2° lezione S.O.

Due Date	@March 11, 2025
Multi-select	
Status	Done

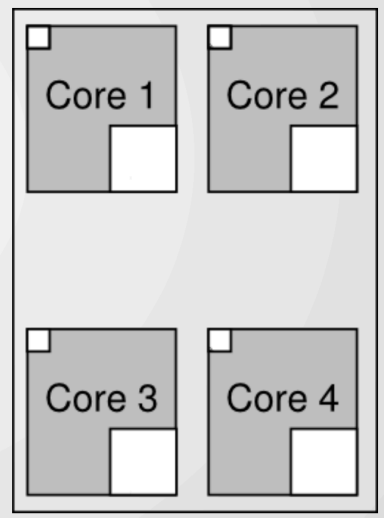
Ripasso Lezione Precedente

Nell'architettura dei sistemi operativi, l'esecuzione del codice in **modalità utente** impone restrizioni sull'accesso diretto alle risorse hardware e alle funzionalità privilegiate del sistema operativo. Per superare questi vincoli, i processi utente ricorrono alle **chiamate di sistema (system call)**, che fungono da interfaccia controllata tra il software applicativo e il kernel.

Il passaggio dalla **modalità utente** alla **modalità kernel** avviene attraverso un'**istruzione privilegiata**, tipicamente un **trap handler**, che trasferisce il controllo al sistema operativo. Questo meccanismo garantisce che l'esecuzione di operazioni sensibili, come la gestione della memoria o l'accesso ai dispositivi hardware, avvenga in un ambiente protetto, riducendo i rischi di instabilità e compromissione del sistema.

Un concetto analogo è rappresentato dagli **interrupt**, che, al pari delle chiamate di sistema, provocano un cambio di contesto e l'esecuzione di codice in modalità kernel. Tuttavia, mentre una **trap** è generata esplicitamente da un processo per invocare servizi del sistema operativo (ad esempio, tramite un'istruzione `int` su x86 o `svc` su ARM), gli **interrupt hardware** sono eventi asincroni, generati da periferiche o dal processore stesso per segnalare condizioni che richiedono l'intervento del sistema operativo, come il completamento di un'operazione I/O o una richiesta di scheduling.

Il supporto al multithreading



L'idea alla base di questa tecnica è l'ottimizzazione dell'efficienza della CPU, riducendo i **tempi morti**, ovvero quei cicli in cui il processore rimane inattivo in attesa di dati dalla memoria . Anche se l'uso della cache può ridurre questi tempi di attesa, la latenza di accesso alla RAM è comunque elevata rispetto alla velocità di esecuzione della CPU.

Per ridurre i tempi morti, il **multithreading simultaneo** prevede l'inserimento di **due set di registri** all'interno della CPU, permettendo di mantenere il contesto di due thread separati. Anche se l'unità di elaborazione rimane unica (quindi non vi è vero parallelismo hardware), la CPU può rapidamente **commutare** tra i due contesti in caso di attesa di risorse.

Ad esempio, se il **thread A** è in attesa di un dato dalla memoria, la CPU può **switchare** immediatamente al **thread B**, utilizzando il secondo set di registri senza dover salvare e ripristinare lo stato in memoria. Questo consente un utilizzo più efficiente della CPU, riducendo i tempi di inattività e migliorando le prestazioni complessive del sistema.

Il software, in particolare il sistema operativo e gli strati di astrazione dell’hardware, è in grado di rilevare se l’esecuzione avviene su una CPU fisica o su una CPU virtuale fornita da un hypervisor. Questa distinzione è fondamentale per garantire un corretto funzionamento della gestione delle risorse e dell’allocazione dei processi. Se si hanno due CPU con la virtualizzazione e si suppone di avere due processi questi non verranno mai eseguiti in parallelo ma si avrà una parvenza di parallelismo. Si ha anche il caso di due CPU nella stessa scheda madre in questo caso si ha il parallelismo vero e proprio.

Che cos’è il troutput

Il **throughput** di un sistema rappresenta la quantità di lavoro completata per unità di tempo ed è una metrica chiave per valutare le prestazioni di un'architettura hardware e software. Se un sistema dispone di una singola CPU, la sua capacità di elaborazione sarà necessariamente limitata dal numero di istruzioni e processi che può eseguire in parallelo.

Quando il carico di lavoro aumenta e il numero di processi da gestire cresce, una singola CPU potrebbe diventare un collo di bottiglia, riducendo il throughput complessivo. Viceversa, con più CPU (o CPU multicore), il sistema può distribuire il carico di lavoro tra più unità di esecuzione, migliorando l'efficienza e aumentando il numero di operazioni completate per unità di tempo.

Il throughput può essere misurato in diverse scale temporali (ad esempio, processi completati al secondo, al minuto o all'ora), a seconda del contesto applicativo. È una misura fondamentale per valutare la produttività di un sistema, sia in ambienti monolitici con una singola CPU sia in architetture multi-threaded e multi-core, dove l'obiettivo è massimizzare l'utilizzo delle risorse disponibili per migliorare le prestazioni complessive.

La principale distinzione tra l'evoluzione dell'hardware e il software che lo utilizza risiede nella progettazione delle CPU e nella loro interazione con i programmi. La miniaturizzazione dei componenti rappresenta una sfida ingegneristica complessa, poiché l'integrazione di numerose funzionalità in un chip sempre più piccolo aumenta la difficoltà di progettazione. Tuttavia, questo processo offre vantaggi significativi: riducendo le dimensioni dei transistor, è possibile integrare un numero maggiore di core all'interno dello stesso chip, migliorando la velocità di comunicazione tra di essi e ottimizzando lo scambio di informazioni.

