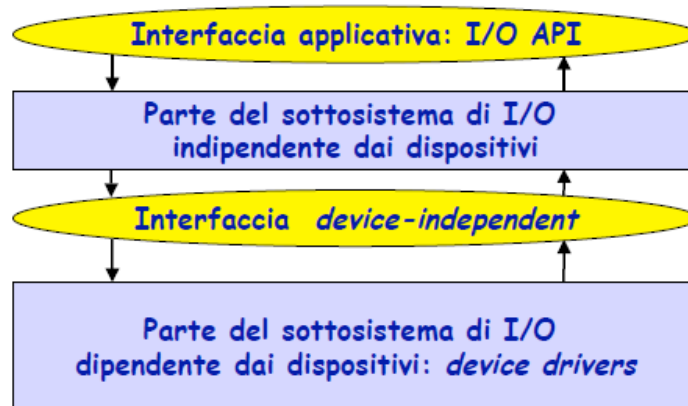


Gestione periferiche I/O

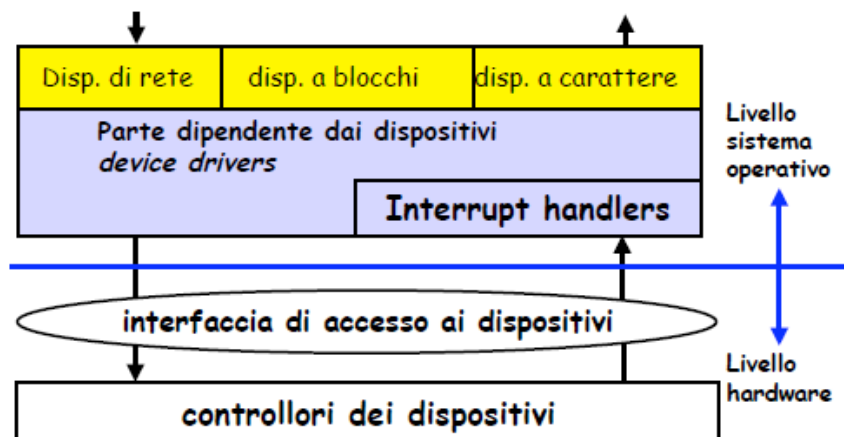
Compiti del sottosistema IO

- Nascondere al programmatore i dettagli delle interfacce hw e dei dispositivi (*utilizzo di controllori*)
- Omogeneizzare la gestione di dispositivi diversi
 - Velocità di trasferimento diverse
 - Tipologie di dispositivi (*a carattere, a blocchi, speciali* come il timer)
- Gestione i malfunzionamenti che si possono verificare durante un trasferimento di dati
 - Eventi eccezionali (es. mancanza carta, EOF)
 - Guasti transitori (es. disturbi elettromagnetici)
 - Guasti permanenti (es. rottura testina)
- Definire lo spazio dei nomi (*naming*) con cui vengono identificati i dispositivi
 - Nomi unici (numerici) per identificazione nel sistema
 - Nomi simbolici da parte dell'utente
 - Uniformità col meccanismo di naming del file system
- Garantire la corretta sincronizzazione tra processo applicativo che ha attivato un trasferimento e l'attività del dispositivo
 - Gestione sincrona (processo bloccato fino al termine del trasferimento)
 - Gestione asincrona
 - Necessità di *bufferizzazione dati*

Architettura



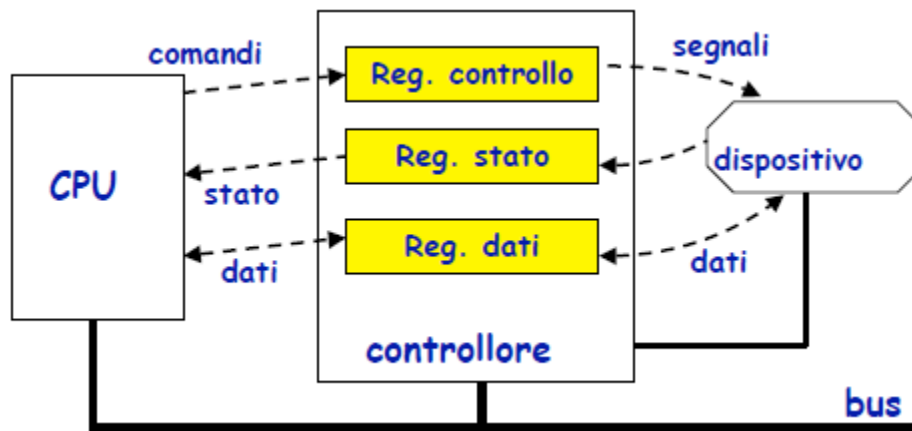
Architettura parte dipendente dai dispositivi



