Lezione 25 [13/12/22]

Capitolo 13: Sistemi di I/O

- Introduzione
- Hardware di I/O
- Interfaccia di I/O per le applicazioni
- Sottosistema per l'I/O del kernel
- Trasformazione delle richieste di I/O in operazioni hardware
- STREAMS
- Prestazioni

Introduzione

I due compiti di un computer sono l'I/O e l'elaborazione

Il ruolo di un sistema operativo nell'I/O di un computer è quello di gestire e controllare le operazioni e i dispositivi di I/O

I dispositivi di I/O sono diversi per funzioni e velocità quindi abbiamo diversi metodi di controllo

Questi metodi rappresentano il **sottosistema di I/O** del kernel

Questo sottosistema separa il resto del kernel dalla complessità di gestione dei dispositivi di I/O

Esiste una grande varietà di dispositivi di I/O che rende difficile il lavoro di integrazione

I **driver dei dispositivi** offrono al sottosistema di I/O un'interfaccia uniforme per l'accesso ai dispositivi, così come le system call forniscono un'interfaccia uniforme tra le applicazioni ed il sistema operativo

Architetture e dispositivi di I/O

Esistono tre categorie generali di dispositivi di I/O:

- memoria secondaria e memoria terziaria:
  - dischi, nastri, etc
- dispositivi di trasmissione:
  - modem, schede di rete
- interfaccia uomo macchina:
  - tastiera, schermo, videocamere digitali, etc.

Ogni dispositivo comunica con il sistema di calcolo inviando segnali attraverso un cavo o attraverso l'etere

- Comunica con il calcolatore tramite un punto di connessione, cioè una **porta**

I dispositivi di I/O comunicano tra loro e con il nucleo di sistema mediante un **bus**:

- un insieme di fili ed un protocollo ben definito che specifica l’insieme dei messaggi che si possono inviare attraverso i fili

I messaggi sono inviati tramite configurazioni di livelli di tensione elettrica applicati ai fili con una scansione temporale ben definita

- Quando un dispositivo A ha un cavo che si connette ad un dispositivo B
- Ed il dispositivo B ha un cavo che si connette ad un dispositivo C che a sua è collegato ad una porta di calcolatore
- Si ottiene il cosiddetto **collegamento a margherita** (**daisy chain**) che di solito funziona come un bus

I bus sono diversi tra loro per formato di segnali, throughput e metodo di connessione

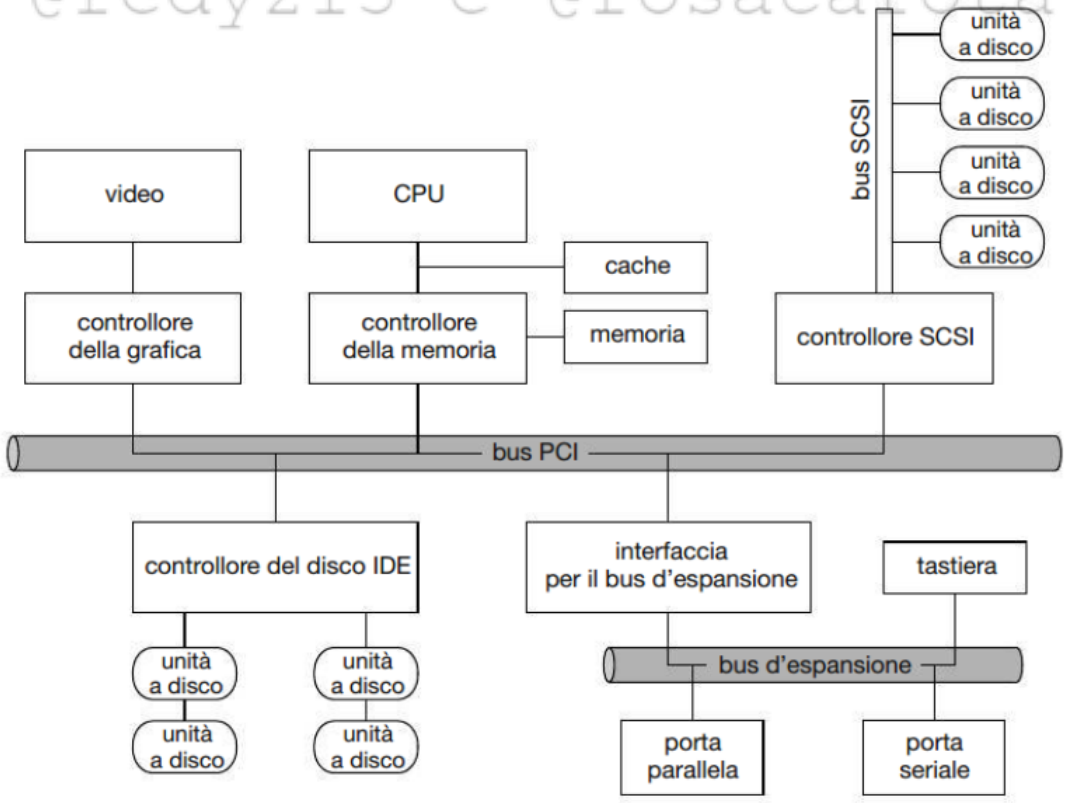
Un classico esempio di bus, che è divenuto uno standard come bus di sistema nei PC è il **bus PCI**:

- Il comune bus di sistema dei PC che connette il sottosistema CPU-memoria ai dispositivi veloci

Un **bus di espansione** viene utilizzato per connettere al sistema i dispositivi relativamente lenti

- tastiera, porte seriali o parallele, etc

Tipica struttura del bus di un PC:



In figura sono presenti quattro dischi collegati ad un bus SCSI (**small computer system interface**) inserito nel relativo controllore. E’ progettato per eseguire il trasferimento

di dati a bus in modalità parallela

Altri bus utilizzati:

