

# Sistemi Operativi

Alexej Nardovičk

October 19, 2022

## 1 Contenuti del corso

- **Funzione e struttura** di un Sistema Operativo
- **Sistemi a processi**: proprietà di processi e thread
- Gestione dell'**unità centrale** - Alg.s di scheduling
- Gestione della **memoria centrale**
- Gestione del **file system**
- Gestione della **memoria secondaria** e i dispositivi di I/O
- **Casi di studio** - Unix/Linux. Linee generali di Windows e MacOS.

# Contents

<b>1 Contenuti del corso</b>	<b>1</b>
<b>2 Introduzione</b>	<b>4</b>
2.1 Cos'è un Sistema Operativo? . . . . .	4
<b>3 Storia dei Sistemi Operativi</b>	<b>4</b>
3.1 I generazione – 1945 - 1955 . . . . .	4
3.2 II generazione – 1955 - 1965 . . . . .	5
3.3 III generazione – 1965 - 1980 . . . . .	5
3.4 IV generazione – 1980 - presente . . . . .	8
3.4.1 Cenni di storia di Internet e World Wide Web . . . . .	12
3.5 V generazione – 1990 - presente . . . . .	13
<b>4 Cos'è un sistema operativo</b>	<b>14</b>
<b>5 Sistemi timesharing</b>	<b>15</b>
<b>6 Componenti hardware</b>	<b>15</b>
6.1 componenti hardware - CPU . . . . .	15
6.2 componenti hardware - Scheda madre . . . . .	18
6.3 componenti hardware - Memoria . . . . .	18
6.3.1 memoria volatile . . . . .	19
6.3.2 memoria non volatile . . . . .	20
6.4 componenti hardware – I/O - interrupt . . . . .	22
6.5 componenti hardware – Direct Memory Access (DMA) . . . . .	23
6.6 componenti hardware - bus . . . . .	23
<b>7 Sistemi operativi come base e interfaccia delle app.</b>	<b>25</b>
7.1 Tipi e scopi . . . . .	26
7.1.1 Esempi di tipi . . . . .	27
<b>8 Concetti di Base S.O.</b>	<b>28</b>
8.1 Protezione . . . . .	28
8.2 Interruzioni . . . . .	28
8.3 Avvio . . . . .	29
8.4 Processi . . . . .	30
8.5 File System . . . . .	31

8.6	Plug and Play . . . . .	33
8.7	Cache, buffer, spool . . . . .	33
8.8	Memoria virtuale . . . . .	34
8.9	Chiamate di sistema . . . . .	34
<b>9</b>	<b>Componenti e Obiettivi</b>	<b>36</b>
<b>10</b>	<b>Architetture</b>	<b>36</b>
10.1	Architettura Monolitica . . . . .	36
10.2	Architettura a livelli . . . . .	38
10.3	Architettura Microkernel . . . . .	39
<b>11</b>	<b>Macchina Virtuale</b>	<b>40</b>
<b>12</b>	<b>Sistemi Operativi di Rete e Sistemi Operativi Distribuiti</b>	<b>40</b>
12.1	Sistema Operativo di Rete . . . . .	40
12.2	Sistema Operativo Distribuito . . . . .	41

## 2 Introduzione

### 2.1 Cos'è un Sistema Operativo?

- macchina estesa
- gestore di risorse

## 3 Storia dei Sistemi Operativi

### 3.1 I generazione – 1945 - 1955

La prima generazione degli elaboratori elettronici è caratterizzata dall'utilizzo della tecnologia delle valvole e delle tavole di commutazione (*plug boards*), insieme ad una programmazione avvenuta direttamente in linguaggio macchina. Alcuni antecedenti di queste macchine furono la *Macchina analitica* del matematico Charles Babbage (1837 ca.) e il lavoro di Lady Ada Lovelace, considerata la prima programmatrice della storia.

La programmazione in questa epoca avveniva senza alcun sistema operativo di riferimento, bensì direttamente in linguaggio macchina o cablando manualmente i circuiti del calcolatore.

Gli anni '40 videro la nascita di alcuni dei primi calcolatori *general purpose*, (*i.e.* macchine senza un unico programma di esecuzione ma programmabili per eseguire calcoli generici), tra i quali il *MARK I* della Harvard University costruito da IBM, l'*ENIAC* (*Electronic Numerical Integrator and Calculator*) e successivamente l'*EDVAC*, (*Electronic Discrete Variable Automatic Computer*), la prima macchina basata sulla tecnologia dei transistor che segue l'architettura di von Neumann.

Queste macchine erano molto ingombranti fisicamente, e offrivano una potenza di calcolo minima rispetto a quello che ci offrono oggigiorno i Personal Computer: erano noti infatti i problemi relativi all'affidabilità, alla complessità d'uso e alla lentezza, dovuti tra le altre cose ad una mancata distinzione tra i ruoli di costruzione, progettazione, programmazione e manutenzione delle macchine.

La memorizzazione di dati e programmi avveniva prima attraverso schede perforate, successivamente tramite transistor.

## 3.2 II generazione – 1955 - 1965

A partire da metà anni '50 prese piede lo sviluppo di linguaggi di programmazione ad alto livello, tra i quali ricordiamo sicuramente il linguaggio assembly e Fortran. La memorizzazione di dati e programmi avveniva ancora tramite schede perforate e transistor, e i calcolatori erano caratterizzati dalla cosiddetta **monoprogrammazione**: i programmi utente memorizzati su schede perforate e in attesa di esecuzione venivano caricati ed eseguiti nelle macchine **uno alla volta**. Questi programmi o insiemi di programmi prendono il nome di **job**, e la loro esecuzione avveniva in differita attraverso un operatore che eseguiva dei **comandi batch**.

Questa generazione vede anche la distinzione tra i seguenti ruoli:

- Costruttori
- Programmatori (programmavano in *high level language*)
- Operatori (gli effettivi utilizzatori dei calcolatori; gestivano le schede di programma, le schede JCL (Job Control Language) e l'I/O)

In questi anni cominciano a nascere i primi sistemi operativi per semplificare l'uso dei calcolatori, che funzionavano appunto secondo il concetto di *batch*:

- 1) I programmi e i dati venivano caricati in sequenza su un nastro
- 2) Questi nastri venivano spostati sul computer ed eseguiti a lotti (da qui *batch*)
- 3) L'output veniva stampato su nastro
- 4) Altri computer più piccoli si occupavano della lettura/scrittura da/verso nastro

## 3.3 III generazione – 1965 - 1980

La terza generazione di calcolatori conserva dagli anni precedenti l'utilizzo di sistemi di elaborazione *batch*, tuttavia con una grande novità: quella della **multiprogrammazione**: secondo questo principio, un job può usare il processore, mentre altri job usano le unità periferiche di cui hanno bisogno. Questo portò ad una partizione della memoria, assegnandone le varie parti

Figure 1: Struttura tipica di un job - Fortran

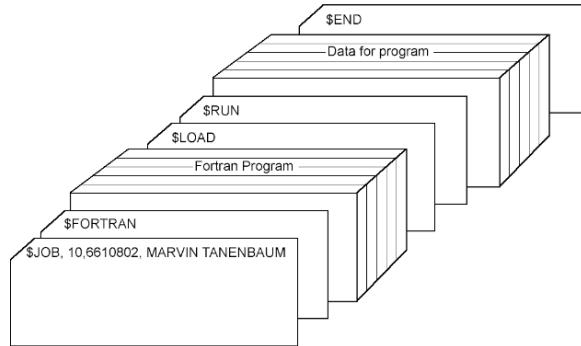
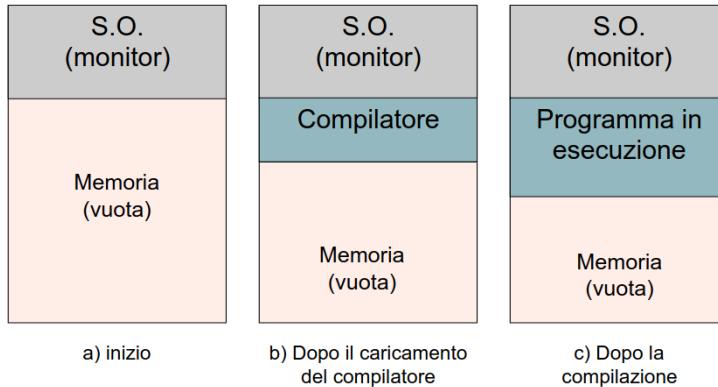


Figure 2: Organizzazione della memoria durante l'esecuzione di un sistema *batch*



a diversi job, in favore di un'esecuzione contemporanea di diversi job e una potenza di calcolo decisamente superiore.

Per gestire questo *pool* di job da eseguire, nacque la necessità di algoritmi di **scheduling** della CPU: algoritmi efficienti grazie ai quali il processore sa come, quando e quanto allocare risorse ai programmi che più ne hanno bisogno in un determinato momento. Vennero quindi progettati i primi **scheduler**, software del sistema operativo adibiti a questo compito.

In questi anni si progettarono nuovi linguaggi di programmazione ad alto livello (*e.g.* C) e si diffuse l'uso di editor testuali e grafici.

Un'altra novità tecnologia di questa generazione è lo sviluppo dei **circuiti integrati**, che permisero una riduzione dei costi ottenendo anche una mag-

gior velocità, e la nascita di sistemi operativi più complessi.

Tra le macchine degne di nota ricordiamo la famiglia di elaboratori *System/360* di produzione IBM e finalizzata all'uso per *mainframe* e i minicalcolatori PDP, caratterizzati dai costi contenuti e dalle prestazioni limitate.

