

u1-s1-introduzione-so

Sistemi Operativi

Unità 1: Introduzione

Introduzione ai Sistemi Operativi

[Martino Trevisan](#)

[Università di Trieste](#)

[Dipartimento di Ingegneria e Architettura](#)

Argomenti

1. Componenti di un sistema di elaborazione
 2. Architettura di un sistema di elaborazione
 3. Definizione di sistema operativo
 4. Componenti di un sistema operativo
 5. Definizioni relative ai sistemi operativi
 6. Tipologie di sistemi operativi
-

Le quattro componenti di un sistema di elaborazione

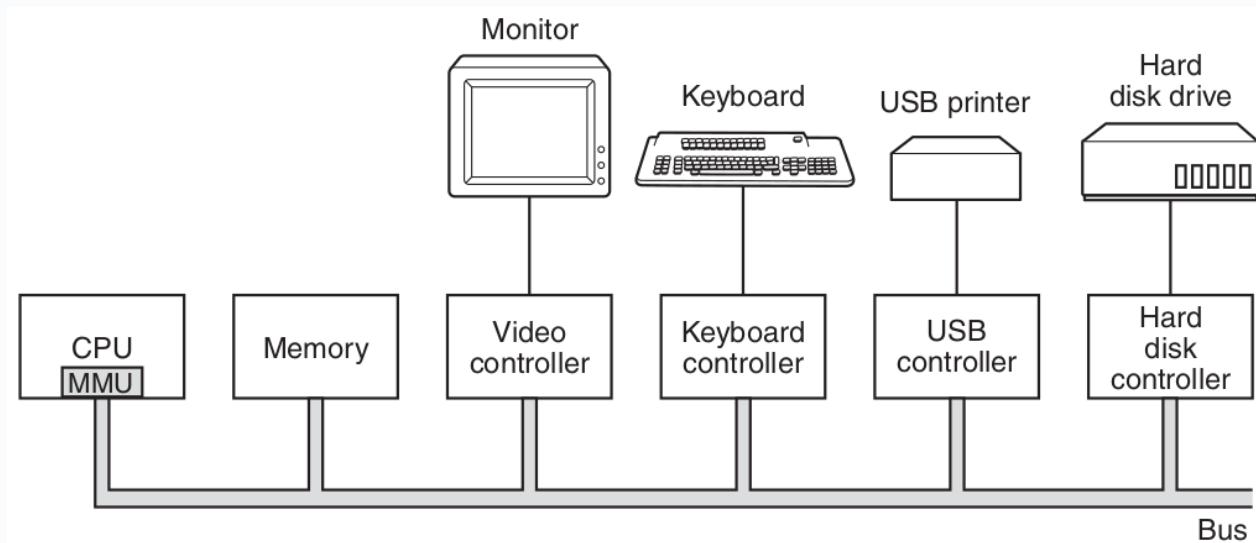
Un sistema di elaborazione si può suddividere in quattro componenti:

- *Hardware*
 - Fornisce le risorse fisiche: CPU, memoria, disco
- *Sistema Operativo*
 - Gestisce l'accesso all'hardware da parte dei programmi
- *Programmi Applicativi*
 - Eseguono i compiti desiderati dagli utenti o dal sistema
- *Utenti*
 - Lanciano e usano programmi

I *programmi applicativi* usano il *sistema operativo* per interagire con l'*hardware*.

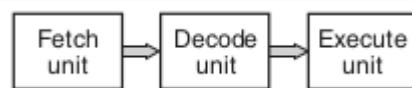
Architettura di un sistema di elaborazione (ripasso)

- Composto da diverse unità
- Il **bus** mette in comunicazione le componenti del sistema



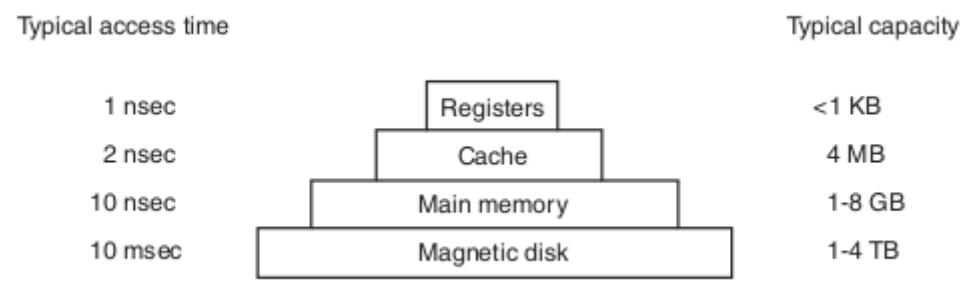
LA CPU.

- La **CPU** esegue le istruzioni che preleva dalla memoria.
- **Moduli fondamentali:**
 - Arithmetic Logic Unit
 - Control Unit
 - Registri
- Tre fasi per eseguire ogni istruzione: fetch, decode e execute. In particolare con i cicli **executre** e **fetch** si ha l'**interazione col mondo esterno**



LA GERARCHIA DELLE MEMORIA.

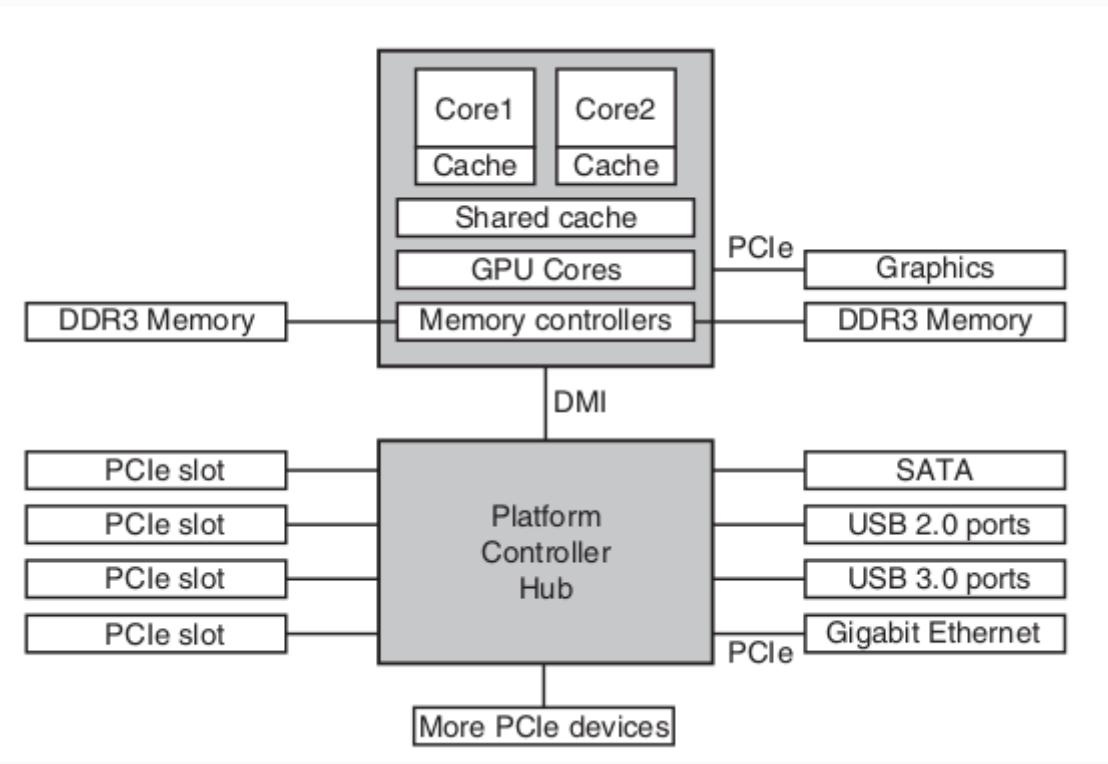
- La **Memoria** fornisce un modo per salvare i dati.
- Un sistema ha vari tipi di memoria, con caratteristiche diverse
 - Organizzazione in una gerarchia di memoria



Come esempio di "**magnetic disk**" si ha la **HDD** o la **SSD**, come esempio di "**main memory**" si ha la **RAM** e i "**registers**" si trovano nella **CPU**.

