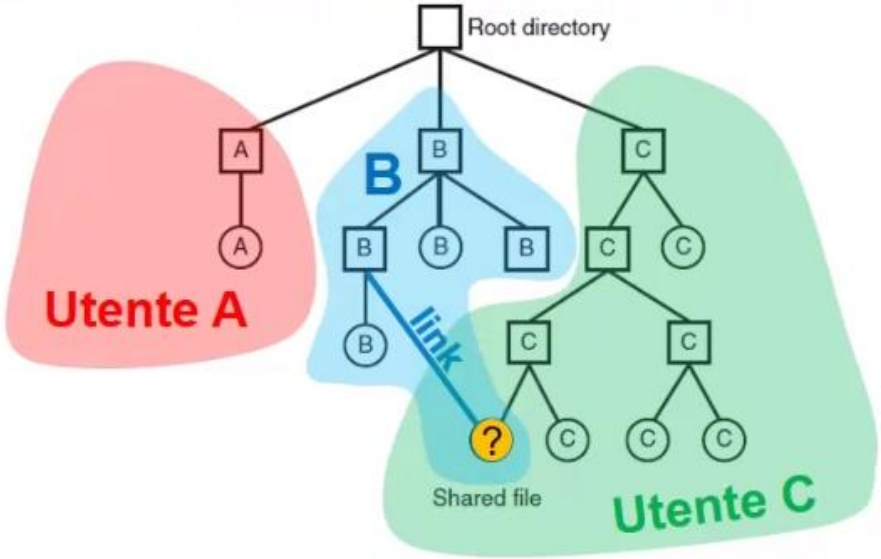
File condivisi

- Quando più utenti lavorano insieme su un progetto, hanno bisogno di condividere file. In questi casi è conveniente far apparire il file condiviso nelle diverse directory degli utenti.
- Il collegamento che permette ad un utente di vedere in una sua directory il file condiviso è chiamato **link**.



- Il file system degenera così in un **grafo aciclico orientato** o, semplicemente, **DAG** (**Directed Acyclic Graph**).

Problemi con i file condivisi

- La condivisione è comoda, ma introduce dei problemi.
- Se le directory contengono gli indirizzi del disco, quando un file è collegato occorre copiarli. A questo punto se uno dei due utenti appende dati, i nuovi blocchi sono visibili solo all'utente che li ha inseriti vanificando il concetto di condivisione, due soluzioni:

UNIX: i blocchi non sono elencati nella directory ma in una struttura dati associata al file (**i-node**).

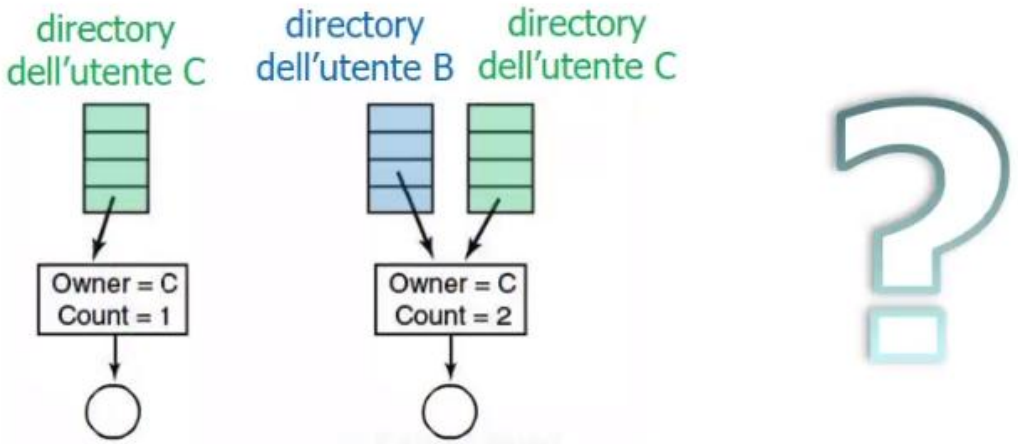
Link simbolico: il collegamento crea un nuovo file di tipo link che permette al sistema operativo di accedere al file sorgente (quello di C).

