# CAPITOLO 13: SISTEMI DI I/O

I due compiti principali di un calcolatore sono l’I/O e l’elaborazione. Spesso il compito principale è costituito dall’I/O mentre l’elaborazione è semplicemente accessoria.

Il ruolo di un sistema operativo nell’I/O di un calcolatore è quello di gestire e controllare le operazioni e i dispositivi di I/ O.

## 13.1 INTRODUZIONE

Il controllo dei dispositivi connessi a un calcolatore è una delle questioni più importanti che riguardano i progettisti di sistemi operativi. Poiché i dispositivi di I/O sono così largamente diversi per funzioni e velocità (es un mouse, un disco e un jukebox di CD-ROM) altrettanto diversi devono essere i metodi di controllo.

Tali metodi costituiscono il *sottosistema di I/O* del kernel.

La tecnologia dei dispositivi di I/O mostra due tendenze tra loro in conflitto. Da una parte, si osserva la crescente uniformazione standard delle interfacce fisiche logiche, d’altra parte, però, si assiste a una crescente varietà di dispositivi di I/O; alcuni di loro sono tanto diversi dai dispositivi precedenti dal rendere molto difficile il compito di integrarli nei calcolatori e nei sistemi operativi esistenti. Questo problema si affronta strutturando gli elementi di base dell’architettura di I/O in modo da potervi connettere un’ampia varietà di dispositivi di I/O e strutturando il kernel del sistema operativo in moduli di driver di dispositivi allo scopo di incapsulare i dettagli e le particolarità dei diversi dispositivi.

I driver dei dispositivi offrono al sottosistema di I/O un’interfaccia uniforme per l’accesso ai dispositivi, così come le chiamate di sistema forniscono un’interfaccia uniforme tra le applicazioni e il SO.

## 13.2 ARCHITETTURE E DISPOSITIVI DI I/O

I calcolatori fanno funzionare un gran numero di tipi di dispositivi: la maggior parte rientra nella categoria dei dispositivi di memorizzazione secondaria e terziaria, dispositivi di trasmissione (schede di rete – modem), interfaccia uomo – macchina (schermi, tastiere). Un dispositivo comunica con un sistema di calcolo inviando segnali attraverso un cavo o attraverso l’etere e comunica con il calcolatore tramite un punto di connessione (porta), ad esempio una porta seriale. Se uno o più dispositivi usano in comune un insieme di fili. La connessione e detta bus. Un bus è un insieme di fili e un protocollo rigorosamente definito che specifica l’insieme dei messaggi che si possono inviare attraverso i fili. In termini elettronici, i messaggi si inviano tramite configurazioni di livelli di tensione elettrica applicate ai fili con una definita scansione temporale. Quando un dispositivo A ha un cavo che si connette a un dispositivo B e il dispositivo B ha un che si connette un dispositivo C che a sua volta è collegato a una porta di un calcolatore, si ottiene il cosiddetto **collegamento a Margherita** (daisy chain)**,** che di solito funziona come un bus.

I bus sono ampiamente usati nell’architettura dei calcolatori e differiscono tra loro per metodo di segnale, velocità, throughput e metodi di connessione. La figura sottostante mostra una tipica struttura di bus di un PC; si tratta di un **bus PCI** (il comune bus di sistema dei PC) che connette il sottosistema CPU-memoria ai dispositivi veloci, e di un **bus d’espansione** cui si connettono i dispositivi relativamente lenti come la tastiera e le porte seriali e parallele.

Un **controllore** è un insieme di componenti elettronici che può far funzionare una porta, un bus o un dispositivo. Un controllore di porta seriale è un semplice controllore di dispositivo; di tratta di un singolo circuito integrato nel calcolatore che controlla i segnali presenti nei fili della porta seriale. Alcuni dispositivi sono dotati di propri controllori incorporati.

L’unità d’elaborazione dà comandi e fornisce dati al controllore per portare a termine trasferimenti di I/O tramite uno o più registri per dati e segnali di controllo. La comunicazione con il controllore avviene attraverso la lettura e la scrittura, da parte dell’unità d’elaborazione, di configurazioni di bit in questi registri. Un modo in cui questa comunicazione può avvenire è tramite l’uso di speciali istruzioni di I/O che specificano il trasferimento di un byte o una parola a un indirizzo di porta I/O.

l’istruzione di I/O attiva le linee di bus per selezionare il giusto dispositivo e trasferire bit dentro o fuori dal registro di dispositivo.

In alternativa, il controllore di dispositivo può disporre dell’I/O associato alla memoria o mappato in memoria. In questo caso i registri di controllo del dispositivo si fanno corrispondere ad un sottoinsieme dello spazio di indirizzi della CPU, che esegue le richieste di I/O usando le ordinate istruzioni di trasferimento dati per leggere e scrivere i registri di controllo del dispositivo.

Certi sistemi usano entrambe le tecniche. I PC, ad esempio, usano istruzioni di I/O per controllare alcuni dispositivi e l’I/O mappato in memoria per controllare altri. il controllore della grafica ha alcune porte di I/O per le operazioni di controllo di base, ma dispone di un’ampia regione mappata in memoria, detta memoria grafica, che serve a mantenere i contenuti dello schermo. Il processo scrive sullo schermo inserendo i dati nella regione mappata in memoria; il controllore genera l’immagine dello schermo sulla base del contenuto di questa regione di memoria.

Questa tecnica è semplice da usare; inoltre, la scrittura di milioni di byte nella memoria grafica è molto più veloce dell’invio di milioni di istruzioni di I/O.

La facilità di scrittura in memoria è controbilanciata da uno svantaggio: un comune errore di programmazione è la scrittura in una regione di memoria sbagliata causata da un errato puntatore. Ciò rende i registri dei dispositivi mappati in memoria vulnerabili ad accidentali modifiche.