DISPOSITIVI I/O

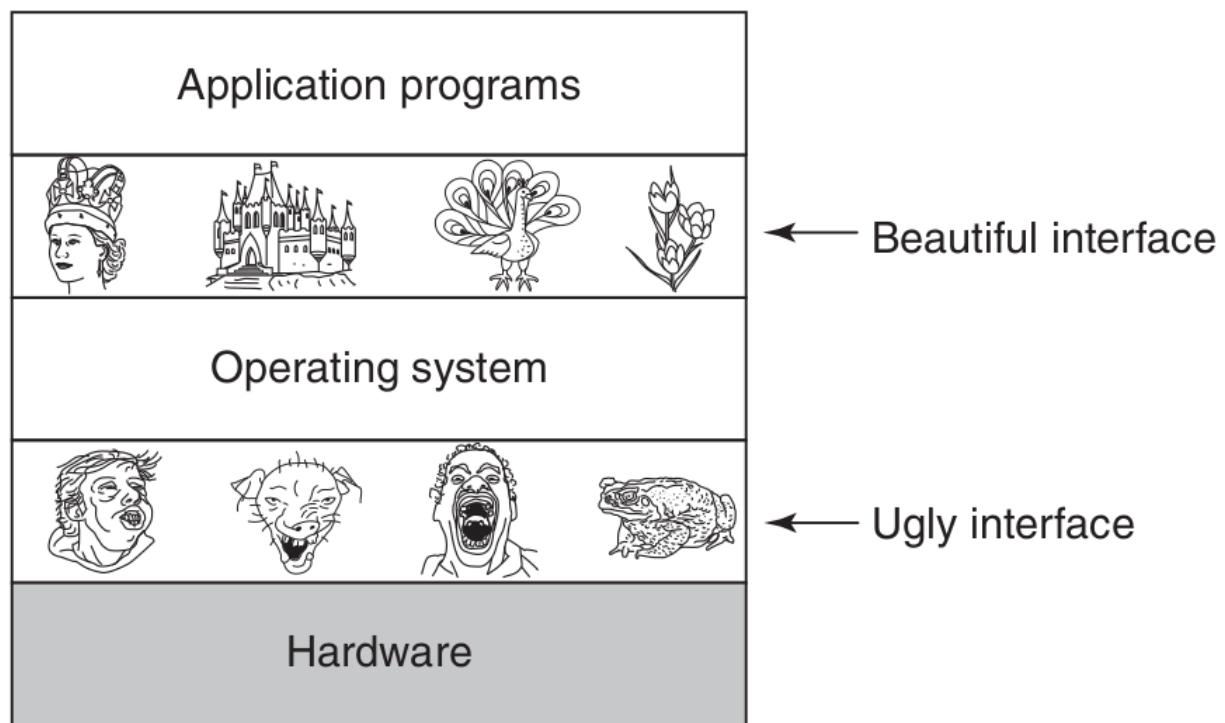
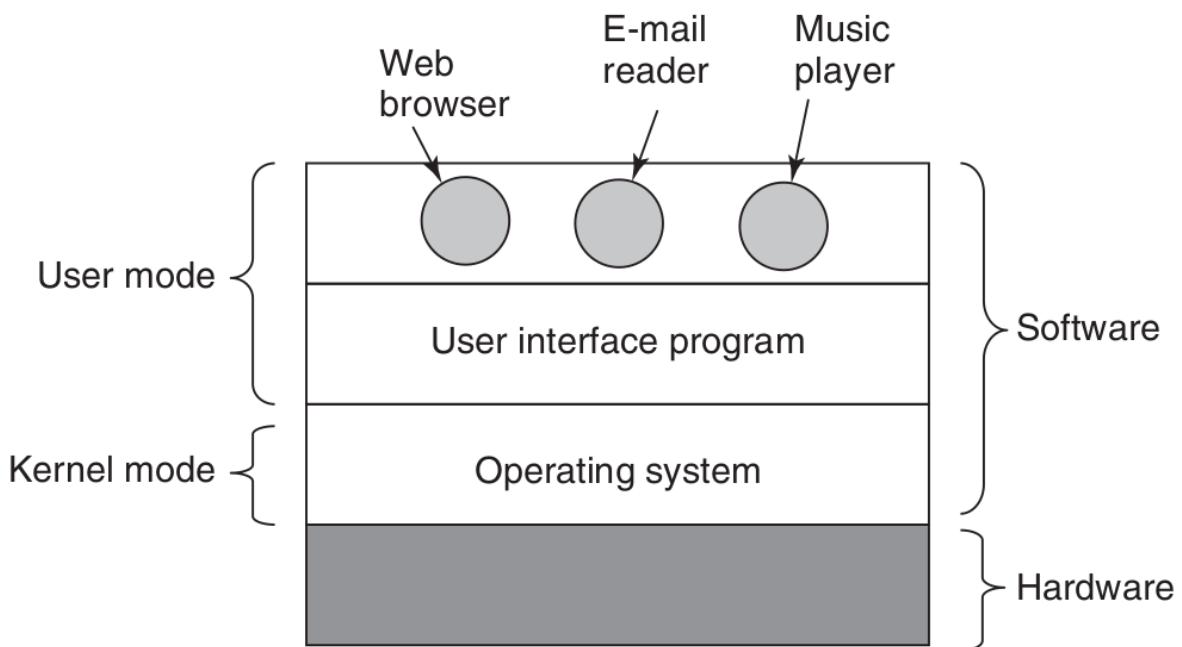
- Un sistema di elaborazione interagisce con l'esterno tramite dispositivi di **Input/Output**
 - Schermo
 - Tastiera
 - Mouse
 - Rete
 - **Sensori** (questi sono particolarmente importanti! sono presenti soprattutto nei smartphone)
- Comunicano con la CPU tramite il bus



Definizione di sistema operativo

IL COMPITO DEL SISTEMA OPERATIVO.

- Il compito dei sistemi operativi è:
 - Interagire con l'**hardware**
 - Fornire all'utente un **modello** di computer più semplice (ovvero con ciò che si chiama **UI**)
 - In un certo senso, i sistemi operativi rendono gradevole ciò che ha un'interfaccia sgradevole. Ovvero di **mediare** la comunicazione tra le **applicazioni** e **hardware**. Infatti, ogni **dispositivo I/O** ha una sua **logica**, quindi il compito di un **sistema operativo** è anche quello di **permettere la portabilità delle applicazioni**.