Un aspetto chiave di questa evoluzione è la riduzione della distanza fisica tra i componenti, che si traduce in una maggiore velocità di trasferimento dei dati e in una riduzione del consumo energetico. Questi benefici risultano particolarmente evidenti in scenari dove l'efficienza energetica è cruciale, come nei data center, dove il controllo della temperatura è una sfida costante, o nei dispositivi mobili, dove l'autonomia della batteria rappresenta un vincolo progettuale primario.

Di conseguenza, i sistemi multicore trovano maggiore diffusione nei dispositivi mobili rispetto ai desktop o ai laptop di fascia bassa, in cui l'obiettivo principale

è aumentare la capacità di calcolo senza incrementare eccessivamente il consumo energetico. Nei sistemi multicore, la gestione dei processi e la comunicazione tra i core avviene in modo trasparente per il software. Sebbene il parallelismo migliorato permetta un aumento dell'efficienza computazionale, esso non sempre corrisponde a un’esecuzione realmente parallela di tutte le operazioni.

Un ulteriore vantaggio del parallelismo nei sistemi multicore è l'aumento dell'affidabilità: in caso di guasto di un core, il sistema può continuare a funzionare utilizzando le risorse rimanenti, garantendo così una maggiore resilienza e continuità operativa.

Se si sfruttassero due macchine?

Si vanno a distribuire i vari processi su delle macchine(grid computing). Ha degli svantaggi come i costi perchè si devono comprare tutti i componenti eccetera. Nel caso si sotra, invece, i componenti sono condivisi tra le due CPU questo ha un impatto positivodal punto di vista dei costi. Tipicamnte si riesce a raggiunge un troputput maggiore a parità di costi, si dev etener conto che le risorse devono essere sufficienti per ospitare i carichi di loro che provengono dalle due o da più CPU. Inoltre si ha un aumento della capità di elaborazione senza dover aggingere altri costi.

L'architettura **multicore** consente di ottimizzare le comunicazioni e lo scambio di informazioni tra i core all'interno dello stesso chip. Operando su una scala più ridotta, la distanza fisica tra i componenti si riduce, migliorando così la velocità di trasferimento dei dati e la sincronizzazione tra i core. Questo incremento dell'efficienza comunicativa si traduce in una maggiore reattività del sistema e in una riduzione della latenza nei processi di elaborazione.

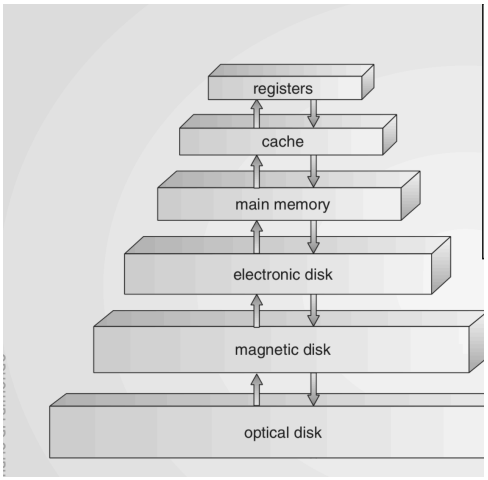
Un ulteriore vantaggio riguarda **l'efficienza energetica**: riducendo le distanze e migliorando la gestione del calcolo parallelo, i sistemi multicore generano meno calore rispetto a un’architettura con singolo core che lavora a frequenze elevate. Ciò consente di abbassare il consumo energetico complessivo, migliorando la dissipazione termica e aumentando l'affidabilità del sistema, soprattutto in ambienti critici come **data center e dispositivi mobili**. IL multicore non influisce dugli algoritmi di scheduling.

Un altro tipo di parallelismo interessante è quello utilizzato nelle GPU (Graphic Processing Units), che sono progettate principalmente per gestire operazioni grafiche, come il rendering. Le GPU moderne sono equipaggiate con centinaia

o migliaia di core piccoli, ognuno in grado di eseguire calcoli paralleli su piccole quantità di dati, come le operazioni di rendering. Questi sistemi di elaborazione parallela sono ottimizzati per applicazioni specifiche come i giochi o l'intelligenza artificiale.

La gestione del parallelismo nelle GPU è generalmente più localizzata all'interno delle applicazioni stesse, piuttosto che nel sistema operativo. Quest'ultimo non gestisce direttamente il parallelismo delle GPU, ma consente di allocare risorse per applicazioni che ne fanno uso, con un focus sull'ottimizzazione delle operazioni per le specifiche necessità dell'applicazione.

La memoria



Per quanto riguarda la memoria, quando si parla di sistemi multicore, è importante comprendere la gerarchia della memoria, che include registri, cache, RAM e altre memorie. I registri sono la memoria più veloce e più limitata, seguita dalle cache, che servono a mitigare la lentezza della RAM. La cache memorizza copie dei dati usati di frequente, riducendo i tempi di accesso ai dati. La RAM è più ampia, ma più lenta, e in alcuni casi, se i dati non sono trovati nella cache, si verificano i cosiddetti "**cache miss**", che obbligano a un accesso più lento alla memoria.

Si ha anche il “cache hit” quando il dato che si sta cercando si trova nella chache.

Il **principio di località** è cruciale in questo contesto: i dati richiesti sono spesso vicini tra loro nella memoria, e questo principio viene sfruttato per caricare

intere linee di cache, non solo i singoli dati, al fine di migliorare l'efficienza complessiva.

La gestione delle cache viene realizzata tramite vari algoritmi di sostituzione delle linee di cache. Ad esempio, si utilizzano algoritmi come LRU (Least Recently Used) per determinare quale linea di cache deve essere sostituita quando è necessario caricare nuovi dati.