Funzioni:

- Fornire i device drivers
- Offrire l'interfaccia device – independent (es. read)

Gestore di un dispositivo



Gestore a polling e a interrupt

```
// *****  
// DEVICE DRIVER A POLLING  
// *****  
// Utilizzo dei bit di stato nel REG_STATO  
// Attesa attiva  
process esterno  
{  
  while (true)  
  {  
    while(start == 0);  
    << comando >>;  
    << registro esito del comando >>;  
    flag = 1;  
  }  
}  
  
process applicativo  
{  
  << prepara il comando >>;  
  << invia il comando >>;  
  start = 1;  
  while (flag == 0);  
  << utilizzo esito >>;  
}  
  
// *****  
// DEVICE DRIVER A INTERRUPT  
// *****  
// Un semaforo inizializzato a 0 per periferica  
// Abilitare le interruzioni del dispositivo (REG_CONTROLLO)  
semaphore datoDisponibile = 0;  
  
process applicativo  
{  
  << prepara il comando >>;  
  << invia il comando >>;  
  P(datoDisponibile);  
  << utilizzo esito >>;  
}
```

```

InterruptHandler
{
    // ...
    V(datoDisponibile);
    // ...
}

```

Descrittore di dispositivo

- Indirizzi registri di controllo, stato e dati
- Semaforo dato Disponibile
- Contatore dati da trasferire (così riattivo il processo solo al termine dei trasferimenti)
- Puntatore al buffer di memoria
- Esito del trasferimento

Funzione di lettura

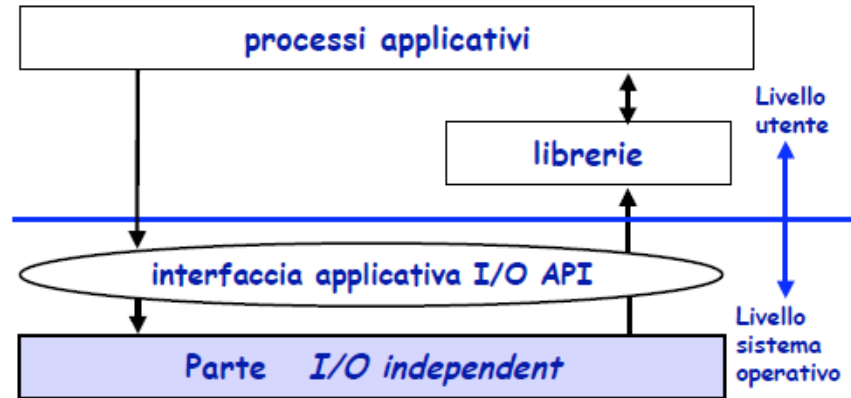
```

int _read(int dispositivo, char* buffer, int counter)
{
    descrittore[dispositivo].contatore = counter;
    descrittore[dispositivo].buffer = buffer;
    << attivazione dispositivo >>;
    P(descrittore[dispositivo].datoDisponibile);
    if (descrittore[dispositivo].esito == << errore >>)
        return -1;
    else
        return counter - descrittore[dispositivo].contatore;
}

void InterruptHandler()
{
    char b;
    << leggo registro di stato >>;
    if (<< bit di errore >> == 0)
    {
        // Ramo normale
        b = << registro dati >>;
        *(descrittore[dispositivo].buffer) = b;
        descrittore[dispositivo].buffer++;
        descrittore[dispositivo].contatore--;
        if (descrittore[dispositivo].contatore != 0)
            << riattivazione dispositivo >>;
        else
        {
            descrittore[dispositivo].esito = << terminazione corretta >>;
            << disattivazione dispositivo >>;
            V(descrittore[dispositivo].datoDisponibile);
        }
    }
    else
    {
        // Ramo eccezionale
        << gestione errore >>;
        if (<< errore non recuperabile >>)
        {
            descrittore[dispositivo].esito = << terminazione con errore >>;
            V(descrittore[dispositivo].datoDisponibile);
        }
    }
}