Una porta di I/O consiste in genere in quattro registri: *status, control, data-in e data-out.*

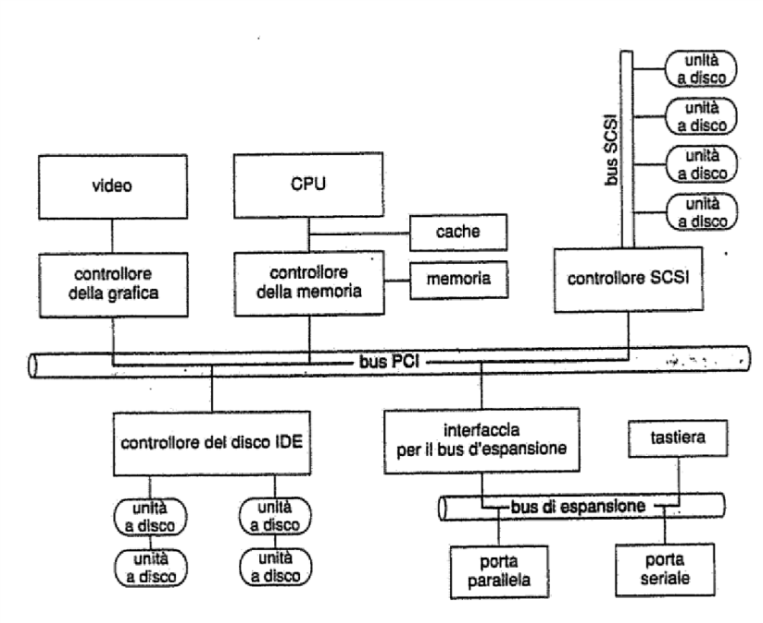
1)La CPU legge il registro data-in per ricevere dati.

2)La CPU scrive nel registro data-out per emettere dati.

3)Il registro status contiene alcuni bit che possono essere letti e indicano lo stato della porta; ad esempio, indicano se è stata portata a termine l’esecuzione del comando corrente

4)Il registro control può essere scritto per attivare un comando o per cambiare il modo di funzionamento del dispositivo. Ad esempio, un certo bit nel registro control della porta seriale determina il tipo di comunicazione tra *half-duplex e full-duplex*

La tipica dimensione dei registri di dati varia tra 1 e 4 byte. Certi controllori hanno circuiti integrati FIFO che possono contenere parecchi byte per l’immissione e l’emissione dei dati, in modo da espandere la capacità del controllore oltre la dimensione del registro di dati.



## 13.2.1 Interrogazione ciclica

Il protocollo completo per l’interazione fra la CPU e un controllore può essere intricato, ma la fondamentale nozione di negoziazione *(handshaking)* è semplice, ed è illustrata con un esempio.

Il controllore specifica il suo stato per mezzo del bit busy del registro status; pone a 1 il bit busy quando è impegnato in un’operazione, e lo pone a 0 quando è pronto a eseguire il comando successo. La CPU comunica le sue richieste tramite il bit comand-ready nel registro command; pone questo bit a 1 quando il controllore deve eseguire un comando.

1. La CPU legge ripetutamente il bit busy finché non valga 0.
2. La CPU pone a 1 il bit write del registro dei comandi e scrive un byte nel registro data-out
3. La CPU pone a 1 il bit command-ready.
4. Quando il controllore si accorge che il bit command-ready è posto a 1, pone a 1 il bit busy
5. Il controllore legge il registro dei comandi e trova il comando write; legge il registro data-out per ottenere il byte da scrivere, e compie l’operazione di scrittura nel dispositivo
6. Il controllore pone a 0 il bit command-ready, pone a 0 il bit error nel registro status per indicare che l’operazione di I/O ha avuto esito positivo, e pone a 0 il bit busy per indicare che l’operazione è terminata.

(La sequenza appena descritta si ripete per ogni byte.)

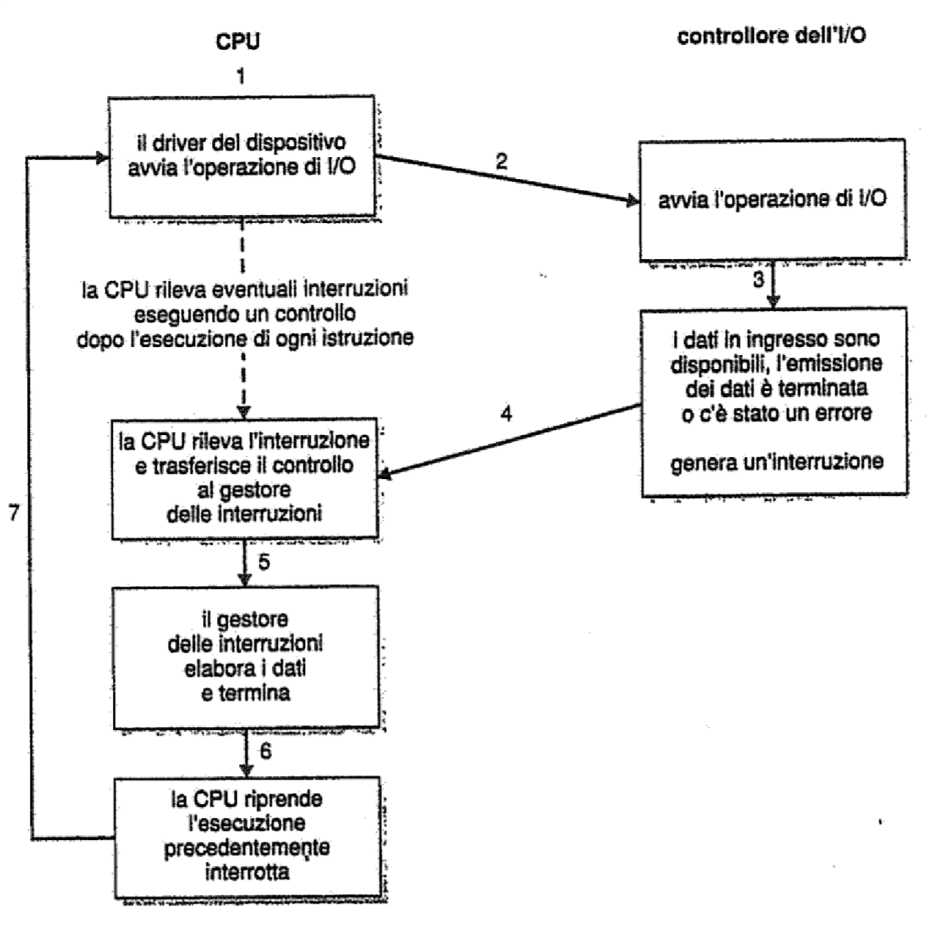
Durante l’esecuzione del passo 1, la CPU è in attesa attiva (*busy-waiting)* o in interrogazione ciclica (polling): itera la lettura del registro status finché il bit busy assume il valore 0. Se il controllore e il dispositivo sono veloci, questo metodo è ragionevole, ma se l’attesa rischiasse di prolungarsi, sarebbe probabilmente meglio se la CPU si dedicasse a un’altra operazione.