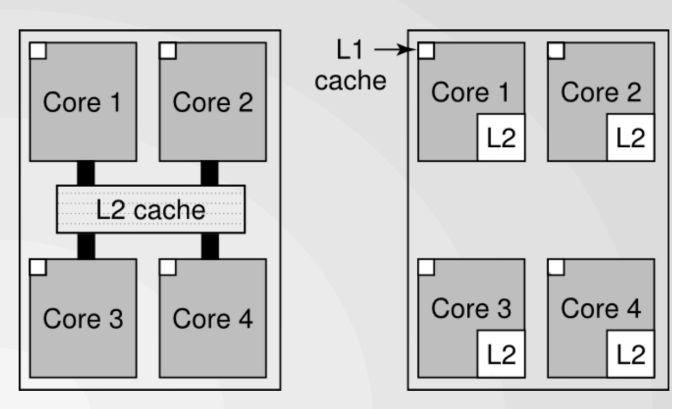
Anche se le tecniche di gestione della cache sono implementate principalmente a livello hardware, vedremo che simili meccanismi vengono applicati a livello software, ad esempio nella gestione dei dischi. In questo caso, il sistema operativo implementa una cache che funge da intermediario tra il disco e la memoria principale, per ottimizzare l'accesso ai dati e ridurre i tempi di latenza.

In generale, quando un'informazione è condivisa tra più utilizzatori, questo può far scattare dei semi-problemi legati alla conformità, poiché si potrebbe perdere buona parte della coerenza dei dati. In particolare, questo è un tema legato alla concorrenza e al parallelismo. Anche se il problema sembra essere legato principalmente all'hardware, le problematiche pratiche si manifestano quando bisogna gestire la coerenza tra diverse cache. Per esempio, un controller di cache può avere un compito molto più complicato rispetto a un altro che gestisce una cache singola per un unico utilizzatore.

Ogni **core** all'interno di una CPU è dotato di una **cache privata**, che consente di ridurre i tempi di accesso ai dati frequentemente utilizzati. Tuttavia, nei sistemi **multicore**, la gerarchia della cache è strutturata su più livelli per bilanciare prestazioni ed efficienza energetica.

Le cache sono organizzate tipicamente in:

- **Cache di primo livello (L1)**: È la più veloce e vicina al core, suddivisa in cache per le istruzioni e cache per i dati. Ha una capacità ridotta ma garantisce tempi di accesso estremamente bassi.
- **Cache di secondo livello (L2)**: Generalmente più ampia della L1, può essere privata per ciascun core o condivisa tra più core. Pur essendo più lenta della L1, è più economica in termini di area e consumo energetico.
- **Cache di terzo livello (L3)**: Di capacità maggiore e generalmente condivisa tra tutti i core della CPU, funge da buffer per ridurre il traffico di accesso alla memoria principale.



La cache può essere indipendente o condivisa

- **indipendente:** ogni core ha la propria cache
- **condivisa:** tutti i core accedono alla stessa cache

In generale condividere ha delle complicazione perché deve essere controllata più complesso da gestire. nel secondo caso si hanno delle complicazioni nascoste ma altrettanto subdole e complicate da gestire. Una di queste è che le varie cache devono essere coerenti tra di loro.

Questa struttura gerarchica ottimizza il **flusso di dati** e minimizza i colli di bottiglia dovuti alla latenza della RAM. Nei sistemi avanzati, alcune CPU possono includere anche una **cache di quarto livello (L4)**, utilizzata per migliorare ulteriormente l'efficienza nei workload intensivi.

Esempio

Supponiamo di avere due core con una cache di secondo livello, supponiamo di avere due processi sui rispettivi core A, B. Si suppone che entrami i proccesi facendo uso di una variabile x condivisa tra i due processi.

Ogni core potrebbe memorizzare una copia di x nella propria cache L2. Se uno dei processi dovesse modificare x, l'altra copia nell'altro core diventa obsoleta generando una possibile incongruenza nei dati

Questo è un problema di coerenza delle cache.

Normalmente, senza le cache, ogni modifica fatta da A verrebbe immediatamente visibile a B, ma quando la stessa informazione è duplicata in più cache, diventa difficile garantire che tutte le copie siano aggiornate correttamente in tempo reale. Questo è il cuore del problema di coerenza delle cache.

Il sistema di cache è un meccanismo di ottimizzazione delle prestazioni, ma diventa un ostacolo quando la coerenza dei dati non è garantita. La comunicazione tra i processi tramite la condivisione delle variabili è fondamentale, e quando la cache non è sincronizzata correttamente, ciò può portare a incoerenze nei dati.

Inoltre, il problema non riguarda solo le cache di secondo livello. Anche le cache di primo livello, che sono più piccole e più veloci, possono contenere copie duplicate della stessa variabile condivisa, generando lo stesso tipo di problema. La differenza sta nel fatto che le cache L2 sono più complesse da gestire, in quanto sono più grandi e distribuite su più core, mentre le cache di primo livello sono più indipendenti e locali.

In ogni caso, la gestione della coerenza delle cache deve essere ben progettata per evitare che l'esistenza delle cache diventi un ostacolo anziché un aiuto. Questo diventa particolarmente importante quando si ha a che fare con sistemi paralleli, dove molteplici processi possono cercare di accedere e modificare le stesse informazioni contemporaneamente.

Le **cache** rappresentano solo una componente del sistema di memoria e, pur rispondendo a molte esigenze di accesso rapido ai dati, non sono sufficienti da sole. Il sistema di memoria è strutturato su più livelli, ciascuno con caratteristiche specifiche in termini di **velocità, costo e persistenza dei dati**.