```

Architettura parte indipendente dai dispositivi



Funzioni:

- **Naming**
- **Buffering:** area di memoria tampone per contenere i dati oggetti di trasferimento
 - Gestione differenti velocità processo e periferica
 - Quantità di dati da trasferire (es. per dispositivi a blocchi un dispositivo può chiedere di trasferire meno dati del blocco)
- **Gestione malfunzionamenti**
 - Soluzioni: risoluzione del problema (mascheramento) o propagazione a livello applicativo dopo gestione parziale
 - Tipi: eventi propagati dal livello inferiore (es. guasto hw permanente), eventi di questo livello (es. accesso a dispositivo inesistente)
- **Allocazione dei dispositivi ai processi**
 - Dispositivi condivisi e mutua esclusione
 - Dispositivi dedicati a cui i client possono inviare messaggi
 - Spooling

Temporizzatore

- Funzioni
 - Aggiornamento data
 - Gestione quanto di tempo
 - Valutazione impegno CPU
 - Gestione system call ALARM
 - Gestione time-out (watchdog timer)
- Per ogni timer c'è un registro contatore inizializzato da CPU e decrementato dal timer e quando raggiunge 0 parte l'interrupt alla CPU
- Nel descrittore
 - N semafori privati per bloccare un processo che chiama la delay
 - N contatori per sapere quanti quanti di tempo prima di riattivare un processo

Dischi

- **Indirizzo di un blocco fisico:** (f, t, s)
 - f = numero faccia
 - t = numero tracci sulla faccia
 - s = numero settore entro la faccia
 - Tutti i settori di un disco sono trattati come array
 - M tracce per faccia

- N settori per traccia
- Un settore (f, t, s) ha indice $f * M * N + t * N + s$

- **Calcolo dei tempi**

- TF = tempo medio di trasferimento per leggere o scrivere un settore
- TA = tempo medio per posizionare la testina all'inizio del settore
- TT = tempo di trasferimento dei dati del settore
- ST = tempo di seek, per posizionare la testina sulla traccia desiderata
- RL = rotational latency, per posizionarci su una traccia in quel settore
- T = tempo per compiere un giro completo
- S = numero di settori per traccia

$$TF = TA + TT$$

$$TA = ST + RL$$

$$TT = t / s$$

$$TF = ST + RL + TT$$

- **Miglioramento del tempo di accesso**

- Intervento sul *metodo di allocazione dei file*
- Intervento sulle *politiche di scheduling delle richieste*

- **Politiche di scheduling delle richieste**

- First Come First Served: servono rispettando il tempo di arrivo, non c'è starvation ma nessuna ottimizzazione
- Shortest Seek Time First: vado da chi richiede tempo di seek minimo dalla posizione attuale; può causare starvation
- SCAN Algorithm: la testina si porta ad una estremità del disco e si sposta verso l'altra estremità, servendo le richieste man mano che viene raggiunta una traccia, poi torna indietro.
- C-SCAN (Circular SCAN): fornisce un tempo di attesa più uniforme, arrivato alla fine del disco, la testina torna all'inizio.

RAID

- **Obiettivo:** unico disco virtuale caratterizzato da grande capacità, alta velocità di ingresso e alta affidabilità
- **RAID0:** parallelizzazione accessi
- **RAID1:** duplicazione dei dati
- **RAID2:** correzione di un errore, rilevazione di 2, correzione su ogni disco
- **RAID3:** bit di parità per ogni parola su un disco diverso
- **RAID4:** bit di parità per settore su un disco diverso
- **RAID5:** bit di parità per settore su vari dischi