Studiamo più nel dettaglio alcuni concetti della multiprogrammazione:

- **SPOOL** (*Simultaneous Peripheral Operation On Line*): tecnologia che permette l'esecuzione di operazioni concorrenti e l'esecuzione di job parallela alle operazioni I/O e al trasferimento dati.
- **Sistemi timesharing**: variante della multiprogrammazione, sviluppata per supportare molti utenti interattivi simultanei ai terminali. Questa tecnologia prevedeva di dividere il tempo della CPU in **quanti di tempo**, di lunghezza arbitraria, al termine dei quali il job viene interrotto e si assegna la CPU al job successivo (*prelazione*).  
Questo concetto comportava dunque:
  - Cambi di contesto frequenti (**context switch**, che avvengono tuttora nelle architetture moderne)
  - Attenzione alla protezione e sicurezza dei dati e della memoria (un job non può poter accedere ai dati di un altro job)
  - Principio di sviluppo della tecnologia della **memoria virtuale**
  - Tempo di risposta ridotto a minuti/secondi

Per quanto questo concetto possa sembrare estremamente efficiente, bisogna selezionare la dimensione del quanto di tempo con una certa cognizione di causa: un quanto troppo breve causerebbe un overhead troppo grande dovuto alla maggior frequenza di cambi di contesto, mentre un quanto troppo lungo rallenterebbe job di minore entità in favore di job più complessi.

Alcuni esempi di tecnologie di timesharing introdotte in questo periodo furono il sistema **CTSS** (*Compatible Time Sharing System*) da parte dell'MIT, evoluto poi nel **MULTICS** (*Multiplexed Information and Computer Service*), in cui venne introdotto il concetto di **processo**. Questi sistemi, insieme al

**TSS** (*Time Sharing System*, su cui si basa *Unix*, di cui parleremo in seguito) e al **CP/CMS** (*Control Program/Cambridge Monitor System*) includono il concetto di **memoria virtuale**, per dare ai processi l'illusione di avere maggior memoria di quella fisicamente disponibile.

Nacquero anche i sistemi **real-time**, atti a fornire una risposta entro un dato periodo di tempo limitato.

Abbiamo già menzionato *Unix*, sistema operativo derivato da CTSS e MULTICS che implementa le tecnologie di timesharing multiprogrammati che abbiamo visto finora. Nel 1974 *Unix* fece registrare una doppia licenza, commerciale e libera, e venne pubblicato il suo codice sorgente: per rendere compatibili le diverse versioni che si vennero a creare di questo sistema operativo, fu introdotto lo standard *POSIX*, appartenente allo standard *IEEE*, dal momento che disponendo del codice sorgente ognuno aveva la possibilità di creare una nuova versione del sistema operativo. Questa possibilità fu anche favorita dallo sviluppo dei primi semplici Personal Computer, costituiti da una tecnologia a microprocessori.

Infine, in questa generazione si iniziò a sviluppare anche la tecnologia Internet e i protocolli TCP/IP da parte del Dipartimento della Difesa americano, usati in ambienti militari e accademici per una massiccia e veloce trasmissione dei crescenti volumi di dati.

### 3.4 IV generazione – 1980 - presente

A partire dagli anni '80 crebbe fortemente lo sviluppo delle **workstation** e dei **personal computer**, il cui utilizzo venne fortemente semplificato grazie all'introduzione delle **interfacce grafiche (GUI)**. Il trasferimento di informazioni tra computer tramite **reti** divenne molto più pratico ed economico, e l'elaborazione dei dati diventò man mano sempre più **distribuita** ai siti in cui veniva richiesta.

Alcuni esempi di sistemi operativi introdotti in questa generazione furono:

- *CP-M80 Digital*, divenuto poi *MS-DOS*
- *LisaOS* (1983), primo SO per PC con GUI, seguito da *MacOS* (1984) e *MacOSX* (1999)
- *Windows* (1985), *Windows 3* (1990) che adottò la tecnologia della

**memoria virtuale**, *Windows 95* (1995), *Windows NT* (1998), *Windows XP* e *Windows Millennium Edition* (2001), *Windows 7* (2017), *Windows 8* (2012) per sistemi touch, *Windows 10* (2015).

Tutti i sistemi operativi di *Microsoft* accennati sopra sono stati inizialmente pensati per girare su PC, ma vennero poi anche adattati per le *workstation*.

- *Linux*, *Unix* - dotati di interfaccia *Xwindows* (basato su *X11* del MIT), poi *BSD*, *Xenix*, *SunOS*, *Solaris*, *FreeBSD* (da cui derivò anche *Mac OSX*)

Nacque in questa generazione la distinzione tra **cliente** e **servente**, concetti chiave su cui si svilupperà poi il relativo modello di elaborazione:

- I **clienti** richiedono diversi servizi ai serventi
- I **server** eseguono le richieste di servizio

Subì una forte evoluzione anche l'area dell'ingegneria del software, promossa anche dal governo degli Stati Uniti per un controllo rigoroso dei progetti software del Dipartimento della Difesa, realizzando alcuni traguardi della storia dell'informatica quali i concetti di **riusabilità del codice**, un maggior grado di **astrazione nei linguaggi di programmazione** e il **multithread** di istruzioni da poter eseguire in modo indipendente.

Il decennio successivo vide la nascita delle applicazioni e delle **reti di calcolatori**, tra cui ricordiamo il *World Wide Web* e *Microsoft Office*. In questo periodo sistemi operativi e calcolatori erano già molto simili a quelli che usiamo al giorno d'oggi, infatti è a partire dagli anni '90 che si fondarono le basi dei dispositivi che usiamo tuttora: PC, smartphone e tablet (con relativi sistemi operativi), sistemi distribuiti, cloud computing...

Tutte queste innovazioni furono favorite dall'importante **decrescita del costo** della capacità di elaborazione e della memoria, che rese possibile l'esecuzione di programmi grandi e complessi su PC e ampliò la disponibilità di sistemi economici per la memorizzazione di grandi database e l'elaborazione di job, riducendo quindi la necessità di mainframe centralizzati: diretta conseguenza di ciò fu lo sviluppo di metodi e tecniche per l'elaborazione di calcolo distribuito, tramite appunto i cosiddetti **sistemi distribuiti**, cioè sistemi di elaborazione indipendenti che cooperano per raggiungere un obiettivo comune.

- **Sistemi paralleli**

- Un sistema con un ampio insieme di unità di elaborazione
- Accoppiamento stretto, comunicazione rapida
- Risorse condivise (*e.g.*, la memoria)
- Alta affidabilità e prestazioni

- **Sistemi distribuiti**

- Sistema costituito da un insieme di unità di elaborazione complete interagenti e cooperanti, collegate da linee di comunicazione
- Omogeneo o eterogenei
- Architetture
- Condivisione di risorse, prestazioni, affidabilità e trasparenza

Sempre in questo decennio vennero creati degli **standard** di sistemi operativi che supportano **networking tasks**, per l'aumento della **produttività e comunicazione**, ed è in questo periodo che la **Microsoft Corporation** divenne dominante, grazie all'introduzione nei sistemi operativi *Windows* di tecnologie presenti nei primi Macintosh che permettevano agli utenti di navigare più applicazioni concorrenti con una certa facilità.

Importante fu anche lo sviluppo della **tecnologia a oggetti**:

- Molte applicazioni si scrivevano in **linguaggi di programmazione orientati agli oggetti** (*e.g.*, C++ o Java)
- **Sistemi operativi orientati agli oggetti**, oggetti appunto che rappresentano componenti del SO
- Concetti come **eredità** e interfacce sfruttati per creare sistemi operativi **modulari** e che permettevano di mantenere ed estendere il software rispetto alle tecniche precedenti

Va adesso fatta una digressione storica sulla questione della filosofia *open-source*: negli anni '90 la maggior parte dei software commerciali era venduta sotto forma di **codice oggetto** già compilato, non includeva il codice sorgente vero e proprio; questo serviva ai produttori di sistemi operativi per

nascondere le tecniche di programmazione e informazione proprietaria. In questo decennio si diffuse sempre di più l'ideologia del software libero, ***open-source*** appunto, distribuito insieme al codice sorgente, che ne permette l'esame e la modifica. Alcuni software estremamente famosi utilizzati tutt'ora, come **Linux** e **Apache Web Server**, sono *open-source*.

Una pietra miliare della storia dell'*open-source* fu la nascita del progetto ***GNU***, a cura di Richard Stallman, che consisteva in un progetto di **software libero** che si poneva l'obiettivo di ricreare ed estendere gli strumenti per il sistema operativo **Unix** di *AT&T*, schierandosi contro il concetto di costo per l'uso di software.

Sulla scia di Richard Stallman nacquero altre iniziative a favore della promozione dell'*open-source*, tra le quali ricordiamo la ***Open Source Initiative, OSI***:

- Organizzazione nata per promuovere i benefici della programmazione *open-source*
- Facilita il rapido miglioramento dei prodotti software, permettendo a chiunque di **testare**, eseguire il **debug** e **migliorare le prestazioni**
- Aumenta la probabilità di riconoscere e risolvere i *bugs* per risolvere anche problemi relativi alla sicurezza dei programmi
- Gli individui e le aziende possono modificare il codice sorgente per creare software personalizzato secondo l'esigenza del relativo ambiente (**adattabilità**)

Possiamo concludere questa rassegna sulla IV generazione dei sistemi operativi ribadendo alcuni concetti chiave:

- I sistemi operativi diventano sempre più di facile uso (*user friendly*)
- Le caratteristiche delle ***GUI*** introdotte da *Apple* e adottate poi da tutte le altre grandi aziende sono sempre più diffuse e migliorate
- Le funzionalità **"Plug-and-play"** sono integrate nei sistemi operativi: permettono agli utenti di aggiungere e rimuovere dinamicamente componenti hardware, senza il bisogno di riconfigurare manualmente il sistema operativo
- I **servizi web** cominciano la loro ascesa verso l'utilizzo commerciale:

- Comprendono un insieme di **standard** correlati
- Componenti software ready-to-use su Internet
- Permette a due qualsiasi applicazioni di comunicare e scambiare dati
- Fanno largo uso di **middleware**: un insieme di software che collegano applicazioni separate e diverse (spesso tramite rete e anche fra macchine eterogenee, dotate di diverse architetture)

### 3.4.1 Cenni di storia di Internet e World Wide Web

Prima di affrontare l'ultima generazione della storia dei sistemi operativi, è importante soffermarci su tempi e modalità che alla fine del II millennio portarono alla nascita del sistema di reti del *World Wide Web*.