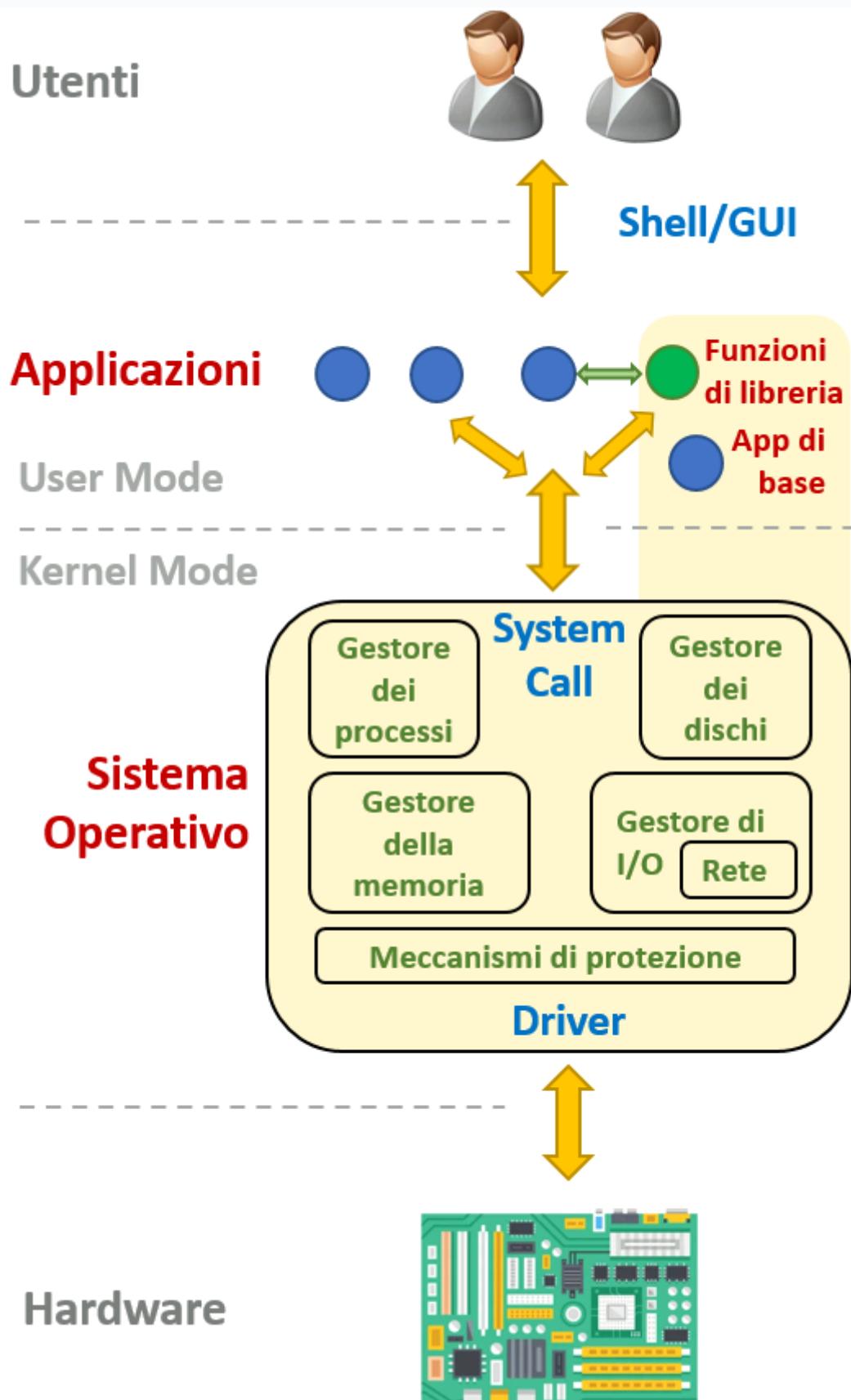
COME L'OS FA DA MEDIATORE?

- Il sistema operativo si interfaccia con i dispositivi hardware tramite dei *moduli software* chiamati *Driver*. Quindi si ha una situazione del tipo *OS ↔↔ Driver ↔↔ Hardware*.
- Offre servizi alle applicazioni tramite *API* chiamate *System Call*, che permettono alle applicazioni di *usare l'hardware facilmente*.
- Organizzato in moduli

Componenti di un sistema operativo

VISIONE GENERALE.

- Un sistema di elaborazione è composto di diversi moduli
 - Che offrono servizi a utente
 - Ma interagiscono anche tra loro



OSSERVAZIONI.

- I *system call* sono molto *fondamentali*, dal momento che per le *applicazioni* è *strettamente proibito* interagire direttamente con gli hardware. Infatti, si ha una situazione del tipo
 - *Software* \leftrightarrow *System call* \leftrightarrow *OS* \leftrightarrow *Driver* \leftrightarrow *Hardware*
- Qui il ruolo del *sistema operativo* diventa fondamentale

LE COMPONENTI SOFTWARE DEL SISTEMA OPERATIVO.

Principalmente un sistema operativo ha *tre gestori*, per gestire la mediazione *software-hardware*.

- **Gestore dei processi.** Crea e gestisce i *processi* (un processo è un programma in esecuzione) e *trova le risorse* di CPU e memoria *necessarie* all'esecuzione
- **Gestore della memoria.** La memoria di un elaboratore è un unico vettore (dato che stiamo implicitamente usando le architetture *Von-Neumann*) e il SO gestisce l'*allocazione di memoria ai processi* e la *condivisione della memoria* tra programmi.
- **Gestore dei dischi.** La memorizzazione su disco (come un HDD) è persistente e il SO organizza la memoria su disco in una struttura ad albero di directory e file
- **Gestore dei dispositivi di I/O (e di rete).** I dispositivi di I/O *non* sono gestiti *direttamente* dalle applicazioni (sarebbe infatti complicato e creerebbe problemi di funzionamento); i SO utilizza invece i *driver* per pilotare i dispositivi. La connessione di rete rappresenta un caso particolare di I/O: ha un trattamento speciale nei moderni SO.
- **Gestore dei meccanismi di protezione.** Nei SO ci sono tanti *utenti* con *diversi permessi di accedere alle risorse* (per accedere a file, dispositivi, configurazione, ecc...). I SO implementa le policy di accesso; vedremo in seguito il caso particolare di *Linux*.

Dopodiché ci sono altre componenti che agevola ai processi di fare ulteriori cose.

- **Sincronizzazione.** Permette ai processi di *comunicare* e di *sincronizzarsi* tra loro, dal momento che *ogni processo* va per *conto suo*, ma vanno comunque *coordinati*.
- **Virtualizzazione.** Per supportare la creazione di macchine virtuali o simili (e.g., container). Se il SO offre funzionalità per la virtualizzazione, *le VM saranno molto più veloci* perché possono avere *accesso diretto* ad (alcune) risorse. Ultimamente la *virtualizzazione* divenne un tema fondamentale per i *sistemi operativi*.

Definizioni relative ai sistemi operativi

Adesso diamo delle *definizioni* relativi ai *sistemi operativi*.

1. Kernel

1. Kernel

- Si dice il "*cuore del SO*"
- Include *tutti i moduli* visti in precedenza
 - Gestisce le operazioni fondamentali e a più basso livello
- Modulo software *sempre in esecuzione*
 - Con *privilegi speciali*, ovvero *massimi*
 - Tutte le altre applicazioni (di utente o di sistema) hanno *privilegi minori* (ad esempio a loro è *proibito* interagire con i dispositivi I/O)
 - Si *appoggiano al kernel*
- Diverse tipologie di kernel
 - *Monolitici*: il kernel è un unico programma che esegue tutto il codice necessario.
 - *Micro-Kernel*: cercano di delegare alle applicazioni più funzionalità possibile (per rendere più modulare un sistema operativo).
 - *A livelli stratificati*: il kernel (e i programmi) sono organizzati in una gerarchia di processi a privilegi crescenti (questo per creare meno distinzioni tra app e kernel).

2. Processo

2. **Processo**

- Programma in esecuzione
 - Con certi *privilegi e risorse* della CPU o RAM
- Accede alla CPU *a turno* con gli altri programmi
 - Il SO deve permettere che il suo stato sia conservato quando viene messo in pausa
 - Richiede salvataggio di registri di CPU
 - Ogni processo identificato da un numero detto *PID* (*Process Identifier*)
- Un processo può creare altri processi detti *figli*
 - Si dice che i processi sono organizzati in un *Albero dei Processi*

3. Thread

3. **Thread** (da inglese *"filo")

- Un processo *raggruppa le risorse* di un programma in esecuzione
- Un *thread individual* raggruppa a sua volta *un flusso di esecuzione*
- Un processo può avere invece al suo interno *uno o più flussi* in esecuzione (come ad esempio *Chrome* con le sue tab)
- Identificato da un identificatore detto *TID*

4. System call

4. **System call**

- Sono delle *funzioni* messe a disposizione dal SO alle applicazioni
- Offrono i servizi del SO per *creare processi, accedere ai dischi*, ecc...

- Il kernel **"serve"** (nel senso di compiere) le richieste di System Call dei programmi
- **ATTENZIONE!** Da non confondere con le **Funzioni di Libreria!!!**
 - Che sono **moduli software** (**pezzi di codice**) che svolgono compiti comuni a più applicazioni
 - Sono software comuni con stessi **privilegi di applicazione** (ovvero a "**user-level**")
 - Possono (ma non sempre) chiamare delle System Call
 - Non possono **interagire direttamente** con l'**hardware**
- Sintassi delle system call.:**
- Le system call sono funzioni C. Hanno aspetti diversi

POSIX su Linux

```
C
int read (int fd, void *buffer, size_t nbytes);
```

Win32/Win64 API su Windows

```
C
BOOL ReadFile (
    HANDLE fileHandle,
    LPVOID dataBuffer,
    DWORD numberOfBytesToRead,
    LPDWORD numberOfBytesRead,
    LPOVERLAPPED overlappedDataStructure
);
```

Esempi di System Call comuni.

