



# RSO: Sistemi Operativi

spitfire

A.A. 2023-2024

# Contents

<b>1</b>	<b>Struttura e servizi</b>	<b>3</b>
1.1	Componenti di un sistema di elaborazione . . . . .	3
1.2	Requisiti per i sistemi operativi . . . . .	4
1.3	La maledizione della generalità . . . . .	4
1.4	Struttura dei sistemi operativi . . . . .	5
1.5	Servizi offerti da un sistema operativo . . . . .	6
1.6	Chiamate di sistema e Application Programming Interfaces . . . . .	6
1.7	Programmi di sistema . . . . .	7
1.7.1	Intefaccia utente: l'interprete dei comandi . . . . .	8
1.7.2	Interfaccia utente: le interfacce grafiche . . . . .	9
1.7.3	Intefaccia utente: Le interfacce touch-screen . . . . .	10
1.8	L'implementazione dei programmi di sistema . . . . .	10
<b>2</b>	<b>Processi e thread: i servizi</b>	<b>11</b>
2.1	Programmi e processi . . . . .	11
2.2	Struttura di un processo . . . . .	11

# 1 Struttura e servizi

Cosa sappiamo sui sistemi operativi? Sappiamo che, per esempio, i principali sono **linux, Windows e MacOS**; che il sistema operativo è il **primo programma che viene eseguito dopo il boot**. Di solito un sistema operativo fornisce un **ambiente desktop a finestre e ci permette di installare nuove applicazioni**. Ci permette inoltre di eseguire tante applicazioni **contemporaneamente**, anche più dei **core dei processori**. Inoltre, esso **mantiene e organizza i nostri dati sotto forma di file e cartelle**. Quindi, cos'è un **sistema operativo**? Esso è:

- Un insieme di **programmi** (Software)
- Che gestiscono **gli elementi fisici di un computer** (Hardware)

E a cosa serve un sistema operativo?

- Fornire una **piattaforma di sviluppo per le applicazioni**, che permette loro di **condividere e astrarre** le risorse HW.
- Agisce da **intermediario** tra utenti e computer, permettendo agli utenti di **controllare l'esecuzione dei programmi applicativi** e l'assegnazione delle risorse HW ad essi
- **Protegge le risorse degli utenti** (e dei loro programmi) dagli altri utenti (e dai loro programmi) e da eventuali **attori esterni**

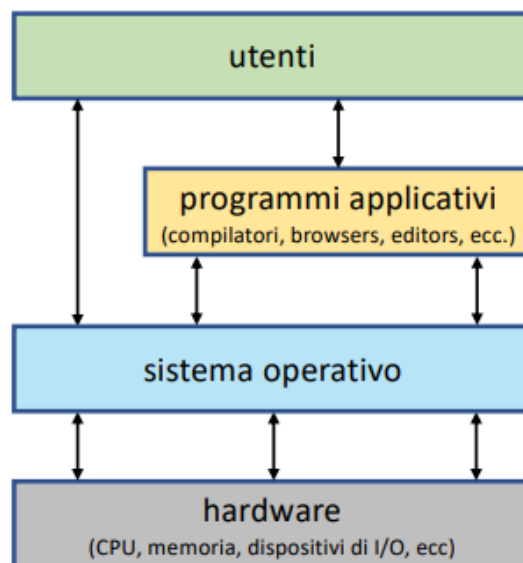
Un sistema operativo è quindi in primo luogo una **piattaforma di sviluppo**, ossia un insieme di funzionalità software che i programmi applicativi possono usare. Tali funzionalità permettono ai programmi di poter usare in maniera conveniente le risorse hardware di condividerle:

- Da un lato il sistema operativo **astrae** le risorse hardware, presentando agli sviluppatori di programmi applicativi una visione delle risorse hardware più facile da usare e più potente rispetto alle risorse hardware "native".
- Dall'altro, il sistema operativo **condivide** le risorse hardware tra molti programmi contemporaneamente in esecuzione, suddividendole tra i programmi in maniera equa ed efficiente e controllando che questi le usino correttamente.

## 1.1 Componenti di un sistema di elaborazione

Le componenti di un sistema di elaborazione sono:

- **Utenti**: Persone, macchine, altri computer, ecc...
- **Programmi applicativi**: Risolvono i problemi di calcolo degli utenti
- **Sistema operativo**: Coordina e controlla l'uso delle risorse hardware
- **Hardware**: Risorse di calcolo (CPU, periferiche, memoria di massa, ...)