Come abbiamo già accennato, gran parte dello sviluppo di Internet fu promosso dai governi, specialmente quello degli Stati Uniti d'America: alla fine degli anni '60, il Dipartimento della Difesa americano creò e implementò **ARPA***net*, una sorta di predecessore di Internet, una rete progettata per trasmettere dati tra i principali sistemi delle istituzioni finanziarie da ARPA **senza controllo centralizzato**. Questo antecedente di Internet era già allora capace di una trasmissione di informazioni quasi istantanea tramite e-mail.

Successivo fu poi lo sviluppo del **Transmission Control Protocol/Internet Protocol (TCP/IP)**, cioè di un insieme di regole per comunicare su *ARPA**net*. Questi protocolli **gestiscono la comunicazione fra applicazioni**, assicurandosi che i messaggi siano **instradati** correttamente dal mittente al destinatario, limitando gli errori tramite **tecniche di correzione dell'errore**. Gli stessi protocolli vennero poi riutilizzati quando la rete venne aperta anche all'uso commerciale più generale.

Nel 1989, nel suo studio al CERN di Ginevra, Tim Berners-Lee progettò quella che diventerà poi la rete su cui, attualmente, verranno scambiate tutte le informazioni tramite Internet: il **World Wide Web**.

Venne ideato come rete per **individuare e visualizzare documenti multimediali** su argomenti qualsiasi, ma soprattutto per la **condivisione di informazioni** attraverso documenti di testo con **collegamenti ipertestuali**.

ali. I documenti presenti in questa rete sono definiti secondo lo standard del linguaggio **HTML** (**HyperText Markup Language**), e il protocollo (cioè le regole di comunicazione) è definito dall'**HTTP** (**HyperText Transfer Protocol**).

### 3.5 V generazione – 1990 - presente

La caratteristica principale degli elaboratori moderni è sicuramente la **mobilità**: dagli anni '90 infatti lo sviluppo della tecnologia per sistemi mobili crebbe a ritmi esponenziali, partendo dai primi palmari quali il *Nokia N9000* per arrivare ai moderni *smartphone* (termine coniato nel 1997) dotati di sistemi operativi complessi, quali *Android*, basato su Linux e di proprietà di Google, e *iOS*, di proprietà di Apple.

Ciò che contraddistingue i dispositivi mobili moderni è una serie di fattori:

- Risorse limitate
- Alimentazione a batteria
- Diversi dispositivi di I/O
- Uso intensivo di applicazioni, reti e protocolli (*cloud computing*)
- Integrazione in altri oggetti
- Attenzione alla sicurezza

Tutti i moderni calcolatori possono interagire tra di loro attraverso l'**Internet of Things (IoT)**, grazie all'evoluzione della rete sulla quale possono scambiare informazioni oggetti statici e mobili (e.g., elettrodomestici, abbigliamento, impianti, macchine, attrezzature...).

Alla base di questa tecnologia si trova il concetto di *ubiquitous computing*, in italiano *computazione ubiqua*, cioè un modello di interazione uomo-macchina secondo il quale l'elaborazione delle informazioni è integrata all'interno di oggetti e attività di tutti i giorni (gli oggetti citati sopra). Per permettere queste implementazioni bisogna chiaramente progettare e sviluppare un sistema operativo anche per questi oggetti, e anche per essi, come per un tradizionale calcolatore, bisogna preoccuparsi di garantirne sicurezza e privacy.

L'informazione può essere scambiata anche attraverso i cosiddetti *smart objects* (cioè tecnologie *RFID*, codici *QR*...).

## 4 Cos'è un sistema operativo

Un S.O. è un software che controlla l'hardware.

Un S.O. è un programma che gestisce e controlla l'esecuzione di un insieme di applicazioni, agisce come interfaccia tra le applicazioni e l'hardware del calcolatore e gestisce le risorse hardware.

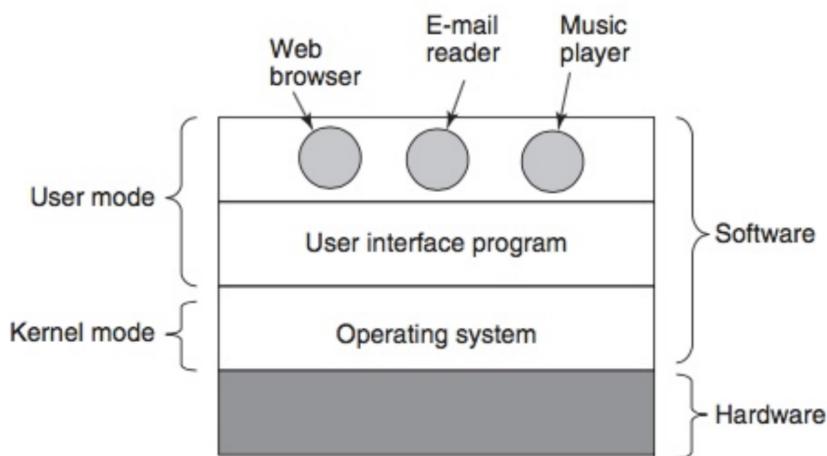
Il S.O. è (prevalentemente) eseguito con il processore in modalità kernel. Le applicazioni sono eseguite principalmente in modalità utente. Un sistema operativo ha applicazioni separate dall'hardware utilizzato:

- livello software
- gestione software e hardware per produrre i risultati desiderati

Un Sistema operativo innanzitutto è un gestore di risorse:

- Risorse hardware;
  - processore
  - periferiche I/O
  - memoria
  - periferiche di comunicazione
  - ecc.
- Applicazioni software

componenti di un sistema operativo:



## 5 Sistemi timesharing

- Variante della multiprogrammazione, fu sviluppato per supportare più utenti interattivi nello stesso momento sul terminale
- Il tempo di CPU è suddiviso in **quanti di tempo**
- Al termine del quanto il job viene interrotto e si assegna la CPU al job successivo (prelazione)
- Il tempo di risposta viene ridotto a minuti o secondi

Progettazione dei sistemi timesharing:

- Gestione del processore: scheduling
- Gestione della memoria: memoria virtuale
- Protezione delle risorse (memoria, filesystem, ...)

## 6 Componenti hardware

Per progettare un sistema operativo devo conoscere le risorse hardware e software che deve gestire:

- processori
- memoria interna / memoria secondaria
- altre periferiche I/O
- processi
- thread
- file
- database

### 6.1 componenti hardware - CPU

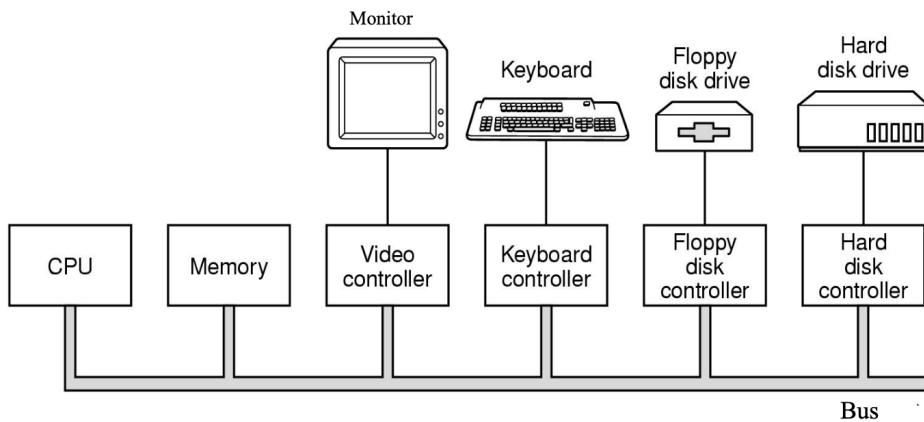
Il processore è quella parte dell'hardware che **esegue** in linguaggio macchina, la CPU esegue le **istruzioni** di un programma.

All'interno della CPU troviamo i **registri**, ovvero delle memorie ad alta velocità situati sui processori. I dati devono essere nei registri prima che un processore possa operarvi.

Il **tempo di elaborazione si misura in cicli**, fornito dal generatore di clock di sistema.

La velocità del processore è misurata in **GHz** (miliardi di cicli per secondo).

## Componenti Hardware



Componenti di un semplice personal computer

## Evoluzione dei microprocessori con architettura Intel

Microprocessore	Bit	Anno	Transistor	Produttore
4004	4	1971	2,25 K	Intel
8080	8	1974	5 K	Intel
Z80	8/16	1976	6 K	Zilog
8088	8/16	1979	29 K	Intel
80286	16	1982	134 K	Intel,Amd
80386	32	1985	275 K	Intel,Amd
80486	32/64	1989	1,2 M	Intel,Amd
Pentium	32/64	1993	3,1 M	Intel
Pentium III	32/64	1999	9,5 M	Intel
Athlon	32/64	1999	22 M	Amd
Pentium IV	32/64/128	2000	40 M	Intel
Opteron	32/64/128	2003	100 M	Amd
Itanium 2	32/64/128	2004	220M	Intel
Opteron quad core	32/64/128	2006	460 M	Amd
Core 2 Quad	32/64/128	2006	582 M	Intel
Xeon MP 6 core	32/64/128	2008	1.900 M	Intel
Itanium quad core	32/64/128	2008	2.000 M	Intel

La **CPU** si trova in diverse modalità, in mod **nucleo (kernel)** o **utente**. Quando viene chiamata una TRAP (istruzione) viene cambiata la modalità da utente a kernel.