- Gestione dei **processi**: **fork exec wait kill**
- Gestione dei **file su disco**: **open close read write**

5. I sistemi di libreria

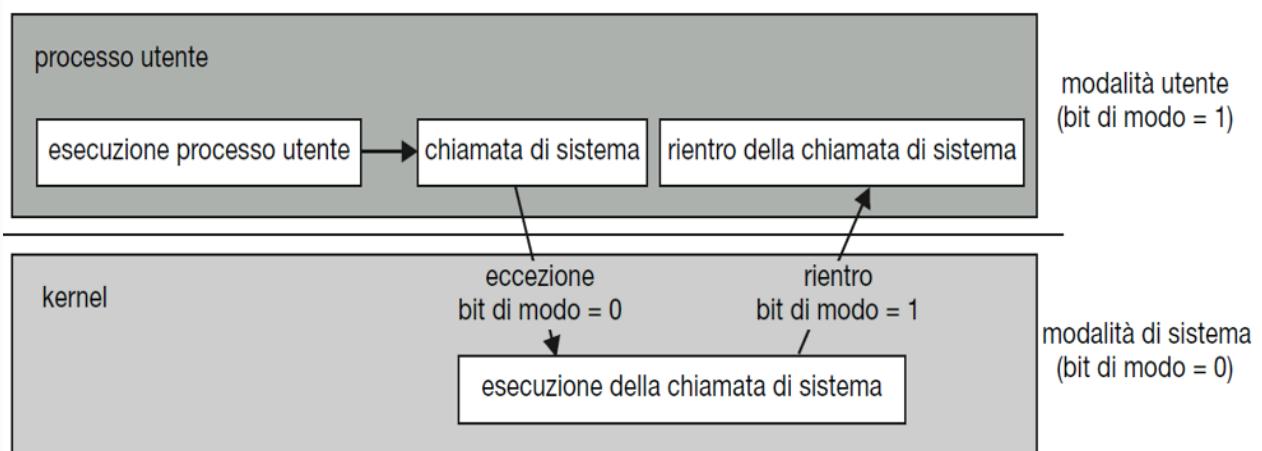
I SISTEMI DI LIBRERIA.

- **Nota:** i SO operativi offrono anche molte **funzioni di libreria**
 - Alcune funzioni di libreria non necessitano di System Call
 - Calcolare la radice quadrata: **double sqrt(double arg)**. Non necessita di una System Call
 - Scrivere su schermo: **int printf(char *format, arg list ...)**. Formatta la stringa da scrivere e poi chiama la System Call **write**
- **ATTENZIONE!** Quindi da questo discende che la funzione **printf(...);** **NON** può essere una **system call!** Questa **invoca** una system call, ma non lo è!

6. Kernel/User Mode

5. **Kernel/User mode** nella CPU: Ci sono principalmente *due livelli di privilegio*.

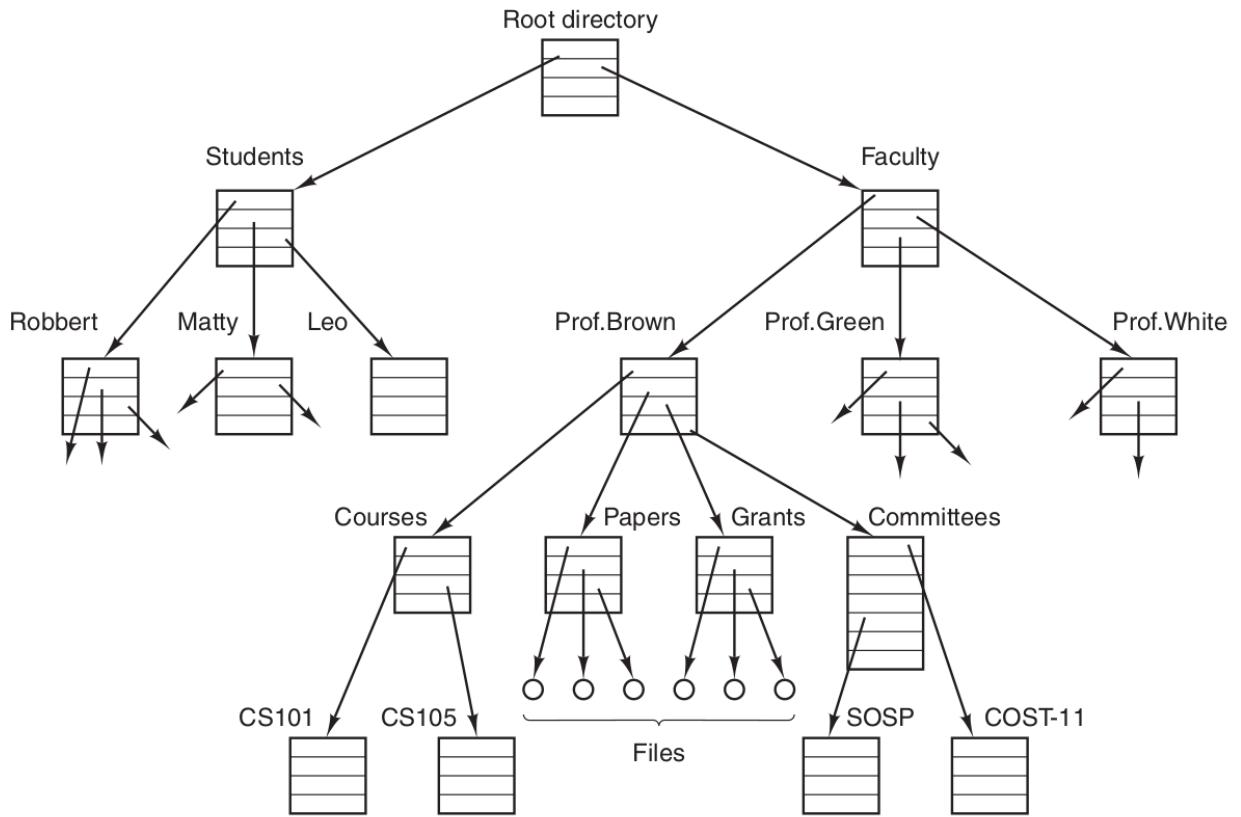
- **Kernel Mode (alto livello)**: Il kernel ha accesso *completo* all'hardware
 - Può accedere a ogni locazione di memoria e registro di dispositivo di I/O; può accedere a **TUTTO** per quanto concerne i dispositivi I/O
 - **Solo** codice di ottima qualità. Gli errori sono potenzialmente *distruttivi!!!*
- **User Mode**: le applicazioni hanno possibilità limitate
 - Accedono a uno spazio di indirizzamento limitato (*memoria virtuale*); quindi **non** possono accedere a "*memorie virtuali di altro processi*".
 - Non possono accedere direttamente ai dispositivi di I/O; quindi **niente Assembly**!
 - Necessaria cooperazione della CPU che implementa specifiche funzionalità
 - Le *funzioni di libreria* si eseguono in *User Mode*



7. File system

7. **File System**: ci sono due *significati* di *file system*

- Struttura che include un *insieme di file e cartelle*
- Organizzato secondo un *albero* (ovvero un *file system* è l'*albero*)



8. I path

8. Organizzazione dei path.

- **Root Directory:** la *radice di tutti le cartelle*. Si *identifica* con **/** in Linux
- **Working Directory:** directory dove un processo viene lanciato
Ci sono *due modi* per *esprimere un path*.
- **Path Assoluto:** *inizia* con **/** e identifica un percorso a partire dalla *Root Directory*, dalla *radice*
- **Path Relativo:** *non inizia* con **/** ma con un nome. Identifica un path relativo alla *Working Directory* del processo

9. Bootloader, login e shell

9. Bootloader:

- Il codice che carica in memoria il kernel al momento dell'accensione del sistema
- Contenuto in *ROM/EEPROM immunificabile*
- Cosa succede quando *accendiamo un computer*? La *CPU* accede un *indirizzo noto della memoria*, che a sua volta contiene una *ROM* con il *bootloader*.