## 1.2 Requisiti per i sistemi operativi

Oggigiorno i computer sono ovunque: vi sono molteplici tipologie di computer utilizzati in scenari applicativi molto diversi. In quasi tutti i tipi di computer si tende ad installare un sistema operativo allo scopo di gestire l'hardware e semplificare la programmazione. Ma ogni scenario applicativo in cui viene usato un computer richiede che il sistema operativo che vi viene installato abbia caratteristiche ben determinate. Che cosa si richiede quindi ad un sistema operativo per supportare uno determinato scenario applicativo? Vediamo qualche scenario:

- **Server e Mainframe:** massimizzare le performance, rendere equa la condivisione delle risorse tra molti utenti
- **Laptop, PC e tablet:** massimizzare la facilità d'uso e la produttività della singola persona che lo usa
- **Dispositivi mobili:** Ottimizzare i consumi energetici e la connettività
- **Sistemi embedded:** funzionare senza, o con minimo, intervento umano e reagire in tempo reale agli stimoli esterni (interrupt)

## 1.3 La maledizione della generalità

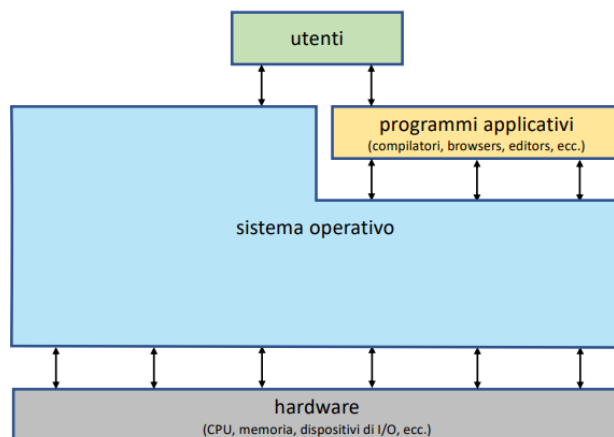
Nella storia (ed anche oggi) alcuni sistemi operativi sono stati utilizzati per scenari applicativi diversi. Ad esempio, Linux è usato oggi nei server, nei computer desktop e nei dispositivi mobili (come parte di Android). La **maledizione della generalità** afferma che, se un sistema operativo deve supportare un insieme di scenari applicativi troppo ampio, non sarà in grado di supportarne nessuno particolarmente bene. Esempio di questo si è visto con **OS/360**, il primo sistema operativo che doveva supportare una famiglia di computer diversi (la linea 360 IBM).

## 1.4 Struttura dei sistemi operativi

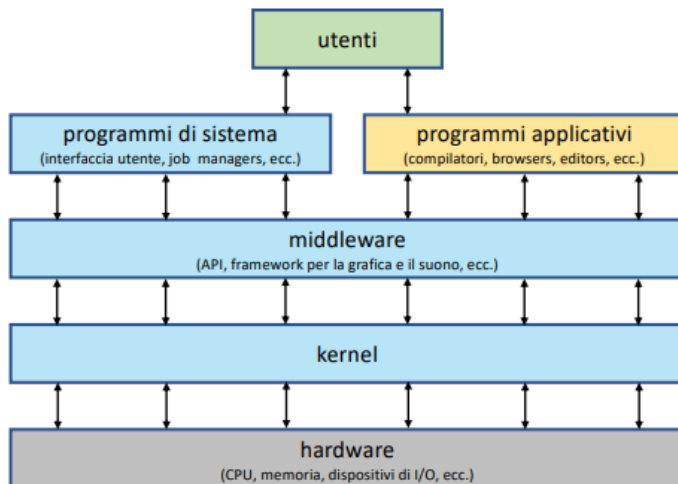
Non c'è una definizione universalmente accettata di quali programmi fanno parte di un sistema operativo. In generale però un sistema operativo almeno comprende:

- **Kernel:** Il "programma sempre presente" che si "impadronisce" dell'HW, lo gestisce, ed offre ai programmi i servizi per poterlo usare in maniera condivisa ed astratta
- **Middleware:** servizi di alto livello che astraggono ulteriormente i servizi del kernel e semplificano la programmazione di applicazioni (API, framework per grafica e per suono,...)
- **Programmi di sistema:** Non sempre in esecuzione, offrono ulteriori funzionalità di supporto e di interazione utente con il sistema (gestione di processi e jobs, UI, ...)