Problemi con i file condivisi

- Ciascuno di questi metodi ha i suoi svantaggi:

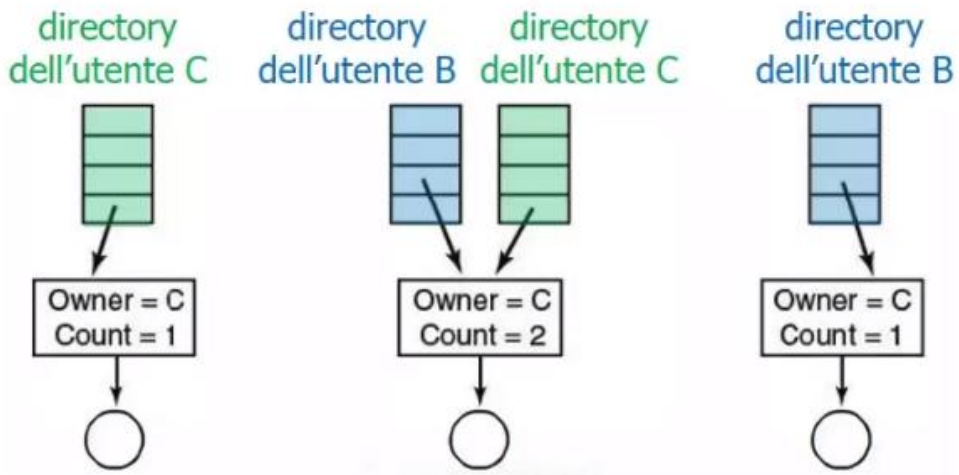
1. Quando B collega il file di C, l'i-node registra C come proprietario del file e il contatore dei riferimenti al file a 2 (Count).

- Se ora C rimuove il file, se si cancella l'i-node, B avrà una voce che punta ad un i-node non valido oppure, se l'i-node è stato riassegnato ad un altro archivio, ad un file scorretto.



Problemi con i file condivisi

- L'unica cosa possibile è cancellare la voce dalla Directory di C e mantenere l'i-node in C (con Count=1 e Owner=C).
- B è l'unico utente che ha una voce di directory per un file di proprietà di C.



Problemi con i file condivisi

2 Con i collegamenti simbolici questo problema non si verifica perché solo il vero proprietario ha un puntatore per l'i-node. Quando il proprietario rimuove il file, viene distrutto.

- I collegamenti simbolici richiedono più tempo di gestione.
- Il file contenente il path deve essere letto, quindi il percorso analizzato e seguito fino a raggiungere l'i-node.
- Tutte queste attività possono richiedere accessi supplementari al disco. Inoltre è richiesto un i-node aggiuntivo per ogni link simbolico.
- I link simbolici hanno il vantaggio che possono essere utilizzati per collegare facilmente file che risiedono su macchine dislocate ovunque nel mondo.
- I link (simbolici o hard), a causa della presenza di più voci nelle directory, favoriscono l'eccesso di copia durante le operazioni di duplicazione o di backup.

File system basati su log strutturati

- I cambiamenti nella tecnologia spingono l'evoluzione dei file system: le CPU diventano più veloci; i dischi, le memorie e le cache sono sempre più grandi e economiche.
- Il collo di bottiglia diventa così il tempo di ricerca su disco (per quelli meccanici).
- Per questa ragione è stato progettato un nuovo tipo di file system: il **Log-structured File System** (**LFS**).
- L'idea è di soddisfare la maggior parte delle letture direttamente dalla cache del file system (**senza avere accesso al disco**).

File system basati su log strutturati

- Quindi maggior parte degli accessi al disco sarà di scrittura.
- Alcuni sistemi effettuano piccole scritture che sono altamente inefficienti:
  - una scrittura da 50 μs impiega 10ms per il tempo di seek e 5 ms per il ritardo di rotazione.
- I progettisti dell'**LFS** hanno deciso di re-implementare il file system UNIX in modo da raggiungere la **larghezza di banda** del disco.
- L'idea di base è quello di strutturare l'intero disco come un file di log:
  - tutte le scritture (piccole e inefficienti) sono registrate nella memoria e raccolte in un unico segmento che verrà poi appeso al log sul disco.

File system basati su log strutturati

- Un singolo segmento può contenere un miscuglio di i-node, blocchi di directory e dati.
- All'inizio di ogni segmento c'è un segmento di sommario che spiega il contenuto del segmento stesso.
- Se si riesce a mantenere il segmento medio intorno al MB, può essere sfruttata l'intera larghezza di banda del disco.
- Per facilitare la ricerca degli i-node, divenuta non più accessibile con l'«**i-number**», è introdotta una mappa che indicizza gli i-node con i corrispondenti «**i-number**».
- La mappa viene conservato sul disco, ma è anche memorizzato nella cache, in modo da mantenere in memoria gli i-node più utilizzati.