E necessario che la CPU serva certi tipi di dispositivi rapidamente, o si potrebbero perdere alcuni dati.

Quando, ad esempio, i dati affluiscono in una porta seriale o della tastiera. Il piccolo buffer del controllore diverrà presto pieno, e se la CPU attende troppo a lungo prima di riprendere la lettura dei byte, si prenderanno informazioni.

In molte architetture di calcolatori sono sufficienti tre cicli di istruzioni di CPU per in-terrogare ciclicamente un dispositivo: read, lettura di un registro del dispositivo; logical-and, configurazione logica usata per estrarre il valore di un bit di stato, e branch, salto a un altro punto del codice se l’argomento è diverso da zero.

Anziché richiede alla CPU di eseguire un’interrogazione ciclica, può essere più efficiente far sì che il controllore comunichi alla CPU che il dispositivo è pronto. Il meccanismo dall’architettura che permette tale comunicazione si chiama interruzione della CPU o, più brevemente, interruzione *(interrupt).*



## 13.2.2 Interruzioni

La CPU ha un contatto, detto linea di richiesta dell’interruzione, del quale la CPU controlla lo stato dopo l’esecuzione di ogni istruzione. Quando rileva il segnale di un controllore nella linea di richiesta dell’interruzione, la CPU salva lo stato corrente e salta alla routine di gestione dell’interruzione *(interrupt-handler routine),* che si trova e un indirizzo prefissato di memoria. Questa procedura determina le cause dell’interruzione, porta a termine l’elaborazione necessaria ad esegue un’istruzione return from interrupt per far sì che la CPU ritorni nello stato in cui si trovava prima della sua interruzione. Il controllore del dispositivo *genera* un segnale d’interruzione della CPU lungo la linea di richiesta delle interruzioni, che la CPU *rileva* e *recapita* al gestore delle interruzioni, che a sua volta *evade* il compito corrispondente servendo il dispositivo. Il meccanismo di base delle interruzioni permette alla CPU di rispondere ad un evento asincrono, Come quello di un controllore di un dispositivo che diventa pronto per essere servito.

Nei sistemi operativi moderni necessarie capacità di gestione delle interruzioni più raffinate.

1. Si deve poter posporre la gestione dell’interruzione durante le fasi critiche dell’elaborazione.
2. Si deve disporre di un meccanismo efficiente per passare il controllo all’appropriato gestore delle interruzioni, senza dover esaminare ciclicamente tutti i dispositivi *(polling)* per determinare quale abbia generato l’interruzione.
3. Si deve disporre di più livelli d’interruzione, di modo che il sistema possa distinguere le interruzioni ad altra priorità da quelle a priorità inferiore, servendo le richieste con la celerità appropriata del caso.

In un calcolatore moderno queste tre caratteristiche sono fornite dalla CPU e dal controllore delle interruzioni.

La maggior parte delle CPU ha due linee di richiesta delle interruzioni. Una è quella delle interruzioni non mascherabili, riservata a eventi quali gli errori di memoria irrecuperabili. La secondo linea è quella delle interruzioni mascherabili: può essere disattivata dalla CPU prima dell’esecuzione di una sequenza critica di istruzioni che non deve essere interrotta. Il meccanismo delle interruzioni accetta un indirizzo, nella maggior parte delle architetture questo indirizzo è uno scostamento relativo a una tabella detta vettore delle interruzioni, contenente gli indirizzi di memoria gli specifici gestori delle interruzioni. In pratica, tuttavia, i calcolatori hanno più dispositivi (e quindi, più gestori delle interruzioni) che elementi nel vettore delle interruzioni. Una maniera diffusa nel risolvere questo problema consiste nel servirsi di una tecnica detta **concatenamento delle interruzioni** (*interrupt chaining)*, in cui ogni elemento del vettore delle interruzioni punta alla testa di una lista di gestori nella lista delle interruzioni. Quando si verifica un’interruzione, si chiamano uno alla volta i gestori nella lista corrispondente finché non se ne trova uno che può soddisfare la richiesta.

Il meccanismo delle interruzioni realizza anche un sistema di livelli di proprietà delle interruzioni. Esso permette alla CPU di differire la gestione delle interruzioni di bassa priorità senza mascherare tutte le interruzioni, e permette a un’interruzione di priorità alta di sospendere l’esecuzione della procedura di servizio di un ‘interruzione di priorità bassa. Un sistema operativo moderno interagisce con il meccanismo delle interruzioni in vari modi. All’accensione del machina esamina i bus per determinare quali dispositivi siano presenti, e installa gli indirizzi dei corrispondenti gestori delle interruzioni nel vettore delle interruzioni. Durante l’I/O, i vari controllori di dispositivi generano i segnali d’interruzione della CPU quando sono pronti per un servizio.

Il meccanismo delle interruzioni si usa anche per gestire un’ampia gamma di eccezioni, come la divisione per zero, l’accesso a indirizzi di memoria protetti o inesistenti o il tentativo di eseguire un’istruzione privilegiata in modalità utente. Gli eventi che producono segnali di interruzione hanno proprietà in comune: inducono la CPU a eseguire urgentemente una procedura autonoma. Un sistema operativo può fare altri usi proficui di un ufficiale del meccanismo dell’architettura che memorizza una piccola quantità di informazioni sullo stato della CPU e poi richiama una procedura del kernel. ad esempio, molti sistemi operativi usano il meccanismo delle interruzioni per la gestione della memoria virtuale.