- **PCI Express**
- **HyperTrasport**

Hardware di I/O

Hardware di I/O: controllori

La parte hardware che svolge il ruolo di interfaccia tra il bus e i componenti di I/O è chiamata **controllore**

Un controllore è un insieme di componenti elettronici che può far funzionare una porta, un bus o un dispositivo

La complessità hardware di un controllore dipende dall'attività che esso deve svolgere su dispositivo

Esistono controllori semplici, come ad es. il controllore delle porte seriali:

- un singolo circuito integrato che controlla i segnali presenti nei fili della porta seriale

Altri, invece, possono disporre di un proprio microprocessore, di una propria memoria, dei propri registri e addirittura di piccoli sistemi operativi microcodificati per svolgere le operazioni dedicate esclusivamente al controllore

- Ad es. il controllore del bus SCSI è spesso realizzato con una scheda circuitale separata (**adattatore**) che contiene un'unità di elaborazione, microcodice e memoria locale che gli consentono di elaborare ci messaggi del protocollo SCSI

Hardware di I/O: scambio di dati

Il processore dà comandi e fornisce dati al controllore per portare al termine trasferimenti di I/O tramite i suoi registri di dati e di controllo

Quindi la comunicazione tra processo e controllore avviene mediante la scrittura e la lettura di configurazioni di bit in tali registri da parte del processore

La comunicazione può avvenire in due modi:

- **Utilizzo di speciali istruzioni di I/O:** L'istruzione di I/O attiva le linee di bus che selezionano il giusto dispositivo ed trasferiscono bit nei (o dai) registri del dispositivo stesso
- **I/O associato alla memoria (I/O memory mapped):** Ai registri del controllore viene assegnato una sequenza di indirizzo in memoria centrale, in modo che ogni operazione di lettura o scrittura a questi indirizzi comporta un trasferimento diretto di dati con i registri del dispositivo.
  - **Vantaggi:** Aumento dell'efficienza del sistema; il S.O. non deve preoccuparsi di cercare spazio disponibile in memoria dove inserire le informazioni
  - **Svantaggi:** Rischio di avere errori software (scritture in una regione sbagliata della memoria): necessità di tecniche di protezione della memoria

Hardware di I/O: porte di I/O

Una **porta I/O** permette la trasmissione di informazioni tra il sistema centrale e un dispositivo

Tipicamente consiste di 4 registri:

- **Data-in**: Mantiene i dati che vengono letti dalla CPU (host)
- **Data-out**: E' il registro in cui l'host scrive i dati in output
- **Status:** Contiene dei bit che possono essere letti dall'host e indicano lo stato della porta
  - Ad es. indicano se è stata portata a termine l'esecuzione del comando corrente, se un byte è disponibile per essere letto dal registro data-in, se si è verificato un errore nel dispositivo
- **Control:** Viene scritto dall'host per attivare un comando o per cambiare il modo operativo del dispositivo
  - Ad esempio, un bit nel registro può decidere la lunghezza delle parole (7 o 8 bit), il tipo di comunicazione, altri la velocità che la porta seriale può sostenere

La tipica dimensione dei registri varia 1 e 4 byte

Indirizzi delle porte dei dispositivi di I/O nei PC (elenco parziale):

indirizzi per l'I/O (in esadecimale)	dispositivo
000-00F	controllore DMA
020-021	controllore delle interruzioni
040-043	timer
200-20F	controllore dei giochi
2F8-2FF	porta seriale (secondaria)
320-32F	controllore del disco
378-37F	porta parallela
3D0-3DF	controllore della grafica
3F0-3F7	controllore dell'unità a dischetti
3F8-3FF	porta seriale (principale)

Interrogazione ciclica (Polling)

Con il crescere del numero dei dispositivi cresce l'esigenza di avere un controllo continuo e asincrono sull'I/O

Il modo in questo avviene è attraverso la negoziazione (**handshaking**)

Spiegazione handshaking con un esempio:
Si assuma l'uso di due bit per controllare la relazione di tipo produttore/consumatore fra il controllore e la CPU

Il controllore specifica il suo stato per mezzo del bit **busy** del registro **status**

Pone a 1 il bit busy quando è impegnato in una operazione e lo pone a 0 quando è pronto per il comando successivo

La CPU comunica le sue richiesta tramite il bit **command ready** del registro **command**

Pone questo bit a 1 quando il controllore deve eseguire un comando

In questo esempio, la CPU scrive in una porta coordinandosi con un controllore tramite handshaking in questo modo:

1. La CPU legge ripetutamente il bit **busy** fino a che esso non vale 0
2. La CPU pone a 1 bit **write** nel registro **command** e scrive un byte nel registro **data out**
3. La CPU pone a 1 il bit **command ready**
4. Quando il controller si accorge che il bit **command ready** è posto a 1, pone a 1 il bit busy
5. Il controller legge il registro **command** e riceve il comando **write**, legge il registro **data-out** per ottenere il byte da scrivere, ed effettua l'operazione di I/O nel dispositivo
6. Il controller pone a 0 il bit **command-ready.** Se l'operazione di I/O è stata eseguita correttamente pone a 0 il bit **error** nel registro status e pone a 0 il bit **busy** per indicare che l'operazione è terminata

Questa sequenza viene ripetuta per ogni byte

Durante l'esecuzione nel passo 1 la CPU è in **attesa attiva** o in **interrogazione ciclica (polling):**

- Itera la lettura del registro **status** fino a che il bit busy assume il valore 0

Nell maggior parte delle architetture basano tre istruzioni alla CPU per effettuare il polling di un dispositivo:

- *read* → lettura di un registro del dispositivo
- *logical-and* → usata per estrarre il valore di un bit di stato
- *branch* → salto ad un altro punto del codice se l'argomento è diverso da 0

Il polling risulta inefficiente se le ripetute interrogazioni trovano raramente un dispositivo pronto: in tali casi è preferibile utilizzare **interruzioni**, cioè il controllore comunica alla CPU che il dispositivo è pronto

