File System

I dispositivi di I/O sono componenti hardware mirati a fornire input e direzionare output; il SO deve gestire e controllare tutti i dispositivi e le loro operazioni.
I dispositivi possono essere divisi in due categorie: **device a blocchi** e **device a caratteri**. I primi operano su blocchi di dati di dimensione prefissata e l'accesso è random; i secondi ricevono o inviano flussi di byte e l'accesso e sequenziale. I dispositivi di rete non rientrano in nessuna delle due categorie.

1 Hardware

Un dispositivo generalmente consiste di due parti: il **dispositivo** stesso e il **controllore**. Il dispositivo comunica con il computer tramite il suo controller attraverso un punto di connessione, chiamato **porta**.

1.1 Controllori

Il controllore (o adattatore) è un chip o un insieme di chip che controllano fisicamente il dispositivo. Il controllore accetta comandi dal SO e li esegue. L'interfaccia tra il controllore e il dispositivo è standard ma è di basso livello; il controllore presenta un'interfaccia più semplice nascondendo i dettagli del dispositivo.
La CPU e i controllori comunicano tra loro sul bus di sistema. I computer moderni hanno multipli bus che definiscono diversi protocolli di comunicazione. Un controllore può essere un singolo chip nella scheda madre del computer oppure un circuito stampato che si inserisce nel computer tramite uno slot d'espansione. Alcuni dispositivi hanno i loro controllore (**controllori periferici**), che permettono al controllore del computer di inviare comandi ad alto livello al controllore periferico.

1.2 Interazione tra CPU e dispositivi

Nel modello di riferimento di sistema, CPU, memoria e dispositivi sono connessi su un singolo bus e comunicano tra di loro attraverso questo. Nel modello di riferimento del dispositivo, l'interfaccia è l'interfaccia che il controllore del dispositivo presenta al resto del sistema; la struttura interna varia da implementazione a implementazione ed implementa l'astrazione che il controllore presenta.
Il controllore dei dispositivo ha dei **registri di controllo** che vengono acceduti dalla CPU e che permettono l'interazione CPU-dispositivo: i **data registers** sono usati dalla CPU per controllare i trasferimenti di dati dal e al dispositivo, gli **status registers** sono usati dalla CPU per controllare lo stato del dipositivo, i **command registers** possono essere scritti dalla CPU per controllare cosa il dispositivo farà. Inoltre, molti dispositivi hanno un **buffer dati**, usato per memorizzare dati che non possono essere passati direttamente al SO o dati che stanno per essere scritti sul dispositivo.

1.3 Port-mapped, memory-mapped

Sono utilizzati due approcci per garantire l'interazione tra CPU e dipositivo: **I/O mappato a porte** e **I/O mappato a memoria**. I due approcci possono essere combinati.
COn I/O mappato a porta, ad ogni registro del dispositivo è assegnato un **numero di porta I/O**, l'insieme dei quali forma l'**I/O port space**, accessibile solo in modalità kernel. Vengono

usate istruzioni speciali e privilegiate per comunicare. Con questo approccio, non viene consumata memoria, non bisogna gestire il caching e non vanno prese misure speciali in caso di bus multipli. Con I/O mappato a memoria, i registri del dispositivo sono mappati nella memoria, e ad ogni registro è assegnato un indirizzo di memoria, di solito a tempo di boot. Le istruzioni sono istruzioni di trasferimento dati standard. Con questo approccio, lo sviluppo di driver è semplificato (possono essere sviluppati in C), non è richiesto nessun meccanismo di protezione addizionale per impedire ai processi utente di eseguire I/O a basso livello ed il design delle CPU può essere semplificato (ogni istruzione che può referenziare la memoria può anche referenziare i registri del dispositivo).

