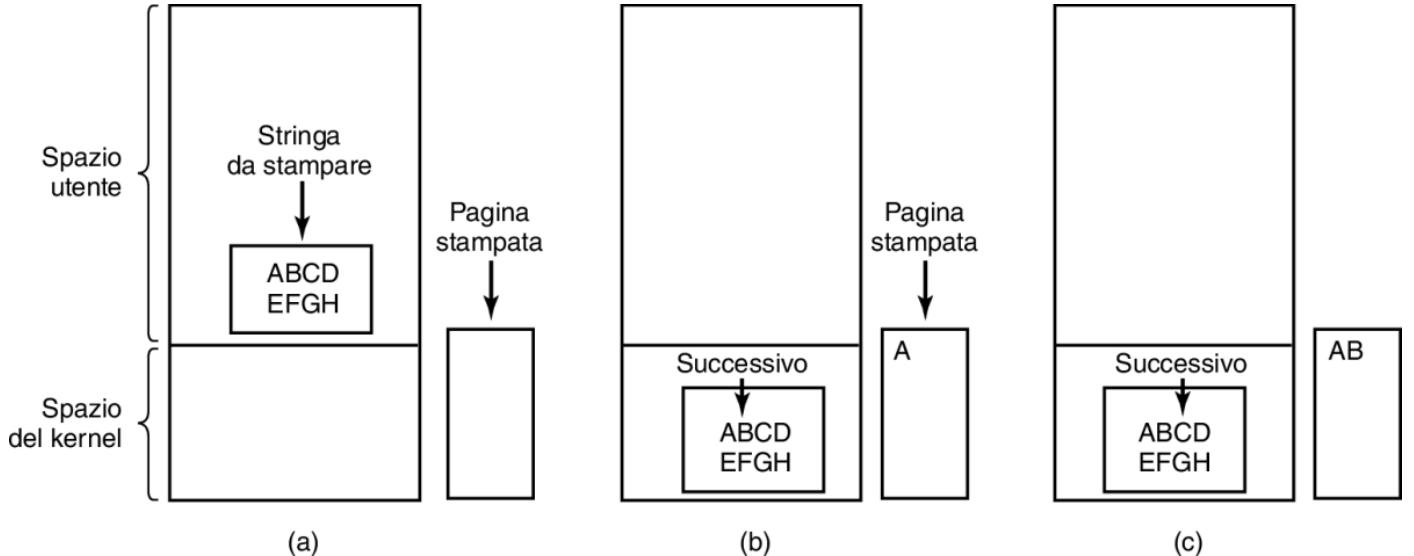
I/O Programmato

I/O Guidato dagli Interrupt

I/O con DMA

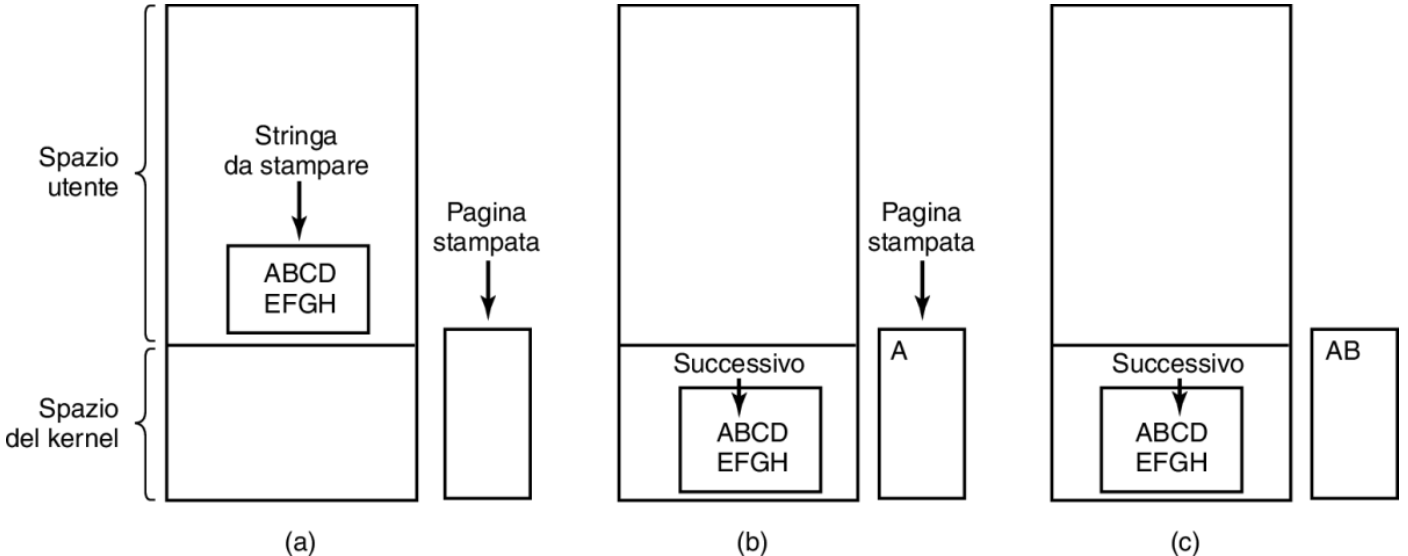
I/O PROGRAMMATO - PROCESSO E ESEMPIO PRATICO

- **Definizione di I/O Programmato**: la CPU gestisce direttamente tutto il processo di trasferimento dei dati.
- **Esempio Pratico con Riferimento alla Figura 5.7**:
  - Un processo utente prepara una stringa "ABCDEFGH" in un buffer dello spazio utente.
  - Il processo effettua una chiamata di sistema per stampare la stringa, dopo aver ottenuto l'accesso alla stampante (a).



I/O PROGRAMMATO - PROCESSO E ESEMPIO PRATICO

- **Azione del Sistema Operativo**: copia il buffer in uno spazio del kernel.
  - Invia i caratteri alla stampante uno alla volta, aspettando che questa sia pronta per ogni carattere (b e c).
- **Polling o Busy Waiting**:
  - Il sistema operativo entra in un ciclo di polling, controllando il registro di stato della stampante e inviando un carattere alla volta.



ESEMPIO DI CODICE E LIMITI DELL'I/O PROGRAMMATO

```
copy_from_user(buffer, p, count);                    /* p è il buffer del kernel */
for (i = 0; i < count; i++) {                        /* ripeti per tutti i caratteri */
        while (*printer_status_reg != READY) ;       /* ripeti finché lo stato diventa READY */
        *printer_data_register = p[i];               /* invia in output ogni carattere */
}
return_to_user();
```

- **Esempio di Codice per I/O programmato**:
  - Utilizza `copy_from_user(buffer, p, count)` per copiare i dati dal buffer utente a quello del kernel.
  - Un ciclo `for (i = 0; i < count; i++)` gestisce il trasferimento carattere per carattere alla stampante.
- **Svantaggi dell'I/O Programmato**:
  - **Occupa la CPU a tempo pieno durante il processo di I/O**, facendo continuamente polling sullo stato della stampante.
  - **Inefficiente** in sistemi complessi dove la CPU ha altre attività importanti da gestire.

