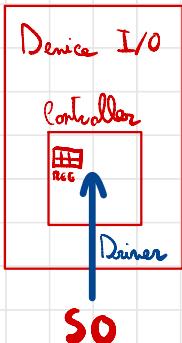


3 dispositivi di I/O sono costituiti da due parti

- un controller  $\Rightarrow$  è un chip che accetta comandi dal SO e fa



interfaccia al dispositivo (es. leggere i dati del device o trasferirli). Nel controller c'è presente un piccolo computer che assembla i bit in parole e le salva in memoria.

- il dispositivo vero  $\Rightarrow$  essi hanno un interfaccia molto semplice in modo da poter essere STANDARD, quindi gestibile da ogni controller SATA.

Un controller è un piccolo calcolatore con la propria CPU e memoria che mantiene nel proprio buffer l'intera della system call e lo trasmette alla RAM.

Tutto quello che il sistema operativo needs è l'interfaccia del controller. Ogni controller è diverso dall'altro ed è gestito dal SO attraverso i driver del dispositivo.

Ogni controller ha dei registri che usa per comunicare. Per attivare il controller, il SO invia un comando al driver, che lo traduce nei valori appropriati da scrivere nei registri. La raccolta di tutti i registri dell'unità costituisce lo **registro di una porta di I/O**. L'input e l'output memory gestiti dal SO in tre modi diversi:

### 1) **Busy waiting**

Un programma invia una system call, il Kernel la traduce in una chiamata ad una procedura del driver appropriato. Il driver invia al dispositivo di I/O e attende che quest'ultimo si liberi (lo stato del dispositivo è visibile attraverso l'interrogazione di un opportuno bit).

Il SO, attraverso i driver I/O, legge continuamente dal controller la porta (**polling**) che contiene il bit che identifica che l'informazione è presente e pronta per essere trasferita in memoria.

Il polling è un'altra attesa che costituisce uno spreco di risorse in quanto spiega CPU per eseguire continuamente processi in Kernel mode e non si dedica ai processi dell'utente.

- Chiama chi di sistema l'intero (bloccanti)  $\Rightarrow$  condiziona I/O devicem
 

Bloccare il processo in quanto per proseguire l'esecuzione del programma non necessita di conoscere l'entro della richiesta di I/O
- Chiama chi non bloccanti
 

l'esecuzione del codice può proseguire indipendentemente dall'entro della system call

## 2) Interrupzioni

Il driver invia direttamente al dispositivo e richiede un'interruzione. Termina, a quel punto il driver ritorna.

Un interrupt-controller modifica la richiesta di interrupt remota obbligando la CPU ad eseguire continuamente un processo di polling.

## 3) Direct memory access (DMA)

Il DMA è un meccanismo che si occupa di accedere direttamente alla memoria centrale in modo asincrono e indipendente dalla CPU, consentendo quindi di non occupare cicli di CPU per il trasferito.

dati. Questo consente di snellire molto la gestione di un interrupt poiché trasferita i dati dalla memoria centrale di buffer che sono controller sono interessare CPU e SO, inviare tutto tramite hardware.

## Bus

Non esiste un solo bus ma ci sono molti bus specializzati con velocità di trasmissione e scopi differenti. Essi devono essere in grado di garantire un throughput che sia alla pari delle esigenze della CPU.

La CPU comunica con la memoria attraverso un bus DDR3 veloce con una periferica grafica esterna attraverso un bus PCIe e con tutti gli altri dispositivi attraverso un HUB sul bus DMA.

## => Software

Non esiste un sistema operativo universale in grandi contesti differenti mostrando esigenze differenti

### - SO per mainframe / server

sistemi dove i job (richieste) sono molto elevati e per riuscire a gestire deve avere elevatissime priorizzazioni

- continuità di CPU
- alto carico di I/O (scalabile)
- multiprogrammazione
- sistema stabile perché destinato allo Storage

### - SO per personal computer

sistemi per utilizzatori giornalieri, anche quelli recenti da multiprogrammazione e multiutente. Diversi processi possono essere in esecuzione contemporaneamente (anche in background nel caso di domande). Si introduce l'esigenza di avere delle GUI (graphical user interface) per facilitare l'utilizzo per l'utente.

### - SO per palme / smartphone

Anche qui si necessita di multiprogrammazione. Il contesto è comunque l'uso quotidiano e con risorse limitate (batteria) che richiede quindi l'amministrazione dell'utilizzo della CPU. Qui allora in esecuzione ulteriore della CPU.

### - SO per sistemi integrati (embedded)

Sono tutti quei sistemi operativi che hanno una scena specifica che non possono essere riprogrammati dall'esterno e che interagiscono con un hardware ad hoc. Non ne ha la capacità di aggiungere processi o di isolare l'esecuzione.

### - SO per sistemi real time

Sono sistemi che si occupano della gestione di attività in maniera tempestiva, l'impiego sono i contesti come catene di montaggio. Per i SO real time è necessario riuscire a controllare sensori e attuatori nella maniera più tempestiva possibile. Sono sistemi operativi nei quali non è necessario aggiungere sicurezza a causa della prevedibilità.