1.3.1 PIO, Interrupt-Driven

Ci sono tre approcci per eseguire le operazioni I/O: **I/O programmato**, **I/O guidato da interrupt** e **I/O tramite DMA**.
Con **I/O programmato** la CPU richiede un'operazione di I/O al controllore e ne attende l'esecuzione (**busy waiting**).
Con **I/O guidato da interrupt**, la CPU chiede al controllore di eseguire una richiesta I/O, blocca il processo che l'ha richiesta ed esegue un'altra operazione. Quando il controllore individua la fine di una richiesta I/O, solleva un interrupt per segnalarne il completamento, asserendo un segnale sul bus. Il segnale è gestito dal **controllore degli interrupt**. Il controllore degli interrupt ignora l'interrupt se interrupt con maggiore priorità sono in attesa. Quando è pronto, il controllore gestisce l'interrupt mettendo un numero sul bus specificando quale dispositivo richiede attenzione ed inviando un interrupt alla CPU. Questo numero è un indice in una tabella chiamata **interrupt vector**: ogni cella contiene l'indirizzo di memoria che punta all'inizio di un interrupt handler. Quando l'interrupt controller prende in gestione l'interrupt, la CPU serve la richiesta di interrupt, caricando il nuovo PC dall'interrupt vector e mandando un segnale all'interrupt controller. Ora la CPU è pronta a gestire un nuovo interrupt. Quando l'interrupt handler ha finito, la CPU riprende l'esecuzione del processo che era stato interrotto.

1.3.2 Interrupt precisi e imprecisi

Essendo le CPU moderne in grado di eseguire più istruzioni contemporaneamente, quando avviene un interrupt non tutte le istruzioni potrebbero essere state completate in tempo. Un **interrupt preciso** è un interrupt che lascia la macchina in uno stato ben definito: tutte le istruzioni prima del PC sono state eseguite e nessuna istruzione dopo il PC è stata portata a termine (è permesso che l'istruzione puntata dal PC sia stata eseguita o meno). Se una di queste proprietà viene violata, si ha un **interrupt impreciso**.

1.3.3 DMA

Con **I/O tramite DMA**, la CPU si appoggia ad un processore speciale detto **controllore DMA**: il controllore DMA gestisce i trasferimenti di dati tra memoria e dispositivi usando I/O programmato, inviando un interrupt alla CPU al termine delle operazioni. Il DMA contiene registri che possono essere letti e scritti dalla CPU. Lo schema di trasferimetno usato dal DMA è detto **fly-by mode**: i dati non passano dal controllore del DMA, ma sono eseguiti dal controllore del dispositivo, il controllore del DMA comunica al controllore del dispositivo quali operazioni deve svolgere. Un approccio alternativo è il **flow-through mode**: i dati passano dal controllore del DMA, dove sono memorizzati in un buffer interno al DMA, ma richiede due cicli di bus invece di uno.

1.4 Cycle-stealing, burst-mode

Il controllore del DMA e la CPU si contendono l'accesso alla memoria. Ci sono due approcci: **cycle-stealing** e **burst mode**.
Con approccio **cycle-stealing**, per ogni parola da trasferire il controllore DMA deve prima acquisire il bus e rilasciarlo quando il trasferimento è completato: avendo il DMA priorità sul bus superiore alla CPU, il controllore occasionalmente ruba cicli di bus alla CPU.
Con approccio **burst mode**, il controllore DMA comunica al dispositivo di ottenere il bus, iniziare una serie di trasferimenti e poi rilasciarlo: è un approccio più efficiente ma può bloccare la CPU ed altri dispositivi se il burst è molto lungo.
I controllori DMA possono controllare più trasferimenti dati allo stesso momento.

2 Software

2.1 Interazione CPU-SO

Ci sono tre approcci: **I/O programmato**, **I/O guidato da interrupt** e **I/O tramite DMA**.
Con **I/O programmato**: il SO legge/scrive il dispositivo e continua ad interrogarlo per vedere se ha terminato una operazione (**busy waiting**).
Con **I/O guidato da interrupt**: il SO invia una richiesta I/O e blocca il processo che l'ha richiesta, gestendo nel mentre altri processi, finchè non riceve un interrupt dal dispositivo. Quando tutte le operazioni sono finite, il SO sblocca il processo.
Con **I/O tramite DMA**, il controllore del DMA serve la richiesta senza appoggiarsi alla CPU (è come PIO ma svolta dal controllore del DMA al posto della CPU).

