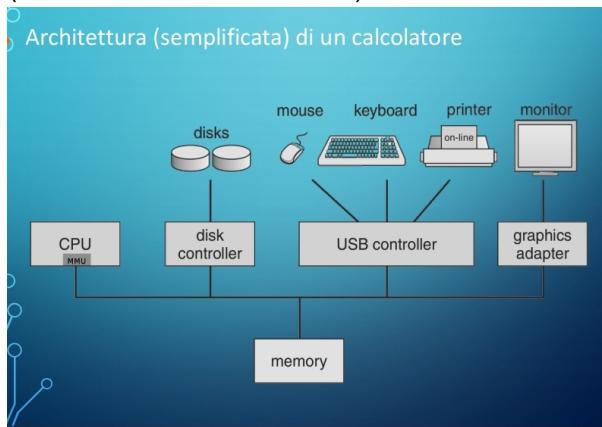


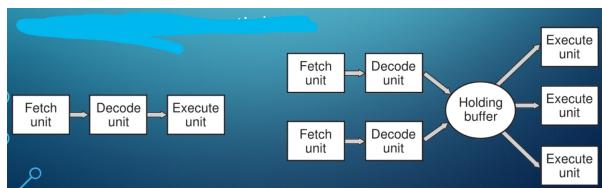
16-03-2023

Il SO deve conoscere esattamente tutte le componenti hardware che comunicano fra loro tramite i **BUS**. (di istruzioni, dati e controllo).



Abbiamo visto il processore e la memoria (parti chiavi per il SO). Le operazioni del processore:

- cerca l'istruzione e la preleva (fetch)
- Analizza l'istruzione
- Esecuzione
- **repeat()**



Nella CPU ci sono diversi **REGISTRI** per le **variabili** o registri per **dati temporanei**.

Il SO deve conoscere tutti i registri e ce ne sono alcuni specifici:

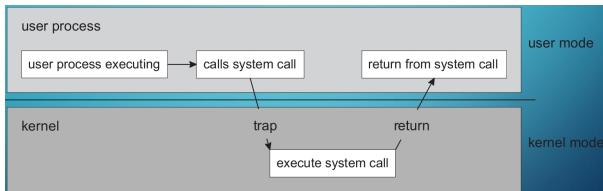
- **Program Counter** (ci sono l'istruzione successiva da eseguire)
- **Stack Pointer** (informazioni riguardo l'attività del SO, variabili utilizzate in quel momento ecc)
- **Program Status Word** (informazioni relative al programma, modalità, priorità -> non tutti i programmi sono eseguiti allo stesso modo). Quando un programma viene fermato, esso deve partire esattamente da dove si è fermato. Con chiamate di sistema I/O

Le CPU moderne permettono di svolgere più istruzioni allo stesso tempo. Mentre inizia un'istruzione e dopo un ciclo si preleva la successiva:

- **pipeline**: accelerano le operazioni. Eseguo il programma n che decodifica e il programma n+1 che preleva l'istruzione. L'unità che elabora le istruzioni è **unica**
- **cpu superscalare**: ci sono unità di esecuzione distinte specializzate. Una gestisce operazioni aritmetiche con interi, un'altra con float ecc... La prima unità che si libera preleva i dati necessari (**coerenti**) dal buffer.

Modalità esecuzione dei programmi

Una piccola parte viene eseguita in modalità **KERNEL** e la restante parte in modalità **UTENTE**. Le **TRAP(chiamate di sistema)** costano e quindi non devono essercene troppe altrimenti creano rallentamento del sistema. Si usa ridurre al minimo quello che c'è al livello kernel.



Thread: programma suddiviso in parti indipendenti e vengono eseguite in maniera **quasi parallela**.

Multithread: più thread sullo stesso programma. Il SO deve conoscere l'esistenza dei thread per gestirli nel miglior modo possibile.

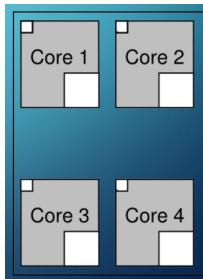
Ogni programma è diviso in **sottoprogrammi** indipendenti. I thread comandano il programma principale.

Esempio: Ogni processo ha 4 di thread ed è come se sta simulando 4 CPU

Multicore: gestione più delicata. Gli obiettivi del SO, in questo caso, massimizzare l'obiettivo (più programmi nel minor tempo possibile) e mantenere il corretto **bilanciamento di carico** per **sfruttare a pieno i vari processori o core**.