Ad esempio, la **memoria RAM** è una memoria volatile, il che implica che i dati in essa contenuti vengono persi quando il sistema viene spento. Al contrario, le **memorie di archiviazione secondaria**, come i dischi rigidi (HDD) o le unità a stato solido (SSD), sono **non volatili** e consentono di conservare i dati anche in assenza di alimentazione. Tuttavia, queste ultime, pur garantendo una maggiore capacità di memorizzazione, presentano tempi di accesso significativamente superiori rispetto alla RAM, influenzando le prestazioni complessive del sistema.

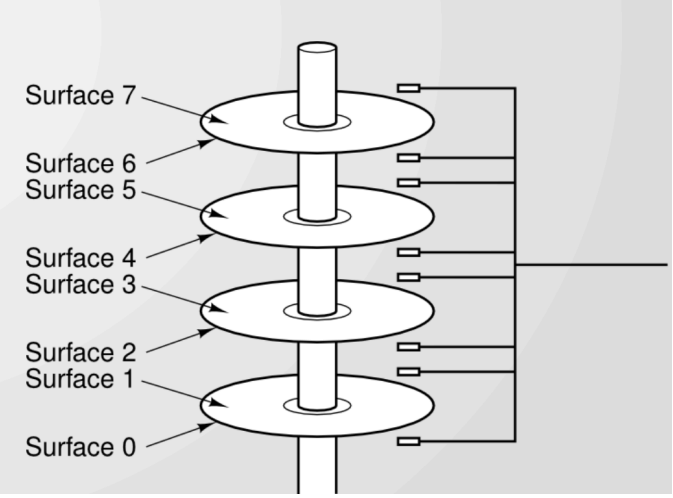
Nel contesto delle tecnologie di memorizzazione, sia i dischi magnetici che le memorie elettroniche perseguono lo stesso obiettivo: **archiviare i dati in modo affidabile e durevole**, garantendone l'integrità nel tempo. Gli **HDD**, basati su un meccanismo elettromeccanico, offrono capacità elevate a un costo ridotto ma con prestazioni inferiori rispetto alle **memorie a stato solido (SSD)**, che, essendo prive di componenti meccaniche, assicurano tempi di accesso e trasferimento dati significativamente più rapidi.

La scelta tra diverse tecnologie di memoria dipende quindi da una serie di compromessi tra **costo, velocità, capacità e persistenza dei dati**, in base alle esigenze specifiche del sistema e dell'applicazione.

Nel caso della **memoria RAM**, i dati devono essere caricati al suo interno prima di poter essere elaborati dalla CPU. Questo processo è essenziale poiché la CPU non può operare direttamente sulla memoria secondaria (HDD o SSD) a causa della sua latenza significativamente più elevata rispetto alla RAM.

Con i dischi, sia meccanici che solidi (SSD), il sistema operativo gestisce questi dispositivi tramite il paradigma del file system. Nonostante le differenze tecnologiche tra un disco meccanico e un SSD, il funzionamento base, ossia la gestione dei dati tramite file, rimane simile.

Un hard disk funziona con una serie di dischi rotanti, ciascuno rivestito da un materiale magnetico, su cui vengono scritti i dati. Le testine di lettura/scrittura si spostano sopra questi dischi per leggere o scrivere informazioni. La superficie dei dischi è suddivisa in **tracce concentriche**, e ogni traccia è a sua volta divisa in **settori numerati.** Le traccie possono essere indentificate dalla distanza che si ha tra il perno e la traccia.



Quando si accede ai dati, il braccio che muove la testina deve spostarsi sulla traccia giusta, attendere che il disco ruoti fino a far passare la sezione corretta sotto la testina, e poi leggere i dati.

Ci sono t**re componenti principali che influenzano il tempo di accesso a un disco meccanico**:

1. **Tempo di posizionamento (seek time):** è il tempo che impiega la testina a spostarsi dalla sua posizione attuale a quella corretta, per raggiungere la traccia giusta. Questo tempo dipende dalla distanza tra la posizione attuale e quella finale.
2. **Tempo di rotazione:** è il tempo necessario affinché il settore contenente i dati si posizioni sotto la testina. Il disco deve ruotare finché il dato richiesto non arriva sotto la testina.
3. **Tempo di trasferimento:** è il tempo necessario per trasferire i dati dal disco alla memoria. Questo tempo è proporzionale alla quantità di dati che devono essere letti.

Quando si leggono più dati, il sistema cerca di ridurre i tempi di accesso, cercando di leggere dati contigui in un unico passaggio. In questo caso, i tempi di posizionamento e di rotazione vengono spesi una sola volta, mentre si trasferiscono più dati contemporaneamente, riducendo i ritardi.

Passando ai dischi a stato solido (SSD), questi sono più veloci dei dischi meccanici perché non hanno parti mobili. Tuttavia, i dischi SSD sono più costosi per capacità di memoria rispetto agli HDD.