ESEMPIO DI CODICE E LIMITI DELL'I/O PROGRAMMATO (2)

```
copy_from_user(buffer, p, count);                    /* p è il buffer del kernel */
for (i = 0; i < count; i++) {                        /* ripeti per tutti i caratteri */
        while (*printer_status_reg != READY) ;       /* ripeti finché lo stato diventa READY */
        *printer_data_register = p[i];               /* invia in output ogni carattere */
}
return_to_user();
```

- **Applicazioni e Contesti Efficaci**:
  - L'I/O programmato è **efficace quando il tempo di elaborazione** di un carattere è **breve**.
  - Adatto a sistemi embedded dove la CPU non ha altre attività significative.
- **Necessità di Metodi di I/O Alternativi**:
  - Nei sistemi più complessi, il busy waiting diventa un approccio inefficiente.
  - Ricerca di metodi di I/O che liberino la CPU da costanti attività di polling.

I/O GUIDATO DAGLI INTERRUPT - PROCESSO DI STAMPA

- **Scenario di Stampa senza Buffer**:
  - Una stampante che stampa un carattere alla volta, con un ritardo di 10 ms per carattere,
  - permettendo alla CPU di eseguire altri processi durante l'attesa.
- **Utilizzo degli Interrupt**:
  - Dopo la chiamata di sistema `copy_from_user` per stampare una stringa, il buffer viene copiato nello spazio del kernel e il primo carattere viene inviato alla stampante.
  - La CPU poi passa l'esecuzione ad altri processi mentre attende che la stampante sia pronta per il carattere successivo.
- **Cambio di Contesto e Blocco del Processo**:
  - Il processo che ha richiesto la stampa viene bloccato fino a quando non è stampata l'intera stringa.
  - La CPU attiva lo scheduler per eseguire altri processi durante l'attesa.

```
/* copia dati dall'utente al kernel.*/
copy_from_user(buffer, p, count);

/*abilita gli interrupt*/
enable_interrupts();

/*attende che la stampante sia pronta
  a ricevere caratteri */
while (*printer_status_reg != READY) ;

/*invia il primo carattere alla stampante.*/
*printer_data_register = p[0];

/*passa il controllo a un altro processo.*/
scheduler();
```

Codice eseguito al momento della chiamata di sistema per la stampa.

I/O GUIDATO DAGLI INTERRUPT - PROCESSO DI STAMPA (2)

- **Generazione dell'Interrupt da Parte della Stampante**:
  - La stampante genera un interrupt quando è pronta per il carattere successivo, interrompendo il processo corrente e salvandone lo stato.
- **Esecuzione della Procedura di Servizio Interrupt**:
  - Viene eseguita la procedura di servizio di interrupt per la stampante.
  - Se ci sono altri caratteri da stampare, il gestore stampa il successivo.
- **Sblocco del Processo e Ritorno dall'Interrupt**:
  - Se tutti i caratteri sono stati stampati, il gestore degli interrupt esegue azioni per sbloccare il processo utente.
  - Riconosce l'interrupt e ritorna al processo interrotto, che riprende l'esecuzione da dove era stato lasciato.
- **Problema**: Interrupt ad ogni carattere!

```
/* Controlla se tutti i caratteri sono stati stampati. */
if (count == 0) {
    /* Sblocca il processo utente che ha richiesto la stampa */
    unblock_user();
} else {
    /* Invia il carattere successivo alla stampante. */
    *printer_data_register = p[i];
    /* Decrementa il conteggio dei caratteri rimanenti. */
    count = count - 1;
    /* Passa al carattere successivo nel buffer. */
    i = i + 1;
}
/* Riconosce l'interrupt ricevuto dalla stampante. */
acknowledge_interrupt();
/* Ritorna dall'interrupt, permettendo alla CPU di riprendere altre operazioni. */
return_from_interrupt();
```

Procedura di servizio interrupt per la stampante.

I/O CON DMA: EFFICIENZA E GESTIONE DEI PROCESSI

- **Principio del DMA**:
  - Il DMA riduce il numero di interrupt, passando da uno per ogni carattere a uno per buffer.
  - Libera la CPU per eseguire altre attività durante il trasferimento di I/O.
- **Setup e Inizio del Trasferimento (a)**:
  - **Preparazione dei Dati**: `copy_from_user`.
  - **Configurazione DMA**: `set_up_DMA_controller`.
  - **Ottimizzazione delle Risorse CPU**: `scheduler`.
- **Gestione dell'Interrupt e Conclusione (b)**:
  - **Gestione dell'interrupt generato** dal completamento del trasferimento DMA: `acknowledge_interrupt`.
  - **Ripresa del Processo Utente**: `unblock_user`.
  - **Ritorno dal Contesto dell'Interrupt**: `return_from_interrupt`.

```
/* Copia i dati dall'utente al kernel. */
copy_from_user(buffer, p, count);
/* Impostazione del controller DMA per il trasferimento */
set_up_DMA_controller();
/* La CPU esegue altri processi mentre il DMA gestisce il
          trasferimento. */
scheduler();
```