2.2 Obiettivi del software I/O

Gli obiettivi del software I/O è garantire:

- **indipendenza dei dispositivi**: poter scrivere programmi che possono accedere a qualsiasi dispositivo
- **denominazione uniforme**: il nome di un file o dispositivo non dovrebbe dipendere dal dispositivo
- **gestione degli errori trasparente**: gli errori dovrebbero essere gestiti vicino all'HW
- **supporto a più modalità di trasferimento**: la maggior parte dell'I/O fisico è **asincrono** (guidato da interrupt), ma i programmi sono più facili da sviluppare se le operazioni di scrittura sono **sincrone** (bloccanti), il SO dovrebbe far sembrare operazioni asincrone come sincrone ai programmi
- **gestione del del buffering**: a volte i dati in uscita dal dispositivo non possono essere gestiti direttamente e vanno memorizzati in un buffer
- **gestione dei dispositivi condivisi e dedicati**

2.3 Gestori degli interrupt

Quando avviene un interrupt, il **gestore degli interrupt** associato deve gestirlo. Per prima cosa, il SO salva il contesto del processo corrente; viene impostato il contesto e lo stack del gestore degli interrupt; il SO riscontra il controllore degli interrupt ed esegue il gestore degli interrupt finchè questo non esegue l'istruzione di ritorno dall'interrupt; infine il SO schedula un altro processo e lo esegue.

2.4 Driver dei dispositivi

I **driver** nascondono i dettagli per il controllo dei dispositivi agli strati superiori. Ogni driver di solito gestisce un tipo di dispositivo o classe di dispositivi, e sono di solito sviluppati dal produttore. I driver forniscono i gestori degli interrupt e sono eseguiti in modalità kernel: devono accedere i registri del controllore posizionati sotto il resto del SO (driver in modalità utente non sono largamente utilizzati). Struttura generale di un driver:

1. Accettare la richiesta I/O astratta dagli strati superiori
2. Controllare la validità dei parametri in input, e ritornare errore se non sono validi
3. Controllare se il dipositivo è in uso, iniziando le operazioni se non lo è
4. Inviare una sequenza di comandi a basso livello al dispositivo
5. Quando i comandi sono stati inviati: se PIO, attendere il completamento dell'operazione; se guidato da interrupt bloccare fino all'arrivo di un interrupt
6. Alla fine dell'operazione, controllare se si sono verificati errori
7. Ritornare informazioni di stato e errori
8. Selezionare la richiesta successiva o attendere

Un driver deve essere **rientrante**, ossia deve poter essere richiamato anche dopo essere stato interrotto. Inoltre devono essere capaci di gestire rimozione o aggiunta di dispositivi a runtime.

2.5 Software indipendente dal dispositivo

Il software deve includere funzioni che siano comuni a tutti i dispositivi e fornire un'interfaccia uniforme per tutti i livelli.
L'interfaccia uniforme tra il SO e i driver è realizzata tramite una API e uno schema di denominazione uniforme. Per l'API, il SO definisce per ogni classe di dispositivi le funzioni che il driver deve fornire. In UNIX, i dispositivi sono acceduti tramite nomi nel filesystem (**file speciali** o **file dispositivo**): ogni file speciale ha un **major device number** (per selezionare il driver appropriato) ed un **minor device number** (per selezionare il dispositivo I/O).
Il buffering è un problema sia per dispositivi a blocchi sia per dispositivi a caratteri. Senza buffering si hanno troppi interrupt; con buffering in spazio utente si hanno problemi legati alla paginazione; con buffering in spazio kernel, il buffer viene copiato in spazio utente quando è pieno (operazione atomica), ma a volte un buffer non è abbastanza quindi se ne usano 2 oppure buffering circolare.

