# Ch12- I/O System

Servono diversi pezzi di sw per gestire i dispositivi di IO → DEVICE TRACKERS

* da chi produce i dispositivi di IO

**Tipi di dispositivi** che abbiamo:

* Storage
* Trasmissione
* Interfaccia con persone

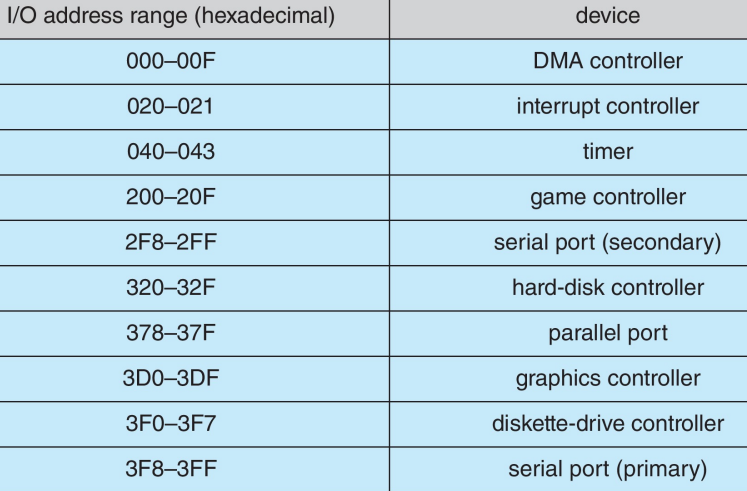
Tutti questi **dipositivi hanno**:

* Immagine che contiene testo, schermata, diagramma, schermo

  Descrizione generata automaticamente**Porte**: dove si connette il dispositivo
* **Bus**: canali di comunicazione
  + **PCI:** bus molto veloce che tra gli altri collega processore, parte grafica
  + **Expansion bus:** connette dispositivi relativamente lenti (tastiera, porte usb)
  + **SAS:** interfaccia comune per i dischi, memorie di massa
* **Controller**: si affaccia al dispositivo e parla con il bus e con il micro-processore

Dal punto di vista **hw I/O**: possono essere semplici ma anche molto complicati; in generale, dal punto di vista della CPU, si tratta di vederli come dei registri o delle celle di memoria che rispondono a determinati indirizzi:

* **Dispositivo I/O -mapped**
  + All’indirizzo 1000 risponde un indirizzo di IO
  + Ma c’è anche un indirizzo 1000 per la ram
  + Quindi se scrivi qualcosa all’indirizzo 1000 sull’address bus, c’è qualcosa che potenzialmente può rispondere. Per decidere chi, ci va un altro filo che indica chi dei due ha risposto
    - In base ad istruzione usata si va quindi a settare il valore corretto su quel filo
* **Memory mapped** → dispositivo che risponde a delle istruzioni
  + Può essere messo ad indirizzo 10000, se messo indirizzo 10000 e risponde load/Store ad indiirizzo 10000, vuol dire che a quell’indirizzo non c’è ram, c’è dispositivo di IO



Non c’è bisogno di tanti indirizzi di I/O → bastano indirizzi da 10/12 bit

**Disp di IO hanno**:

* Dato-in
* Dato-out
* Status
* Controll → programmare o chiedere qualcosa al dispositivo

### Polling vs interrupt:

#### POLLING

* Loop
* Se scrivo il sw ok, se penso a prestazioni no
* Per ogni byte di I/O
  + Controllare il busy bit dal registro di stato
    - Se 1, looppa continuando a leggere
  + Se libero:
    - Seleziona l’operazione (read o write)
      * Read → dato arriva dal dispositivo
      * Write → dico che voglio write e metto dato sul dispositivo
        + Scrivo su data-out
  + Setto il bit di command-ready ad 1
  + Controller setta busy bit ad 1
  + Pulisco il controller, busy bit, errori bit, pulisco command-ready → trasferimento finito

Dunque, si ha BUSY WAIT nel primo punto → va bene per dispositivi veloci che attenderanno poco, non per dispositivi lenti

* + Ok per dispositivi veloci
  + Non ok per dispositivi lenti → resto bloccato troppo tempo

#### Immagine che contiene testo, schermata, Carattere, diagramma Descrizione generata automaticamenteINTERRUPT

* **reagisce ad un evento asincrono**
  + c’è un filo che arriva alla cpu che viene verificato in un momento particolare alla fine di un’istruzione
    - se interrupt attivo → alla prossima istruzione innesca il protocollo di interruzione
      * salva contesto e attiva altro programma
* **interrupt handler** → gestore di interrupt attivato quando viene riconosciuto un interrupt
* **interrupt vector** → vettore di interrupt che viene acceduto tramite un numero ricevuto dall’interrupt → numero fornisce informazioni sul tipo di interrupt per fare in modo che la cpu sappia cosa fare
  + dipatcha interrupt al gestore corretto
  + si basa su priorità
  + Concatenamento di interrupt se più di un dispositivo ha lo stesso numero di interrupt

Dunque, una parte dell’interrupt viene gestita a livello generico ed un’altra parte sarà compito del device driver.

Gli interrupt sono **usati anche per**:

* Eccezioni:
  + Terminazione processo, crash del sistema a causa di errore hw
* Trap: esecuzione system calls

## DMA: Directory Memory Acess

Immagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamenteDispositivo che serve ad aiutare la cpu a fare I/O verso dispositivi a blocchi:

* serve ad evitare l’IO programmato
* prende il controllo del bus dalla cpu
  + prende i dati e ci pensa lui
    - trasferisce direttamente da dispositivo di IO alla memoria
* SO dice al **DMA controller** cosa vuole
  + Indirizizzo destinazione e arrivo
  + Modalità read/write
  + Numero di byte
    - Dma controller ruba effettua **Cycle stealing** → ruba cicli alla cpu, gestione a livello HW.