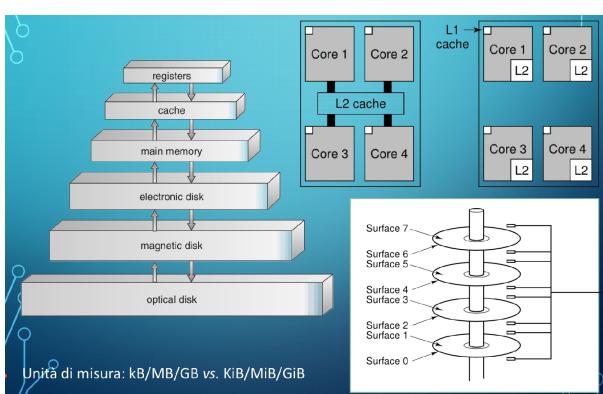
GPU: in parallelo fa diversi calcoli, rendering di parti grafiche.

Il SO lavora in diversi modi in base alle casistiche sopra citate. Usa al meglio i processori e gestisce al meglio i sottoprogrammi in modo da ottenere le massime prestazioni.



Memorie

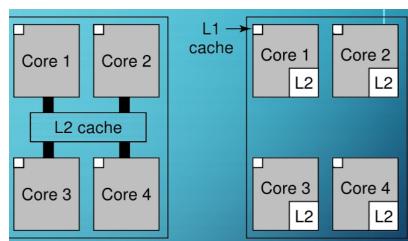
L'utente vede un'**unica memoria** veloce e di **grande quantità** di dati. Ci sono insiemi di memorie a vari **livelli**. Viene rappresentata attraverso l'**uso gerarchico** delle memorie.



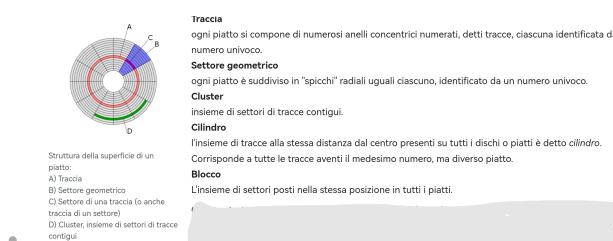
- I **registri** sono memorie piccole, veloci e costose
- La **cache** si trova nell'unità di elaborazione o nei pressi. All'interno di essa vengono memorizzati i dati che vengono usati **più frequentemente**. Se il dato viene trovato in cache si dice **CACHE HIT** e

quindi il dato viene prelevato dalla cache stessa senza analizzare le memorie più lente. Al contrario si ha **CACHE MISS** e il dato viene cercato nelle altre memorie e, allo stesso tempo, memorizzo tale dato nella cache.

- Nella cache di un processore ci sono i dati che sono stati usati per determinati processi. Si cerca di far eseguire i programmi negli stessi processori dove la cache è già popolata con i dati interessati
- La **cache L1** è dentro l'unità di elaborazione e risponde immediatamente
- La **cache L2** può essere di 2 tipi ed è nelle vicinanze dell'unità di elaborazione:
 - condivisa fra tutti i core
 - ogni core ha una sua cache L2
- I processori Intel usano la cache L2 per ogni core
- I processori AMD usano una cache L2 condivisa fra i core



- La **RAM** (che è **volatile**) contiene la **ROM** (che non è volatile) e serve per l'**avvio del calcolatore**
- **Hard Disk:** E' molto grande ed economico ma molto lento
 - I dati sono descritti in **cerchi concentrici** e ad ogni posizione del braccio, la testina legge una regione (**traccia**)
 - Tutte le tracce in una specifica zona formano un **cilindro**.
 - Ogni traccia è divisa in **settori**
 - Muovere il braccio da un cilindro ad un altro richiede circa **1ms** di tempo



- Disco ottico: per prelevare dei dati specifici si deve sempre partire dall'inizio, quindi si ha un **accesso sequenziale**

Memoria virtuale: ogni programma genera dei propri indirizzi virtuali che vengono associati a indirizzi fisici. Se ne possono usare molti di più di quelli che sono effettivamente gli indirizzi reali fisici. Di ciò si occupa **MMU (Memory Management Unit)** ed esegue la **mappatura (astrazione)** di questi indirizzi (**fisico->virtuale**).

Dispositivi I/O

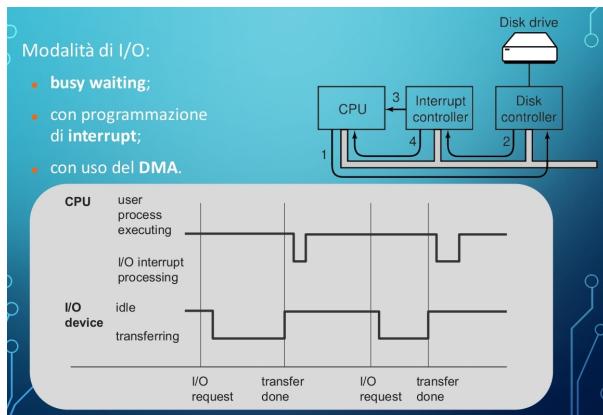
Dialogano spesso con il sistema operativo. Essi sono formati da:

- **controller:** offre un'**interfaccia semplice** per leggere i dati e comunicare con il SO. E' un **chip** (o insiemi di chip) che controlla il dispositivo. **Accetta i comandi richiesti** dal SO. Rende l'interfaccia molto semplice per il SO
- **dispositivo in sè:** ha anch'esso un'interfaccia semplice ma complicata da usare. (Esempio disco SATA: disco magnetico usato su molti PC).