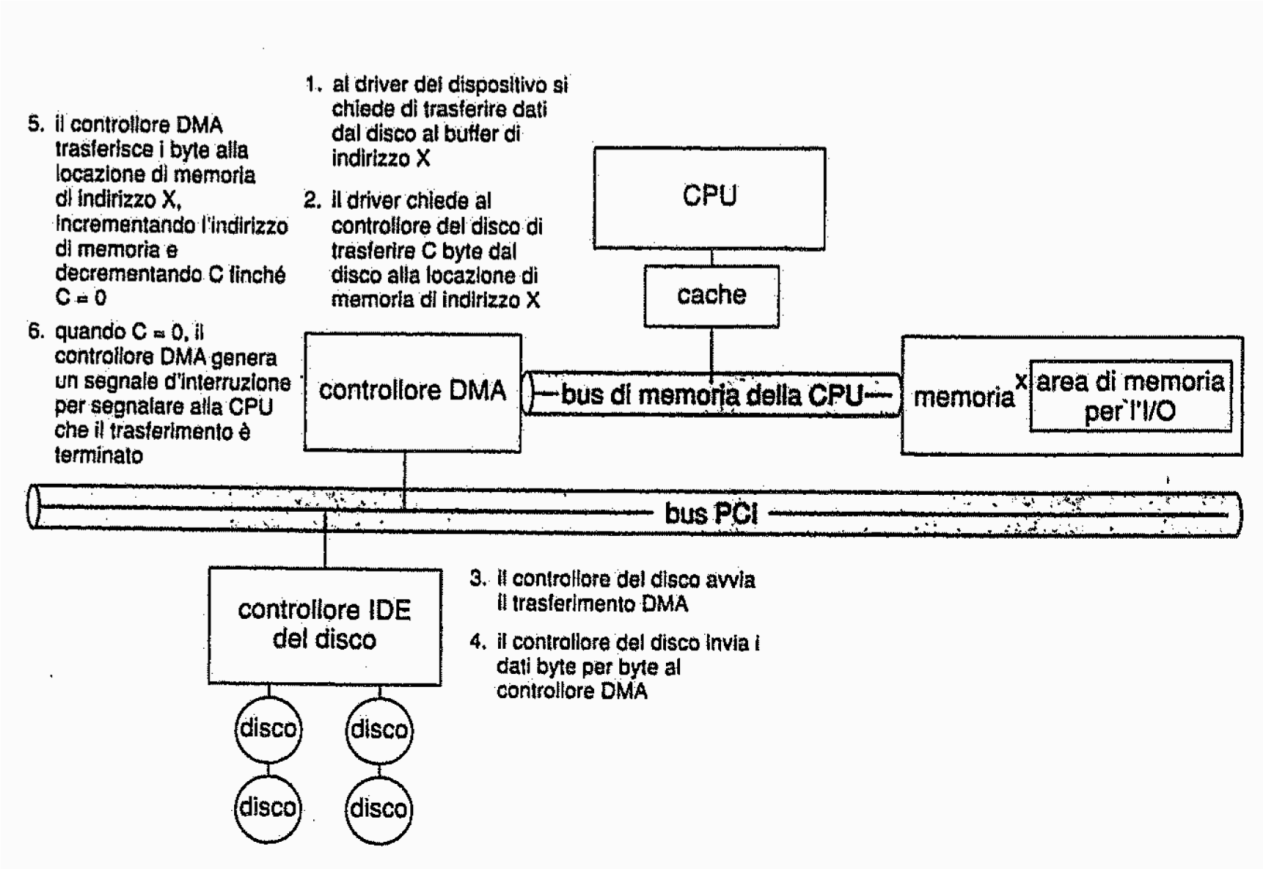
Un altro esempio è dato dall’esecuzione delle chiamate di sistema. Solitamente i programmi sfruttano routine di libreria per eseguire chiamate di sistema. La routine controlla i parametri passati dall’applicazione, li assembla in una struttura dati appropriata da passare al kernel, e infine esegue una particolare istruzione detta interruzione software o trap. Quando la chiamata di sistema esegue l’istruzione di eccezione, l’architettura delle interruzioni memorizza le informazioni riguardanti lo stato cui era giunta l‘esecuzione del codice utente, passa al modo supervisore e recapita l’interruzione alla procedura del kernel che realizza il servizio richiesto. Le interruzioni si possono inoltre usare per gestire il controllo del flusso all’interno del kernel. Si consideri ad esempio l’elaborazione richiesta per completare una lettura da un disco. Un passo necessario è quello i copiare dati dalla regione di memoria usata dal kernel al buffer dell’utente. Questa azione richiede tempo, ma non è urgente e non dovrebbe bloccare la gestione delle interruzioni con priorità più alta. Un altro passo è quello di avviare l’evasione desse successive richieste di IO relative a quell’unita a disco. Questo passo ha priorità più alta: se le unta a disco si devono usare in modo efficiente, è necessario avviare l’evasione della successiva richiesta di IO non appena la precedente sia stata soddisfatta.

Un’architettura del kernel basata su thread è adatta alla realizzazione di più livelli di priorità delle interruzioni e a dare la precedenza alla gestione delle interruzioni rispetta alle elaborazioni in sottofondo delle procedure del kernel e delle applicazioni. Riassumendo i segnali d’interruzione sono usati diffusamente dai SO moderni per gestire eventi asincroni e per eseguire in modalità supervisore le procedure del kernel. Per far si che i compiti più urgenti siano portati a termine per primi, i calcolatori moderni usano un sistema di priorità delle interruzioni. I controllori dei dispositivi, gli errori e le chiamate di sistema generano segnali d’interruzione al fine di innescare l’esecuzione di procedure del kernel.

## 13.3.2 Accesso diretto alla memoria DMA

Quando un dispositivo compie trasferimenti di grandi quantità di dati, come nel caso di un’unita a disco, l’uso di una costosa CPU per il controllo dei bit di stato e per la scrittura di dati nel registro del controllore byte alla volta, detto IO programmato, sembra essere uno spreco. In molti calcolatori si evita di sovraccaricare la CPU assegnando una parte di questi compiti a un’unita di elaborazione specializzata, detta controllore dell’accesso diretto alla memoria DMA. Per dar avvio a un trasferimento DMA la CPU scrive in memoria un comando strutturato per il DMA, esso contiene un puntatore alla locazione dei dati da trasferire, un altro puntatore alla destinazione dei dati e il numero di byte da trasferire. La CPU scrive l’indirizzo di questo comando nel controllore del DMA, e prosegue con la sua esecuzione. Il controllore DMA agisce quindi direttamente sul bus della memoria, presentano al bus gli indirizzi di memoria necessarie per eseguire il trasferimento senza l’aiuto della CPU. Un semplice controllore DMA è un componente o ordinario dei PC e le schede di IO dette bus mastering di un PC includono di solito componenti DMA ad alta velocità.