Interruzioni

(gestione degli eventi asincroni)

L'hardware della CPU ha un contatto (un input) detto **linea di richiesta dell'interruzione** del quale controlla lo stato dopo l'esecuzione di ogni istruzione

Quando rileva il segnale di un controllore, la CPU memorizza lo stato del processo corrente e salta alla **procedura di gestione delle interruzioni (interrupt-handler routine)** che si trova ad un indirizzo prefissato: tramite questa procedura gestirà l'interruzione

Questa procedura determina le cause dell'interruzione, porta a termine l'elaborazione necessaria, ripristina lo stato ed esegue un'istruzione **return from interrupt** per far si che la CPU ritorni nello stato in cui si trovava prima della sua interruzione

1. Il controllore genera un'interruzione della CPU sulla linea di richiesta delle interruzioni
2. La CPU la rileva e recapita al gestore delle interruzioni
3. Il gestore delle interruzioni gestisce l'interruzione corrispondente servendo il dispositivo

Nei sistemi più moderni però ci devono essere maggiori capacità per la gestione delle interruzioni:

- Si deve poter postporre in caso di fasi critiche di elaborazione
- Si deve disporre di un meccanismo efficiente per passare il controllo all'appropriato gestore delle interruzioni, senza dover effettuare polling per determinare quale abbia generato l'interruzione
- Si deve disporre di più livelli di interruzione per definire la priorità di ogni interruzione

Queste caratteristiche nei sistemi moderni sono garantite dal **controllore hardware delle interruzioni**

La maggior parte delle CPU ha due linee di richiesta delle interruzioni:

- **Interruzioni non mascherabili:** Riservata agli errori di memoria irrecuperabili
- **Interruzioni mascherabili:** può essere mascherata dalla CPU prima di una sequenza di critica di istruzioni; è usata da i controllori per richiedere un servizio

Il meccanismo di interruzioni accetta un **indirizzo**: un numero che seleziona una specifica procedura di gestione delle interruzioni da un insieme ristretto. Questo è un offset nel **vettore delle interruzioni**

Vettore delle interruzioni:

- Contiene gli indirizzi di memoria degli specifici gestori delle interruzioni
- Tabella che associa ad ogni interrupt d'indirizzo di una corrispondente routine di gestione

- Lo scopo è di ridurre la necessità che un singolo gestore debba individuare tutte le possibilità fonti di interruzione per determinare chi ha richiesto il servizio

  I dispositivi hanno più gestori delle interruzioni che elementi nel vettore delle interruzioni: la risoluzione di questo problema è la **concatenazione delle interruzioni** in cui ogni elemento del vettore delle interruzioni punta alla testa di una lista di gestori. Quando si ha un'interruzione si chiamano uno alla volta i gestori pe verificare chi può soddisfare la richiesta

Il meccanismo delle interruzioni ha anche un sistema di **livelli di priorità delle interruzioni che** permette alla CPU di differire la gestione delle interruzioni di bassa priorità e permette ad una interruzione di priorità alta di sospendere l'esecuzione di quella di una bassa

Il meccanismo delle interruzioni viene usato anche per gestire le **eccezioni** (divisione per 0, tentativo di accesso a indirizzi di memoria protetti)

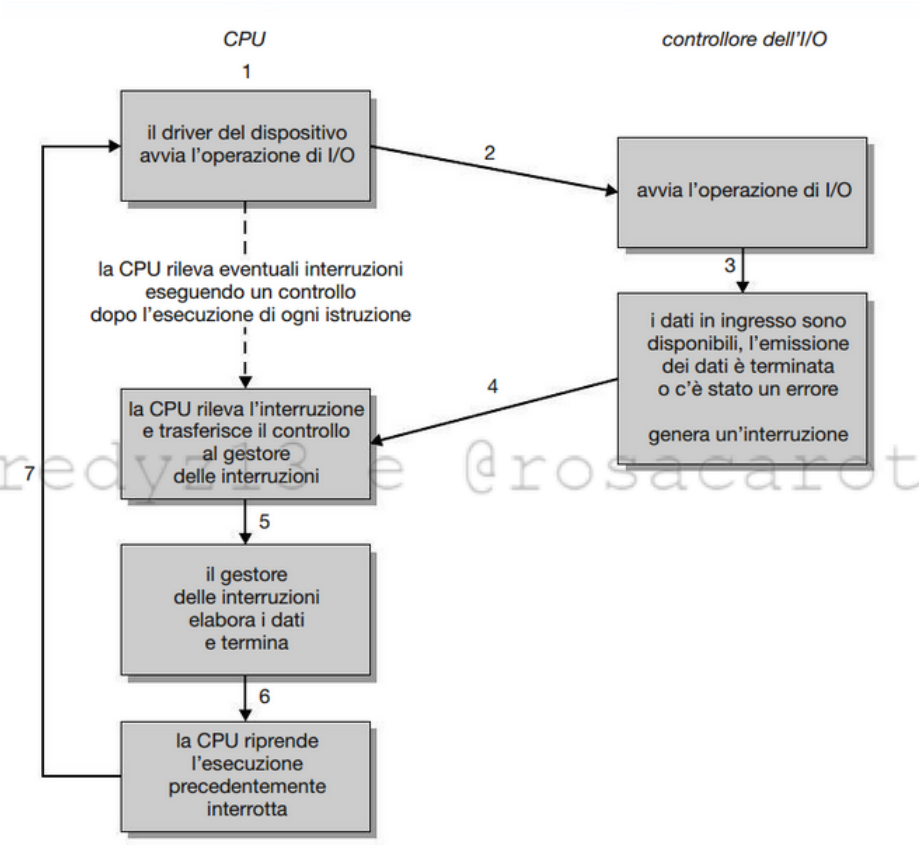
Gestione delle **chiamate a sistema**:
Solitamente i programmi sfruttano routine di libreria per eseguire chiamate al sistema: la routine controlla i parametri pasati dall'applicazione, li racchiude in una struttura dati da passare al kernel ed esegue un'**interruzione software o trap.** Questa istruzione ha un operando che identifica il servizio kernel desiderato
Quindi, l'hardware delle interruzioni salva lo stato del codice utente, passa in modalità kernel e recapita l'interruzione alla procedura del kernel che realizza il servizio