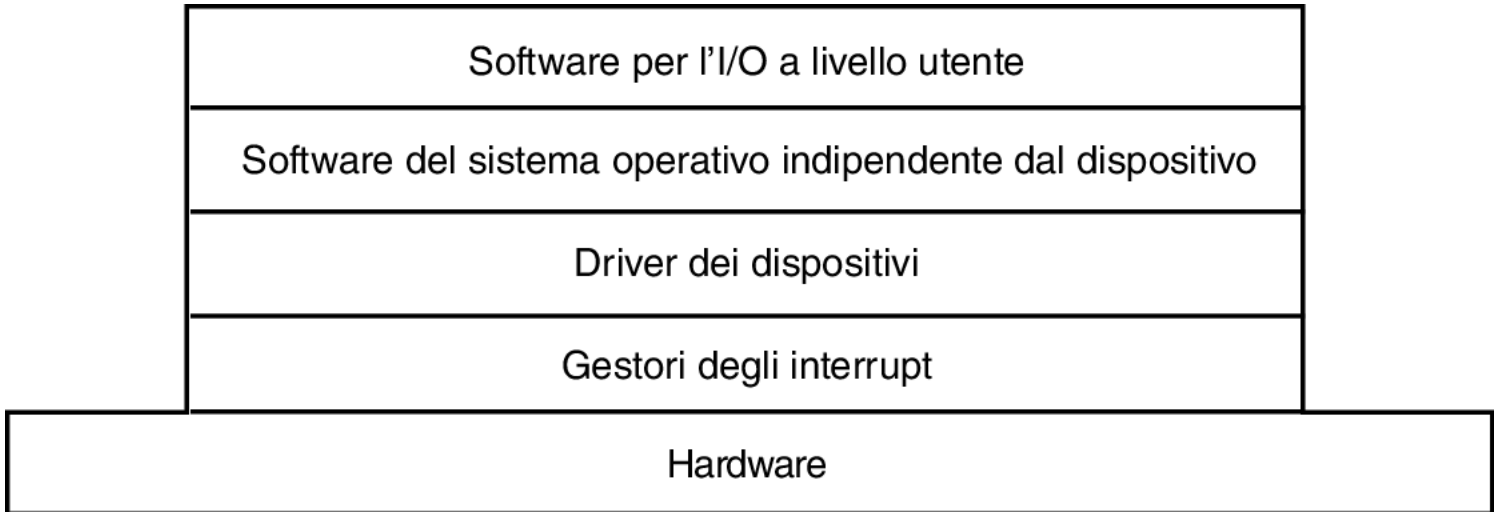
(a)

```
/* Riconosce l'interrupt ricevuto dal DMA. */
acknowledge_interrupt();
/* Sblocca il processo utente dopo che il trasferimento è completo. */
unblock_user();
/* Ritorna dall'interrupt, consentendo alla CPU di proseguire con altre operazioni. */
return_from_interrupt();
```

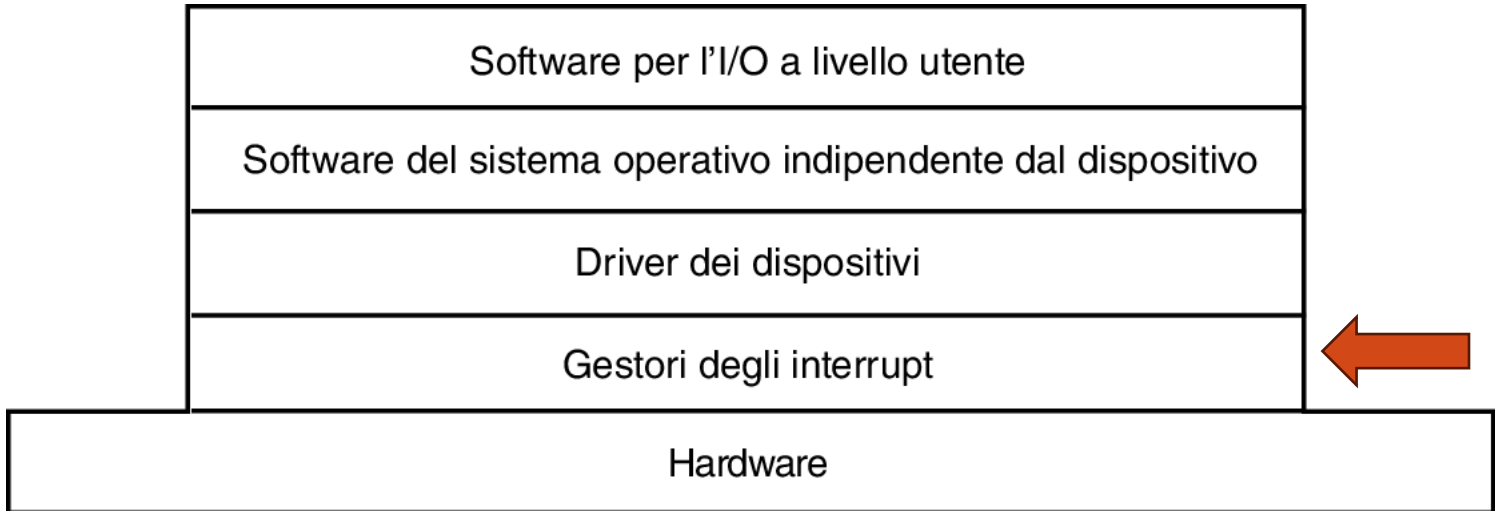
(b)

STRUTTURA DEL SOFTWARE DI I/O

- **Organizzazione a Quattro Livelli**:
  - Il software di I/O è strutturato in quattro livelli distinti.
  - Ogni livello ha funzioni e interfacce specifiche.
- **Interfacce e Funzionalità**:
  - Le funzionalità e le interfacce variano a seconda del sistema operativo.
  - L'analisi dettagliata parte dal livello più basso.



YOU ARE HERE


Software per l'I/O a livello utente
Software del sistema operativo indipendente dal dispositivo
Driver dei dispositivi
Gestori degli interrupt
Hardware

GESTIONE DEGLI INTERRUPT NEL SISTEMA OPERATIVO

- **Blocco dei Driver**: Durante l'I/O, i driver vengono bloccati (es. con semafori, anche se è più complicato) fino al completamento dell'I/O e all'arrivo dell'interrupt.
- **Gestione Complessa**: La gestione degli interrupt richiede diversi passaggi, inclusi salvataggio dei registri, impostazione di contesti e conferme al controller degli interrupt.
- **Impatto della Memoria Virtuale**: Su sistemi con memoria virtuale, la gestione degli interrupt richiede passaggi aggiuntivi per gestire MMU, TLB e cache, aumentando la complessità e i cicli macchina necessari.
- **Elaborazione Non Banale**: L'elaborazione di un interrupt richiede numerosi cicli CPU e varia notevolmente a seconda del sistema e dell'architettura.