File system basati su log strutturati

- Se il sistema continua a scrivere i segmenti nel file di log, si satura il disco!
- I segmenti possono contenere blocchi che sono cambiati, per esempio per effetto di una modifica del contenuto di un file o di una sua cancellazione.
- Così **LFS** ha un **thread cleaner** che esegue la scansione circolare del log per compattarlo (quando possibile).
- Le misurazioni mostrano che **LFS** sorpassa un classico file system UNIX di un ordine di grandezza su piccole scritture, mentre ha le stesse prestazioni per letture/scritture di grandi dimensioni.

File system basati su journaling

- I file system basati su log strutturato sono un'idea interessante, ma non sono diffusi a causa della loro incompatibilità con i file system esistenti.
- L'idea di base è quella di tenere un diario di ciò che il file system sta per fare prima che lo faccia, in modo che in caso di crash si possa recuperare il lavoro.
- Tali file system sono chiamati **file system journaling** e sono ampiamente utilizzati:
  - **NTFS** di Microsoft.
  - **ext3** di Linux.
  - **ReiserFS**.

File system basati su journaling

- Affinche il journaling funzioni occorre che le operazioni nel log devono essere **idempotenti**: possono essere ripetute tutte le volte che serve senza produrre effetti collaterali.

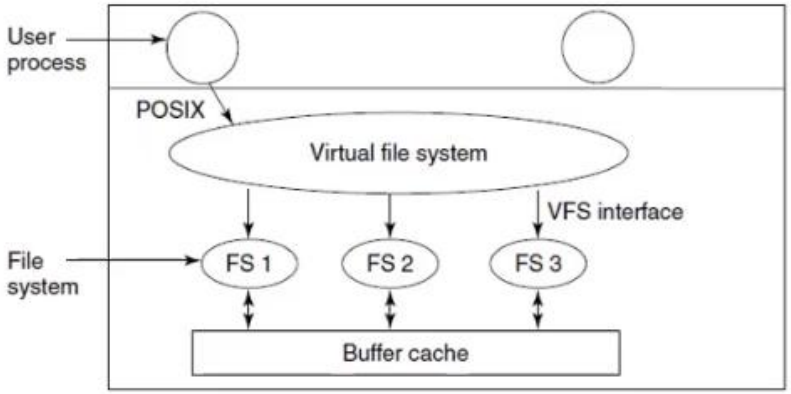
- Per migliorare l'affidabilità, un file system può introdurre il concetto di **transazione atomica**.
  - più operazioni possono essere raggruppate in una entità unica e svolta in modo atomico.

File system virtuali

- I sistemi operativi sono in grado di gestire diversi file system.
- Un sistema **Windows** potrebbe avere un file system principale **NTFS**, ma anche un drive **FAT-32** o **FAT-16**.
- **Windows** gestisce questi file system identificando ciascuno con una differente lettera di unità (a partire da **C:**, **D:**, ...).
- Non c'è alcun tentativo di integrare file system eterogenei in un uno solo unificato.
- Al contrario, tutti i moderni sistemi **UNIX** cercano di integrare più file system in una sola gerarchia sotto **root**.
- Un sistema Linux potrebbe avere **ext2** come il file system di root, con una partizione **ext3** montata su **/usr** e un secondo disco rigido con un file system **ReiserFS** montato su **/home** e così via...

File system virtuali

- I sistemi UNIX usano il **Virtual File System** (**VFS**): la parte comune dei file system è posta su un livello astratto più alto, mentre il codice specifico su un livello più basso .



- Con questo approccio è semplice aggiungere nuovi file system:
  - i progettisti prima analizzano le tipologie di funzioni che il **VFS** può ricevere e poi scrivono il layer sottostante che permette al **VFS** di soddisfarle.
  - se il file system esiste già, occorre realizzare delle funzioni di wrapping verso il **VFS**.

INPUT/OUTPUT

- Tra i compiti del sistema operativo troviamo la gestione dei dispositivi di I/O:
  - Controllare i dispositivi.
  - Inviare comandi ai dispositivi.
  - Catturare gli interrupt dei dispositivi.
  - Gestire gli errori provenienti dai dispositivi.
  - Fornire un'interfaccia semplice e device-independent per la gestione dei dispositivi.