La procedura di negoziazione tra il controllore del DMA e il controllore del dispositivo si svolge grazie a una coppia di fili detti DMA request e DMA acknowledge. Il controllore del dispositivo manda un segnale sulla linea DMA request quando una parola di dati è disponibile per il trasferimento. Questo segnale fa si che il controllore DMA prenda possesso del bus di memoria, presenti l'indirizzo desiderato ai fili d’intirizzimento della memoria e mandi un segnale lungo la linea DMA acknowledge. Quando il controllore del dispositivo riceve questo segnale, trasferisce in memoria la parola di dati e rimuove il segnale dalla linea DMA request. Quando l’intero trasferimento termina, il controllore del DMA interrompe la CPU. Quando il controllore del DMA prende possesso del bus di memoria, la CPU è temporaneamente impossibilitata ad accedere alla memoria centrale, sebbene abbia accesso ai dati contenuti nella sua cache primaria e secondaria. Questo fenomeno noto come sottrazione di cicli può rallentare la computazione della CPU; ciononostante l’assegnamento del lavoro di trasferimento di dati a un controllore DMA migliora e in generale le prestazioni complessive del sistema. In alcune architetture per realizzare la tecnica DMA si usano gli indirizzi della memoria fisica, mentre in altre s’impiega l’accesso diretto alla memoria virtuale, in questo caso si usano indirizzi virtuali che poi si traducono in indirizzi fisici.



## 13.3 INTERFACCIA DI I/O PER LE APPLICAZIONI

In questo paragrafo si discutono le tecniche e le interfacce di un SO che permettono un trattamento uniforme dei dispositivi di I/O. Si spiega, ad esempio, come un’applicazione possa aprire n file residente in un disco senza sapere di che tipo di disco si tratti e come si possano aggiungere al calcolatore nuove unita a disco e latri dispositivi senza che si debba modificare il sistema operativo.

I metodi qui esposti coinvolgono l’astrazione, l’incapsulamento e la stratificazione dei programmi. In particolare, si può compiere un procedimento di astrazione rispetto ai dettagli delle differenze tra i dispositivi per l’IO identificandone alcuni tipi generali. A ognuno di questi tipi si accede per mezzo di un unico insieme di funzioni- un’interfaccia. Le differenze sono incapsulate in moduli del kernel detti driver dei dispositivi che comunicano con l’esterno per mezzo di interfacce uniformi.

Lo scopo dello strato dei driver dei dispositivi è di nascondere al sottosistema di IO del kernel le differenze tra i controllori dei dispositivi in mood simile a quello con cui le chiamate di sistema di IO incapsulando il comportamento dei dispostivi in alcune classi generiche che nascondo le differenze alle applicazioni.

Sfortunatamente per i produttori di dispositivi, ogni tipo di sistema operativo ha le sue convenienze riguardanti l’interfaccia dei driver dei dispositivi. I dispositivi possono differire in molti aspetti:

* **Trasferimento a flusso di caratteri o a blocchi.** Un dispositivo del primo tipo trasferisce dati un byte alla volta mentre uno del secondo tipo ne trasferisce un blocco alla volta.
* **Accesso sequenziale o diretto.** Un dispositivo del primo tipo trasferisce dati sentendo ordine prestabilito, mentre l’utente di un dispositivo ad accesso diretto può richiedere l’accesso a una qualunque delle possibili locazioni di memorizzazione
* **Dispositivi sincroni o asincroni.** Un dispositivo sincrono trasferisce dati con un tempo di risposta prevedibile, mentre un dispositivo asincrono ha tempi di risposta irregolari
* **Condivisibili o riservati.** Un dispositivo condivisibile può essere usato in modo concorrente da diversi processi mentre ciò è impossibile se un dispositivo è riservato.
* **Velocità di trasferimento.** Può variare da alcuni byte/s fino a qualche gb/s
* **Lettura e scritture, solo lettura o solo scrittura.** Alcuni dispositivi possono emettere e ricevere dati, ma altri consentono solo una delle due possibilità.

Per ciò che riguarda l’accesso delle applicazioni ai dispositivi, molte di queste differenze sono nascoste dal sistema operativo e i dispostivi sono raggruppati in poche classi convenzionali.

## 

## 13.3.1 Dispositivi con trasferimento a blocchi o a caratteri //da migliorare

L’interfaccia per i dispostivi a blocchi sintetizza tutti gli aspetti necessari per accedere all’unita a disco e ad altri dispositivi basati sul trasferimento di blocchi di dati. Il SO e certe applicazioni particolari come quelle per la gestione delle basi di dati possono trovare più conveniente trattare questi dispositivi come una semplice sequenza lineare di blocchi. In questo caso si parla di IO a basso livello. La tastiera è un esempio di dispositivo al quale si accede tramite un’interfaccia a flusso di caratteri.

## 

## 13.3.2 Dispositivi di rete //da migliorare

Poiché i modi di indirizzamento e le prestazioni tipiche dell’IO di rete sono notevolmente differenti da quelli dell’I/O dell’unita a disco, la maggior parte dei SO fornisce un’interfaccia per l’I/O di rete diversa da quelle delle normali operazioni read(),write() e seek().Un’interfaccia disponibile in molti SO è l’interfaccia di rete socket. Una volta creata una socket si possono usare le normali operazioni di I/O.

## 13.3.3 Orologi e timer

La maggior parte dei calcolatori ha timer e orologi che forniscono 3 funzioni essenziali:

* Segnare l’ora corrente;
* Segnalare il tempo trascorso;
* Regolare un timer in modo da avviare l’operazione “x” al tempo “t”;

Il dispositivo che misura la durata di un lasso di tempo e che può avviare un’operazione si chiama timer programmabile. Lo scheduler usa questo meccanismo per generare un segnale d’interruzione che sospende un processo quando il suo quanto di tempo è scaduto. Il SO può inoltre fornire un’interfaccia per permettere ai processi utenti di usare il timer.

In certi casi, simulando orologi virtuali, il SO può anche gestire un numero di richieste d’uso dei timer maggiore del numero dei timer fisici. Per far ciò il driver del timer (kernel) mantiene una lista ordinata cronologicamente delle interruzioni richieste dagli utenti e dalle proprie procedure, e imposta il timer per la prima scadenza. Quando il timer genere il segnale di interruzione, il kernel avvisa il richiedente, e imposta il timer per la scadenza successiva.

Nella maggior parte dei calcolatori, l’orologio è costruito sulla base di un contatore ad alta frequenza.