9. Login:

- *Autenticazione* di un utente nel sistema, solitamente tramite *username e password*

9. Shell:

- *Programma* che legge *comandi* da tastiera, li *esegue* e ne stampa l'*output*
 - **NOTA.** Come input viene *accettato solo la tastiera*

- Metodo di accesso tradizionale
 - *Non fa parte del kernel*
 - Quando un sistema viene avviato, il kernel avvia sempre una shell o l'interfaccia grafica
 - Ne faremo un pesante utilizzo nel corso
-

Domande

Cosa é un processo?

- a) • Un programma
- b) • Un algoritmo
- c) • Un programma in esecuzione

Le System Call sono usate da:

- a) • SO per interagire con l'hardWare
- b) • Dai processi per interagire col SO

Le Funzioni di Libreria vengono eseguite:

- a) • In User Mode
- b) • In Kernel Mode

Il File System é organizzato come:

- a) • Un Grafo contentente cicli
- b) • Un Grafo NON contentente cicli

Risposte

C, B, A, B

u1-s2-tipologie-storia

Sistemi Operativi

Unità 1: Introduzione

Storia e Tipologie di Sistemi Operativi

[Martino Trevisan](#)

[Università di Trieste](#)

[Dipartimento di Ingegneria e Architettura](#)

Argomenti

1. Storia dei sistemi operativi
 2. Tipologie di sistemi operativi
 3. Linux
-

Storia dei sistemi operativi e degli elaboratori

I Primi passi ai Sistemi Operativi

Condizione necessaria per un SO: Avere un sistema di elaborazione

LA PRIMA IDEA: Primo elaboratore *progettato* da Charles Babbage nella *prima metà dell'800*

- Puramente meccanico.
- Non fu mai costruito

I PRIMI ELABORATORI: I primi elaboratori vennero *costruiti* negli anni '40 del '900

- Basati su *valvole*, oggi si usano i *transistori* che sono più *veloci ed efficaci* (e meno grandi)
- Programmati direttamente in linguaggio macchina
- Nessun sistema operativo. L'elaboratore eseguiva un programma per volta

FIGURA: Elaboratore a valvole

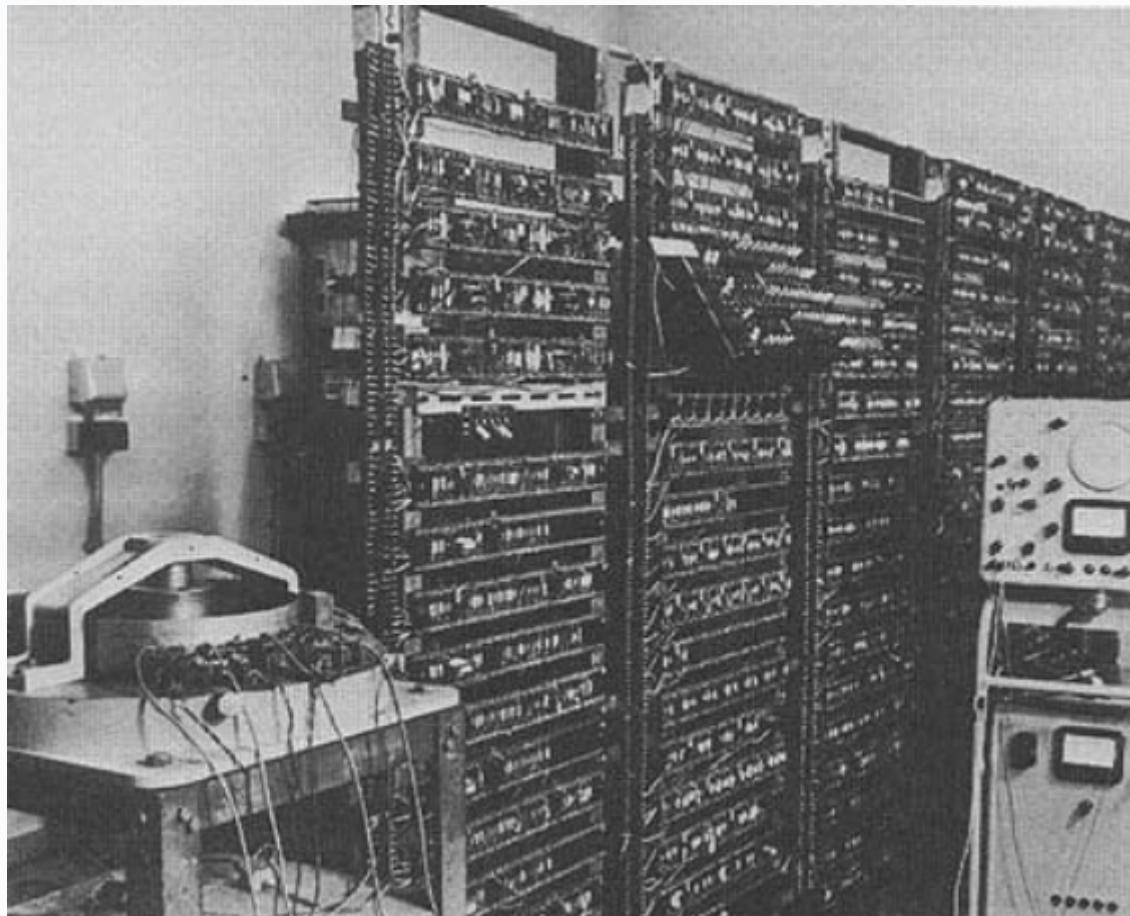
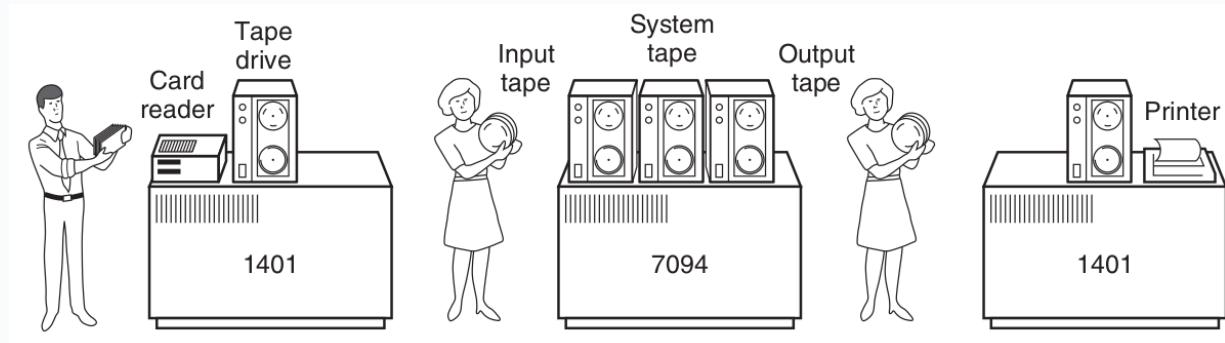