- La parte di software responsabile della gestione dell'I/O rappresenta una frazione significativa dell'intero sistema operativo.

Dispositivi di I/O

- I dispositivi di I/O possono essere suddivisi in due tipologie principali:

1. **Dispositivi a blocchi**: dischi rigidi, CD-ROM, chiavette USB,...
   - Gestiscono informazioni in blocchi di dimensione fissa, ciascuna con un proprio indirizzo.
   - I trasferimenti avvengono con uno o più blocchi (consecutivi).

2. **Dispositivi a caratteri**: stampanti, interfacce di rete, mouse,...
   - Gestiscono flussi di caratteri, a prescindere da qualsiasi struttura.
   - Non sono indirizzabili e non offrono alcuna operazione di ricerca.

Altri dispositivi periferici

- Ci sono altri dispositivi che non sono indirizzabili a blocchi e non generano flussi di caratteri, quindi non rientrano nelle precedenti categorie:
  - I **clock**, producono interruzioni ad intervalli di tempo definiti.
  - La **RAM** della **scheda video**, è una memoria dentro il controller video che rappresenta con una bitmap l'immagine sullo schermo.

Dispositivi di I/O

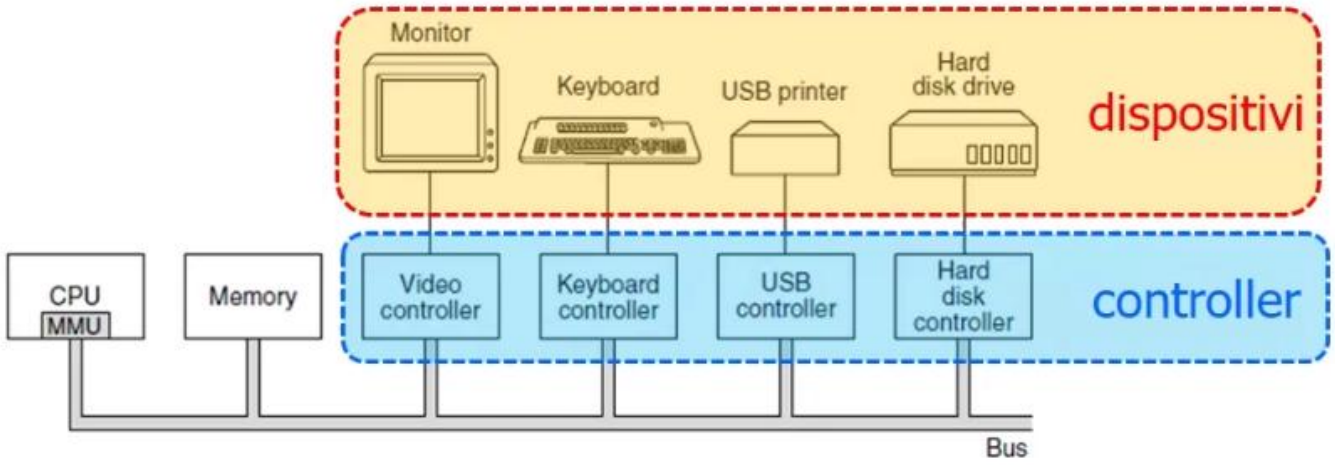
- I dispositivi I/O possono avere velocità molto diverse tra loro, quindi è il software che deve cercare di sfruttare al meglio le potenzialità del dispositivo durante il trasferimento dei dati.

- La maggior parte di questi dispositivi tende a divenire sempre più veloce con il passare del tempo.

Dispositivo	velocità trasf.
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

Controller dei dispositivi di I/O

- I dispositivi di I/O sono in realtà costituiti da una componente meccanica e una elettronica.
- La componente elettronica è chiamata **controller del dispositivo** o **adattatore**.
- Sui PC è un chip sulla scheda madre o una scheda che può essere inserita in uno slot di espansione (es. PCI).
- La componente meccanica è il dispositivo stesso.