GESTORI DEGLI INTERRUPT – PROCESSO E IMPLEMENTAZIONE

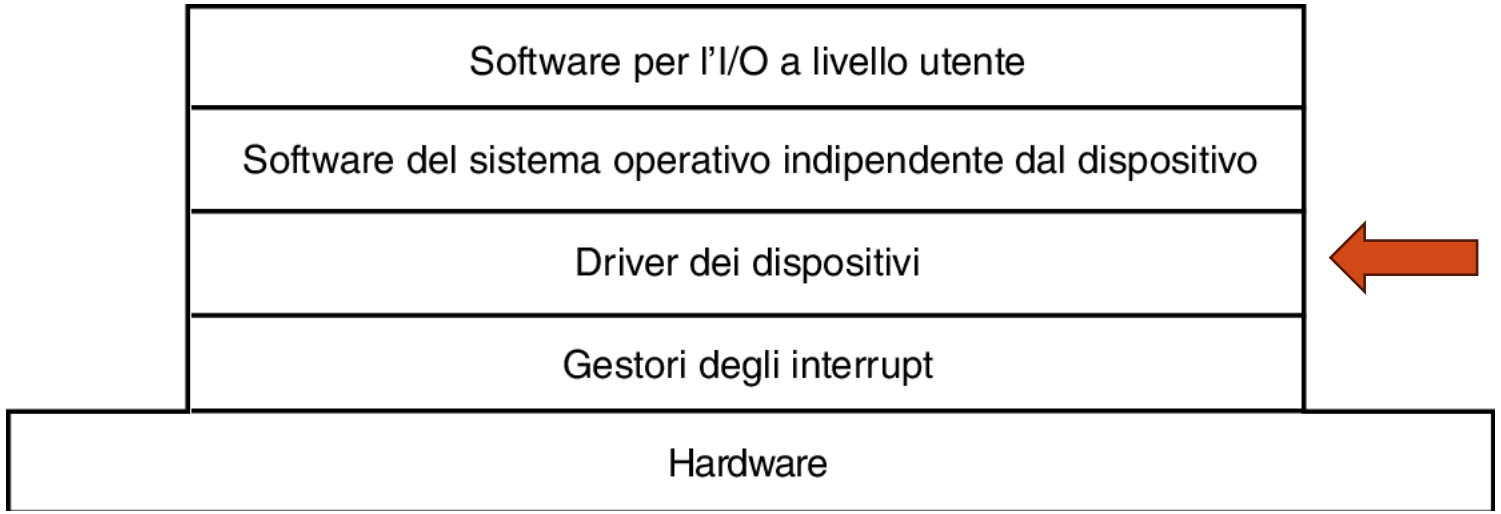
- Di seguito una serie di passaggi da eseguire nel software dopo il completamento dell'interrupt hardware.
  - i dettagli dipendono molto dal sistema (in alcune macchine i seguenti passi potrebbero essere ordinati differentemente o non esserci)

1. **Salvataggio dei Registri**: Salvataggio di tutti i registri, inclusi quelli non salvati dall'interrupt hardware.
2. **Impostazione del Contesto**: Impostazione di un contesto per la procedura di servizio dell'interrupt, incluso il setup di TLB, MMU e una tabella delle pagine.
3. **Impostazione dello Stack**: Configurazione di uno **stack** per la procedura di servizio dell'interrupt.
4. **Conferma al Controller degli Interrupt**: Conferma al controller degli interrupt e riabilitazione degli interrupt, se necessario.
5. **Copia dei Registri nella Tavola dei Processi**: Copia dei registri salvati nella tabella dei processi.

GESTORI DEGLI INTERRUPT – PROCESSO E IMPLEMENTAZIONE (2)

6. **Esecuzione della Procedura di Servizio dell'Interrupt**: Estrazione delle informazioni dai registri del controller del dispositivo che ha generato l'interrupt.
7. **Scelta del Processo Successivo**: Determinazione di quale processo eseguire come successivo, potenzialmente uno con priorità alta sbloccato dall'interrupt.
8. **Impostazione del Contesto per il Nuovo Processo**: Impostazione del contesto della MMU e potenzialmente del TLB per il processo successivo.
9. **Caricamento dei Nuovi Registri del Processo**: Caricamento dei registri, inclusi PSW, del processo successivo.
10. **Avvio del Nuovo Processo**: Inizio dell'esecuzione del processo selezionato.

YOU ARE HERE


Software per l'I/O a livello utente
Software del sistema operativo indipendente dal dispositivo
Driver dei dispositivi
Gestori degli interrupt
Hardware

DRIVER DI DISPOSITIVO - INTRODUZIONE E RUOLO

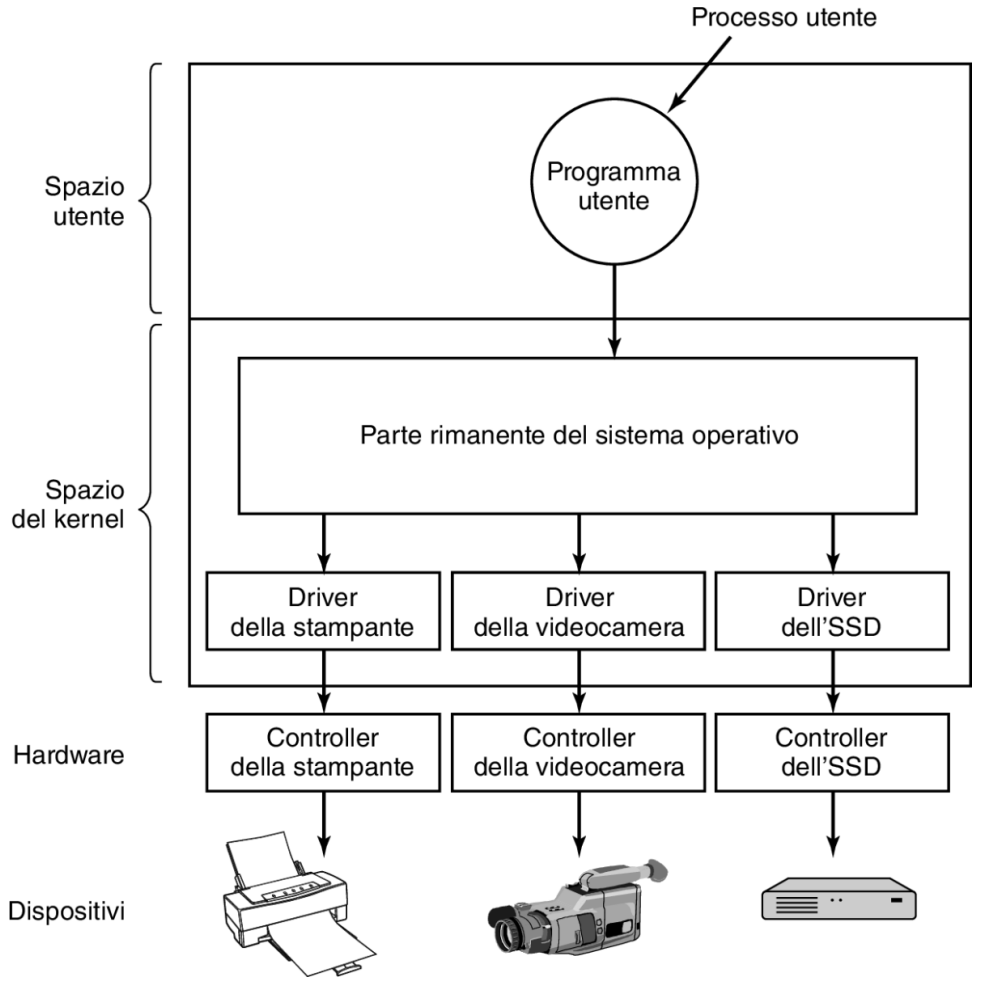
- **Ruolo dei Driver di Dispositivo**: Gestiscono i dispositivi di I/O attraverso registri di dispositivi specifici.
  - **Diversi per ciascun tipo** di dispositivo (es. mouse vs disco rigido), al più gestiscono un tipo o una classe di dispositivi correlati (ma spesso un unico dispositivo).
  - **Ogni dispositivo necessita di un codice specifico**, noto come driver di dispositivo, solitamente fornito dal produttore.
- **Esempi di Tecnologie Basate su Driver Comuni**: Tecnologie come USB utilizzano una pila di driver per gestire una vasta gamma di dispositivi.
  - Livelli diversi per gestire aspetti specifici dei dispositivi USB, ogni livello «parla» con il livello inferiore
  - **Livello di Base**: Gestione dell'I/O seriale e delle questioni hardware.
  - **Livelli Superiori**: Trattano pacchetti dati e funzionalità comuni condivise dalla maggior parte dei dispositivi USB.
  - **API di Alto Livello**: Forniscono interfacce specifiche per diverse categorie di dispositivi.