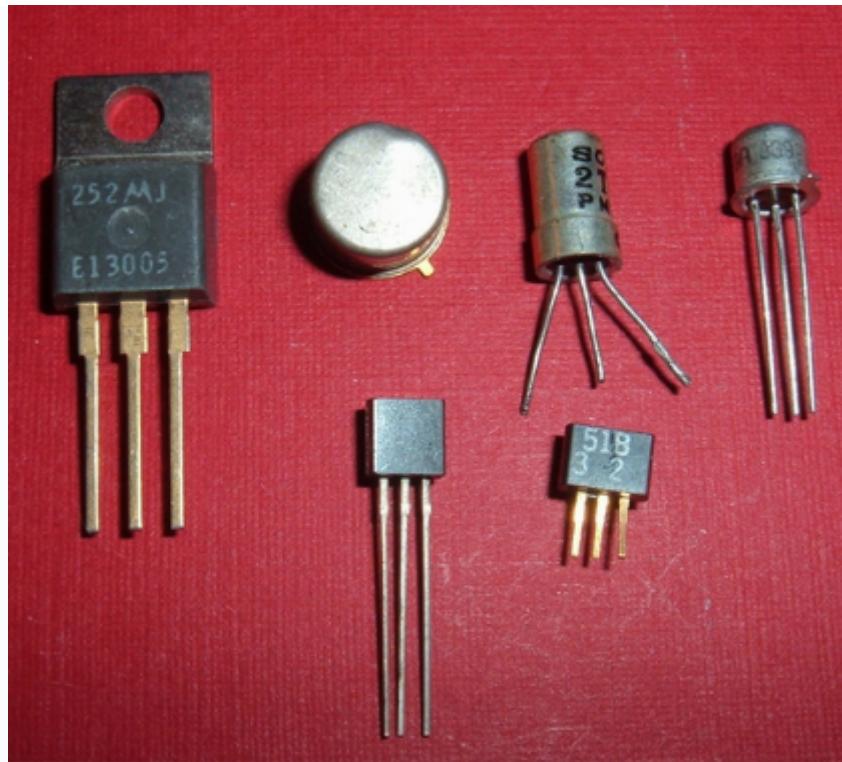
Il passaggio ai transistor

Elaboratori a transistor (1955-1965) (nel *dopoguerra*)

- Mainframe custoditi in locali e da tecnici specializzati
- Programmi scritti su schede perforate o nastri magnetici
- Primi linguaggi di programmazione (e.g., *FORTRAN*)
- I programmi venivano *eseguiti in sequenza*.
- Il *sistema operativo* aveva il *solo* compito di *eseguire programmi* in sequenza: il codice veniva rappresentato dalle *schede perforate*.

FIGURA: MACCHINA A TRANSISTOR





I circuiti integrati e i Primi Sistemi Operativi

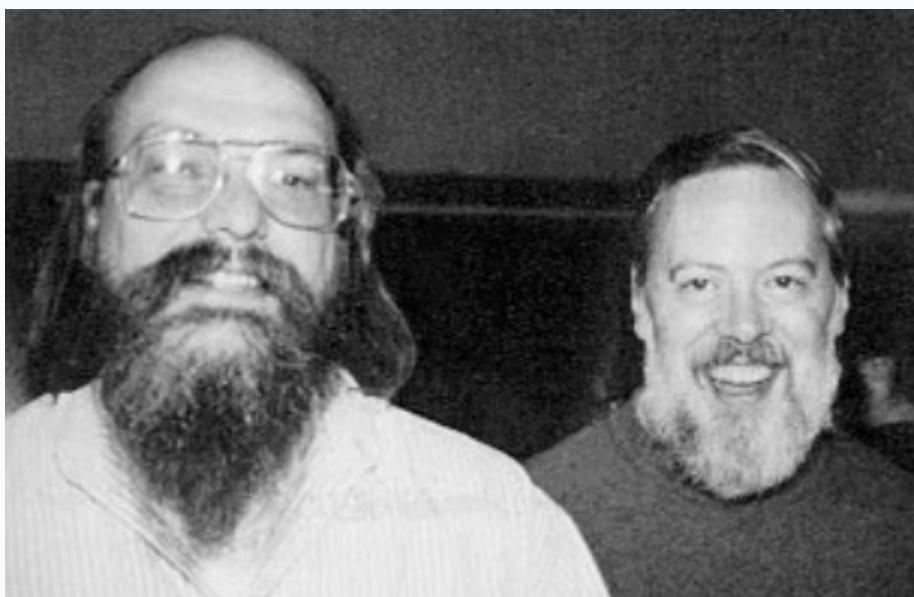
Circuiti Integrati (1965-1980)

- Prestazioni maggiori. Prezzi e dimensioni minori: infatti i *circuiti integrati* sono delle *singole schede* con molti *transistor*
- IBM crea la linea di computer IBM 360, col suo SO detto **OS/360**
 - Introduce multiprogrammazione: più processi in esecuzione contemporaneamente
 - Usato ancora oggi in alcuni campi
- Il MIT assieme a partner industriali (3) sviluppa **MULTICS**
 - Progettato per *main frame molto potenti*
 - Permette l'utilizzo a *centinaia di utenti*
 - Poco successo: *MULTICS* rimane comunque importante, dato che ha posto le *fondamenta* per la costruzione di altri *sistemi operativi*. Infatti, da MULTICS nasce *Unix* nei Bell Labs (New Jersey), come progetto personale di Ken Thompson, Dennis Ritchie e altri: poi da UNIX nascerà la famiglia Linux

FIGURA: Elaboratore a circuito integrato



FIGURA: Ken Thompson e Dennis Ritchie



Apple e Microsoft

I personal Computer (1980-oggi)

- Grazie a sviluppi nei Circuiti Integrati fu possibile produrre computer a prezzi bassi.
- Nei primi anni '80 nasce **Microsoft/DOS**, inizialmente pensato per computer IBM con CPU Intel (famiglia 8086)
- La **Apple** inventa un sistema operativo con **GUI** che ottiene molto successo, da cui **Windows** si ispirerà

FIGURA: Bill Gates e Steve Jobs



Microsoft: Fondata nel 1975 da Bill Gates e Paul Allen

- Nel 1981 commercializza **MS-DOS**
- Nel 1985 commercializza **Windows**
 - Sistema operativo con *interfaccia grafica a finestre* (da cui il nome)
 - Orientato a *processori Intel*
- Nel tempo ha commercializzato versioni a
 - 16 bit (Windows 1.0, 1985 – Windows 3.1, 1992)
 - 16/32 bit (Windows 9x, 1993-2000)
 - 32/64 bit (da Windows NT in poi)
 - Oggi abbiamo Windows 11

Apple: fondata nel 1976 da Steve Jobs, Steve Wozniak e Ronald Wayne

- Dal 1984 al 2001 commercializza Mac OS
 - SO completamente grafico
 - Raggiunge *limiti strutturali* di sviluppo alla fine del '90, non permettendo
 - *Multitasking preemptivo*: questo si rivelò come la "*falla fondamentale*" del sistema operativo **MAC OS**, in quanto con questo sistema *era il singolo processo a dire quando interrompere la CPU*. Se si ha un "*processo buggato*", allora l'intero processore sarebbe bloccato!
 - Memoria protetta
- Ricostruendo tutto *da capo*, nel 2001 commercializza Mac OS X
 - Nato per computer Macintosh
 - Inizialmente retro-compatibile con Mac OS
 - Basato su architettura UNIX

I SISTEMI OPERATIVI DI OGGI.

- Windows e Apple Mac OS continuano lo sviluppo fino ad ora.
 - Windows 11 e Mac OS 13 Ventura
- Dagli anno '90 in boom dei **telefoni cellulari**, porta alla nascita di sistemi operativi dedicati. Nascono:
 - (Symbian: morto nel 2011)
 - Android
 - Mac OS (*iOS*)

Tipologie di sistemi operativi

Diverse varietà di SO. Alcune ancora vive, altre morte e sepolte.

1. SO per mainframe

- Per elaboratori *enormi in grandi compagnie*
- Supportano tanti utenti e risorse
- In declino in favore di SO general purpose (Linux)
- Esiste ancora OS/390, discendente di OS/360 di IBM. Ad esempio ancora oggi i *server delle banche* usano questi OS.

2. SO per PC (ad uso personale)

- Sono i più diffusi.
- *Basati su interfaccia grafica*
- Pensati per un solo utente, *non esperto*
- Esempi: Windows, MacOS, Ubuntu (?), ...