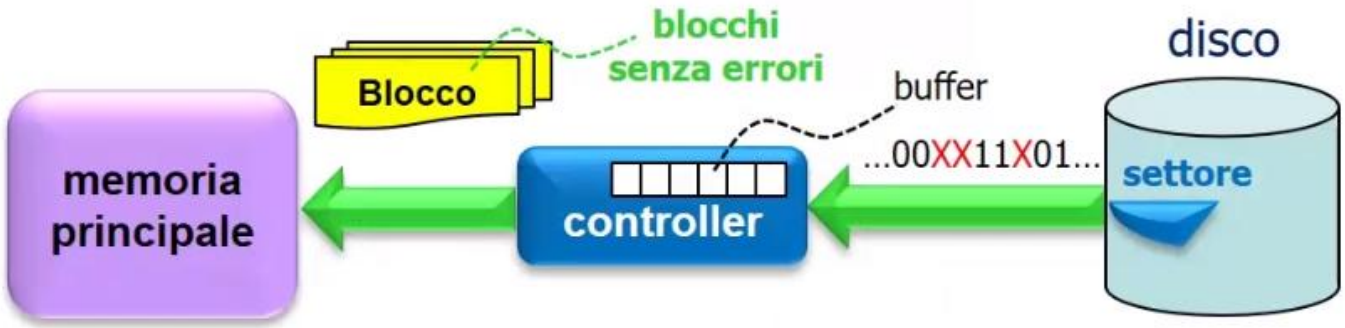


Controller dei dispositivi di I/O

- Molti controller sono in grado di gestire più dispositivi identici.
- Se l'interfaccia tra il controllore e il dispositivo è standard, possono esistere due mercati separati di produttori.

Esempio

- Un disco fornisce un flusso di bit (preambolo, dati del settore e ECC). Il controllore accoda in un buffer lo stream di bit, esegue l'eventuale correzione agli errori e spedisce i blocchi alla memoria principale.



Fasi elementari per la lettura di una parola

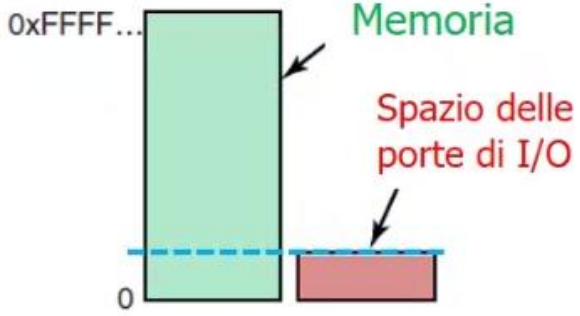
- Quando la CPU vuole leggere una parola (dalla memoria o da una porta di I/O):
  - Pone l'indirizzo sull'address bus.
  - Asserisce il segnale READ sul bus di controllo.
  - Asserisce un segnale per dire da dove si svolge la lettura (I/O o memoria).
  - Attendere la risposta della memoria o dal dispositivo.

Comunicazione CPU←→Controller

- Ogni controllore ha alcuni **registri di controllo** che sono utilizzati per la comunicazione con la CPU.
- Scrivendo questi registri, il sistema operativo comanda al dispositivo di inviare dati, accettarli, accendersi o spegnersi,...
- Leggendo questi registri, il sistema operativo può capire lo stato del dispositivo e se è disposto ad accettare un nuovo comando.
- Oltre ai registri di controllo, molti dispositivi hanno un **buffer dati** che il sistema operativo può leggere e scrivere.
- La CPU comunica con il dispositivo in tre differenti modalità:
  1) Come un vero e proprio dispositivo di I/O (**port-mapped I/O** o **I/O isolato**).
  2) Come parte della memoria (**memory-mapped I/O**).
  3) Entrambi (**hybrid-mapped I/O**).

port-mapped I/O

- Ad ogni registro di controllo viene assegnato un numero di porta di I/O (8 o 16 bit)
- L'insieme di tutte le porte di I/O forma lo **spazio delle porte di I/O** che è protetto: i programmi utente non possono accedervi (tranne il sistema operativo).
- Le istruzioni utilizzate dalla CPU per leggere/scrivere i registri di controllo all'indirizzo PORT sono:



Memory-Mapped I/O

- Il secondo approccio consiste nel mappare tutti i registri di controllo nella memoria (utilizzato per la prima volta da PDP-11)
- A ciascun registro di controllo viene assegnato un indirizzo di memoria univoco (nella parte superiore dello spazio di indirizzo) non più utilizzabile come spazio di memoria principale.
- Questo sistema è chiamato **memory-mapped I/O**.