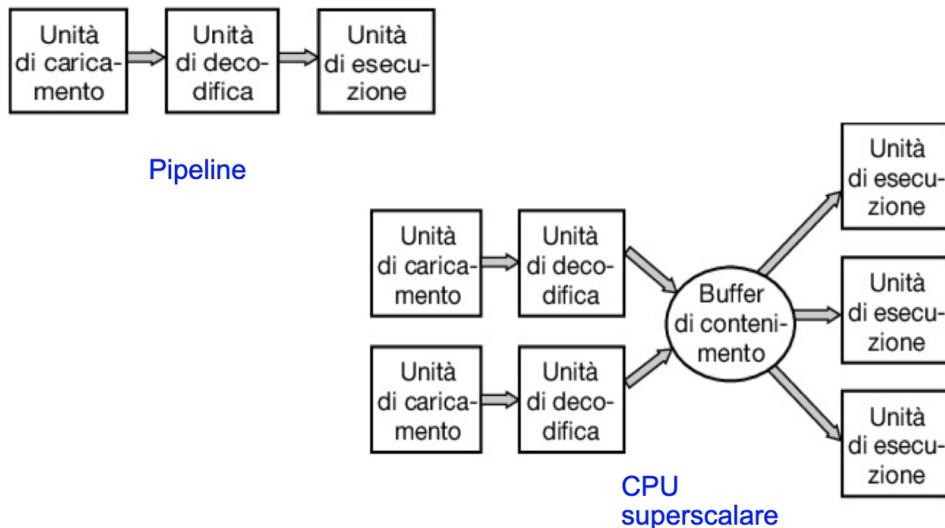
#### Prestazioni di una CPU

$$T = N_i / IPS \quad IPS = F \times IPC = F / CPI$$

T	tempo di esecuzione
N <sub>i</sub>	numero di istruzioni di un programma
IPS	numero di istruzioni per secondo
F	frequenza di clock del processore
IPC	numero di istruzioni per ciclo di clock
CPI	cicli di clock per istruzione

T può migliorare con

- **aumento di F** → miniaturizzazione
- **riduzione N<sub>i</sub>**, per lo stesso lavoro → CISC / RISC  
(Complex/Reduced Instr. Set)
- **aumento di IPC** ovvero **riduzione CPI** → **pipeline**  
→ CPU superscalare

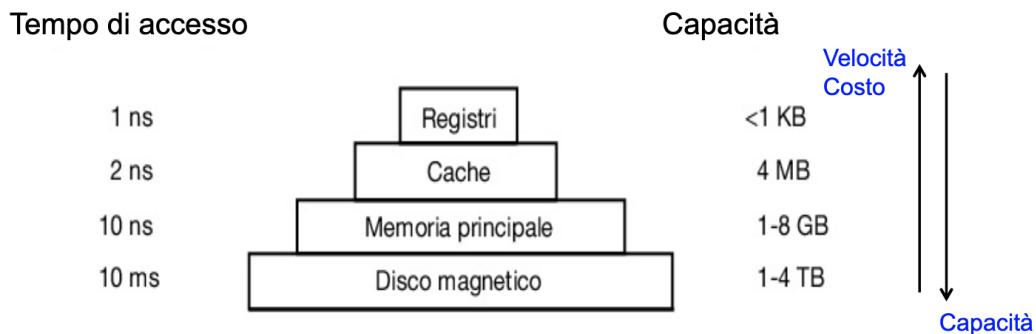


## 6.2 componenti hardware - Scheda madre

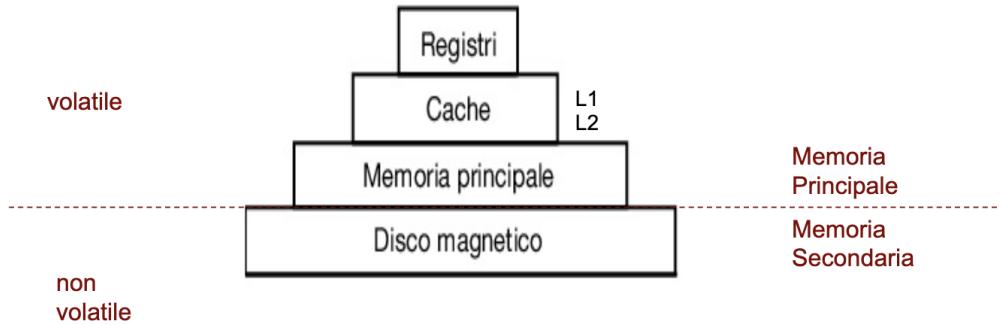
Scheda con circuiti elettronici stampati:

- Componente hardware che fornisce collegamenti elettrici tra i dispositivi
- La scheda madre è il circuito stampato (**PCB**) centrale nel sistema:
  - I dispositivi come CPU e memoria principale sono attaccati
  - include **chip** per eseguire operazioni di basso livello (**BIOS – Basic Input Output System**).
  - BIOS controlla quanta RAM è disponibile, le componenti collegate, scansiona i bus e rileva i dispositivi, poi determina il dispositivo di avvio, carica e avvia il primo settore.

## 6.3 componenti hardware - Memoria

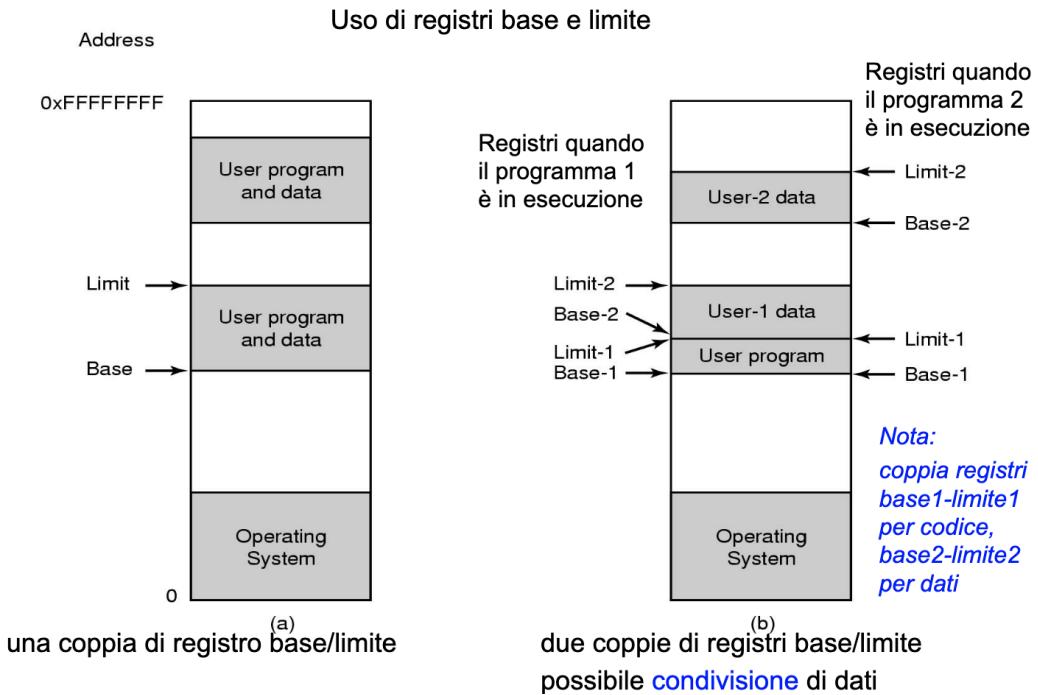


Tipica **gerarchia di memoria**. Con memoria non volatile intendiamo tutta la memoria che non viene cancellata una volta tolta l'alimentazione. In genere nella memoria principale abbiamo i dati del livello più basso a cui la CPU fa direttamente riferimento.



### 6.3.1 memoria volatile

- **Registri**, sono intrni alla CPU, hanno una capacità limitata, ovvero:
  - 32x32 bit per CPU a 32 bit
  - 64x64 bit per CPU a 64 bit
- **Chache**, è composta da livelli, ogni livello è più lento del precedente. La chache è estremamente utile per ridurre i tempi, ma è difficile da progettare.
- **Memoria principale**, si divide in:
  - **RAM**, Random Access Memory, è una memoria volatile con accesso diretto ovunque. Si divide in **DRAM** (dinamica, richiede l'aggiornamento del circuito) e **SRAM** (statica, non richiede aggiornamento). La **Banda** è una caratteristica importante della RAM e consiste nella quantità di dati che possono essere trasferiti per unità di tempo.
  - **ROM**, Read Only Memory, non volatile, veloce, economica, programmata dal costruttore.
  - **EEPROM**, Memoria Flash, non volatile, riscrivibile, molto più lenta della RAM.
  - **CMOS**, volatile, spesso per memorizzare data e ora.