Gli errori sono molto comuni nel contesto dell'I/O. Gli errori sono specifici al dispositivo ma il framework per gestire gli errori è indipendente dai dispositivi. Gli **errori di programmazione** si verificano quando il chiamante richiede operazioni o parametri non validi. Gli **errori di I/O** si verificano quando qualcosa va storto durante la richiesta I/O.
Alcuni dispositivi sono dedicati (possono essere usati da un solo processo alla volta) e il SO li deve gestire per evitare deadlock.

2.6 I/O nello spazio utente

La maggior parte del SW di I/O viene eseguito in modalità kernel. Le chiamate di sistema possono essere invocate via funzioni di libreria. I programmi sono detti **daemon** e sono spesso associati ad un **sistema spooler**.

3 Dischi

3.1 Dischi magnetici

Ogni **superficie** di un piatto è divisa logicamente in **tracce** circolari, che sono suddivise in **settori**. Ogni settore contiene lo stesso numero di bit di dati; i settori sono separati da spazi senza bit di dati, che servono ad identificare i settori; l'insieme delle tracce che si trovano sotto un braccio costituisce un **cilindro**.
Il controllore del disco può scrivere/leggere dati solo a multipli di settore e leggere/scrivere tutti i dati in una traccia senza muovere il braccio del disco. I controllori permettono **seek sovrapposti**, ma il trasferimento dei dati tra controllore e memoria principale non può sovrapporsi.
Nei dischi moderni, la geometria virtuale è diversa dalla geometria fisica: la lunghezza fisica della traccia aumenta all'aumentare della distanza dal centro, quindi i dischi contengono più settori nelle zone esterne che interne (**Zone Bit Recording**). Per nascondere questo al SO, i controllori presentano una geometria virtuale (LBA$\leftrightarrow$CHS).

3.2 Performance

La performance del disco è caratterizzata da:

- **tempo di seek**: tempo per posizionare il braccio del disco sul cilindro desiderato
- **latenza rotazionale**: tempo impegato dal settore desiderato per ruotare sotto la testina
- **velocità rotazionale**: la velocità con cui il motore fa ruotare il disco
- **tasso di trasferimento dei dati**: il tasso in bit/sec al quale i dati scorrono dalla superficie del disco al controllore
- **tempo di posizionamento**: somma del tempo di seek e latenza rotazionale
- **tempo d'accesso**: somma del tempo di posizionamento e tempo di trasferimento

3.3 Affidabilità

In caso di **fallimento totale del disco**, il disco non può svolgere ulteriori operazioni e deve essere rimpiazzato. Il **tasso di fallimento annuale** è il numero di fallimenti ogni anno e il **tasso di fallimento orario** è il numero di fallimenti ogni ora.
La **disponibilità** è la frazione di tempo il sistema è disponibile a servire le richieste degli utenti. Il tasso di fallimento segue il modello a vasca: i fallimenti sono più frequenti all'inizio ed alla fine del tempo di servizio.

3.4 Formattazione a basso livello

Prima che un disco possa essere usato per memorizzare dati, deve ricevere una **formattazione a basso livello**, che definisce la geometria fisica del disco. Ogni settore ha i seguenti campi: **preambolo**, che identifica l'inizio di un settore, **dati** e **error correcting codes**. Quando un settore viene scritto, il controllore del disco aggioran l'ECC con un valore calcolato da tutti i byte del campo dati; quando un settore viene letto, il controllore ricalcola l'ECC e lo compara al valore in memoria.
Le tracce non sono numerate partendo dalla stessa posizione, ma da posizioni diverse: questo è chiamato **cylinder skew**.
Dopo la formattazione a basso livello, la capacità del disco viene ridotta. Inoltre, impone un tasso masimo di trasferimento dati.
Per ovviare al problema della dimensione del buffer, si usa una tecnica chiamata **sector interleaving**, oppure un buffer molto capiente, che possa memorizzare un'intera traccia.
Al termine della formattazione a basso livello, il disco viene **partizionato**. Logicamente, una partizione è come un disco separato; il primo settore di ogni partizione e la sua dimensione sono memorizzate nella **partition table**, e la partition table è memorizzata nel **Master Boot Record** (MBR). La MBR è memorizzata nel primo settore del disco, che contiene inoltre il **boot loader**. La partition table del MBR supporta fino a 4 **partizioni primarie**: una e solo una può essere la **partizione estesa**, che può contenere più **partizioni logiche**. Ogni partizione logica viene descritta della **Extended Boot Record** (EBR). MBR sta venendo soppiantato dalla **GUID Partition Table** (GPT).