## 

## 13.3.4 IO bloccante e non bloccante

Un altro aspetto delle chiamate di sistema è la scelta fra IO bloccante e non bloccante. Quando un’applicazione impiega una chiamata di sistema bloccante si sospende l’esecuzione delle applicazioni, che passa dalla coda dei processi pronti per l’esecuzione alla coda d’attesa. Quando la chiamata di sistema termina l’applicazione è posta nuovamente nella coda dei processi pronti in modo che possa riprendere l’esecuzione solo allora essa ricevere i valori riportati dalla chiamata di sistema.

Alcuni processi a livello utente necessitano di una forma di IO non bloccante. Un esempio è quello di un’interfaccia utente con cui s’interagisce col mouse e la tastiera mentre elabora dati e li mostra sullo schermo. Sistemi operativi forniscono chiamate di sistema non bloccante per l'I/O. Una chiamata di questo tipo non arresta l'esecuzione dell'applicazione per un lasso di tempo significativo. Al contrario virgola e si restituisce rapidamente il controllo all’applicazione, fornendo un parametro che indica quanti byte di dati sono stati trasferiti. Una possibile alternativa alle chiamate di sistema non bloccanti è costituita dalle chiamate di sistema asincrone. E se restituiscono immediatamente il controllo al chiamante, senza attendere che l/O sia stato completato.

La differenza tra chiamata non bloccante e asincrona è che una read non bloccante restituisce immediatamente il controllo fornendo i dati che è stato possibile leggere. Una chiamata read asincrona richiede un trasferimento di cui il sistema garantisce il completamento ma solo in un momento successivo e non prevedibile.

**13.4 SOTTOSISTEMA PER L’I/O DEL KERNEL**

Il **kernel** fornisce molti servizi riguardanti l’I/O -scheduling dell’I/O , gestione del buffer, delle cache , delle code , uso esclusivo dei dispositivi , gestione degli errori- sono offerti dal sottosistema per l'I/O del kernel e sono realizzati a partire dai dispositivi e dai relativi driver.

## 13.4.1 Scheduling di IO

Fare lo **scheduling** di un insieme di richieste di I/O significa stabilirne un ordine d’esecuzione efficace.

Lo scheduling può migliorare le prestazioni complessive del sistema, distribuire equamente gli accessi dei processi ai dispositivi e ridurre il tempo d’attesa media per il completamento di un’operazione di I/O.

I progettisti di SO realizzano lo scheduling mantenendo una **coda di richieste** per ogni dispositivo.

Quando un’applicazione richiede l’esecuzione di una chiamata di sistema di I/O **bloccante**, si aggiunge la richiesta alla coda relativa al dispositivo appropriato.

Lo scheduler dell’I/O riorganizza l’ordine della coda per migliorare l’efficienza globale del sistema e il tempo medio d’attesa cui sono sottoposte le applicazioni. Il sistema operativo può anche tentare di essere equo in modo che nessuna applicazione ricerca un servizio carente o può dare priorità a quelle richieste la cui corretta esecuzione potrebbe essere inficiata da un ritardo nel servizio.

I kernel che mettono a disposizione l’I/O asincrono devono essere in grado di tener traccia di più richieste di I/O contemporaneamente. A questo fine alcuni sistemi annettono una **tabella dello stato dei dispostivi** alla coda dei processi in attesa. Gli elementi della tabella indicano il dispositivo, l’indirizzo e lo stato.

Lo scheduling dell'I/O è uno dei modi in cui il sottosistema per l'I/O migliora l'efficienza di un calcolatore, Un altro e l'uso di spazio di memorizzazione nella memoria centrale o nei dischi, per tecniche di memorizzazione transitoria, uso di cache e di code per la gestione asincrona dell’I/O.

## 13.4.2 Gestione del buffer

Un buffer è un’area di memoria che contiene dati durante il trasferimento fra due dispositivi o tra un’applicazione e un dispositivo.

Si ricorre al buffer per 3 ragioni: la **prima** è la necessità di gestire la differenza di velocità tra produttore e consumatore. Un modem, ad esempio, che è nettamente più lento di qualsiasi altro dispositivo ha bisogno di una doppia bufferizzazione altrimenti si renderebbe troppo critico il problema della sincronizzazione con il disco.

Un **secondo** uso del buffer riguarda la gestione dei dispositivi che trasferiscono dati in blocchi di dimensione diversa. Questa disparità si verifica soprattutto nelle reti di calcolatori , dove spesso è necessario frammentare e ricomporre messaggi. Quando un mittente spedisce un messaggio molto lungo , esso è spezzato in piccoli pacchetti che si spediscono attraverso la rete ; il sistema destinatario provvede a ricostruire in un buffer l'intero messaggio originario.

Il **terzo** modo in cui si può impiegare un buffer è per la realizzazione della **semantica delle copie** nell’ambito dell’I/O delle applicazioni. La semantica delle copie garantisce che la versione dei dati scritta nel disco sia conforme a quella contenuta nel buffer al momento della chiamata di sistema indipendentemente da ogni successiva modifica.

## 13.4.3 Cache

Una **cache** è una regione di memoria veloce che serve per mantener copie di certi dati: l’accesso a queste copie è più rapido dell’accesso agli originali. Ad esempio le istruzioni di un processo correntemente in esecuzione sono memorizzate in un disco , copiate nella memoria fisica e copiate ulteriormente nella cache primaria e secondaria della CPU.

La differenza tra **buffer** e **cache** consiste nel fatto che il primo può contenere dati di cui non esiste altra copia, mentre la cache, per definizione, mantiene su un mezzo più efficiente una copia d’informazioni già memorizzate.

## 13.4.4 Code ad uso esclusivo dei dispositivi

Una coda di file da stampare(spool) è un buffer contenente dati per un dispositivo che non può accettare flussi di dati intercalati. Sebbene una stampante possa servire una sola richiesta alla volta, diverse applicazioni devono potere richieder simultaneamente la stampa di dati, senza che questi si mischino.

Il SO risolve questo problema filtrando tutti i dati per la stampante: i dati da stampare provenienti da ogni singola applicazione si registrano in uno specifico file in un disco. Quando un’applicazione termina di emettere dati da stampare, si aggiunge tale file alla coda di stampa, quest’ultima viene copiata sulla stampante, un file per volta. In certi sistemi operativi questa funzione viene gestito da un processo di sistema specializzato (demone), in altri da un thread del kernel.