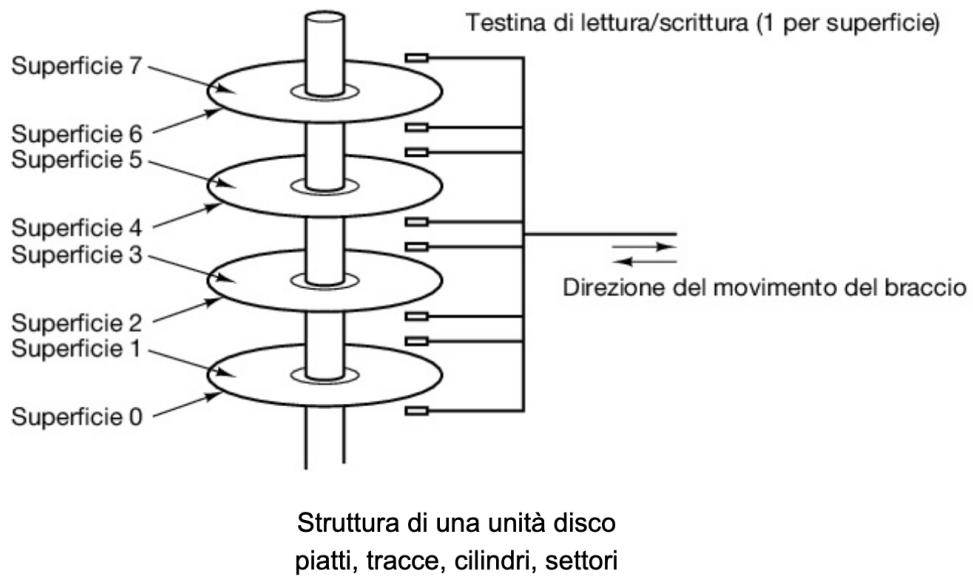
### 6.3.2 memoria non volatile

La **memoria secondaria**, conserva grandi quantità di dati **persistenti** a basso **costo**.

L'accesso ai dati su un disco rigido è molto più **lento** rispetto alla memoria principale.

Il disco funziona grazie ad un movimento meccanico della testa di lettura/scrittura, quindi ci sono dei tempi di trasferimento e una latenza di rotazione.

La memoria secondaria rimovibile facilita il backup e il trasferimento dei dati.



### Dischi:

- **Dispositivo**  
interfaccia semplice
- **Controllore (driver)**  
si interfaccia con il S.O.  
diverso per ogni S.O. che supporta  
su uno o più chip

Come si inserisce un *driver* nel S.O.

- inserzione manuale e riavviare
- in un file del S.O. e riavviare
- senza riavviare      **plug-and-play**

### Driver

ha registri per comunicare  
i registri sono detti spazio di una porta di I/O  
o sono mappati nello spazio indirizzi del S.O. – normali istruzioni  
o sono in una porta speciale di I/O – istruzioni speciali

## Gestione I/O: tre modalità

### - *busy waiting*

chiamate di sistema → chiamate al driver  
→ avvio I/O → attesa **attiva** di fine I/O

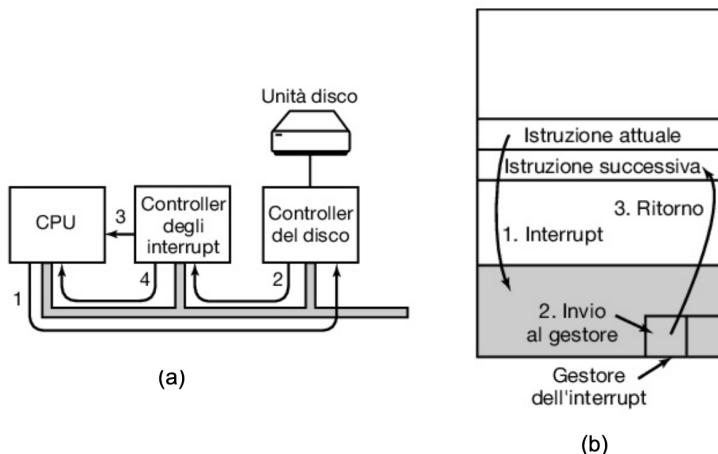
### - *interrupt*

→ avvio I/O → attesa **interruzione del dispositivo** a fine I/O  
→ driver genera **interruzione di I/O**  
→ si seleziona il corrispondente gestore dell'**interruzione**

### - DMA (*Direct Memory Access*)

hardware particolare che svincola la CPU dal controllo di alcuni dispositivi di I/O

## 6.4 componenti hardware – I/O - interrupt



- (a) Passi per l'attivazione di una periferica I/O e gestione dell'interrupt  
(b) Come è interrotta la CPU

## 6.5 componenti hardware – Direct Memory Access (DMA)

La DMA migliora il trasferimento dati fra la memoria e le periferiche I/O.

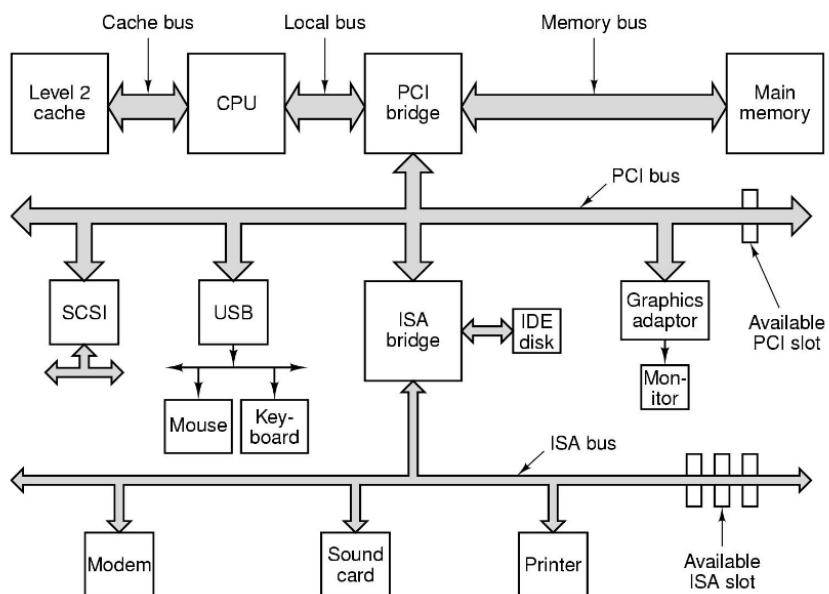
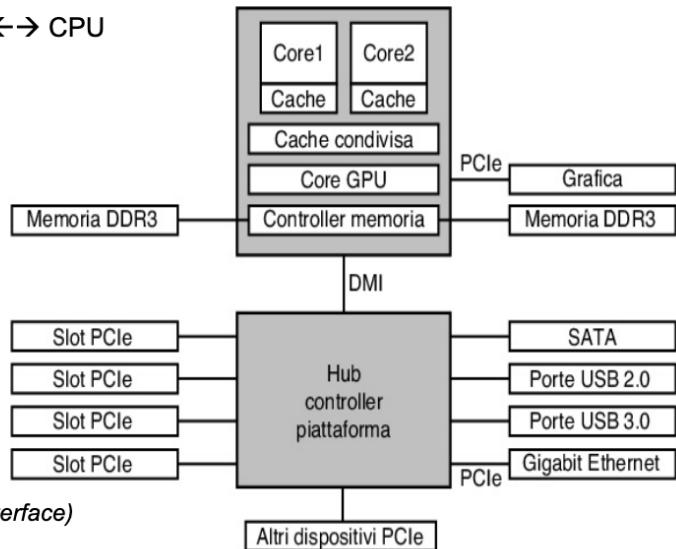
Le periferiche e i controllori **trasferiscono direttamente** i dati da e verso la memoria, il processore è libero di eseguire le istruzioni sw. Il canale DMA usa un controllore I/O per gestire il trasferimento dei dati, notifica al processore quando una operazione I/O è terminata.

Migliora le prestazioni del sistema nel caso di un elevato numero di operazioni of I/O.

## 6.6 componenti hardware - bus

- Un bus è un **insieme di tracce**:
  - le **tracce** sono sottili collegamenti elettrici che trasportano informazioni tra dispositivi hardware
  - una **porta** è un bus che collega solo **2** dispositivi
  - un **canale** di I/O è un bus **condiviso** da diversi dispositivi per eseguire operazioni I/O, che vengono gestite indipendentemente dalla CPU del sistema
  - velocità misurata in MHz
  - **PCIe** (Pheripheal Component Interconnect Express) collega una CPU ai dispositivi; lo standard PCI Express raggiunge fino a 16 o 64 Gb per secondo, velocità che raddoppiano ogni 3-5 anni
  - **AGP** (Accelerated Graphic Port) per schede grafiche

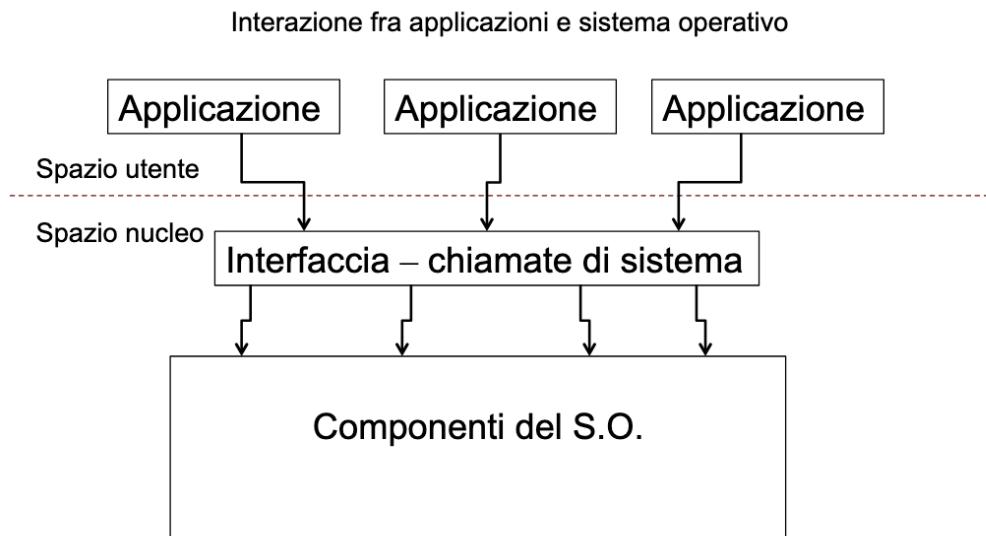
- Limiti alle prestazioni
- Bus addizionali
  - Per I/O
  - Per traffico Memoria  $\leftrightarrow$  CPU
- Esempio di sistema x86  
diversi bus
- bus DMI  
(*Direct Media Interface*)
- bus SATA (*Serial Advanced Technology Attachment*) per hard disk e dischi ottici
- bus USB  
(*Universal Serial Bus*)
- bus SCSI  
(*Small Computer System Interface*)