Alcuni sistemi operativi forniscono "out-of-the-box" anche dei **programmi applicativi** (editor, fogli di calcolo,...) ma non li considereremo come parti del sistema operativo. Data questa lista di componenti, possiamo rivisitare le **componenti di un sistema di elaborazione**:



Che visti in dettaglio diventano:

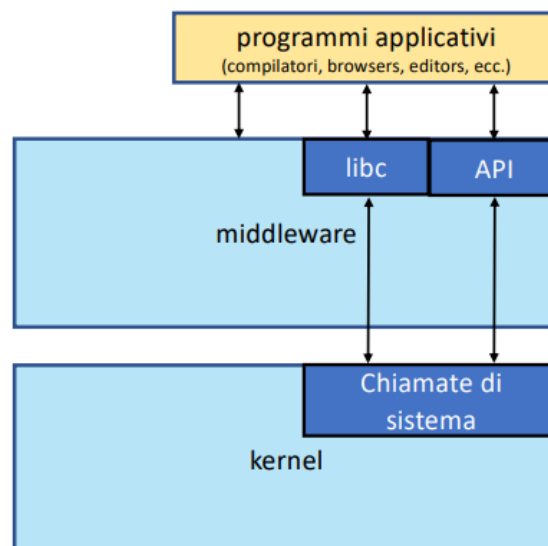


## 1.5 Servizi offerti da un sistema operativo

I principali servizi che un sistema operativo offre sono:

- **Controllo processi:** questi servizi permettono di caricare in memoria un programma, eseguirlo, identificare la sua terminazione e registrarne la condizione di terminazione (normale o errorea)
- **Gestione dei file:** questi servizi permettono di leggere, scrivere e manipolare files e directories
- **Gestione dispositivi:** questi servizi permettono ai programmi di effettuare operazioni di input/output, ad esempio leggere da/scrivere su un terminale
- **Comunicazione interprocesso:** i programmi in esecuzione possono collaborare tra di loro scambiandosi informazioni: questi servizi permettono ai programmi in esecuzione di comunicare
- **Protezione e sicurezza:** permette ai proprietari delle informazioni in un sistema multiutente o in rete di controllarne l'uso da parte di altri utenti e di difendere il sistema dagli accessi illegali
- **Allocazione delle risorse:** alloca le risorse hardware (CPU, memoria, dispositivi di I/O) ai programmi in esecuzione in maniera equa ed efficiente
- **Rilevamento errori:** gli errori possono avvenire nell'hardware o nel software (es. divisione per 0); quando avvengono il sistema operativo deve intraprendere opportune azioni (recupero, terminazione del programma o segnalazione della condizione di errore al programma)
- **Logging:** mantiene traccia di quali programmi usano quali risorse, allo scopo di contabilizzarle

## 1.6 Chiamate di sistema e Application Programming Interfaces



Il kernel offre i propri servizi ai programmi come **chiamate di sistema** (syscalls), ossia funzioni invocabili in un determinato linguaggio di programmazione (C, C++, ...). I programmi però non utilizzano direttamente le chiamate di sistema, ma delle librerie di middleware dette **Application Programming Interface** (API) implementate invocando le chiamate di sistema. Spesso le API sono fortemente legate con le librerie standard del linguaggio di implementazione (es. libc se le API sono implementate in C) al punto che anche queste diventano parte implicita dell'API. Bisogna ricordare che:

- Le API sono **esposte dal middleware**, mentre le chiamate di sistema **dal kernel**
- Le API usano le chiamate di sistema nella loro implementazione
- Le API sono standardizzate (es. POSIX, Win32), le chiamate di sistema no, quindi ogni kernel ha chiamate di sistema differenti
- Le API sono stabili, le chiamate di sistema possono variare al variare della versione del sistema operativo
- Le API offrono funzionalità più ad alto livello e più semplici da usare, le chiamate di sistema offrono funzionalità più elementari e più complesse da usare

**EXAMPLE OF STANDARD API**

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the man page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

return value	function name	parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