DRIVER DI DISPOSITIVO - INTRODUZIONE E RUOLO

- **Posizionamento nel Kernel:**
  - I driver di solito fanno parte del kernel del sistema operativo per poter accedere ai registri del controller del dispositivo.
  - **Se usati nello spazio utente**
    - Più facili da installare
    - Mettono meno «a rischio» il Sistema Operativo
    - Più lenti (occorre passare allo spazio kernel per ogni operazione)

FUNZIONALITÀ E INTERFACCIA DEI DRIVER DI DISPOSITIVO

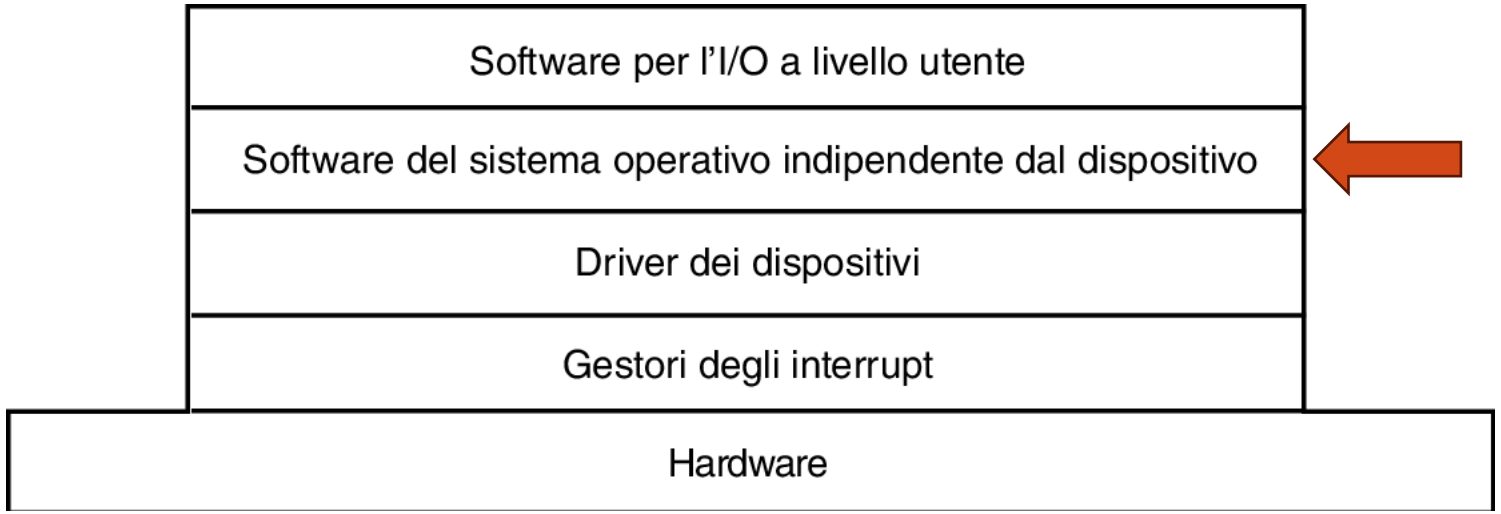
- **Interfaccia con il Sistema Operativo:** Il sistema operativo deve permettere l'installazione di codice scritto da terze parti (driver).
  - I driver si posizionano sotto il resto del sistema operativo.
  - Ogni categoria ha un'interfaccia standard che i driver devono supportare.
- **Classificazione dei Driver**: I sistemi operativi classificano i driver in categorie come dispositivi
  - **a blocchi**: come i dischi, contenenti molteplici blocchi di dati indirizzabili indipendentemente
  - **a caratteri**: come stampanti e tastiere, che generano o accettano un flusso di caratteri
- **Caricamento dei Driver:**
  - In alcuni sistemi, i driver sono inclusi nel programma binario del sistema operativo.
    - Aggiunto un dispositivo nuovo il kernel andava ricompilato!!!
  - Nei sistemi moderni, i driver vengono caricati dinamicamente.



IMPLEMENTAZIONE E COMPLESSITÀ DEI DRIVER DI DISPOSITIVO

- **Funzioni dei Driver**:
  - Gestione di letture e scritture, inizializzazione del dispositivo, gestione dell'alimentazione e del registro degli eventi.
- **Processo Generale di un Driver**:
  - Verifica della validità dei parametri di input, traduzione dei parametri in comandi specifici per il dispositivo, e gestione dell'uso del dispositivo.
- **Gestione dell'I/O e Errori**:
  - I driver gestiscono l'I/O e controllano eventuali errori. In alcuni casi, un driver deve aspettare l'interrupt per completare l'operazione.
- **Complessità e Rientranza dei Driver**:
  - I driver devono essere rientranti per gestire più richieste simultaneamente
    - **Esempio:** mentre sta gestendo un pacchetto di informazioni il driver viene richiamato anche per un altro pacchetto.
  - Gestione delle situazioni complesse come l'aggiunta o la rimozione di dispositivi in sistemi "*hot pluggable*".
    - **Esempio:** Se viene disconnesso un dispositivo mentre si sta leggendo/scrivendo il sistema operativo deve «ripulire» tutte le operazioni in corso e impedire nuove richieste al dispositivo assente.