Struttura di una grande sistema Pentium

- **USB (universal serial bus):**
  - nata per connettere dispositivi lenti
  - oggi USB 3.0 a 5 Gbps
  - non occorre riavviare il sistema per usare i dispositivi
- **SCSI (small computer system interface):**
  - bus ad alte prestazioni
  - tipo hdd, scanner, lettori DVD
  - obiettivo: compatibilità dei dispositivi
  - oggi usato prevalentemente per server e workstations
  - velocità da 5 MBps a 640 MBps

## 7 Sistemi operativi come base e interfaccia delle app.



## 7.1 Tipi e scopi

- Sistemi operativi per **alto livello di astrazione**: Occorre definire speciali requisiti di progetto e supporto hw
  - Grande memoria principale
  - Hardware per usi speciali
  - Grande numero di processi
- Sistemi **integriti** (embedded):
  - Caratterizzato da un **insieme limitato di risorse** specializzare
  - Forniscono funzionalità per vari tipi di dispositivi come telefoni cellulari e PDA (palmari)
  - Gestione efficiente delle risorse fondamentali per la costruzione di un buon sistema operativo
- Sistemi operativi **per mainframe**:
  - Grandi capacità di I/O
  - Servizi: batch, transizioni, time-sharing
  - Es. IBM OS/390, Linux
- Sistemi operativi **per server**:
  - Molti utenti
  - Servizi: archiviazione, web server, ISP
  - Es. Solaris SUN, FreeBSD, Windows Server, Linux
- Sistemi operativi **per multiprocessore**:
  - Molte CPU
  - Computer paralleli, multiprocessori
  - Comunicazione, connessione e coerenza
  - Es. Linux, Windows
- Sistemi operativi **per pc**:

- Multiprogrammazione, un singolo utente
- Es. Linux, FreeBSD, Windows 7, Windows 8, Apple OSX
- Sistemi operativi **per palmari (PDA)**:
  - CPU multicore, fotocamera, sensori, GPS, molte applicazioni
  - Es. Android, iOS
- Sistemi operativi **per sensori**:
  - Es. TinyOS
- Sistemi operativi **per real-time**:
  - Obiettivi con scadenza (deadline)
  - Tipo Hard real-time stretto e improrogabile, che deve essere eseguito in tempo
  - Tipo Soft real-time lasco e con scadenza flessibile
  - Es. controllo di automazione, sistemi audio multimediali
- Sistemi operativi **per smart-card**:
  - Es. pagamento elettronico, trasporti, amministrativi
  - Semplici S.O.

### 7.1.1 Esempi di tipi

- Batch
- Interattivi in time sharing (es. Unix)
- Per PC (es. Windowscagiasbura, Mac OSX, Linuxmerda)
- Real-time (es. telefoni, sistemi di controllo)
- Multimedia (es. video in streaming)
- Transazionali (es. operazioni brevi, banche dati)
- Per dispositivi mobili (es. smartphone, PDA, tablet)
- Embedded (integrati, elettrodomestici, automazione)

## 8 Concetti di Base S.O.

Le architetture di sistemi includono caratteristiche che svolgono **funzioni** del sistema operativo rapidamente in hardware per migliorare le **prestazioni**. In più mantengono caratteristiche che consentono al sistema operativo di **rispettare** rigidamente la **protezione**.

### 8.1 Protezione

Un processore implementa i **meccanismi di protezione** di un sistema operativo:

- Impedisce ai processi di accedere a istruzioni privilegiate o a zone di memoria
- I sistemi in genere hanno diverse modalità di esecuzione:
  - **modalità utente**, dove l'utente può eseguire solo un sottoinsieme delle istruzioni
  - **modalità kernel (kernel mode)**, dove il processore può accedere alle **istruzioni** e alle risorse **privilegiate** per conto dei processi
- Principio del **privilegio minimo** - ad ogni utente minimi privilegi e accessi
- **Protezione** e gestione della **memoria**, previene che i processi accedano alla memoria che non gli è stata assegnata, implementato utilizzando **registri** del processore modificati solo da **istruzioni privilegiate**

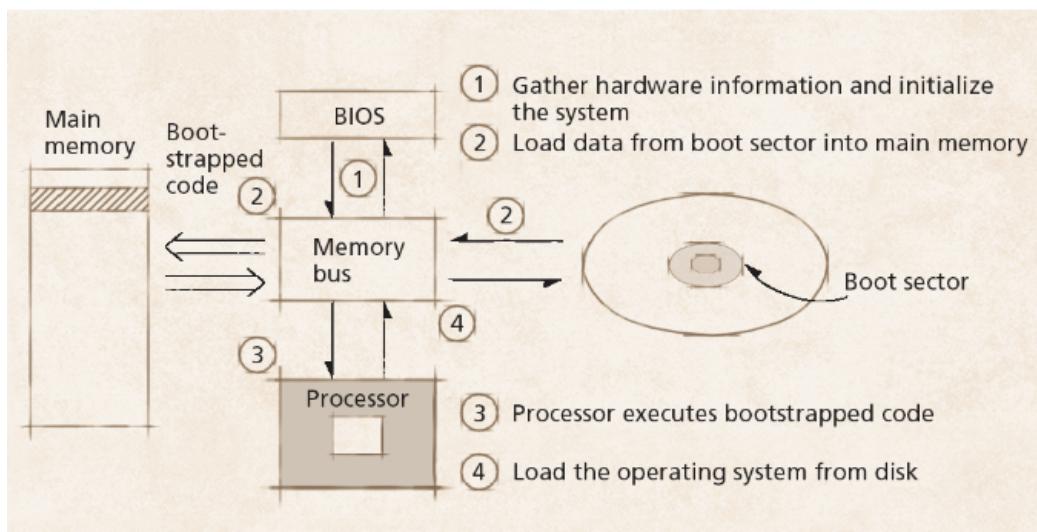
### 8.2 Interruzioni

- **Interrupts** ed Eccezioni:
  - La maggiore parte dei **dispositivi** inviano un segnale chiamato **interrupt** al processore quando accade un evento
  - Le **eccezioni** sono interrupt generati in risposta agli **errori**

- Il Sistema Operativo può rispondere agli interrupts notificandolo ai processi che sono in attesa su questi eventi
- **Timer**; un timer a intervalli **genera periodicamente un interrupt**. I S.O. utilizzano timer a intervalli per evitare che i processi monopopolizzino il processore
- **Clocks**; forniscono una misura di continuità

### 8.3 Avvio

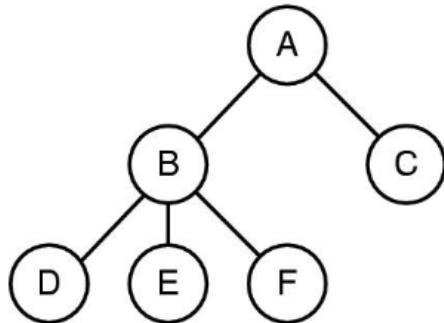
- **Bootstrapping**: caricamento in memoria di componenti del sistema operativo **iniziali**;
  - Eseguita dal Basic Input/Output System (**BIOS**), che inizializza l'hardware di sistema e carica le istruzioni in memoria principale da una regione di memoria secondaria chiamata **settore di avvio** (boot)
  - Se il sistema non è caricato, l'utente non può accedere ad alcuna componente hardware del computer
  - Evoluzione del BIOS: EFI (Extensible Firmware Interface) con interfaccia testuale (shell) e driver. L'utente può accedere ai dispositivi, dischi e rete.



## 8.4 Processi

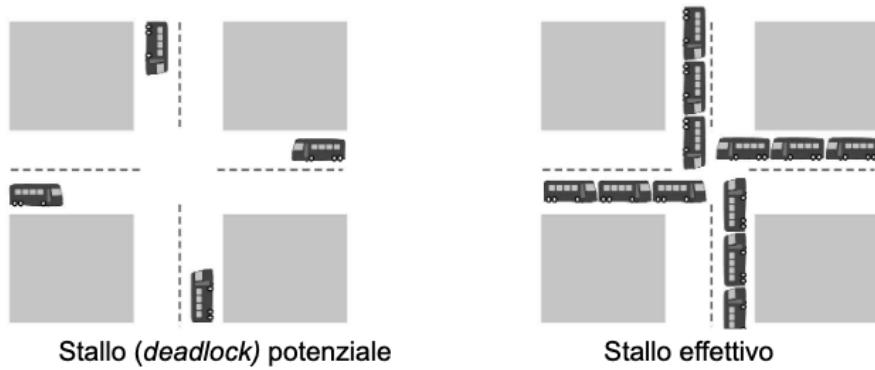
Per **processo** si intende un programma in esecuzione:

- **Processo**: programma in esecuzione
  - Spazio degli **indirizzi**
  - Insieme di **risorse**  
registri, file, segnali,...
  - Descrittore di processo
  - **UID** Identificatore unico di utente
  - Ogni processo ha un UID
  - Tabella di processo
  - Possibilità di creare processi 'figli'
  - Comunicazione e sincronizzazione fra processi
  - IPC (*interprocess communication*)