3. SO per server

- Per *professionisti*
- Spesso dotati di *sola shell*
- Sono varianti di quelli per PC
- Esempi: Linux, Windows Server

4. SO per Smartphone o tablet

- Basati su GUI e input touch
- Esempi: Android, MacOS

5. SO integrati

- Per router, elettrodomestici, veicoli
- Non accettano programmi esterni

6. SO per sensori

- Su dispositivi con risorse *molto limitate* (come ad esempio calcolatrici)

- Molto leggeri e semplici

Usi comuni per le tipologie 5,6: Frighi, elettrodomestici, ...

7. SO real time

- Per applicazioni particolari dove il **tempo è fondamentale**
 - Processi industriali, *aerei*, autoveicoli
- Alcuni compiti devono essere svolti **tassativamente** entro *una deadline*: il più *velocemente possibile*
 - Design del sistema notevolmente più complicato

SO per smartcard

- Le *smartcard* (e.g., Bancomat) hanno un sistema di elaborazione e un SO (*spesso*)
 - Requisiti di **basso consumo** e **sicurezza**
-

Storia del Linux

UNIX

Abbiamo detto che **Unix** nasce negli anni '70 da MULTICS: da sottolineare che è stata nata negli anni '70 (o fine anni '60)! ancora oggi ci stiamo portando la *filosofia* dietro Unix, dal momento che è stato *progettato così bene*.

Nascono *numerose varianti* negli anni '80 che vengono *standardizzate* (in particolare sono standarizzati i *system calls*)

- Standard **ISO C** - 1972
- Standard **Posix** - 1988

Tutte le versioni erano a pagamento, in capo ad *AT&T*

- Il codice era *closed-source* (a *pagamento*), molto lungo e complesso

MINIX

Creato da Andrew *Stuart Tanenbaum*

- Uno degli autori di uno dei libri consigliati in questo corso

E' un *clone di UNIX*:

- *Open-Source*
- A *micro-kernel*
- Pensato per la *didattica*, in particolare per un corso universitario sui *sistemi operativi*
- Non adatto a essere un vero SO

FIGURA: Stuart Tanenbaum



LINUX

Nel 14.03.1991 il giovane finlandese universitario Linus Torvalds crea il kernel **Linux**:

- Sviluppato *a partire da Minix*
- Per esser un *vero SO* (non solo per scopi didattici): quindi anche un uso *professionale*
- Tante distribuzioni: *Ubuntu, Debian, Fedora* e infinite...
- Ormai diffuso globalmente

FIGURA: Linus Torvalds che se la ride



FIGURA: Linus Torvalds che alza un dito medio nel mezzo di una conferenza, gesto per insultare la compagnia NVIDIA



Definizione di Unix, Linux, GNU

Definizioni Generali

- **Unix**: sistema operativo sviluppato negli anni '80 in AT&T (*closed source!*)
- **Linux**: è un kernel Unix-like sviluppato da Linus Torvald dal 1991 (*open source!*)
- **GNU**: sistema operativo open source (*kernel escluso!*) Unix-Like. Può funzionare con diversi kernel; la maggior parte funziona col *linux*. GNU è un acronimo ricorsivo, che sta per *GNU's Not Unix!*⁽¹⁾. La GNU è quella parte che fornisce le *applicazioni di default* e roba del genere.

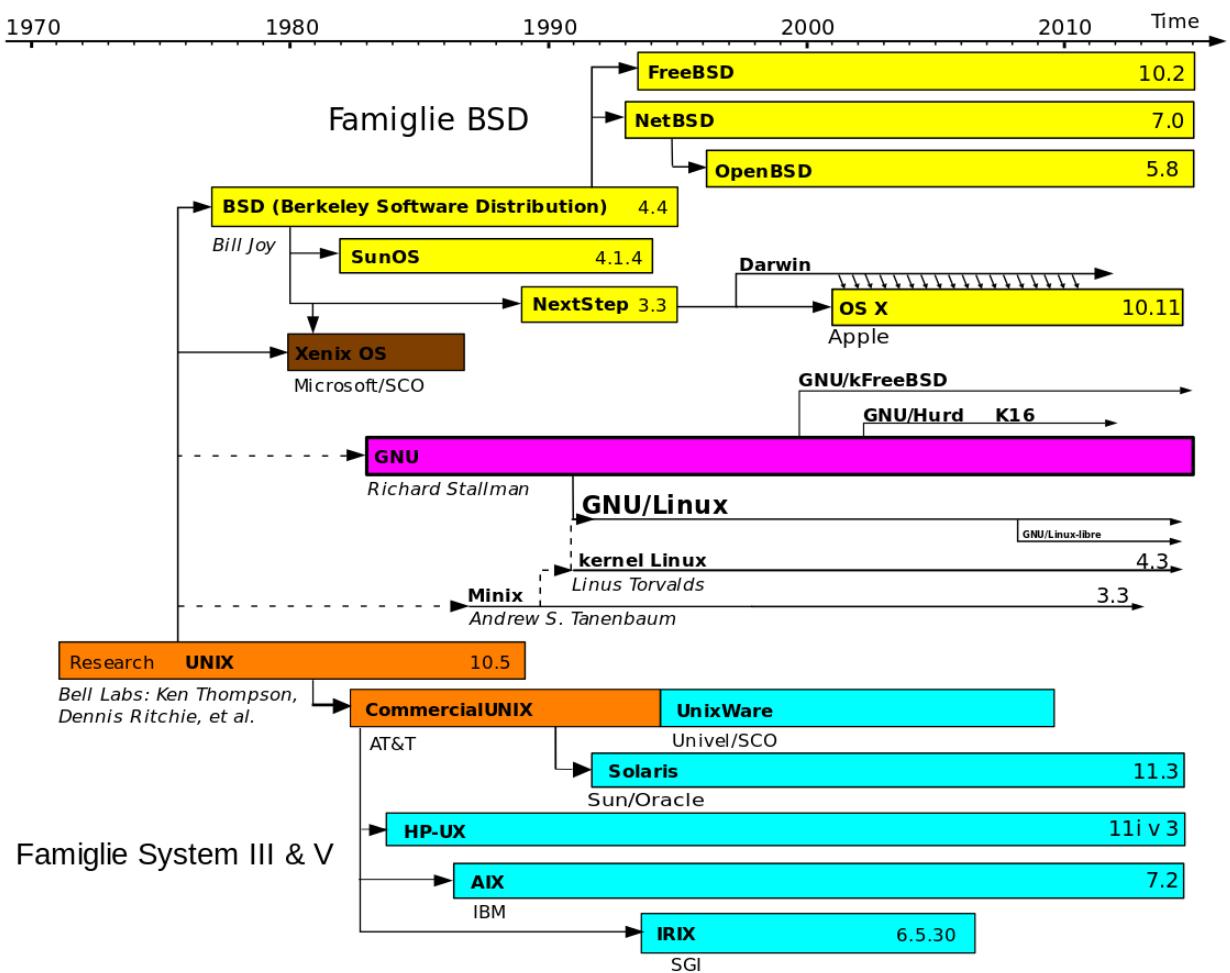
Ora lo standard è **GNU/Linux**: sistema operativo GNU con kernel Linux

Nota: L'evoluzione diretta di Unix é il SO *Berkeley Software Distribution* (BSD), da cui derivano *FreeBSD* e *Mac OS X*

⁽¹⁾. *GNU is a recursive acronym for "GNU's Not Unix!", chosen because GNU's design is Unix-like, but differs from Unix by being free software and containing no Unix code. Stallman chose the name by using various plays on words, including the song The Gnu.*

Diramazioni da UNIX (riassunto generale)

FIGURA: Linea temporale di diramazioni dall'UNIX



NOTA. Quindi da questo schema si sa che l'**OS X** (ovvero il sistema operativo per i dispositivi Apple) **deriva** dall'**UNIX**, che ha la stessa convenzione di **Linux**; è comunque sbagliato dire che **OS X** è **Linux**! Sono due cose comunque diversissime