Le unità di misura

Le **unità di misura della capacità di memorizzazione** si basano su due diverse convenzioni:

1. **Sistema decimale (potenze di 10)** → Utilizzato principalmente dai produttori di dispositivi di archiviazione:
   - **1 kilobyte (KB) = 1.000 byte (10^3 byte)**

- **1 megabyte (MB) = 1.000.000 byte (10^6 byte)**
- **1 gigabyte (GB) = 1.000.000.000 byte (10^9 byte)**
- **1 terabyte (TB) = 1.000.000.000.000 byte (10^{12} byte)**

2. **Sistema binario (potenze di 2)** → Preferito in ambito tecnico e nei sistemi operativi per garantire maggiore precisione:
   - **1 kibibyte (KiB) = 1.024 byte (2^{10} byte)**
   - **1 mebibyte (MiB) = 1.048.576 byte (2^{20} byte)**
   - **1 gibibyte (GiB) = 1.073.741.824 byte (2^{30} byte)**
   - **1 tebibyte (TiB) = 1.099.511.627.776 byte (2^{40} byte)**

Il funzionamento di un'unità di archiviazione e il modo in cui il software interagisce con l'hardware per gestire il sistema dipendono dal ruolo del **controller del disco**, che è responsabile della gestione delle operazioni di lettura e scrittura dei dati.

Un'unità di archiviazione, sia essa un **disco rigido (HDD)** o un'**unità a stato solido (SSD)**, è dotata di un **connettore di interfaccia** che la collega a un controller dedicato. Questo componente, integrato nella scheda madre o direttamente nel dispositivo di archiviazione, gestisce il flusso di dati tra l'unità di memoria e il resto del sistema.

Il **controller del disco** funziona come una piccola unità di elaborazione autonoma, dotata di memoria interna e buffer per ottimizzare il trasferimento dei dati. Il suo ruolo è cruciale, poiché si occupa di coordinare le operazioni di I/O, riducendo il carico computazionale sulla CPU e migliorando l'efficienza complessiva del sistema.

Perché non lasciare che la CPU gestisca tutto? La risposta è che la CPU è molto potente, ma la gestione del disco richiede una precisione temporale che sarebbe troppo gravosa per la CPU. Inoltre, affidare tutto alla CPU potrebbe rallentare altre operazioni. Il controller, quindi, gestisce in modo indipendente le operazioni sui dischi e può farlo in modo più efficiente. Per esempio, il controller si occupa di attivare la testina e di posizionarla nel punto giusto al momento giusto. Se questo non avviene in modo tempestivo, potremmo avere una lettura errata dei dati. Inoltre, il controller può monitorare se la testina sta andando troppo veloce o se si verifica un disallineamento, e in tal caso agire per correggerlo.

Il **controller** riceve comandi dal sistema operativo attraverso un **bus**, che è una connessione tra i vari componenti del computer. Questi comandi sono abbastanza semplici, come "attiva il motore", "sposta a destra", "sposta a sinistra", e così via. Sebbene i comandi siano elementari, la loro gestione è complicata, soprattutto a causa dei **tempi di esecuzione, che devono essere perfettamente sincronizzati.**

Esistono diversi **tipi di interfacce per comunicare con il disco**, ad esempio quelle seriali e parallele. L'importante è che ci sia uno standard, che consenta a dischi e controller di produrre dispositivi compatibili, indipendentemente dal produttore.

Questi standard riducono i costi e migliorano la compatibilità tra i dispositivi, perché tutti parlano lo stesso "linguaggio". Così, se ogni produttore segue lo stesso standard, il controller di un disco di un produttore può interagire correttamente con un disco di un altro produttore, rendendo il sistema più economico e versatile.

Il **controller** è responsabile della gestione di più dischi, non importa chi li ha prodotti. Quando si richiede una lettura, i dati vengono prelevati dal disco e memorizzati nei buffer del controller. I dati rimangono lì finché non vengono trasferiti alla memoria RAM, dove possono essere elaborati dalla CPU. Questo processo consente al sistema operativo di gestire più dischi e più richieste di lettura senza sovraccaricare la CPU.

Come interagisce il software con il controller?

Il software interagisce con il **controller del disco** tramite un'interfaccia dedicata, che espone al sistema operativo una serie di comandi ad alto livello. Tali comandi, come ad esempio "leggi dalla posizione x del disco", vengono interpretati e tradotti dal controller in azioni fisiche precise, come il posizionamento della testina di lettura o la lettura dei dati dalla superficie del disco.

Il controller del disco funge da intermediario tra il sistema operativo e l'hardware fisico del dispositivo di memorizzazione, permettendo al software di interagire con il disco senza necessità di gestire direttamente le operazioni a basso livello. Questo approccio consente al sistema operativo di inviare comandi di I/O in modo astratto e indipendente dall'architettura specifica del dispositivo, mentre il controller si occupa di eseguire le operazioni necessarie per completare le richieste.

Il software interagisce con il controller attraverso un **driver**, che è un programma che traduce i comandi di alto livello inviati dal sistema operativo in sequenze di comandi più bassi che il controller può comprendere. Ogni controller possiede un proprio linguaggio di comunicazione, che viene gestito dal driver, scritto dal produttore del controller stesso. In questo modo, il driver consente al software di "parlare" con il controller nel linguaggio appropriato, assicurando che le operazioni vengano eseguite correttamente.

Inoltre, il controller utilizza un sistema di coordinate per localizzare i dati nel disco. Tradizionalmente, i dischi rigidi (HDD) utilizzano un sistema di coordinate che comprende tre parametri principali: **cilindro**, **testina** e **settore**. Il cilindro rappresenta la posizione verticale della testina all'interno delle tracce, la testina indica quale superficie del disco viene utilizzata, e il settore rappresenta la divisione fisica del disco in unità di lettura e scrittura. Questi tre elementi formano un sistema di indirizzamento che consente di accedere in modo preciso ai dati immagazzinati sul disco.

Il sistema di coordinate permette di mappare un'ubicazione fisica sul disco a un indirizzo logico, facilitando la gestione delle operazioni di lettura e scrittura. Quando il software invia una richiesta di accesso ai dati, il controller traduce il comando in un'operazione che coinvolge il movimento fisico della testina di lettura/scrittura e l'individuazione del settore corretto, rendendo l'accesso ai dati rapido ed efficiente.