Svantaggi del port-mapped I/O

- Per leggere e scrivere i registri di controllo del dispositivo, sono necessarie istruzioni dedicate assembly: **IN** e **OUT** (non esistono istruzioni C o C++ che possono realizzarle).
- L'utilizzo di queste istruzioni aggiunge overhead al controllo dell'I/O.
- È necessario un meccanismo di protezione speciale per controllare lo svolgimento delle operazioni di I/O da parte dei processi utente.
- A differenza dell'I/O mappato in memoria che vede i registri di controllo del device come locazioni di memoria (e quindi variabili di un qualsiasi programma C/C++), occorre un passaggio ulteriore per spostare il contenuto dal registro dalla CPU alla memoria.
- I driver di controllo dei dispositivi non possono essere scritti utilizzando esclusivamente il linguaggio C (o C++).
- Il sistema operativo non riesce ad assegnare in modo semplice e dinamico i dispositivi ai processi utente.

Vantaggi del memory-mapped I/O

- I registri di controllo dei dispositivi sono delle variabili in memoria e possono essere gestiti da un programma C/C++.
- Il sistema operativo deve semplicemente evitare di mettere lo spazio di indirizzo contenente i registri di controllo nello spazio di indirizzamento virtuale di qualsiasi utente.
- Ogni dispositivo ha i suoi registri di controllo su una pagina diversa dello spazio di indirizzamento, in questo modo il sistema operativo può assegnare dinamicamente i dispositivi includendo le pagine desiderate.
- È sufficiente una sola istruzione di test in memoria per controllare lo stato del dispositivo.
- I driver di controllo dei possono essere scritti in C/C++.

Vantaggi del port-mapped I/O

- Poiché le informazioni sono mappate su indirizzi separati dalla memoria:
  - la cache non viene influenzata dai dati che transitano verso/dal dispositivo (contrariamente l'effetto sarebbe disastroso).
  - dispositivi non devono esaminare i riferimenti di memoria per capire quando rispondere.

Svantaggi del memory-mapped I/O

- Il caching di un registro di controllo dispositivo produce effetti disastrosi.
  - l'hardware deve avere la capacità di disabilitare la cache sulle pagine di I/O.
- Se c'è un solo spazio di indirizzamento la memoria e tutti i dispositivi di I/O devono esaminare gli indirizzi che passano sull'**address bus** per vedere a quali rispondere.
  - Semplice su un'architettura a singolo bus.

Svantaggi del memory-mapped I/O

- Se esiste un collegamento dedicato CPU-Memoria, i dispositivi di I/O non vedono gli indirizzi di memoria sul bus e quindi non hanno modo di rispondergli.

- Solo i Pentium hanno molti bus (memoria , PCI, SCSI, USB; ISA,...).

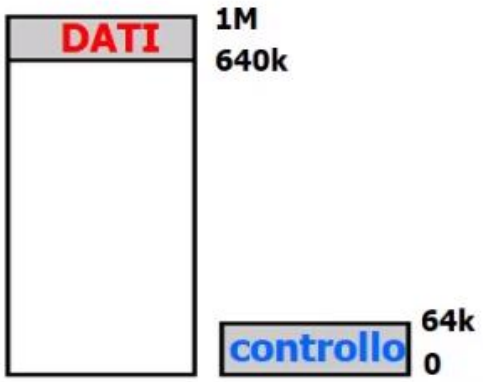
Soluzioni architetture multi-bus

1) La CPU tenta di accedere alla memoria:
   - se le pagine non sono lì, allora prova sugli altri bus (**aumento della complessità hardware ↓**).
2) Si inserisce un filtro:
   - sul bus di memoria che verifica quando l'operazione si riferisce ad un indirizzo di I/O (**i dispositivi potrebbero non lavorare alla velocità della memoria ↓**).
   - all'interno del chip del bridge PCI che verifica quando l'operazione si riferisce ad un indirizzo di I/O (**caricati in fase di avvio ↓**).

Hybrid-Mapped I/O

- Il terzo è uno schema ibrido, con accesso indipendente per:
  - **Buffer dati** (memory-mapped I/O).
  - **Registri di controllo** (port-mapped I/O).

- Il Pentium usa questa architettura, con gli indirizzi da 640K a 1M riservati al buffer dei dati e gli indirizzi da 0 a 64 K per i registri di controllo dei dispositivi di I/O.