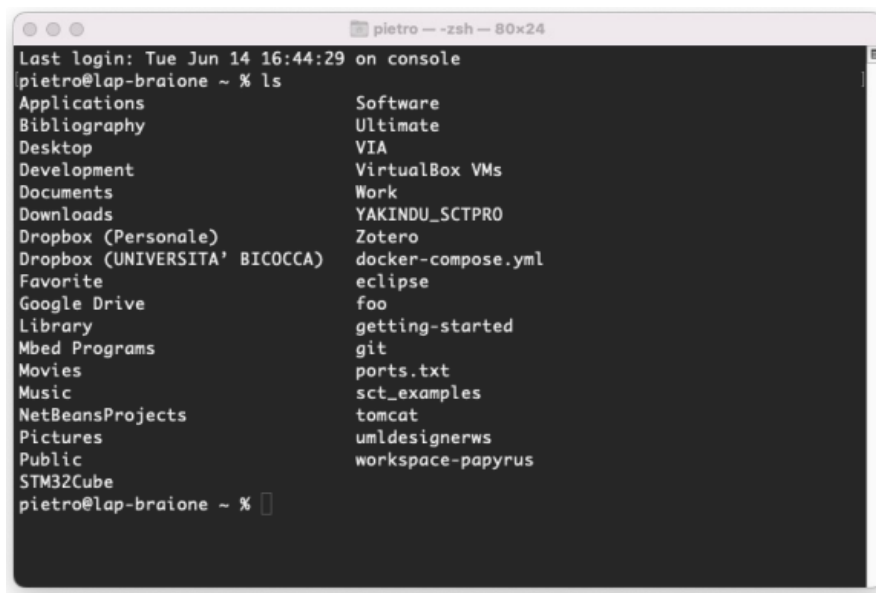
## 1.7 Programmi di sistema

La maggior parte degli utenti utilizza servizi del sistema operativo attraverso i programmi di sistema. Questi permettono agli utenti di avere un ambiente più conveniente

per l'esecuzione dei programmi, il loro sviluppo e la gestione delle risorse del sistema. Vi sono diversi tipi di programmi di sistema:

- **Interfacce utente (UI):** permette agli utenti di interagire con il sistema stesso; può essere grafica (GUI) o a riga di comando (CLI); i sistemi mobili hanno un'interfaccia touch.
- **Gestione file:** creazione, modifica e cancellazione di file e directories
- **Modifica dei file:** editor di testo, programmi per la manipolazione del contenuto dei file (Emacs)
- **Visualizzazione e modifica informazioni di stato:** data, ora, memoria disponibile, processi, utenti, ... fino a informazioni complesse su prestazione, accessi al sistema e debug. Alcuni sistemi implementano un **registry**, ossia un database delle informazioni di configurazione
- **Caricamento ed esecuzione dei programmi:** loader assoluti e rilocabili, linker e debugger
- **Ambienti di supporto alla programmazione:** compilatori, assembleri, debugger, interpreti per diversi linguaggi di programmazione
- **Comunicazione:** forniscono i meccanismi per creare connessione tra utenti, programmi e sistemi; permettono di inviare messaggi agli schermi di un altro utente, di navigare il web, di inviare messaggi di posta elettronica, di accedere remotamente ad un altro computer, di trasferire i file, ecc...
- **Servizi di background:** lanciati all'avvio, alcuni terminano, altri continuano l'esecuzione fino allo shutdown. Forniscono servizi quali verifica dello stato dei dischi, scheduling di jobs, logging, ...

### 1.7.1 Intefaccia utente: l'interpete dei comandi



```
pietro -- -zsh -- 80x24
Last login: Tue Jun 14 16:44:29 on console
pietro@lap-braione ~ % ls
Applications          Software
Bibliography          Ultimate
Desktop              VIA
Development           VirtualBox VMs
Documents             Work
Downloads             YAKINDU_SCTPRO
Dropbox (Personale)   Zotero
Dropbox (UNIVERSITA' BICOCCA) docker-compose.yml
Favorite              eclipse
Google Drive          foo
Library               getting-started
Mbed Programs         git
Movies                ports.txt
Music                 sct_examples
NetBeansProjects      tomcat
Pictures              umldesignerws
Public                workspace-papyrus
STM32Cube
pietro@lap-braione ~ %
```