Ogni controlloer e dispositivo è diverso da un'altro e per questo motivo serve un software di comunicazione. Essi si chiamano **DRIVER** e vengono installati nel SO e ne diventano *componenti*.

I sottoprogrammi che vengono eseguiti si distinguono:

- il loro tempo lo passano maggiormente in esecuzione -> (**CPU Bound**)
- il loro tempo lo passano maggiormente per richiedere I/O -> (**I/O Bound**). In questo caso alterno i programmi per liberare le risorse occupate il più velocemente possibile

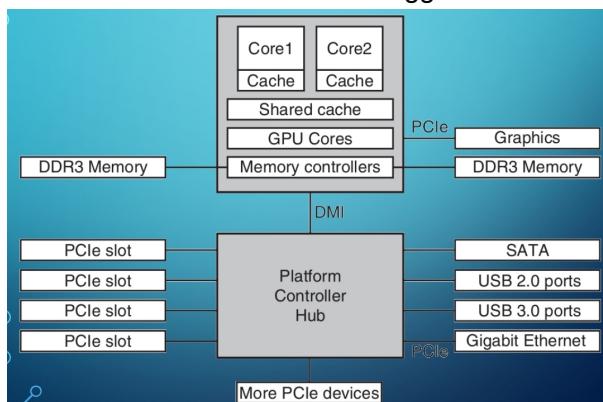


L'attesa può essere gestita in diversi modi:

- l'unità di elaborazione controlla se la risorsa si è liberata o meno ma si ha **busy waiting**. Si occupa **INUTILMENTE** attività dell'unità di elaborazione
- Uso degli **interrupt**: l'unità che controlla gli interrupt comunica con la *CPU* e dice quali dispositivi sono disponibili e quali sono le operazioni da fare. Essi interrompono le attività eseguite in *modalità kernel* perchè le loro azioni sono **determinanti**.
- *Direct Memory Access (DMA)* è un chip che controlla il flusso dei bit fra memoria e alcuni controller **senza l'utilizzo della CPU**. Si ha un trasferimento dal dispositivo direttamente alla memoria. Il *kernel* avviserà mediante *interrupt* che ci sono dei dati messi a disposizione.

Bus

Nei sistemi moderni sono stati aggiunti bus *extra* per le operazioni I/O:



Ogni bus ha una velocità e funzione propria e diversa dagli altri. Il SO deve conoscere l'esistenza e la configurazione di ogni bus.

Il bus principale è il **PCIe (express)** ed è la versione migliorativa rispetto al PCI, cioè vengono trasferiti più dati alla volta.

L'hub mette in comunicazione tutti i dispositivi mediante i bus e mediante il **DMI** (*interfaccia diretta fra dispositivi*).

Lo zoo dei sistemi operativi

In base all'**obiettivo** ci sono Sistemi operativi per:

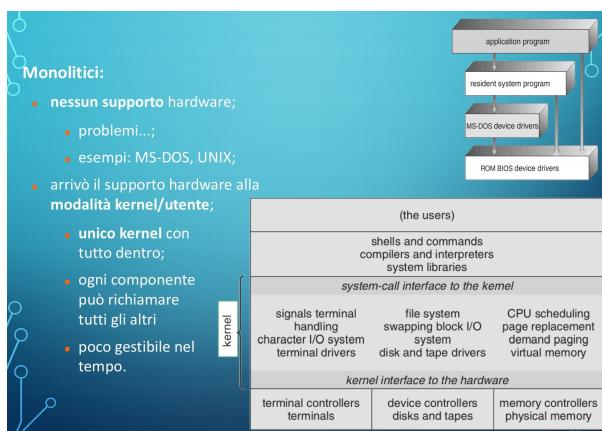
- mainframe/server: gestiscono **grandi quantità di dati** e più dischi (*come UNIX*). Si massimizzano i tempi e usare programmi complicati in un certo (*basso*) intervallo di tempo.
- personal computer: **facilitano l'interazione** con l'utente nel modo più rapido possibile
- palmari/smartphone:
- multiprocessore
- sistemi integrati (embedded): fanno una **specifica attività (ATM)**
- realtime: usati in ambito industriale ed è fondamentale che un'attività avvenga in un **preciso istante** (*catena di montaggio*).
 - **HARD**: avviene in un **preciso istante** altrimenti nulla.
 - **SOFT**: tentano di rispettare i tempi

Strutture per un sistema operativo

Alcune possibili strutture per un SO:

- **Monolitici**
 - A livelli (o a strati)
 - Microkernel
 - A Moduli
 - Macchine virtuali
 - categorie con intersezioni (sistemi ibridi);
 - tassonomia non per forza completa o condivisa

Monolitici



il SO viene eseguito in modalità kernel, visto come una raccolta di procedura che, dopo ogni procedura, ne viene chiamata una nuova. Questa situazione può creare problemi ed errori.