Le tracce più vicine al perno hanno meno settori rispetto alle tracce più esterne, più lunghe e quindi con più settori. Alcuni di questi settori si potrebbero gustare, in questo caso il controller riconosce questo guasto ed evita quel settore, viene sostituito dal punto di vista logico da un altro di riserva.

Il controller del disco ha un proprio linguaggio specifico, che può essere considerato un "dialetto" proprietario. Questo linguaggio è utilizzato per comunicare con il disco attraverso una serie di porte di I/O (Input/Output), che sono essenzialmente punti di accesso per leggere e scrivere dati. Ogni porta corrisponde a un'operazione specifica che il controller esegue in risposta ai comandi ricevuti.

Il software, o meglio il sistema operativo, interagisce con il controller attraverso un driver, che funge da traduttore tra le richieste di alto livello inviate dal sistema operativo e le sequenze di comandi più bassi che il controller può comprendere. Ogni controller ha un linguaggio specifico che è definito dal produttore, e questo linguaggio è solitamente documentato nel manuale del controller stesso. Il driver è responsabile di tradurre le richieste standard, come

"leggi il blocco X dal disco Y", in sequenze di operazioni che il controller può eseguire.

Quale linguaggio usa il controller?

Il linguaggio del controller è un aspetto fondamentale, perché è una sequenza di operazioni molto specifica, che deve essere eseguita in un ordine preciso per garantire che i dati vengano letti o scritti correttamente. Immagina che per interagire con il controller ci siano delle "porte" di I/O, ognuna delle quali richiede un valore da leggere o scrivere. Questo sistema può sembrare complicato, poiché la "sequenza magica" che determina quali porte usare e in quale ordine dipende strettamente dal design del controller.

Il software che interagisce con il controller non deve preoccuparsi di queste sequenze complesse, poiché il driver si occupa della traduzione dei comandi. In effetti, questo è ciò che rende l'interazione con i dischi molto più semplice per il sistema operativo e le applicazioni: possono inviare comandi ad alto livello senza doversi preoccupare di come quei comandi vengono tradotti in azioni fisiche sul disco.

Una complicazione in più è che, mentre l'interfaccia tra il controller e il disco è standardizzata, quella tra il software e il controller è proprietaria e dipende dal produttore del controller stesso. Ciò significa che ogni controller potrebbe avere un "dialetto" diverso che richiede un driver specifico per poter essere utilizzato dal sistema operativo. Per questo motivo, se il sistema operativo vuole accedere a un disco tramite un particolare controller, deve caricare il driver corrispondente, che tradurrà i comandi standard in quelli appropriati per il controller.

In generale, il sistema operativo non conosce i dettagli dei dialetti dei controller, ma può caricare i driver necessari per "parlare" con essi. Il sistema operativo espone un’interfaccia standardizzata che permette alle applicazioni di fare richieste comuni (come leggere o scrivere un file), ma dietro le quinte il driver gestisce la traduzione di questi comandi in azioni fisiche sul controller. Inoltre, i driver sono progettati per essere specifici a ciascun tipo di hardware, e questo significa che se il sistema operativo supporta un particolare disco o controller, deve avere il driver appropriato per interagire con esso.

Infine, le operazioni di lettura e scrittura sui dispositivi di I/O dei controller sono tipicamente eseguite in modalità privilegiata(kernel mode), il che significa che solo il kernel del sistema operativo ha il permesso di accedere direttamente alle

porte del controller. Questo è necessario per garantire la sicurezza e la stabilità del sistema. Tuttavia, esiste un modello alternativo che permette di mappare le porte di I/O del controller su indirizzi di memoria specifici, rendendo possibile a qualsiasi processo, anche in modalità utente, di leggere e scrivere su queste porte senza dover accedere direttamente al kernel. Questo tipo di mappatura è una soluzione utile per delegare l'accesso a determinati dispositivi senza compromettere la sicurezza, ma richiede che il sistema operativo gestisca accuratamente i permessi di accesso.

Sistemi a Microkernel

I **sistemi a microkernel** usano, fondamentalmente, un meccanismo che consente di eseguire un driver in modalità autentica, riducendo così il rischio di distrazioni o malfunzionamenti del sistema. Per ottenere questo, delegano l'uso di alcune porte e controlli a un processo o driver specifico. Una volta che il driver ha bisogno di leggere o scrivere dati su una porta, può farlo in modalità autentica, evitando l'uso di istruzioni privilegiate nel sistema.

Questo approccio implica che **se il programmatore commette un errore**, il **dispositivo potrebbe non funzionare correttamente, ma il sistema nel suo complesso non verrà compromesso** e continuerà a operare normalmente.

Per quanto riguarda la gestione del software, ho detto che, quando un processo richiede la lettura di un file o di una parte di esso, il sistema operativo invia una richiesta al disco. Il disco risponde leggendo uno o più blocchi, che sono necessari per completare l'operazione. Questo processo implica diverse azioni da parte del sistema operativo per ottenere le informazioni necessarie.

Quando un'applicazione richiede un blocco di dati, la sua richiesta deve essere gestita dal sistema operativo. Il processo si blocca in attesa della risposta, perché la lettura da un disco è un'operazione lenta. Questo esempio illustra una "chiamata di sistema lenta", cioè un'operazione che richiede un certo tempo per essere completata, come nel caso di una lettura dal disco. In genere, il sistema operativo deve attendere che l'operazione sia completata prima di riprendere l'esecuzione dell'applicazione.