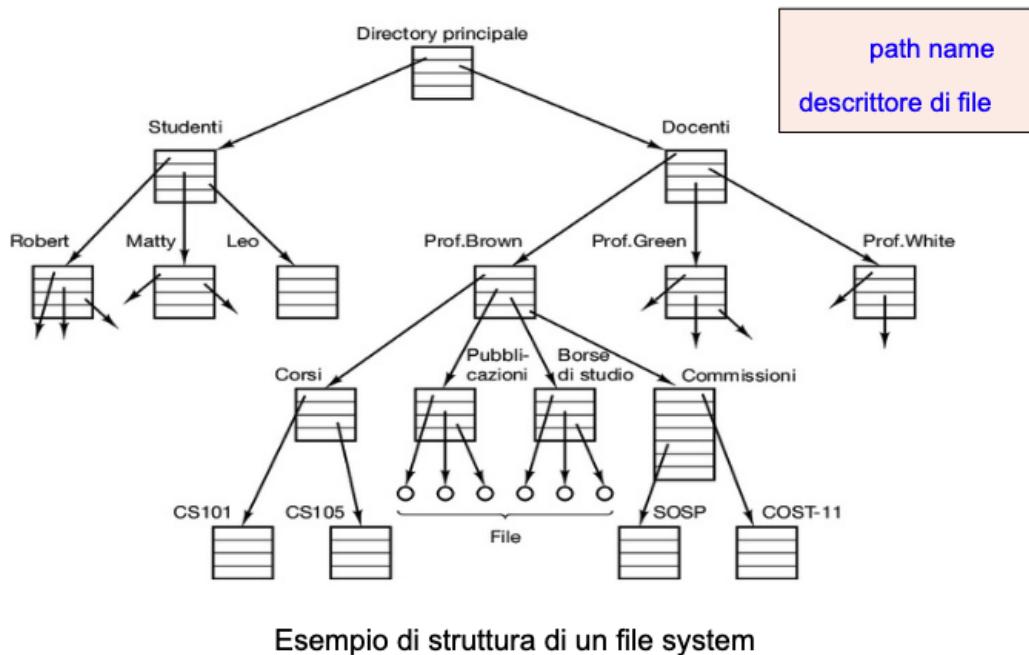


- Un albero di processi
  - A crea due processi figli, B e C
  - B crea tre processi figli, D, E, e F

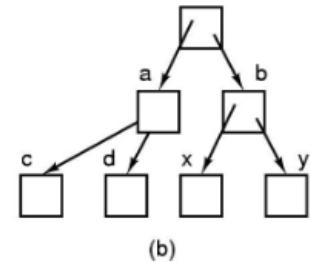
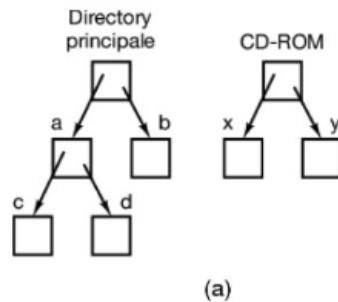
stallo dei processi:



## 8.5 File System



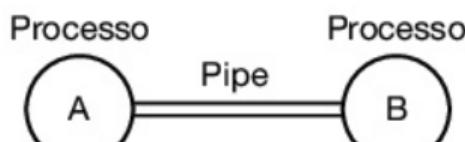
### Montare un file system



- Prima di montare
  - i file sul CD non sono accessibili
- Dopo aver montato il CD sul nodo b,
  - I files sul CD son parte della gerarchia del file system

In Unix

- **File speciali** per vedere e trattare dispositivi di I/O come file  
 a blocchi es. per dischi  
 a caratteri es. per stampanti, modem  
 nella directory /dev
- **Pipe** pseudofile per connessione fra due processi



Due processi connessi da pipe

## 8.6 Plug and Play

La tecnologia **plug and play** consente ai sistemi perativi di **configurare l'hardware** di nuova installazione **senza l'interazione dell'utente**;

- Per supportare plug and play, un dispositivo deve:
  - **Identificarsi** unicamente al sistema operativo
  - **Comunicare** con il sistema operativo per indicare le **risorse e i servizi** che il dispositivo richiede per far funzionare correttamente
  - Identificare il **driver** che supporta il dispositivo e consentire al software di configurare il disp. (tipo assegnare il dispositivo a un canale DMA)

## 8.7 Cache, buffer, spool

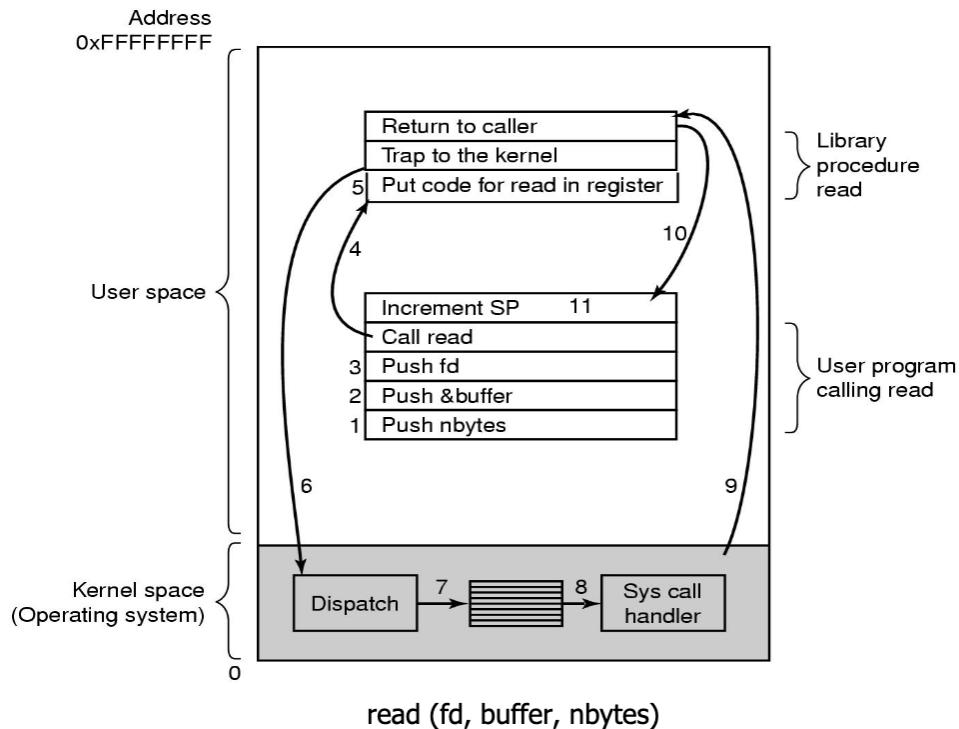
- **Cache:**
  - Memoria relativamente veloce
  - Mantiene **copie di dati** che saranno presto richiesti
  - Aumenta la **velocità** di esecuzione di un programma
  - Cache miss/hit significa: dati presenti o non presenti in cache
- **Buffers:**
  - Area di **memorizzazione temporanea**, una specie di memoria di transito o intermedia
  - Viene usata per il coordinamento delle **comunicazioni** tra i dispositivi a diverse velocità, per memorizzare dati utili all'elaborazione **asincrona**, per permettere ai **segnali** di essere consegnati in modo asincrono.
- **Spooling** (Simultaneous Pheripheal Operations On Line):
  - Tecnica di **buffering** in cui un **dispositivo intermedio** (come il disco) è interposto tra un processo e una periferica I/O lenta
  - Permette ai processi di inviare operazioni di richiesta da una periferica senza aspettare che il dispositivo sia pronto a servire la richiesta

## 8.8 Memoria virtuale

- Possibilità di eseguire programmi con **richieste di memoria maggiori** rispetto alla memoria fisica
- Primo sistema operativo con memoria virtuale  
→ MULTICS
- Oggi in Unix e Windows

## 8.9 Chiamate di sistema

- Nell'interfaccia del S.O.
- Un processo utente attivo attraverso una TRAP
- Il controllo passa al sistema operativo
- Al termine il controllo ritorna all'istruzione successiva del processo utente



### Chiamate di sistema per la gestione di processi in POSIX

Call	Description
pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

### Chiamate di sistema per la gestione di file

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

### Chiamate di sistema per la gestione del file system e delle directories

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

### Altre chiamate di sistema

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Alcune WIn32 API (Application Program Interface) per ottenere servizi del sistema

<b>UNIX</b>	<b>Win32</b>	<b>Description</b>
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

## 9 Componenti e Obiettivi

- Sviluppo dei sistemi di elaborazione
  - I sistemi di elaborazione sono evoluti
  - I primi sistemi non contenevano alcun tipo di sistema operativo
  - successivamente sono stati sviluppati i sistemi con **multiprogrammazione e timesharing**