YOU ARE HERE


Software per l'I/O a livello utente
Software del sistema operativo indipendente dal dispositivo
Driver dei dispositivi
Gestori degli interrupt
Hardware

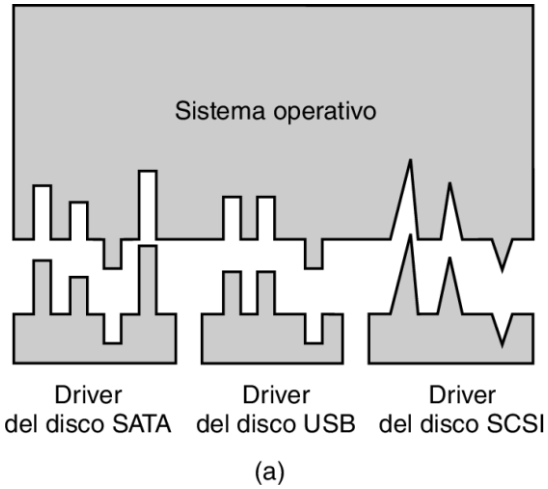
SOFTWARE DI I/O INDIPENDENTE DAL DISPOSITIVO: RUOLO E FUNZIONI

- **Ma il software per I/O dipende sempre dal dispositivo?**
  - Il **software di I/O indipendente dal dispositivo funge da intermediario** tra i driver specifici dei dispositivi e le applicazioni utente.
  - Mira a **semplificare l'interazione con i dispositivi** hardware offrendo un'interfaccia uniforme e gestendo operazioni comuni.

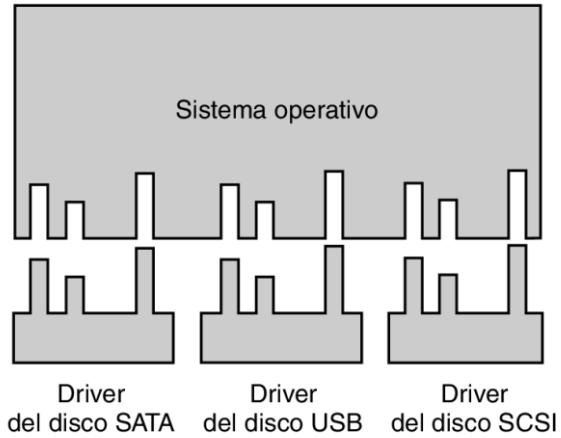
- **Funzioni Chiave:**
  1. **Interfaccia Uniforme dei Driver dei Dispositivi**: Fornisce un'interfaccia standard per diversi tipi di driver di dispositivo.
  2. **Buffering**: Gestisce i buffer per l'efficienza del trasferimento dei dati tra i dispositivi e il sistema.
  3. **Segnalazione degli Errori**: Identifica e comunica gli errori provenienti dai dispositivi all'utente o ad altri sistemi.
  4. **Allocazione e Rilascio dei Dispositivi Dedicati**: Gestisce l'assegnazione e la liberazione di dispositivi dedicati a specifici compiti o utenti.
  5. **Dimensione dei Blocchi Indipendente dal Dispositivo**: Assicura che la dimensione dei blocchi di dati sia gestita in modo uniforme, indipendentemente dalla specificità del dispositivo.

UNIFORMITÀ NELL'INTERFACCIA DEI DRIVER DEI DISPOSITIVI

- **Necessità di Uniformità**:
  - Evita la necessità di modificare il sistema operativo ogni volta che viene introdotto un nuovo dispositivo.
  - Importante per mantenere la consistenza e l'efficienza nel sistema.
- **Interfacce Diverse (a) vs Interfaccia Standard (b)**:
  - Problema con diversi driver aventi interfacce uniche verso il sistema operativo.
  - **Soluzione**: un modello uniforme dove tutti i driver condividono la stessa interfaccia.
- **Definizione di Funzioni per Classe di Dispositivi**:
  - Ogni classe di dispositivi ha un insieme definito di funzioni che i driver devono supportare (es. operazioni di lettura/scrittura per dischi).



(a)



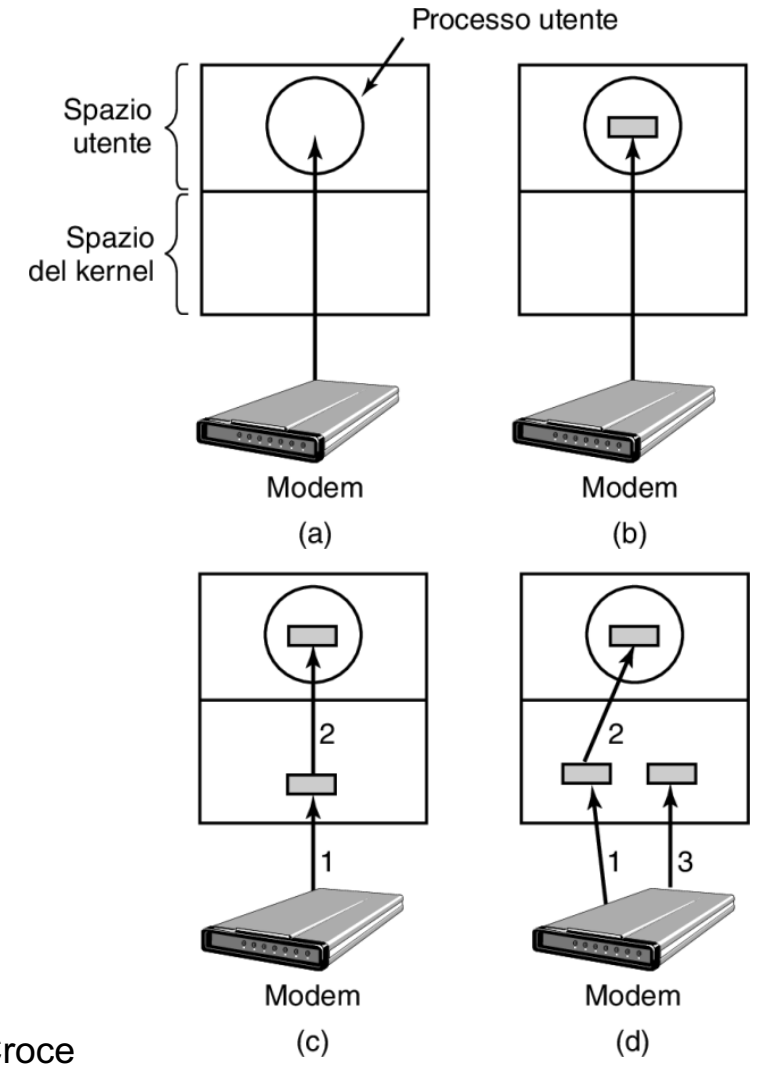
(b)

IMPLEMENTAZIONE E GESTIONE DELL'INTERFACCIA DEI DRIVER

- **Tabella di Puntatori a Funzioni nel Driver**:
  - I driver includono una tabella con puntatori a funzioni richieste, utilizzata dal sistema operativo per facilitare chiamate indirette.
- **Uniformità nella Denominazione e Protezione dei Dispositivi**:
  - Mappatura dei nomi simbolici dei dispositivi ai driver corrispondenti (es. `/dev/disk0` in UNIX).
  - Gestione dei permessi e protezione dei dispositivi simile a quella dei file, consentendo un controllo amministrativo appropriato.
- **Interfaccia e Integrazione nel Sistema**:
  - Questa struttura fornisce un'interfaccia coesa fra i driver e il resto del sistema operativo, semplificando l'integrazione di nuovi dispositivi.