Una volta che il controller ha letto i dati necessari, il sistema operativo deve gestire il trasferimento di queste informazioni dalla memoria del controller alla memoria principale (RAM). Una tecnica comune per gestire questo trasferimento è quella di **"polling"**, cioè controllare continuamente lo stato del controller finché non diventa disponibile l'informazione richiesta. Questo

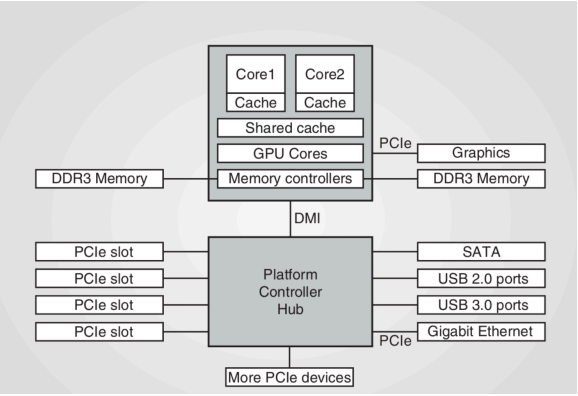
approccio sfrutta i cicli della CPU, ma rappresenta una tecnica inefficiente chiamata **"busy waiting"**, che sprecano risorse della CPU in attesa che un evento si verifichi.

Una tecnica più efficiente per gestire queste operazioni è l'uso degli **interrupt**. Quando il controller ha completato l'operazione, invia un segnale di interrupt alla CPU per notificare che l'operazione è stata completata. In questo modo, la CPU può riprendere l'esecuzione senza dover monitorare continuamente lo stato del controller.

Un'altra tecnica avanzata utilizzata per ottimizzare il trasferimento dei dati è il "**Direct Memory Access**" **(DMA)**. Con DMA, il controller ha accesso diretto alla RAM e può trasferire i dati senza l'intervento della CPU. Questo meccanismo permette di sollevare la CPU dall'onere di trasferire i dati e di ottimizzare il processo, riducendo il carico di lavoro sulla CPU. Sebbene il trasferimento venga gestito direttamente dal controller, un interrupt viene comunque utilizzato per notificare il completamento dell'operazione.

In conclusione, DMA migliora l'efficienza trasferendo i dati direttamente tra il disco e la RAM, riducendo l'uso della CPU. Questo processo può essere utilizzato sia per operazioni di lettura che di scrittura. Quando il sistema operativo desidera scrivere un dato sul disco, può dire al controller, tramite DMA, di prendere i dati dalla RAM e scriverli direttamente sul disco senza l'intervento della CPU.

I BUS



Abbiamo parlato della struttura dei sistemi e ora ci soffermiamo sulla connessione tra CPU e memoria, in particolare sulla presenza di un **bus** dedicato tra la CPU e la cache. Questo meccanismo si occupa di ottimizzare il

flusso di informazioni, influenzato da diversi componenti, tra cui i controller specializzati per la gestione dell'input/output.

Nel tempo, questi controller si sono evoluti, adattandosi alle esigenze sempre più complesse dell'hardware. Oggi molti dispositivi esterni, siano essi rimovibili o installabili sulla scheda madre, utilizzano bus di connessione avanzati. Tra questi, il **PCI Express** ha sostituito il tradizionale PCI, migliorando la velocità e l'efficienza del trasferimento dati tra la CPU e le periferiche, come schede grafiche, schede audio, controller di dischi e altri dispositivi.

La principale innovazione del PCI Express rispetto ai bus precedenti risiede nella sua architettura a connessioni seriali, anziché parallele. Nei vecchi bus paralleli, le informazioni venivano trasmesse su più linee contemporaneamente, con limitazioni dovute alla sincronizzazione dei segnali. Il PCI Express, invece, utilizza una trasmissione seriale ad alta velocità, in cui i dati vengono inviati sequenzialmente, bit dopo bit.

Questa tecnologia permette di assegnare linee dedicate a ciascun dispositivo, migliorando l'efficienza del trasferimento. Inoltre, la velocità è determinata dalla frequenza operativa delle linee di comunicazione, e dispositivi più performanti possono usufruire di più linee per aumentare la larghezza di banda disponibile.

Esistono poi **bus specializzati** per esigenze specifiche, come l'USB, che è nato con l'obiettivo di fornire un'interfaccia standard e universale per dispositivi esterni. L'USB consente il collegamento e la rimozione dinamica di periferiche senza la necessità di riavviare il sistema.

Come un S.O. può adattarsi al tipo di compito a cui è chiamato?

Come un sistema operativo possa adattarsi alle diverse esigenze computazionali. I sistemi operativi sono progettati per gestire una varietà di dispositivi, dai più piccoli ai più potenti, e devono bilanciare diverse priorità in base al tipo di utilizzo.

- **Mainframe**

  nei **mainframe**, sistemi di calcolo di grande capacità, l'obiettivo principale è la gestione efficiente di enormi quantità di dati e richieste simultanee. Questi sistemi sono spesso utilizzati per applicazioni aziendali critiche, database di grandi dimensioni e operazioni batch.

- **Server**

I **server**, invece, pur essendo meno potenti dei mainframe, sono ottimizzati per gestire servizi specifici, come hosting di siti web, gestione della posta elettronica o accesso remoto a risorse aziendali. Un server deve garantire disponibilità e affidabilità, rispondendo nel modo più rapido possibile alle richieste degli utenti.

▼ **Personal Computer**

I personal computer, troviamo sistemi progettati per l'uso individuale, con un focus particolare sull'interattività. L'utente si aspetta un'esperienza fluida e reattiva, in cui applicazioni e interfacce rispondano immediatamente ai comandi. Per garantire questa interattività, i sistemi operativi impiegano tecniche di gestione avanzate, come il time-sharing e la priorità nei processi.