Gestione della lettura di un disco:

Una **coppia** di gestori delle interruzioni realizza il codice del kernel che compie le letture dai dischi

- Gestore ad alta priorità: mantiene le informazioni sullo stato dell'I/O, risponde al segnale di interruzione del dispositivo, avvia il prossimo I/O in attesa e genera un'interruzione a bassa priorità
- Gestore corrispondente all'interruzione a bassa priorità generata in precedenza: completa l'I/O a livello utente copiando i dati dal buffer del kernel a quello dell'applicazione e richiamando poi lo scheduler per aggiungere l'applicazione alla coda dei processi pronti

Ciclo di I/O basato sulle interruzioni:



Vettore delle interruzioni della CPU Intel Pentium:

indice del vettore	descrizione
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19-31	(Intel reserved, do not use)
32-255	maskable interrupts

Accesso diretto alla memoria (DMA)

I/O programmato (programmed I/O - PIO):scrittura di dati nel registro del controllore un byte alla volta. In caso di grandi quantità di dai risulta inefficiente

In molti sistemi di elaborazione si utilizza una unità di elaborazione (un processore) specializzata detta controllore dell'a**ccesso diretto alla memoria (DMA)**

Il controllore di DMA è dedicato al trasferimento diretto delle informazioni dalle periferiche alla memoria centrale

Il DMA viene utilizzato nelle operazioni di trasferimento di grosse quantità di dati:

- Per dare avvio ad un trasferimento DMA, la CPU scrive in memoria un comando che contiene un puntatore alla locazione dei dati, un altro alla destinazione ed il numero di byte da trasferire
- Poi passa l'indirizzo del comando al controllore del DMA e prosegue con l'esecuzione di altro codice

Il controllore DMA agisce direttamente sul bus della memoria, presentando al bus gli indirizzi di memoria necessari al trasferimento senza l'aiuto della CPU

L'handshaking tra il controllore del DMA e il controllore del dispositivo avviene tramite due fili: DMA-request e DMA-acknowledge:

1. Il controllore del dispositivo manda un segnale sulla linea DMA-request quando una word di dati è disponibile per il trasferimento
2. Il controllore DMA prende possesso del bus di memoria
    a. Presenta l'indirizzo desiderato ai fili di indirizzamento della memoria
    b. Manda un segnale lungo la linea DMA-acknowledge
3. Il controllore del dispositivo riceve il segnale: trasferisce, quindi, in memoria la word di dati e rimuove il segnale dalla linea DMA-request

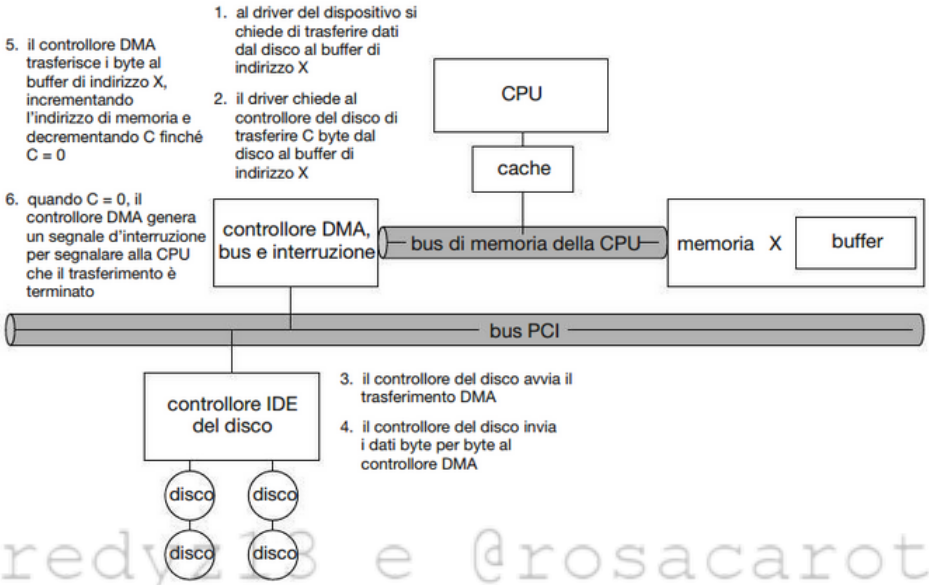
Una volta terminato il trasferimento, il DMA interrompe la CPU

Sottrazione di cicli: Avviene quando il DMA prende possesso del bus di memoria, la CPU si ritrova impossibilitata ad accedere alla memoria centrale, nonostante possa accedere alla

cache primaria e secondaria

La tecnica DMA può essere implementata tramite indirizzi fisici o tramite **l'accesso diretto alla memoria virtuale (direct virtual-memory access - DVMA)**: in questo l'accesso diretto avviene allo spazio di indirizzi virtuali. Gli indirizzi virtuali verranno poi tradotti in quelli fisici

Passi di un trasferimento DMA:



Interfaccia di I/O per le applicazioni

E' necessario avere un trattamento uniforme dei dispositivo di I/O quindi le chiamate di sistema di I/O incapsulano il comportamento dei dispositivi in alcuni tipi generali

A queste tipologie di dispositivi si accede tramite un insieme standard di funzioni, un'**interfaccia**

Le effettive differenze tra i dispositivi sono contenute nei driver

- Moduli del kernel dedicati a controllare ogni diverso dispositivo
- Lo scopo dello strato di driver è di nascondere al sottosistema di I/O del kernel le differenze fra i controllori dei dispositivi (simile a quello che si è detto prima per le chiamate di sistema)
- Sfortunatamente ogni tipo di S.O. ha le sue convenzioni riguardanti l'interfaccia dei driver dei dispositivi
- Così un dato dispositivo sarà venduto con driver diversi, ad es. per MS-DOS, Windows, UNIX, etc.