#### Immagine che contiene testo, schermata, numero, Carattere Descrizione generata automaticamenteINTERFACCIA A LIVELLO APPLICAZIONE

* System call di IO incapsuo le caratteristiche del dispositivo in una classe generica
* Ogni OS ha la propria struttura di sottosistema di IO
* Immagine che contiene testo, schermata, Carattere, Blu elettrico

  Descrizione generata automaticamenteIO può essere

**Dispositivi di IO possono essere divisi in base al OS** in:

* **A blocchi**
  + Accessibili con comandi read write
  + Hanno anche modalità IO diretto o IO raw o accesso filesystem
  + Si può fare memory mapping
  + Si può usare DMA
* **A caratteri**
  + Rispondono a get, put
* **Memory mapped**
* **Network socket**
  + Separa protoclli di rete da operazioni di rete
  + Include la funzionalità select()

**Nota**: Tutti i dipositivi hanno:

* Categoria → condivisa da tutti i dispositivi di una certa categoria
* Immagine che contiene testo, Carattere, bianco

  Descrizione generata automaticamentesottocategoria (sda1, sda2 …) che li distingue tra loro

Nota: **Ioctl** → funzione che serve per porgrammare il dispositivo di IO

### Clock and Timers:

Dispositivi particolari che non trasferiscono dati:

* **Timer**: possono generare
  + Segnari periodici → programmable interval timer
  + Segnali one-shot → fai passare tot tempo e poi avvisami

## IO:

* **Blocking**
  + Immagine che contiene testo, schermata, linea, Carattere

    Descrizione generata automaticamenteProcesso aspetta finchè IO non completato
    - Facile
    - Non sempre accettabile
* **Non blocking**
  + Non aspetta
  + Dopo che si fa la richiesta, c’è un ritorno subito
    - Istruzione termina quando l’IO non è ancora completato
      * Potrebbe esserci già un pezzo di risposta o anche no
      * Può ritornare con il numero di dati disponibili nel caso in cui ci sia un meccanismo che lo permetta
  + Istruzione torna un lavoro non finito, per ottenere successivamente i dati, bisogna eseguire altre informazioni che permettono di completare
    - Si può usare se chiedo i dati e mi servono dopo un po’ e in quello specifico momento vado a rileggerli
* **Asincrono** 
  + Non bloccnte dove però è ben chiaro come completare l’operazione nelle fasi successive
  + Puoi continuare a fare qualcosa e poi aspettare che qualcosa termini
  + È un I/O che non aspetti ma puoi sincronizzarti dopo
    - * Asincrono costa un po’ più di non blocking ma più rifinito
* **Vettorizzato**:
  + Read speciale in cui invece di passare un solo buffer, ne passo più di uno
    - Può essere utile per schedulare il disco passando più ricerche in un colpo solo perhcè permette alle system call di fare operazioni di I/O multiple
      * Es: readve che accetta un vettore di buffers multipli da leggere

## KERNEL i/O SUBSYSTEM

* **Scheduling**
  + Ordinamento di alcune richieste di I/O tramite coda per dispositivo
    - Alcuni sistemi operativi cercano l'equità
    - Alcuni implementano la qualità del servizio (ad esempio IPQOS)
* **Buffering**: strategia importante per **salvare dati in memoria** quando avviene trasferimento tra dispositivi
  + È un passaggio in più
  + Se devi andare da A a C, parcheggia prima in B e poi fai un altro step.
    - È un trasferimento in più
  + Serve a disaccoppiare sorgente e destinazione
    - Se destinazione non è pronta, metti in buffer
    - A regime le velocità dei mittenete e ricevitore devono essere uguali
      * Ma nel mentre, sono ammesse diverse
  + **DOPPIO BUFFER:**
    - Mentre C scarica il buffer che A ha scritto prima, A può scrivere su un secondo buffer
    - (con singolo buffer, mente A scrive, C non può leggere; se C legge, A non può scrivere)
* **Caching**: Dispositivo veloce che conserva la copia dei dati
  + *Semplice copia che permette di migliorare le prestazioni*
  + *In alcuni casi usato insieme a buffering*
* **Spooling**: mantiene l’uscita per un dispositivo
  + caratteristica dedicata a stampanti
    - Possono essere condivise con coda di stampa
* **Reservation**: fornisce accesso esclusivo al dispositivo
  + Dispositivi che possono essere allocati e deallocati

Nota: una read non è vincolata a leggere il blocco

* Dato potrebbe esse anche in cache

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Nota: in molti casi un’istruzione di IO può essere considerata finita e quindi passata a prossima istruzione solo quando IO è completato

Nota: un’operazione di lettura/Scrittura ha un solo indirizzo:

* alcune cpu hanno delle mem to mem da indirizzo ad indirizzo

**Nota: per migliorare le performance:**

* Ridurre il numero di cambi di contesto
* Ridurre la copia dei dati
* Ridurre gli interrupt utilizzando trasferimenti di grandi dimensioni, controller intelligenti, Polling
* Utilizzare DMA
* Utilizzare dispositivi hardware più intelligenti
* Bilanciare le prestazioni di CPU, memoria, bus e I/O per la massima velocità effettiva
* Immagine che contiene testo, schermata, Carattere

  Descrizione generata automaticamenteSposta i processi/demoni in modalità utente nei thread del kernel

Immagine che contiene testo, schermata, diagramma

Descrizione generata automaticamente