In entrambi i casi il SO fornisce **un’interfaccia di controllo** che permette agli utenti e agli amministratori del sistema di esaminare la coda ed eliminare elementi della coda prima che siano stampati, sospendere una stampa e così via.

## 13.4.5 Gestione degli errori

Un SO che usi la protezione della memoria può proteggersi da molti tipo di errori dovuti ai dispositivi o alle applicazioni cosicché il blocco completo del sistema non è l’ordinaria conseguenza di piccoli difetti tecnici.

Dispositivi e i trasferimenti di I/O possono causare errori in molti modi, sia per motivi contingenti, come il sovraccarico di una rete di comunicazione, sia per motivi “permanenti”, come nel caso in cui il controllore dell’unità a disco sia difettoso. I sistemi operativi sono spesso capaci di compensare efficacemente le conseguenze negative dovute a errori generate da cause contingenti , purtroppo, però, è improbabile che il sistema operativo riesco a compensare gli effetti di errori dovuti a disfunzioni permanenti di qualche componente importante. Di norma, una chiamata di sistema per l’I/O riporta un bit di informazione sullo stato di esecuzione della chiamata , denotando con essa la riuscita o l’insuccesso dell'operazione richiesta. Il sistema operativo UNIX usa una variabile intera detta **errno** per codificare piuttosto genericamente il tipo di errore avvenuto.

## 

## 13.4.6 Protezione dell’I/O

Gli errori sono strettamente connessi alla tematica della protezione.

Un processo utente che cerchi di impartire istruzioni I/O illegali può disturbare il funzionamento normale di un sistema, sia che lo faccia intenzionalmente sia accidentalmente.

Onde evitare che gli utenti impartiscano istruzioni di I/O illegali, si definiscono come privilegiate tutte le istruzioni relative all’I/O. Ne consegue che gli utenti non potranno impartire in via diretta alcuna istruzione, ma dovranno farlo attraverso il SO.

Un programma utente per eseguire I/O invoca una chiamata di sistema per chiedere al SO di svolgere una data operazione nel suo interesse.

Inoltre, il sistema di protezione della memoria deve tutelare dall’accesso degli utenti tutti gli indirizzi mappati in memoria e gli indirizzi delle porte di I/O. Il kernel, tuttavia, non può semplicemente negare qualunque tentativo di accesso da parte degli utenti: quasi tutti i videogiochi e i programmi di grafica usano l’accesso diretto al controllore della grafica mappato in memoria. In questi casi il kernel potrebbe applicare dei **lock** per assegnare a un solo processo per volta la porzione della memoria grafica che rappresenta l'appropriata finestra sullo schermo.

## 

## 13.4.7 Strutture dati del kernel

Il **kernel** ha bisogno di mantenere informazioni sullo stato dei componenti coinvolti nelle operazioni di I/O e usa a questo fine diverse strutture dati interne(per es. la tabella dei file aperti).

Il SO Unix, per mezzo del file system, permette l’accesso a diversi oggetti: file degli utenti, dispostivi, spazio degli indirizzi dei processi.

Quando il kernel, ad esempio, deve leggere un file utente ha bisogno di controllare la **buffer cache** prima di decidere l’effettiva esecuzione. Per leggere un disco tramite un’interfaccia a basso livello, il kernel deve accertarsi del fatto che la dimensione dell’insieme dei dati di cui è stato richiesto il trasferimento sia un multiplo della dimensione dei settori del disco e sia allineato con il settore interessato.

Alcuni SO quali windows NT usano metodi orientati agli oggetti in misura più rilevante sullo scambio di messaggi. Quando l’operazione richiesta è di emissione di dati(output) il messaggio contiene i dati da scrivere, quando è di immissione (INPUT) il messaggio contiene l’indirizzo del buffer che si usa per ricevere i dati.

Questo metodo può comportare una minore efficienza rispetto alle tecniche procedurali basate sulla condivisione delle strutture dati, ma semplifica la progettazione e la struttura del sistema di I/O e permette una maggiore flessibilità.

## 13.5 TRASFORMAZIONE DELLE RICHIESTE DI I/O IN OPERAZIONI DEI DISPOSITIVI //da controllare

Si consideri ad esempio la lettura di un file da un’unita a disco. L’applicazione fa riferimento ai dati per mezzo del nome del file: è compito del file system fornire i modi di giungere attraverso la struttura delle directory alla regione del disco appropriata, cioè quella dove i dati del file sono fisicamente residenti.

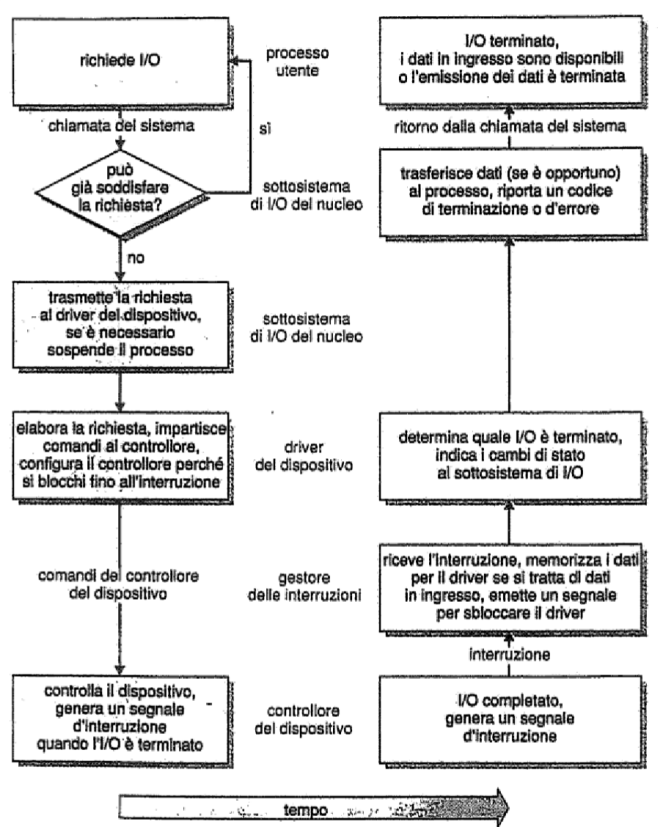
In UNIX il nome è associato a un elemento di una lista di oggetti detti **numeri di *inode***.