Windows e macOS sono esempi di sistemi operativi ottimizzati per l'esperienza utente sui personal computer. Anche Linux, grazie alla sua flessibilità, può essere adattato a diversi contesti, dai server ai dispositivi desktop, fino ai sistemi embedded.

▼ **Dispositivi Mobili**

i dispositivi mobili, come smartphone e tablet, che pur condividendo molte caratteristiche con i personal computer, hanno esigenze particolari legate all'ottimizzazione della batteria e all'uso di interfacce touch. Sistemi operativi come Android e iOS sono progettati per gestire queste necessità, garantendo un consumo energetico efficiente e un'interazione intuitiva tramite il touchscreen.Con il tempo, la tecnologia continua a evolversi, e le interfacce di comunicazione tra hardware e software diventano sempre più sofisticate, migliorando l'efficienza e l'esperienza d'uso su tutte le piattaforme.

Negli ultimi anni, lo sviluppo di sistemi operativi per dispositivi mobili ha richiesto la progettazione di nuove architetture software. Questa transizione ha rappresentato un'opportunità per modernizzare e specializzare diversi aspetti, tra cui la gestione dei permessi delle applicazioni. Sui dispositivi mobili, gli utenti possono controllare in modo più dettagliato le autorizzazioni concesse alle app, una pratica che, pur non essendo rivoluzionaria, è stata più difficile da implementare nei personal computer per ragioni storiche.

Sebbene anche nei PC esistano modelli di sicurezza e di gestione delle risorse, il livello di controllo è meno capillare rispetto a quello dei sistemi

mobili, dove gli utenti possono stabilire, ad esempio, se un’applicazione può accedere alla rete o a determinate periferiche.

Questa tendenza alla specializzazione si estende anche ad altri dispositivi, come router, smart TV e robot domestici, che rientrano nella categoria dei sistemi embedded. Questi dispositivi, pur essendo a tutti gli effetti dei computer, spesso non permettono agli utenti di installare software liberamente o di modificare il sistema operativo. In genere, il software è preinstallato dal produttore e può essere aggiornato solo tramite firmware specifici, mantenendo quindi un ambiente chiuso e controllato.

Il vantaggio di questa chiusura è che sia il produttore che l'utente beneficiano di un'esperienza più stabile e prevedibile, riducendo il rischio di malfunzionamenti causati da software di terze parti o da configurazioni errate. Tuttavia, questa limitazione comporta una minore flessibilità rispetto ai sistemi aperti, come i PC.

▼ **I sistemi Real-Time**

I sistemi **real-time** rappresentano un caso particolare nell’ambito dei sistemi operativi. Questi sistemi sono progettati per garantire tempi di risposta certi e prevedibili, poiché spesso vengono utilizzati in contesti industriali e critici, dove un ritardo nell’elaborazione può avere conseguenze gravi.

Pensiamo, ad esempio, a una caldaia industriale che lavora ad alta pressione. Il sistema deve monitorare costantemente parametri come temperatura e pressione, intervenendo tempestivamente per aprire valvole di sfogo se i valori superano i limiti di sicurezza. Oppure immaginiamo una catena di montaggio automatizzata, dove i robot devono eseguire azioni precise al momento giusto: se un robot non si attiva in tempo, l’intero processo di produzione può essere compromesso.

La caratteristica distintiva di questi sistemi è la **necessità di reattività garantita**. In un normale sistema operativo, possiamo accettare che, nel 95% dei casi, un’operazione venga eseguita entro un certo limite di tempo, ma nei sistemi real-time questo margine di errore può essere inaccettabile. Ad esempio, se nel 5% dei casi un sistema di controllo industriale fallisce nel rispettare una scadenza, il risultato potrebbe essere un guasto grave o un incidente.

Hard real-time vs Soft real-time

A seconda della criticità delle operazioni, i sistemi real-time si dividono in due categorie:

- **Hard real-time**: le scadenze devono essere rispettate sempre, senza eccezioni. Esempi tipici includono i sistemi di controllo di impianti nucleari, avionica, dispositivi medici salvavita e sistemi di frenata automatica nei veicoli.
- **Soft real-time**: le scadenze possono essere violate occasionalmente senza conseguenze critiche. Un esempio è lo streaming audio: se un piccolo ritardo causa un'interruzione momentanea del suono, l'esperienza utente peggiora, ma non si verifica un danno irreparabile.

Gestione dei processi nei sistemi real-time

A differenza dei normali sistemi operativi, in cui le applicazioni competono per l'uso della CPU, nei sistemi real-time la gestione delle risorse è più rigorosa e prevedibile. Chi progetta questi sistemi identifica in anticipo i processi fondamentali e assegna loro priorità fisse.

Nei PC tradizionali, il sistema operativo gestisce più applicazioni contemporaneamente, assegnando la CPU in modo dinamico in base a vari fattori. Nei sistemi real-time, invece, l'allocazione delle risorse è strettamente controllata:

- Ogni processo ha un tempo massimo garantito entro cui deve essere eseguito.
- La CPU è spesso dedicata interamente a una singola applicazione critica, senza interruzioni impreviste.
- L'hardware e il software sono progettati per evitare ritardi dovuti alla competizione tra processi.

Questo approccio rende la gestione della CPU più semplice rispetto a quella di un normale computer, ma anche più rigida: non si possono installare liberamente nuove applicazioni, e ogni cambiamento nel sistema deve essere attentamente pianificato per garantire che le scadenze vengano sempre rispettate.