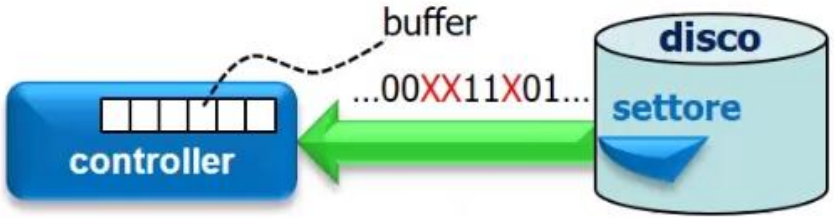


Direct Memory Access (DMA)

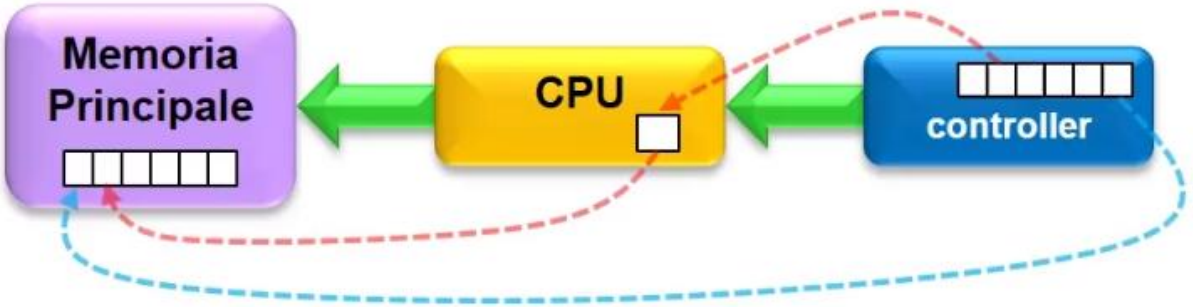
- La CPU accede ai controller dei dispositivi per lo scambio dei dati indipendentemente dal tipo di mappatura per l'I/O.
- La CPU potrebbe richiedere i dati al controller un byte alla volta, sprecando tempo di CPU
- Si usa una diversa tecnica chiamata **DMA** (**Direct Memory Access**) che permette alla CPU di disinteressarsi del trasferimento dati e di svolgere altre attività in parallelo.
- Il sistema operativo può utilizzare il DMA se l'hardware ha il controller DMA.
- I controller possono avere un proprio DMA oppure si può utilizzare un controller DMA (scheda madre) per più dispositivi.
- Il controller DMA è un master del bus, quindi ha accesso al bus di sistema indipendentemente dalla CPU.

Lettura da disco senza DMA

- Il controller del disco legge l'intero blocco (uno o più settori) dal drive in modalità seriale: bit a bit.

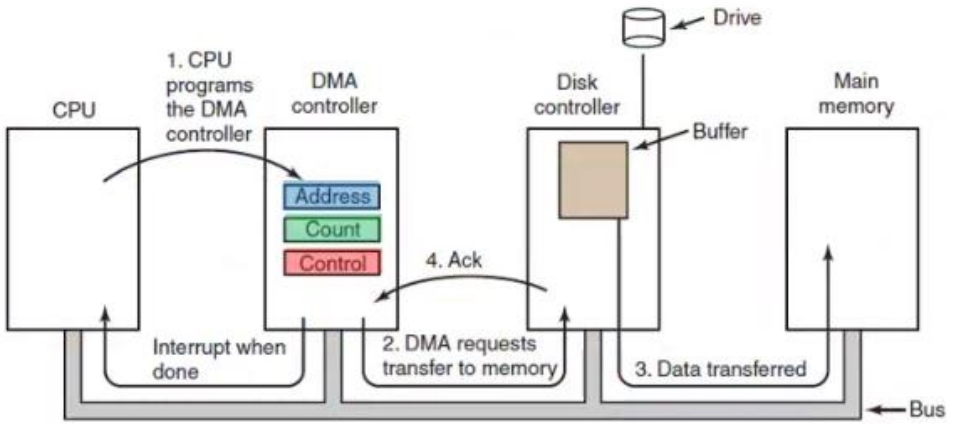


- Quando il blocco è stato caricato tutto nel buffer interno al controllore, si controlla che non si siano verificati errori.
- Il controllore genera un interrupt.
- Il sistema operativo legge con una ISR il blocco dal buffer un byte/parola alla volta e attraverso una MOV lo sposta in memoria.