La prima parte del nome in DOS (c:) identifica il dispositivo e successivamente la collocazione.

Unix utilizza una tabella di montaggio per associare i prefissi dei nomi di percorso ai corrispondenti nomi di dispostivi.

La seguente descrizione del tipico svolgimento di una **richiesta di lettura bloccante** indica che l’esecuzione di un’operazione di I/O richiede una gran quantità di passi:

1. un processo impartisce una chiamata di sistema **read bloccante** relativa a un descrittore di file di un file precedentemente **aperto**.
2. il codice della chiamata di sistema all’interno del kernel controlla la correttezza dei parametri. Se sono presenti nel buffer cache, si passano i dati richiesti al processo chiamante e l’operazione **è conclusa**.
3. altrimenti è necessario eseguire un’operazione fisica di I/O così si rimuove il processo dalla coda **dei processi pronti** per l’esecuzione per inserirlo nella **coda d’attesa** relativa al dispositivo interessato. E si effettua lo scheduling della richiesta di I/O. Infine il sottosistema per l’I/O invia la richiesta al driver del dispositivo; secondo il sistema operativo ciò avviene tramite la chiamata di una procedura o per mezzo dell’emissione di un messaggio interno al kernel.
4. il **driver del dispositivo** assegna un buffer nello spazio d’indirizzi del kernel che serve per ricevere i dati immessi ed esegue lo scheduling dell’I/O. Infine il driver impartisce i comandi al **controllore del dispositivo** scrivendo nei suoi registri.
5. il **controllore** aziona il dispositivo per compiere il trasferimento dei dati.
6. il driver può eseguire **un’interrogazione ciclica** o può avere predisposto un trasferimento DMA nella memoria del kernel (si assume che il trasferimento sia gestito dal controllore DMA, il quale genera un'interruzione al termine dell'operazione).
7. tramite il vettore le interruzioni, si attiva l’appropriato gestore delle interruzioni, che dopo aver memorizzato i dati necessari, avverte con un segnale il driver del dispositivo e restituisce il controllo.
8. il driver riceve il segnale, individua la richiesta di I/O, si accerta della riuscita o del fallimento dell’operazione e segnala al sottosistema per l’I/O del kernel il completamento dell’operazione
9. il kernel trasferisce dati (nel caso di successo) e/o codici di stato (per esempio , per comunicare la mancata riuscita dell'operazione) nello spazio degli indirizzi del processo chiamante , e sposta tale processo dalla coda d'attesa alla coda dei processi pronti per l'esecuzione.
10. nel momento in cui è posto nella coda dei processi pronti per l’esecuzione il processo non è più bloccato: quando lo scheduler gli assegnerà la CPU, esso riprenderà l’elaborazione. L’esecuzione della chiamata di sistema è completata.



## 13.7 PRESTAZIONI //da controllare

L’I/O è uno tra i principali fattori che influiscono sulle prestazioni di un sistema: richiede un notevole impiego della CPU per l’esecuzione del codice dei driver e per uno scheduling equo ed efficiente dei processi quando essi sono bloccati o riavviati. I risultati cambio di contesto sfruttano fino in fondo la CPU e le sue memorie cache. L'imputato Wood, inoltre, rivela le eventuali inefficienze dei meccanismi del kernel per la gestione delle interruzioni virgola e impiega il bus della memoria durante i trasferimenti dei dati tra controllori dei dispositivi e la memoria fisica, e ancora tra le aree di memoria per l’I/O del kernel e lo spazio di indirizzi delle applicazioni.

Sebbene i calcolatori moderni siano capaci di gestire migliaia di interruzioni al secondo, la gestione delle interruzioni è un processo relativamente oneroso: ciascuna di loro fa sì che il sistema cambi stato, esegua il gestore delle interruzioni e infine ripristini lo stato originario. Se il numero di cicli impiegato nell’attesa attiva non è eccessivo, l’I/O programmato può essere più efficiente di quello basato sulle interruzioni. Il completamento di un'operazione di I/O, in genere implica la riattivazione di un processo, comportando così lo svantaggio in termini di efficienza dovuto a un cambio di contesto.

Anche il traffico di una rete può portare un alto numero di cambi di contesto, si consideri, ad esempio, una connessione distanza fra due calcolatori. Ciascun carattere inserito in un calcolatore deve essere comunicato all'altro: quando si inserisce un carattere nel primo calcolatore, la tastiera produce un segnale di interruzione; il carattere arriva tramite il gestore delle interruzioni al driver del dispositivo, al kernel virgola e quindi al processo utente.

Per eliminare cambi di contesto implicati dal trasferimento di ciascun carattere dal demone del kernel, gli sviluppatori di Solaris hanno implementato nuovamente il demone **telnet** tramite thread interni al kernel. Secondo stime della Sun, queste migliorie hanno portato il massimo numero di connessioni contemporanee sostenibili da un grande server da qualche centinaio a qualche migliaio.

Altri sistemi usano unità di elaborazione specifiche per la gestione dell’I/O determinali, riducendo così il carico della gestione delle interruzioni gravante sulla CPU.

Ad esempio, i concentratori di terminali convogliano il traffico di informazioni proveniente da centinaia di terminali a un’unica porta di un grande calcolatore. I canali di I/O sono unità di elaborazione specializzate presenti nei mainframe e altri sistemi di alto profilo; il loro compito è sollevare la CPU di una parte del peso della gestione dell’I/O.

Per migliorare l’efficienza dell’I/O si possono applicare diversi principi:

* Ridurre il numero dei cambi di contesto
* Ridurre il numero di copiature dei dati in memoria durante i trasferimenti fra dispositivi e applicazioni
* Ridurre la frequenza delle interruzioni
* Aumentare il tasso di concorrenza usando controllori DMA intelligenti o canali di I/O per sollevare la CPU dalle semplici copiature di dati
* Realizzare le primitive direttamente tramite dispositivi fisici, così da permettere che la loro esecuzione sia simultanea alle operazioni di bus e di CPU
* Equilibrare le prestazioni della CPU, del sottosistema per la gestione della memoria, del bus e dell'I/O.

Il driver inoltre esegue raffinati algoritmi di gestione degli errori e di recupero dati e poiché hanno un’incidenza notevole sulle prestazioni complessive del sistema, svolge diverse funzioni di ottimizzazione delle prestazioni dell’unità a disco.