Per l'accesso delle applicazioni ai dispositivi, le chiamate di sistema raggruppano tutti i dispositivi in poche classi generali, uniformando i modi di accesso

Solitamente queste classi sono:

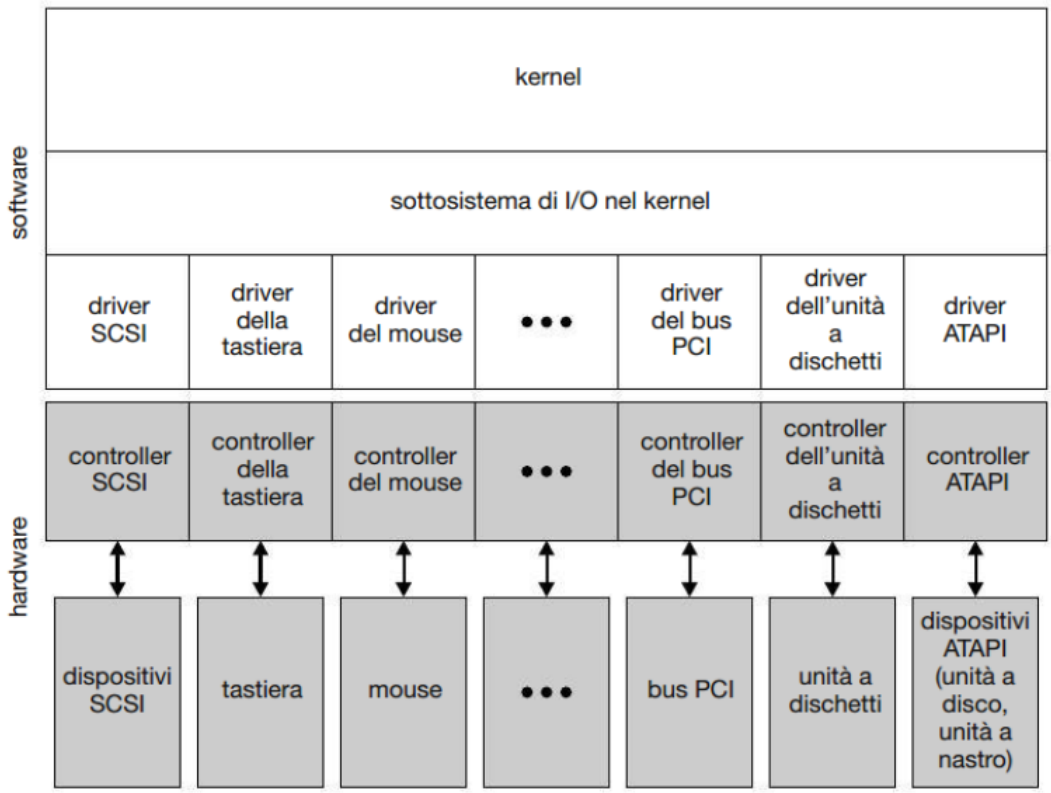
- **I/O a blocchi o a flusso di caratteri:**
  - A Flusso di caratteri: Trasferisce dati un byte alla volta
  - A Blocchi: Trasferisce un blocco di byte alla volta
- **Sequenziale o accesso diretto:**
  - Sequenziale: Trasferisce dati secondo un ordine fisso dipendente dal dispositivo
  - Accesso diretto: Si può richiedere un accesso a qualunque delle possibili locazioni di memorizzazione
- **Dispositivi sincroni o asincroni:**
  - Sincrono: Trasferisce dati con un tempo di risposta prevedibile, in maniera coordinata rispetto al resto del sistema
  - Asincrono: Ha tempi di risposta irregolari o nn prevedibili, non coordinati rispetto al resto del sistema
- Condivisibili e dedicati
  - Condivisibile: Può essere usato in modo concorrente da processu o thread
  - Dedicato: Non può essere usato in modo concorrente
- **Velocità di funzionamento**
- **Lettura e scrittura, solo lettura o solo scrittura**

Le convenzioni i accesso principali includono:

- L'I/O a blocchi o a flusso di caratteri
- L'accesso ai file mappati in memoria
- Socket di rete

Spesso è disponibile una system call scappatoia (**back door**) dove si fa rientrare ciò che non rientra nei casi precedenti (ad es. **ioctl** in UNIX)

Struttura relativa all'I/O di un nucleo



Caratteristiche dei dispositivi di I/O

aspetto	variazione	esempio
modalità di trasferimento dei dati	a caratteri a blocchi	terminale unità a disco
modalità d'accesso	sequenziale casuale	modem lettore di CD-ROM
prevedibilità dell'I/O	sincrono asincrono	unità a nastro tastiera
condivisione	dedicato condiviso	unità a nastro tastiera
velocità	latenza tempo di ricerca velocità di trasferimento attesa fra le operazioni	
direzione dell'I/O	solo lettura solo scrittura lettura e scrittura	lettore di CD-ROM controllore della grafica unità a disco

Dispositivi a blocchi e a caratteri

L'interfaccia per i **dispositivi a blocchi** sintetizza gli aspetti necessari per accedere alle unità a disco e ad altri dispositivi basati sul trasferimenti di blocchi di dati

Comandi tipici:

- *read*
- *write*
- *seek* (in caso di accesso casuale)

Di solito le applicazioni comunicano con i dispositivi tramite un file system che funge da interfaccia