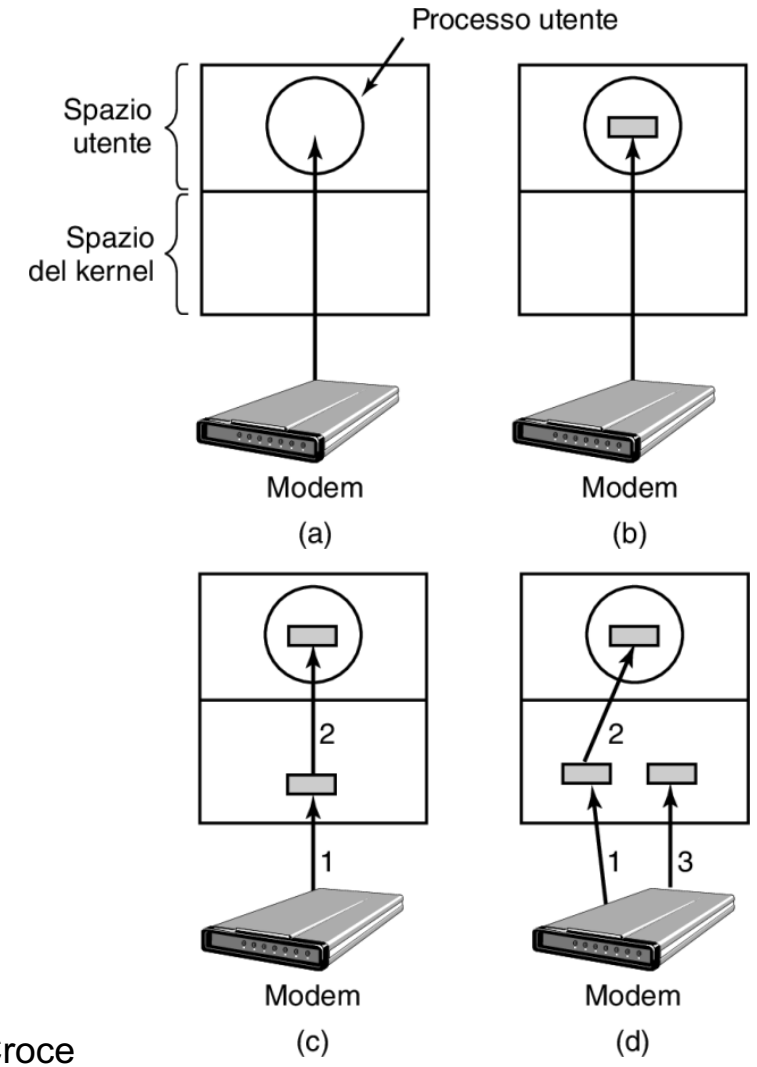
BUFFERING E LA SUA NECESSITÀ NEI DISPOSITIVI DI I/O

- **Scenari di Buffering in Input**:
  - **Esempio di lettura dati da un modem VDSL:** l'input senza uso di buffer (a) è inefficiente poiché richiede un riavvio del processo utente per ogni carattere ricevuto.
  - **Miglioramento con buffer nello spazio utente** (b): il processo fornisce un buffer e si blocca solo quando è pieno.
  - **Problemi di paginazione e soluzione con buffer nel kernel** (c): il buffer nel kernel accumula i caratteri, riducendo il riavvio del processo utente.
- **Doppio Buffer nel Kernel** (d):
  - Soluzione per gestire i caratteri in arrivo durante la lettura del buffer utente dal disco.
  - Utilizzo di due buffer nel kernel che si alternano: uno accumula nuovi input mentre l'altro è in copia nello spazio utente.



BUFFERING IN OUTPUT E LA SUA COMPLESSITÀ

- **Buffering in Output**:
  - Esempio di output su un modem: senza buffering (analogamente a b), il processo utente può restare bloccato per lungo tempo.
  - Soluzione con buffer nel kernel: copia dei dati in un buffer del kernel e sblocco immediato del processo utente.
- **Problemi e Complessità del Buffering**:
  - Il buffering multiplo può influire negativamente sulle prestazioni, come illustrato nella rete
  - Processo di copia multi-stadio: dal buffer utente al kernel, poi al controller, successivamente sulla rete, e infine al buffer del kernel e processo ricevente.
- **Impatto del Buffering sulla Velocità di Trasmissione**:
  - Le molteplici copie richieste per la trasmissione di pacchetti rallentano la velocità effettiva di trasmissione.
  - Sequenzialità delle operazioni di copia aumenta il tempo complessivo di trasmissione.



GESTIONE DEGLI ERRORI DI I/O NEL SISTEMA OPERATIVO

- **Frequenza e Tipi di Errori di I/O**: Gli errori di I/O sono comuni e variano da errori di programmazione a veri errori di I/O.
  - Errori di programmazione includono azioni come la scrittura su un dispositivo di input o la lettura da un dispositivo di output.
- **Risposta ai Diversi Errori**:
  - Errori di programmazione: Ritornano un codice d'errore al processo chiamante.
  - Veri errori di I/O (es. scrittura su un blocco danneggiato): Gestiti dal driver o, se non risolvibili, passati al software indipendente dal dispositivo.
- **Azioni Dipendenti dal Contesto**: In presenza di un utente interattivo: Possibilità di dialogo per scegliere come gestire l'errore (riprova, ignora, termina processo).
  - Senza utente interattivo: La chiamata di sistema fallisce restituendo un codice d'errore.
- **Gestione degli Errori Critici**: In caso di danneggiamento di strutture dati critiche: Visualizzazione di un messaggio d'errore e possibile terminazione del sistema.

GESTIONE DEI DISPOSITIVI DEDICATI NEL SISTEMA OPERATIVO

- **Uso Esclusivo di Alcuni Dispositivi**:
  - Dispositivi come stampanti richiedono l'uso esclusivo da un singolo processo alla volta.
- **Gestione delle Richieste di Uso**:
  - Il sistema operativo valuta le richieste per l'uso del dispositivo, accettandole o rifiutandole a seconda della disponibilità del dispositivo.
- **Metodi di Allocazione e Rilascio**:
- **Approccio Tradizionale**:
  - I processi eseguono una 'open' su file speciali per i dispositivi e la 'close' del dispositivo rilascia il file.
- **Approccio Alternativo**:
  - Meccanismi speciali per richiedere e rilasciare dispositivi: un tentativo di acquisizione non riuscito causa il blocco del processo richiedente.
  - I processi bloccati sono inseriti in una coda e acquisiscono il dispositivo quando diventa disponibile.