Linux Oggi

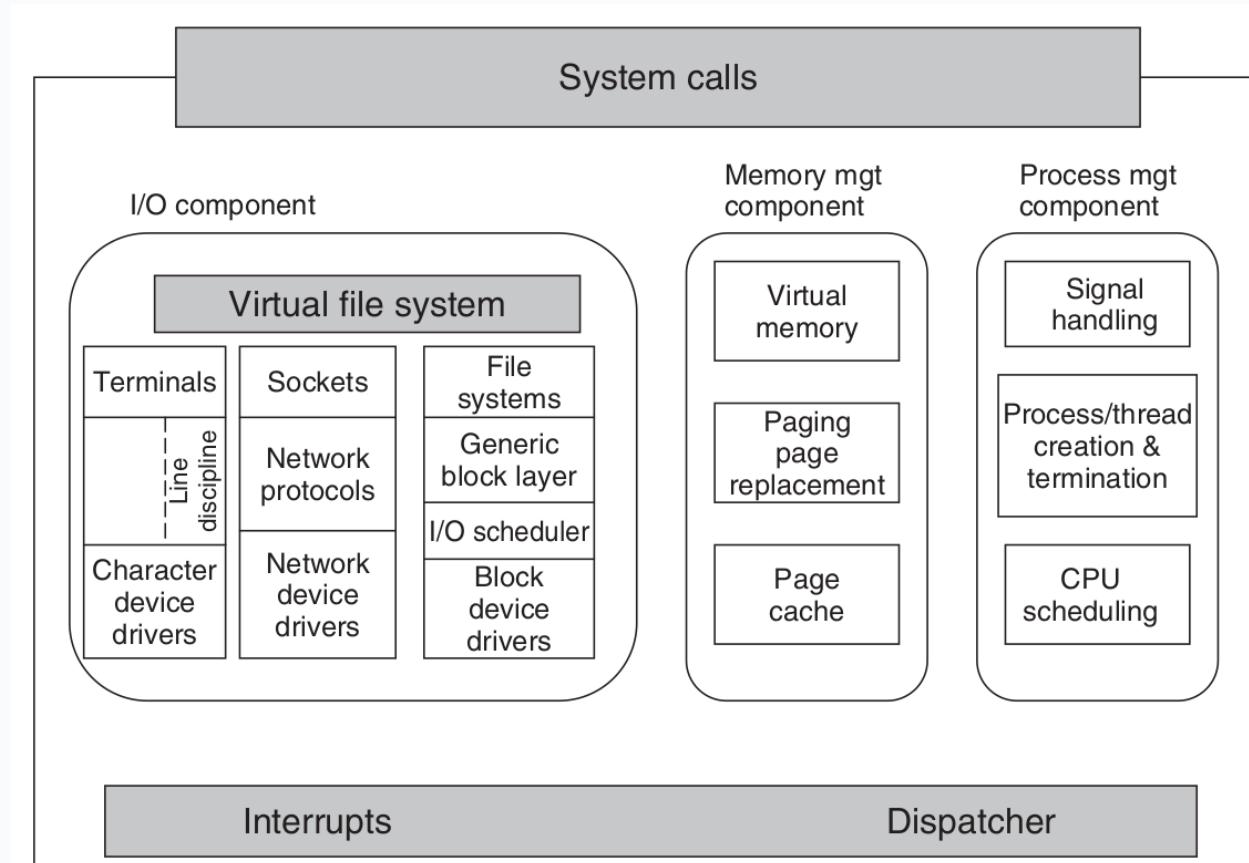
- Stabile, maturo e free (*a livello professionale*)
- Più complesso da usare di Windows
- Alla base di quasi tutte le tecnologie per:
 - Servizi web: **hosting** (e.g. di siti, ...)
 - Archiviazione dei dati: **database**, data warehouse
 - **Sistemi embedded**
 - Piattaforme per **Intelligenza Artificiale**; algoritmi addestrati su server con OS basati sul kernel Linux

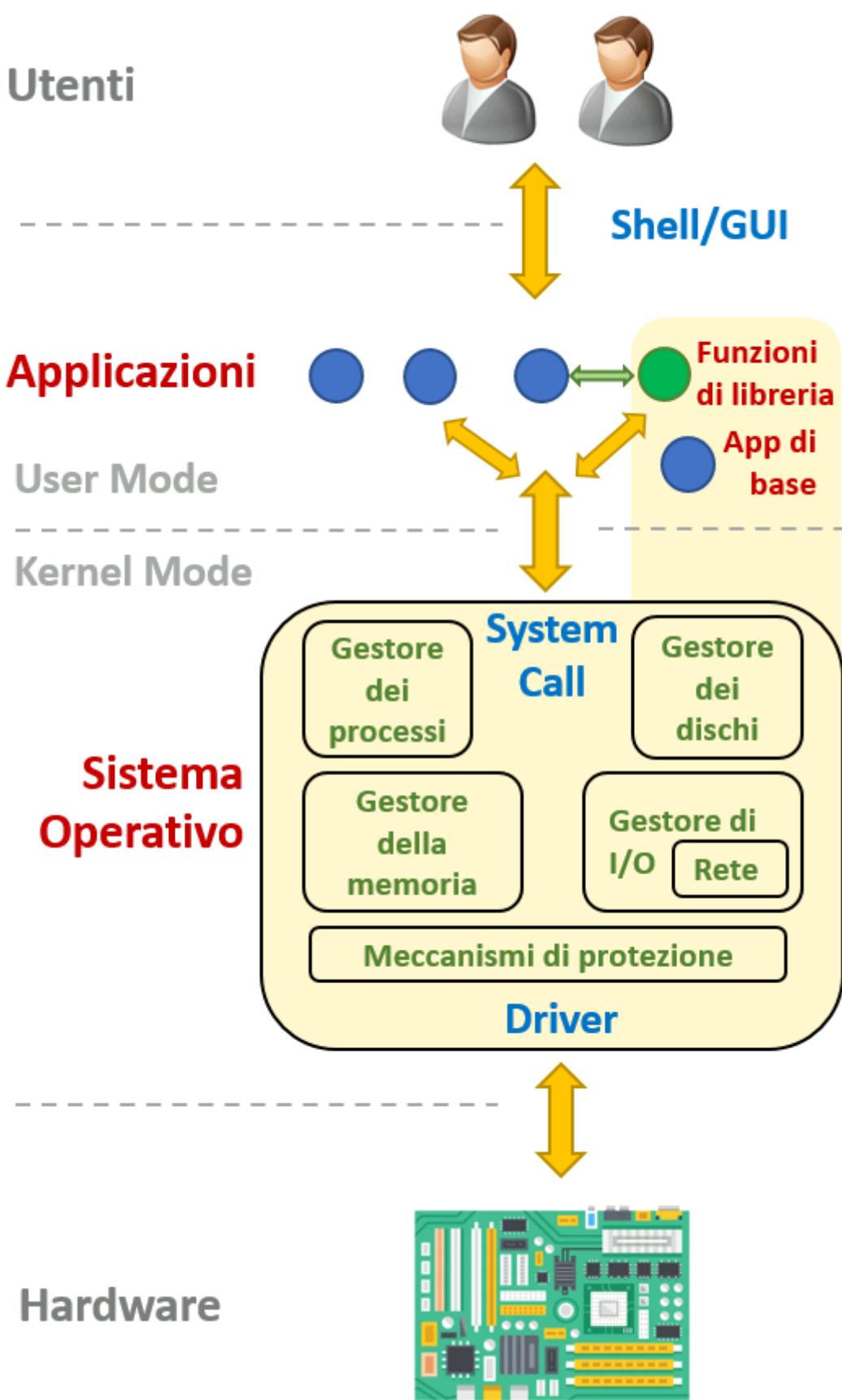
Questo corso si concentrerà sull'uso di Linux, dato che per noi diventerà importantissimo!!!

Kernel Linux

Simile rispetto alla nostra *definizione generica di SO*

FIGURA: Kernel Linux





GNU/Linux

La **GNU/Linux** include **librerie e utility** di default, come i seguenti.

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

Domande

POSIX é:

- Uno standard
- Un SO
- Una famiglia di SO

Risposta: *Uno standard*

UNIX é:

- Closed-Source
- Open-Source

Risposta: *Closed-Source*

Linux é:

- Closed-Source
- Open-Source

Risposta: *Open-Source*