Altre volte invece si usa I/O a basso livello

I/O a basso livello: il S.O. e le altre applicazioni particolari trattano questi dispositivi come una sequenza lineare di blocchi
Esso passa il controllo del dispositivo direttamente all'applicazione togliendo il S.O. di mezzo. Questo implica che non sarà disponibile nessun servizio del S.O. sul dispositivo

I file possono essere associati alla memoria

- Si fa coincidere una parte dello spazio indirizzi virtuale di un processo con il contenuto di un file
  La chiamata a si tema che associa un file a una regione di memoria restituisce l'indirizzo di memoria virtuale di una copia del file
- Gli effettivi trasferimenti sono eseguiti solo quando necessari per soddisfare una richiesta d'accesso all'immagine in memoria

I **dispositivi a flusso di carattere** sono dispositivi che generano o accettano uno stream di dati

- Ad es. tastiera, mouse, porte seriali
- Comandi tipici: *get*, *put* di singoli caratteri o parole

Non è possibile la *seek*

Dispositivi a rete

I modi di indirizzamento e le prestazioni tipiche dell'I/O di rete sono differenti da quelli dei dispositivi a blocchi o caratteri

La maggior parte dei S.O. fornisce un'interfaccia per l'I/O di rete diversa da quella per i dichi

L'interfaccia per l'I/O di rete fornita su molti S.O. è l'interfaccia di rete **socket**

Le system call fornite da questa interfaccia permettono di:

- Creare un socket
- Collegare un socket locale all'indirizzo di un altro punto della rete (permette di collegare questa applicazione alla socket creata da un'altra applicazione)
- Controllare se un'applicazione si sia inserita nel socket locale
- Inviare o ricevere pacchetti di dati lungo la connessione

Una funzione **select()** gestisce un insieme di socket, restituendo informazioni su:

- I socket per i quali sono presenti pacchetti che attendono di essere ricevuti
- Su quelli che hanno spazio per accettare un pacchetto da inviare

La select elimina la necessità di polling e di busy waiting

Orologi e temporizzatori

La maggior parte dei computer ha timer e orologi hardware che offrono tre funzioni essenziali:

- Segnare l'ora corrente
- Segnalare il lasso di 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 è detto **temporizzatore programmabile**

Si può regolare in modo da attendere un certo tempo e poi generare un segnale di interruzione

- Viene utilizzato dallo scheduler anche per segnalare lo scadere della slice di tempo di un processo o per riversare periodicamente nei dischi il contenuto del buffer cache da arte del sottosistema di I/O dell'unità a disco, etc.

Possono anche essere simulati orologi virtuali in modo che si possano soddisfare più richieste rispetto al numero dei timer fisici

Per l'ora corrente vi è un contatore di interrupt ad alta frequenza chiamato **orologio hardware**

I/O bloccante e non bloccante

Un altro aspetto delle chiamate a sistema è la scelta tra I/O bloccante e non bloccante

Una system call **bloccante** sospende l'esecuzione dell'applicazione che l'ha invocata

- Questa passa dalla coda dei processi pronti alla coda d'attesa del sistema
- Quando la chiamata a sistema termina, l'applicazione torna nella coda dei processi pronti

Alcuni processi necessitano di una forma **non bloccante** di I/O

- Ad es., interfaccia utente con cui si interagisce col mouse e la tastiera mentre elabora dati e li mostra su schermo

Una soluzione potrebbe essere la costituzione di applicazioni multithread

Alcuni S.O. mettono a disposizione una system call non bloccante che restituisce rapidamente il controllo all'applicazione che l'ha invocata

- Fornisce un parametro che indica quanti byte di dati sono stati trasferiti

Un'alternativa alle system call **non bloccanti** è costituita dalle **chiamate del sistema asincrone**

- Restituiscono il controllo senza aspettare che l'I/O sia terminato
- L'applicazione continua ad essere eseguita
- Il completamento dell'I/O viene successivamente comunicato all'applicazione per mezzo di una variabile o di un interrupt software o tramite una routine di ritorno (callback)

Non bloccante → Restituisce immediatamente il controllo, fornendo i dati che è stato possibile leggere

Asincrono → Trasferimento di cui il sistema garantisce il completamente ma in un momento successivo e non prevedibile

I/O vettorizzato

I/O vettorizzato: permette ad una chiamata di sistema di eseguire più operazioni di I/O che coinvolgono più locazioni. Questo metodo viene chiamato anche **scatter-gather**

Vantaggi:

- Nessun cambio di contesto
- Nessun overhead per il sistema
- Alcuni metodi forniscono l'atomicità → si evita la corruzione dei dati

Sottosistema per l'I/O del kernel: scheduling

Il kernel fornisce i servizi riguardanti l'I/O

- Molti sono offerti dal sottosistema per l'I/O del kernel e sono realizzati a partire dai dispositivi e dai relativi driver

Fare lo scheduling di un insieme di richieste di I/O significa stabilirne un ordine di esecuzione efficace

Lo scheduling è necessario per:

- Concorrere a migliorare le prestazioni globali del sistema
- Distribuire equamente gli accessi ai dispositivi
- Ridurre il tempo di attesa medio per il completamento di un'operazione di I/O associata ad un dispositivo
    - Ad es. riordinamento delle sequenze di servizio di richieste relative ad operazioni di lettura su disco

I progettisti di S.O. realizzano lo scheduling mantenendo una coda di richieste di I/O associata ad ogni dispositivo

- Quando una applicazione richiede l'esecuzione di una chiamata a sistema di I/O bloccante, si aggiunge la richiesta alla coda relativa al dispositivo
- Lo scheduler dell'I/O riorganizza l'ordine della coda per migliorare l'efficienza globale del sistema ed il tempo medio di attesa cui sono sottoposte le applicazioni