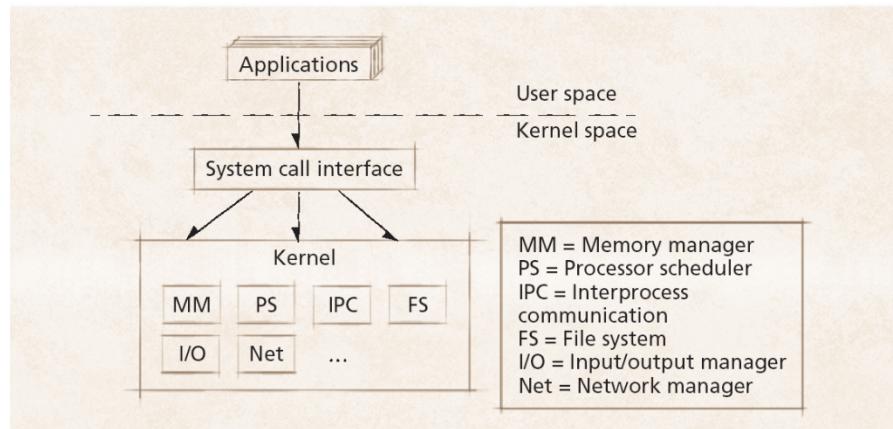
## 10 Architetture

### 10.1 Architettura Monolitica

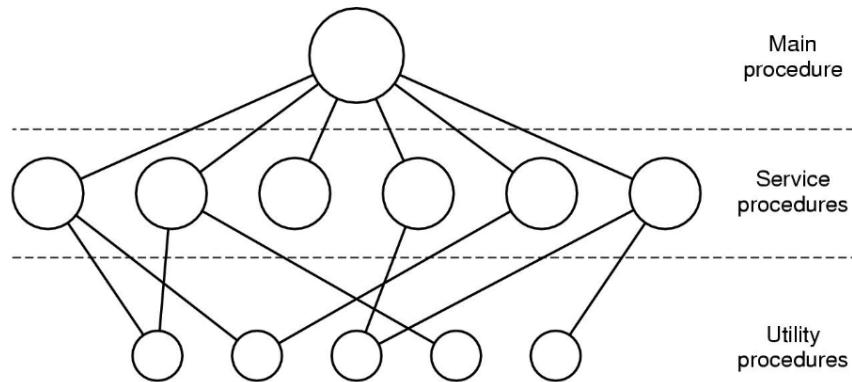
- **Sistemi Operativi Monolitici**
  - Ogni componente è contenuto nel nucleo
  - Ogni componente può comunicare direttamente con qualsiasi altro componente

- L'obiettivo è l'elevata **efficienza**
- Lo svantaggio principale è la difficoltà a identificare eventuali fonti di errore

Architettura di un nucleo di sistema operativo monolitico



- Esempio di architettura monolitica:

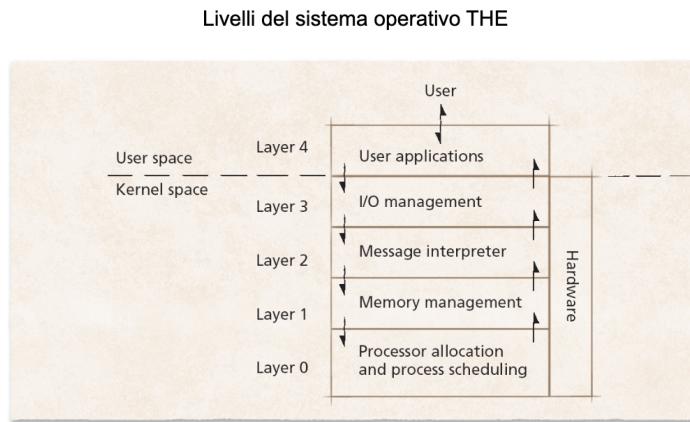


Semplice modello strutturale di un sistema monolitico

## 10.2 Architettura a livelli

- Sistemi Operativi con approccio a livelli

- Obiettivo → migliorare il **progetto** del nucleo monolitico, nei livelli i gruppi di componenti svolgono funzioni semplici.
- **Isolamento:** ogni livello comunica solo con strati immediatamente sopra e sotto
- Le richieste dei processi eventualmente attraversano diversi strati prima del completamento
- Il throughput di sistema può essere meno efficiente dei sistemi con nuclei di SO monolitici. Ulteriori metodi devono essere invocati per passare i dati e il controllo



- esempio:

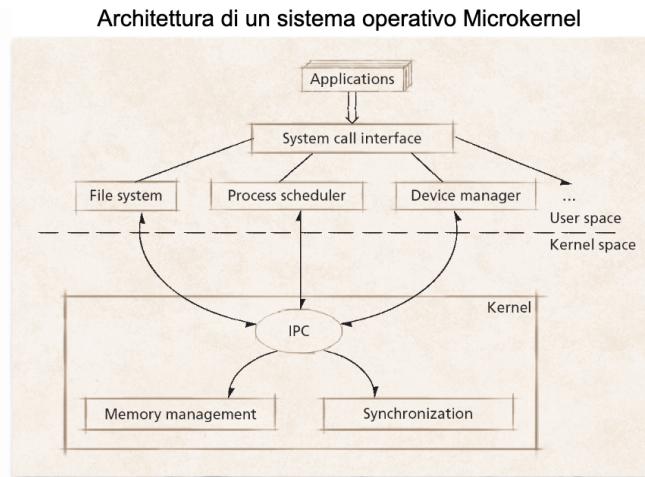
Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Struttura del sistema operativo THE

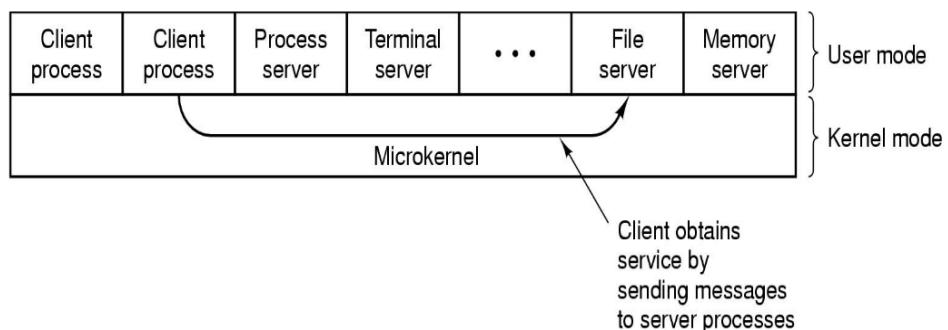
## 10.3 Architettura Microkernel

- Architettura di un S.O. Microkernel

- Fornisce solo **servizi limitati**. Tentativo di contenere le dimensioni del kernel e garantire la scalabilità.
- Elevato grado di **modularità**. Estensibile, portabile e scalabile.
- Aumento del livello di comunicazione fra moduli, può portare ad una degradazione delle prestazioni del sistema.



- esempio:



Modello cliente-servente

## 11 Macchina Virtuale

- Le macchine virtuali (VMs) sono un'astrazione software di un sistema di elaborazione, spesso in esecuzione sopra un SO nativo.

Il sistema Operativo gestisce le risorse fornite dalla macchina virtuale.

Le applicazioni delle macchine V. permettono la coesistenza di **diverse istanze** di un sistema operativo eseguibili contemporaneamente. Si parla di **Emulazione**, software o hardware, che imita le funzionalità di hardware o software non presente nel sistema.

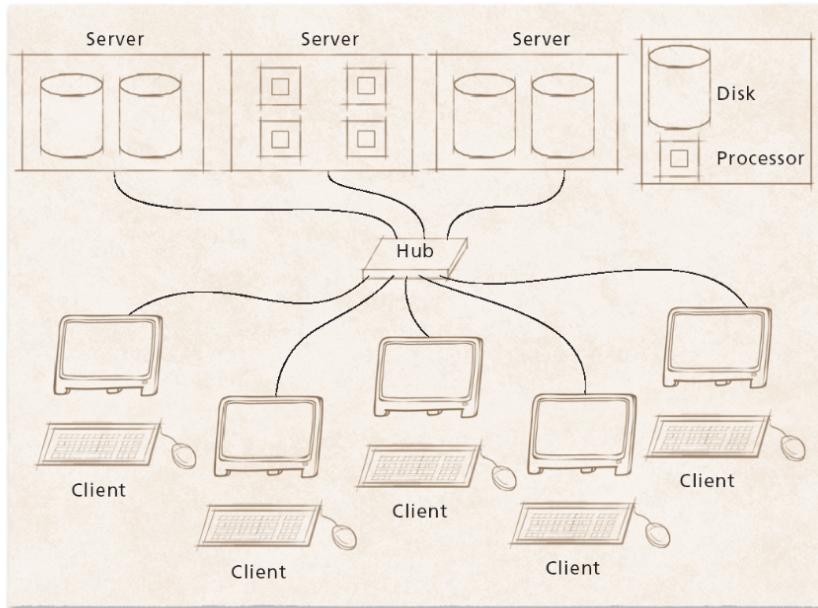
Facilità la **portabilità**.

## 12 Sistemi Operativi di Rete e Sistemi Operativi Distribuiti

### 12.1 Sistema Operativo di Rete

Viene eseguito su un computer, permette ai suoi processi di accedere alle risorse sui compuer remoti.

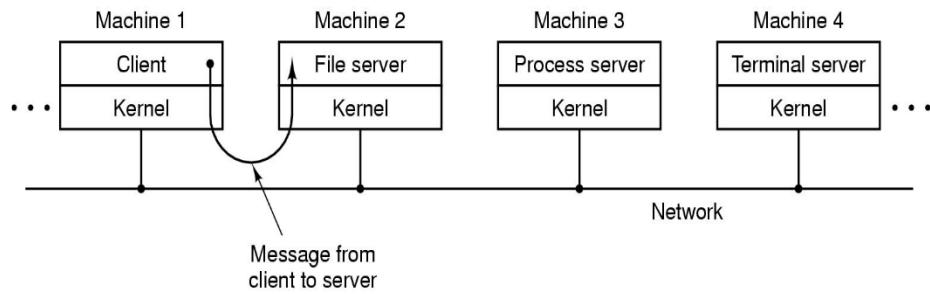
Modello di un sistema operativo di rete di tipo cliente/servente



## 12.2 Sistema Operativo Distribuito

Sistema O. unico, gestisce le risorse su un insieme di sistemi (computer). Gli obiettivi includono; trasparenza di accesso, uso, prestazioni, replicazione ..., Scalabilità, tolleranza ai guasti e consistenza.

**Middleware** è un software per sistemi distribuiti, questo software permette **interazioni** tra più processi in esecuzione su uno o più computer nella rete. Facilita sistemi distribuiti **eterogenei**. **Semplifica** la programmazione delle applicazioni.



Modello cliente-servente in un sistema distribuito