Lettura da disco con DMA

1. La CPU programma il controller DMA impostando i suoi registri (**Address**, **Count** e **Control**) in modo che sappia **cosa** trasferire e **dove** trasferirlo. Poi dice al controller di leggere i dati nel suo buffer interno.
2. Una volta terminato il caricamento del **buffer**, il controller DMA, preso il controllo del bus come master, chiede al controller del disco la lettura del buffer per il trasferimento in memoria.
3. La scrittura in memoria avviene con un ciclo di bus standard.



Direct Memory Access (DMA)

- Molti bus (e anche controller DMA) possono funzionare in due modalità:

1) **Una parola volta**.

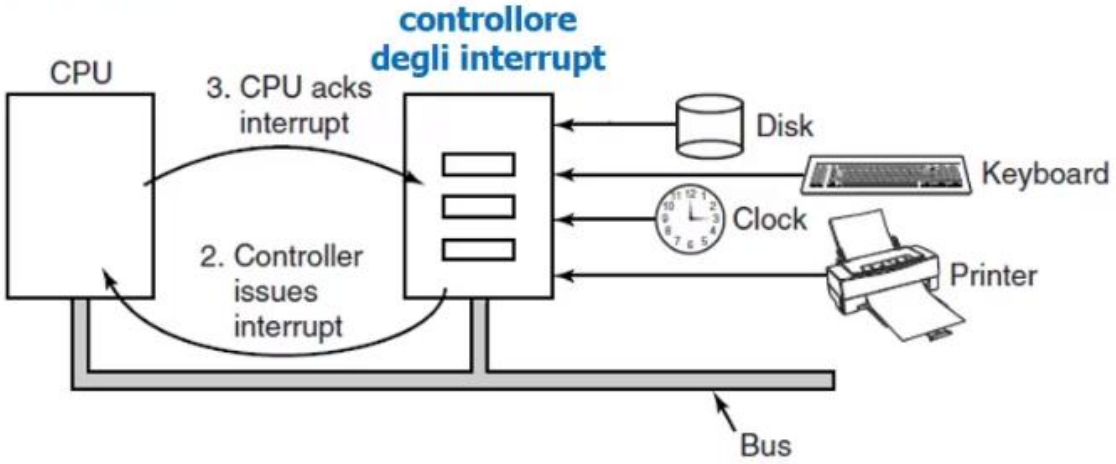
   Il controller DMA richiede il trasferimento di una parola. Se la CPU vuole anch'essa il bus, il contoller DMA deve aspettare. Il meccanismo è chiamato **cycle stealing** perché il controller DMA spia e ruba un ciclo di bus occasionale dalla CPU rallentandola leggermente.

2) **Un blocco alla volta** (**modalità burst**).

   Il controller DMA dice al dispositivo di acquisire il bus, emettere una serie di trasferimenti, quindi rilascia il bus. È più efficiente della precedente per quanto riguarda la durata del trasferimento ma corre il rischio di bloccare la CPU e gli altri dispositivi per un periodo di tempo dipendente dalla dimensione del trasferimento.

Interrupt rivisitati

- Quando un dispositivo di I/O ha terminato il lavoro assegnato provoca un interrupt che invia al **controllore degli interrupt** (o **arbitro**) che decide poi cosa fare:
  - Se nessun altro interrupt è in sospeso, viene elaborata immediatamente l'interruzione.
  - Se è già stato riconosciuto un altro interrupt oppure contestualmente un altro dispositivo ha inviato un altro segnale di richiesta, si decide in base alla priorità assegnata ai dispositivi.



Interrupt rivisitati

- Il controller segnala alla CPU dell'interruzione, attende il riconoscimento e pone un **numero** sulle linee di indirizzo per specificare quale dispositivo ha inviato l'interrupt.
- Il numero è usato come indice in una tabella (**vettore di interruzione**) che restituisce il nuovo valore per il PC: l'inizio della Routine di Servizio dell'Interrupt (**ISR**).
- L'ISR per comunicare al controller degli interrupt che può accettare altre interruzioni, scrive un valore in una delle porte di I/O del controller stesso.

- Prima di avviare la ISR l'hardware deve salvare lo stato della CPU (PC, registri, stack, ...), dove vengono salvate queste informazioni?