I kernel che mettono a disposizione un I/O asincrono devono tener traccia di più richieste di I/O: per questo, viene aggiunta una **tabella dello stato dei dispositivi** alla coda dei processi in attesa. tutti i dispositivi di I/O si trovano in questa tabella; qui sono riportati il tipo, l'indirizzo e lo stato del dispositivo

Lo scheduling dell'I/O è uno dei modi in cui il sottosistema per I/O migliora l'efficienza di un calcolatore

Un altro è 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

Memorizzazione transitoria

Una **memoria di transito (buffer)** è un'area di memoria che contiene dati mentre essi sono trasferiti tra due dispositivi o tra una applicazione e un dispositivo

I buffer vengono utilizzati per tre ragioni principali:

- Gestione della differenza di velocità tra produttore e consumatore per la sincronizzazione tra i due
  - (ad es. modem → disco: Si attua una **doppia bufferizzazione**, quindi si crea un buffer nella memoria principale per accumulare i byte che arrivano dal modem e si necessita di anche un altro buffer in caso si stia effettuando il trasferimento di dati da quello precedente (trasferimento lento))
- Gestione dei dispositivi che trasferiscono blocchi di dati in blocchi di dimensioni diverse (ad es. scambi di messaggi di rete)
- Realizzare la **semantica delle copie**, meccanismo per la coerenza delle informazioni (ad es. applicazione → disco)
  - Si supponga che una applicazione disponga di una area di memoria contente dati da trasferire su disco e che richieda al sistema di effettuare una *write* su disco
  - Cosa succede se dopo che la chiamata a sistema restituisce il controllo all'applicazione questa modifica il contenuto della memoria?
  - La semantica delle copie garantisce che la versione dei dati scritta su disco sia conforme a quella al momento della chiamata a sistema (la chiamata copia i dati in un buffer del kernel prima di restituire il il controllo e il trasferimento avviene da lì)

Cache

Una **cache** è una memoria ad alta velocità utilizzata dai dispositivi di I/O per rendere più efficace il trasferimento di dati
Essa viene utilizzata per mantenere copie di dati contenuti in un dispositivo

L'uso della cache e l'uso dei buffer sono due funzioni distinte
Caching vs buffering (memorie di transito):

- A volte una stessa regione di memoria può essere usata per entrambi gli scopi
  - Ad es., per permettere un efficiente I/O su disco abbiamo buffer in memoria centrale per i dischi che sono anche usati come cache
- Differenza tra un buffer e una cache:
  - Buffer → Contiene dati di cui non esiste un'altra copia
  - Cache → mantiene su un pezzo più efficiente una copia di informazioni memorizzate altrove

Code di spooling e riservazione 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 (ad es. una stampante)

La tecnica dell'accodamento **(spooling)** viene realizzata utilizzando un buffer particolare: lo **spool**

Con spool si intende la sequenza di jobs o la storage area

La tecnica dello spooling viene utilizzata per ottenere un uso esclusivo dei dispositivi

- Per esempio, quando più applicazioni richiedono simultaneamente la stampa del proprio output su di una stampante, è necessario l'utilizzo di questa tecnica per gestire le stampe in modo esclusivo, evitando così stampe di documenti con dati interfogliati

Quindi consente al S.O. di coordinare un output simultaneo

In alcuni S.O. lo spooling è gestito da un processo di sistema specializzato **(daemon di spool)**, in altri da un thread del kernel

Gestione degli errori

Il S.O. deve proteggersi dal malfunzionamento dei dispositivi

Gli errori possono essere:

- Transitori
  - Ad es. rete sovraccarica
- Permanenti
  - Ad es. disco rotto

Nel caso di situazioni transitorie, solitamente il S.O. può tentare di recuperare la situazione

- Ad es. richiedere che si effettui di nuovo l'operazione di I/O

Le chiamate di sistema, quando non vanno a buon fine, segnalano una situazione di errore (restituiscono un bit)

In situazioni con errori permanenti, la gestione degli errori è più difficile

Alcuni S.O., come UNIX forniscono una mappatura degli errori

Alcuni tipi di hardware, poiché gli errori sono in relazione al tipo di dispositivo, forniscono un protocollo per la rilevazione dell'errore

- Ad es. Standard SCSI:
  - Ha 3 livelli di dettaglio:
    - Ha **codici di rilevazione** per identificare la natura generale dell'errore
    - Ha **codici di rilevazione addizionale** che desctive la categoria a cui appartiene l'errore
    - Ha un **qualificatore del codice di rilevazione addizionale** che fornisce ancora più dettagli

Protezione dell'I/O

Per evitare che gli utenti effettuino operazioni illegali, tutte le operazioni di I/O sono operazioni privilegiate: le istruzioni ai dispositivi non sono impartite in maniera diretta, ma tramite il S.O.

Il sistema, una volta ricevuta la richiesta, passa alla modalità privilegiata per poi operare in caso essa sia valida