Era diviso in 3 parti:

- programma principale: riporta la procedura di servizio
- insieme di procedure per realizzare le chiamate di sistema
- procedure di supporto a quelle di servizio (prendono dati e richieste degli utenti)

Questo tipo è poco gestibile nel tempo. **MS-DOS** è uno dei primi esempi ed era a linea di comando

A livelli



SO diviso in gerarchia di livelli. L'interno è più importante di quello esterno:

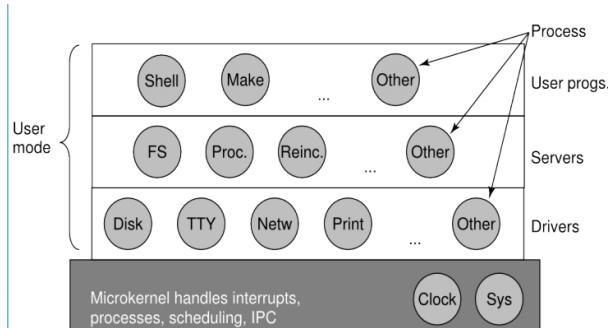
- Il livello centrale è **hardware**
- Al livello superiore -> **attività di esecuzione dei processi**, come si scambiano i programmi, cosa si fa se si termina il tempo assegnato ai programmi, azioni previste in caso di interrupt ecc... Gestione dei programmi e della **CPU**
- al livello successivo -> **gestione della memoria principale**. I programmi non dovevano preoccuparsi della memoria occupata visto che di questo si occupava il livello superiore ad esso.
- al livello successivo ->**gestione della comunicazione fra processi e utente**
- poi: *operazione I/O*
- poi: *organizzazione dei dati*
- L'ultimo livello** contiene l'**interfaccia per interagire con l'utente**

Il passaggio fra livelli **dal livello più alto a quello più basso** viene chiamato **TRAP**, cioè la chiamata di sistema.

In questa struttura ci sono dei problemi:

- I tempi di prestazione non sono eccellenti

Microkernel



Tutti gli strati vengono messi nel kernel e un possibile errore può risultare fatale. Allora si genera un kernel minimale (il più piccolo possibile) per fare poche cose (non affidate ad altre componenti) in modo da ridurre le TRAP e dare maggiore stabilità al sistema.

Il livello più basso, cioè la **modalità kernel**, si occupa di:

- interrupt**
- memoria**
- gestione dei processi (scheduling ->ordine di esecuzione dei processi)**
- gestione della **comunicazione fra processi**

- gestione del **clock**
- sys -> gestore delle chiamate di sistema

Tutte le **altre operazioni** sono in **modalità utente**.

I livelli della modalità utente sono:

1. Drivers:
2. servers: sottoprogrammi che svolgono gran parte dell'attività che deve svolgere il SO. Si verifica anche che tutti i livelli sottostanti stanno funzionando correttamente
3. user progs: rende più stabile il sistema, si riducono le attività rallentamenti dei tempi.

Struttura a moduli



Classica programmazione OOP. Si crea un kernel modulare caricabili dinamicamente e usato da unix, linux, macOS.

Si ha un kernel principale con funzionalità ridotte e altri moduli che si occupa di specifiche attività.

E' una via di mezzo fra microkernel e struttura a moduli.

Si ha:

- l'interfaccia a livello superiore
- mach: gestione della memoria e serve per il supporto alle chiamate remote
- BSD: libreria per la gestione dei thread e così via

Macchine virtuali



L'idea è quella di esagerare con l'astrazione per rappresentare diversi SO su tipologie diverse di macchine. Permette di testare diversi sistemi operativi sul proprio sistema operativo.

Viene lanciata in modalità utente. Deve attivare le richieste per la reale attivazione della modalità kernel.

Lo **HYPERVERISOR** gestisce le virtual machine ed è di tipo:

- 1 -> lavora direttamente sull'hardware dell'host della macchina che lo esegue

- 2 -> le macchine virtuali vengono eseguite in un ambiente del SO convenzionale. E' un processo SO Host ed è più complicato.