UNIFORMITÀ NELLA DIMENSIONE DEI BLOCCHI NEI DISPOSITIVI DI I/O

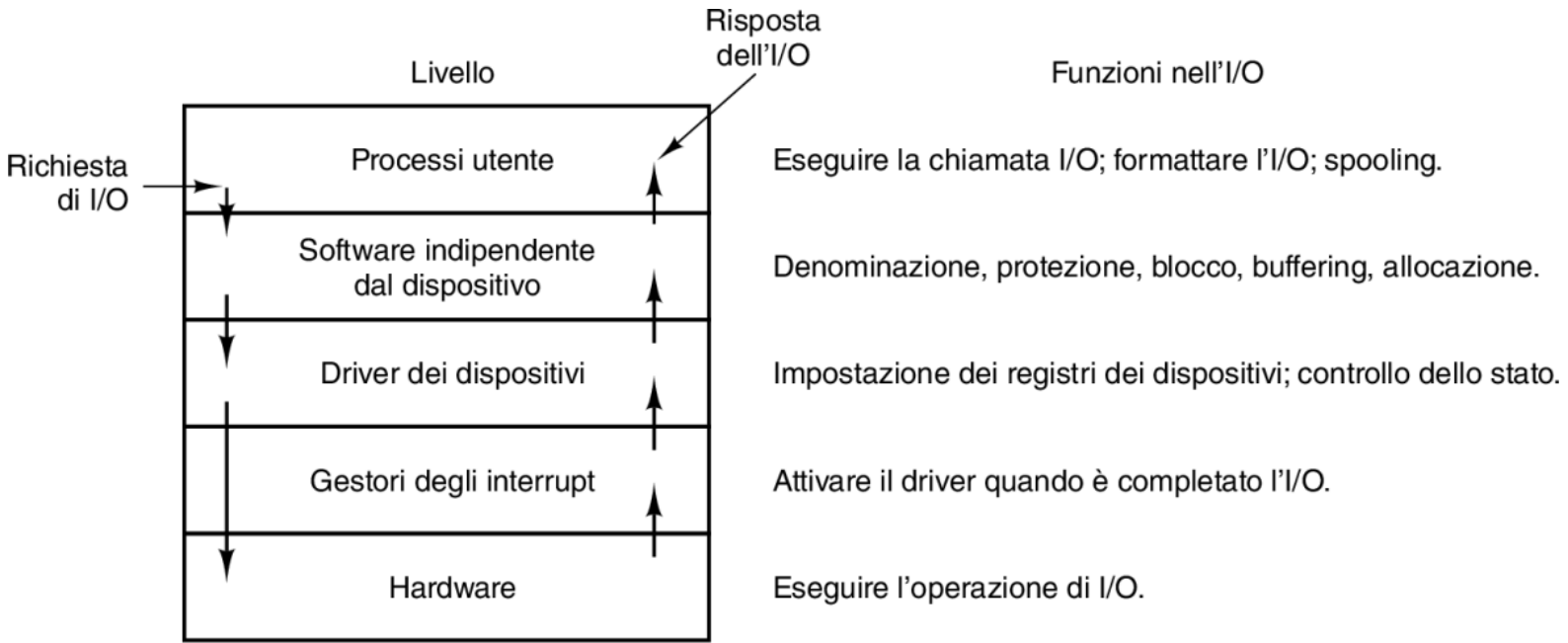
- **Variabilità nelle Dimensioni Fisiche**:
  - Dispositivi come SSD e dischi rigidi presentano dimensioni fisiche di blocchi o settori variabili.
  - Anche i dispositivi a caratteri possono differire nella quantità di dati che trasmettono per volta.
- **Ruolo del Software Indipendente dal Dispositivo**:
  - Nasconde le differenze fisiche nelle dimensioni dei blocchi o settori tra diversi dispositivi.
  - Fornisce una dimensione di blocco logico uniforme ai livelli superiori del sistema.
- **Creazione di Dispositivi Astratti**:
  - Trasforma più settori o pagine flash in un unico blocco logico.
  - Permette ai livelli superiori di interagire con dispositivi "astratti" che utilizzano una dimensione di blocco logico standard, indipendentemente dalle dimensioni fisiche.
- **Occultamento delle Differenze nei Dispositivi a Caratteri**:
  - Gestisce la varianza nella quantità di dati trasmessi dai dispositivi a caratteri (es. mouse vs interfacce di rete), rendendo queste differenze trasparenti ai livelli superiori.

LIBRERIE DI I/O NELL'AMBITO UTENTE

- **Ruolo delle Librerie di I/O**: Facilitano le chiamate di sistema per l'I/O, esemplificato dal metodo `write(fd, buffer, nbytes)` in C.
- **Funzioni di Libreria per Formattazione**: `printf()` e `scanf()` trasformano e gestiscono i dati prima di invocare funzioni di sistema, facilitando operazioni di input e output.
- **Importanza nelle Applicazioni**: Queste librerie semplificano la programmazione di I/O, permettendo ai programmatori di concentrarsi sulla logica dell'applicazione piuttosto che sui dettagli di basso livello delle operazioni di I/O.

SISTEMA DI SPOOLING PER DISPOSITIVI DEDICATI

- **Definizione di Spooling**: Tecnica per gestire dispositivi dedicati in ambienti multiprogrammati, evitando il blocco prolungato da parte di un unico processo.
- **Implementazione Pratica**: Utilizzo di un processo daemon e una directory di spooling, come mostrato in Figura, per ordinare e gestire i lavori di stampa.
- **Benefici del Spooling**: Incrementa l'efficienza nell'uso dei dispositivi dedicati e migliora la gestione delle risorse, permettendo a più utenti o processi di accedere ai dispositivi in modo controllato e sequenziale.



FLUSSO DEL CONTROLLO NEL SISTEMA DI I/O

- **Livelli del Sistema di I/O**: Dall'hardware ai processi utente, come rappresentato in Figura.
- **Interazione e Flusso di Controllo**: Descrive come una richiesta di I/O, ad esempio la lettura di un blocco da un file, attraversa diversi livelli (hardware, gestori degli interrupt, driver dei dispositivi).
- **Gestione delle Richieste di I/O:** Spiega il processo dalla richiesta iniziale all'intervento degli interrupt e al successivo risveglio del processo utente, enfatizzando il ruolo cruciale di ogni livello nel trattamento efficiente delle operazioni di I/O.