3.5 Scheduling del braccio del disco

Le richieste di I/O in attesa di essere servite sono gestite tramite diversi algoritmi di scheduling:

- **First Come First Served** (FCFS): le richieste sono servite nell'ordine in cui arrivano, non è molto veloce
- **Shortest Seek First** (SSF): seleziona la richiesta con il tempo di seek minore dalla posizione attuale, ma può indurre starvation
- **LOOK**: il braccio si muove in una direzione e serve tutte le richieste, poi cambia direzione e serve tutte le altre
- **C-LOOK**: il braccio serve richieste solo in una direzione

3.6 Gestione degli errori

I difetti di produzione sono inevitabili e possono introdurre **bad sectors**, settori che non ritornano i valori che gli sono appena stati scritti. Due approcci per gestire i bad sector: a livello del controllore e a livello del SO.
Il controllore del disco può usare dei **settori di riserva** per rimpiazzare bad sectors, tramite le tecniche di **sector forwarding** (il controllore rimappa uno dei settori di riserva) e **sector slipping** (il controllore muove tutti i settori).
Non tutti i controllori sono in grado si rimappare in maniera trasparente i settori, quindi il SO ottiene la lista dei bad sectors per far in modo che non vengano utilizzati.
Possono anche verificarsi errori di seek, causati da problemi meccanici del braccio, tipicamente risolti da una ricalibrazione.

4 Interfacce utente

Le interfacce utente includono SW di input e output, che collaborano tra loro.

4.1 SW della tastiera

La tastiera contiene un microprocessore che comunica con un chip controllore sulla scheda madre. Un interrupt viene generato ogni volta che un tasto viene premuto o rilasciato. Il numero nel registro di I/O è il numero del tasto, detto **scan code**, che consiste di un singolo byte (7 bit per identificare il tasto, 1 per tasto premuto/rilasciato). Il driver della tastiera converte lo scan code nel character code del **character set** usato dal SO.
I driver della tastiera possono operare in due modi: **noncanonical mode** e **canonical mode**. Il noncanonical mode è orientato a caratteri, mentre il canonical mode è orientato a linee.

4.2 SW del mouse

I mice sono più semplici delle tastiere. Ogniqualvolta il mouse viene mosso o un bottone viene premuto, viene inviato un interrupt al computer. La distanza minima è un **mickey** (0.1 mm). Il messaggio contiene la variazione della posizione sugli assi x e y ed eventuali bottoni premuti.

5 Gestione dell'energia

Per ridurre il consumo energetico, il SO può disattivare parti del computer non utilizzate: il SO deve decidere quale dispositivo spegnere e quando, per evitare sprechi e ritardi nell'utilizzo.
Per quanto riguarda gli hard disk, il break-even point è il lasso di tempo per il quale l'energia consumata per rallentare il disco, tenerlo in questo stato e poi acceleralo è uguale all'energia consumata per tenere il disco ad una velocità costante. Il SO può fare prefetching dei blocchi dal disco in una memoria cache, in modo da non dover riattivare il disco se il blocco è in cache.
Per ridurre il consumo della CPU di possono ridurre tensione e/o la frequenza (DVFS). Le CPU moderne hanno più **performance states** per il DVFS: ogni P-state è denotato da un Px, dove x è un numero da 0 ad un determinato massimo; valori più elevati di x garantiscono un livello di risparmio energetico superiore. Inoltre, le CPU hanno più **power states**: livelli più elevati indicano minore operatività della CPU.