Strutture di dati del kernel

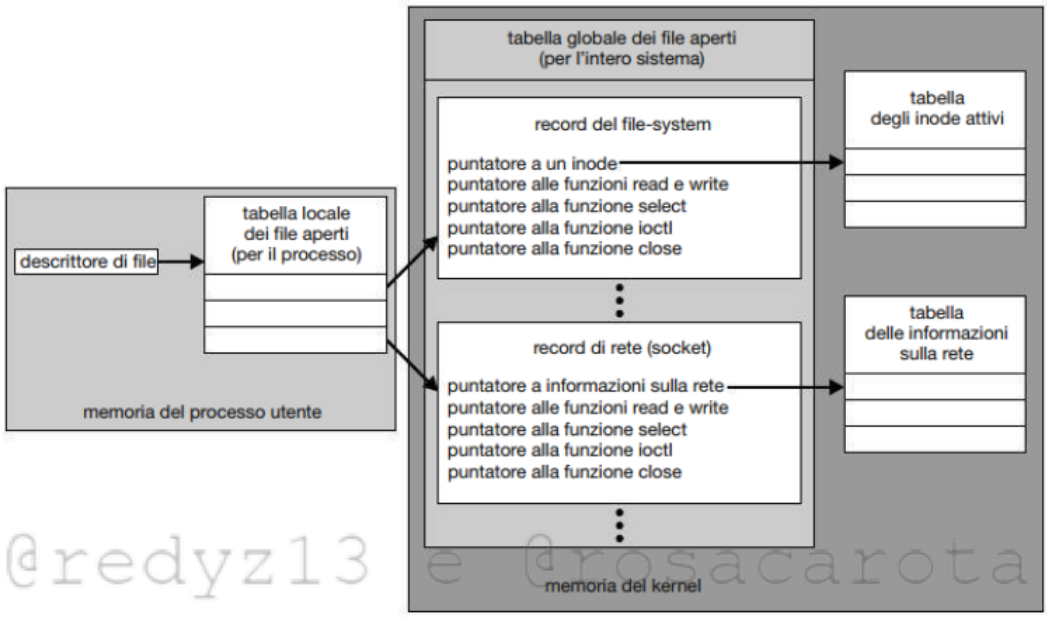
Il S.O. utilizza particolari strutture dati interne al kernel per mantenere informazioni riguardo allo stato dei componenti coinvolti nelle operazioni di I/O

- Come la tabella dei file aperti

Sia UNIX che Windows risolvono il problema dell'ottenere una struttura dati che consenta di effettuare migliori operazioni, seguendo quello che è l'approccio della **programmazione orientata agli oggetti**

- Cioè aggregano nella struttura dati sia dati che metodi (funzioni)

Strutture di dati del nucleo per la gestione dell'I/O in UNIX



Trasformazione delle richieste di I/O in operazioni dei dispositivi

Come si può giungere da un nome di file ad un controllore di un dispositivo?

In MS-DOS la prima parte del nome di un file identifica in modo univoco una periferica:

- Ad es. **C:** è la parte iniziale di ogni nome di file residente nell'unità dischi principali
- Per convenzione del S.O. a **C:** viene associato uno specifico indirizzo di porta per mezzo di una tabella dei dispositivi
- Con i **:** si divide lo spazio dei nomi dei dispositivi dallo spazio dei nomi del file system

In UNIX non c'è nel pathname una vera e separazione fra il dispositivo interessato ed il nome del file, quindi rappresenta i nomi dei dispositivi all'interno dell'ordinario spazio dei nomi del file system

Viene impiegata una **tabella di montaggio (mount table)**, per associare un prefisso di pathname ad uno specifico nome di dispositivo

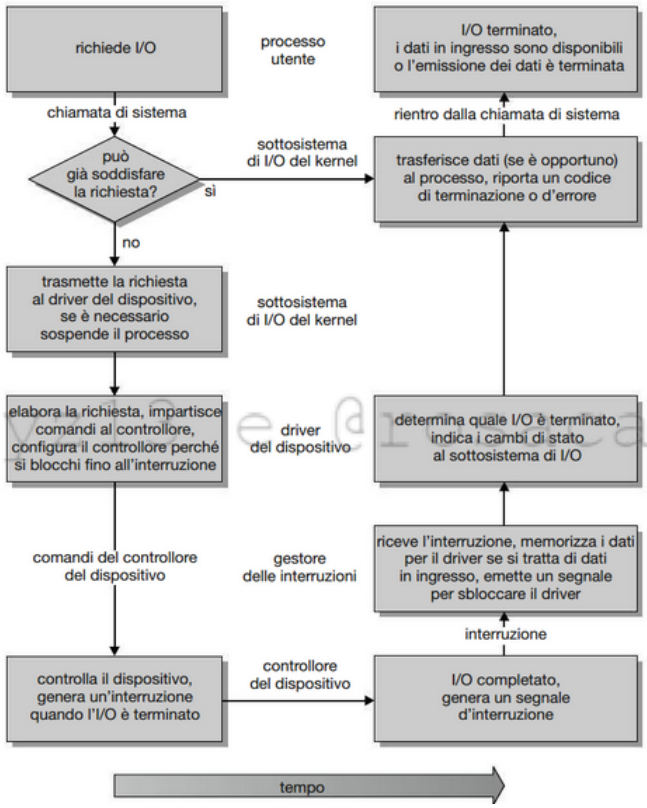
Quando deve risolvere il nome di percorso, il sistema esamina la tabella per trovare il più lungo prefisso corrispondente

- Questo elemento indica il nome del dispositivo voluto

Il nome del dispositivo viene rappresentano come un oggetto del file system

- Non come un inode, ma come una coppia di numeri **<principale, secondario>**
  - **Principale** individua il driver del dispositivo che deve essere usato per gestire l'I/O
  - **Secondario** individua l'indirizzo della porta o l'indirizzo mappato in memoria del controller del dispositivo

Esecuzione di una richiesta di I/O



Prestazioni

L'I/O è un fattore predominante nella performance di un sistema

- Consuma tempo di CPU per eseguire i driver e il codice kernel di I/O
- Continui cambi di contesto all'avvio dell'I/O e alla gestione degli interrupt
- Trasferimenti dati da/per buffer consumano cicli di clock e spazio in memoria
- Il traffico di rete è particolarmente pesante (es. telnet)

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 nella memoria durante i trasferimenti tra dispositivi e applicazioni

- Ridurre la frequenza degli interrupt preferendo trasferimenti di grandi quantità di dati: usare controllori intelligenti e l’interrogazione ciclica
- Aumentare il tasso di concorrenza usando controllori DMA o bus dedicati, per sollevare la CPU dalle semplici copiature di dati
- Implementare le primitive in hardware, dove possibile, per aumentare il parallelismo
- Equilibrare le prestazioni di CPU, del sottosistema per la gestione della memoria, del bus e dell’I/O: il sovraccarico di un elemento comporta l’inutilizzo degli altri