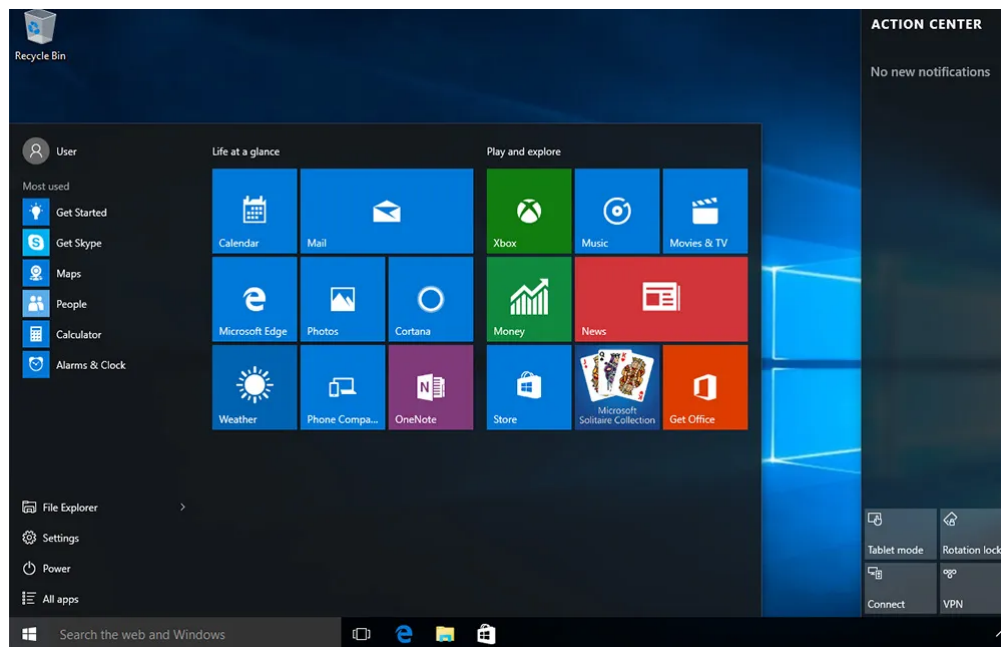


L'interprete dei comandi permette agli utenti di impartire in maniera testuale delle istruzioni al sistema operativo. In molti sistemi operativi è possibile configurare quale interprete dei comandi usare, nel qual caso è detto **shell**. Ci sono due modi per implementare un comando:

- **Built-in:** l'interprete esegue direttamente il comando (tipico dell'interprete dei comandi di Windows)
- **Come programma di sistema:** l'interprete manda in esecuzione un programma (tipico delle shell Unix e Unix-Like)

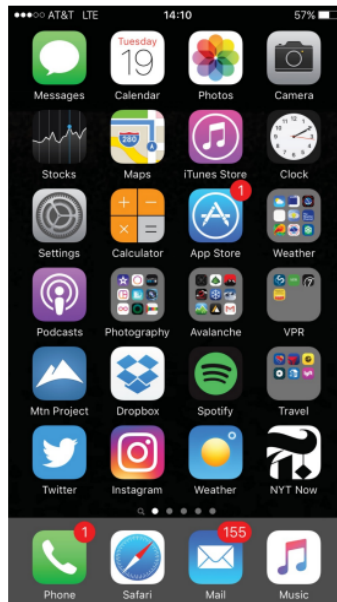
Spesso l'interprete riconosce **un vero e proprio linguaggio di programmazione** (es. Bash).

### 1.7.2 Interfaccia utente: le interfacce grafiche



Le interfacce grafiche(GUI) sono di solito basate sulla metafora della scrivania, delle icone e delle cartelle (corrispondenti alle directory). Nate dalla ricerca presso lo Xerox PARC lab negli anni 70, vennero popolarizzate dai computer Apple Macintosh negli anni 80. Su Linux le più popolari sono KDE e Gnome.

### 1.7.3 Intefaccia utente: Le interfacce touch-screen



I dispositivi mobili richiedono interfacce di nuovo tipo. Esse non prevedono nessun dispositivo di puntamento (mouse); sostituendolo con l'uso dei gesti (gestures). Inoltre esse possono offrire servizi come tastiere virtuali e comandi vocali.

## 1.8 L'implementazione dei programmi di sistema

- **Apri** *in.txt* in lettura
- Se non esiste
  - **Scrivi** un messaggio di errore su terminale
  - **Termina** il programma con codice errore
- **Apri** *out.txt* in scrittura
- Se non esiste, **crea** *out.txt*
- Loop
  - **Leggi** da *in.txt*
  - **Scrivi** su *out.txt*
- End loop
- **Chiudi** *in.txt*
- **Chiudi** *out.txt*
- **Termina** normalmente

I programmi di sistema sono implementati utilizzando le API, esattamente come i programmi applicativi. Consideriamo ad esempio il comando *cp* delle shell dei sistemi operativi Unix-like; la sua sintassi è:

*cp in.txt out.txt*

Esso copia il contenuto del file *in.txt* in un file *out.txt*. Se il file *out.txt* esiste, il contenuto precedente viene cancellato, altrimenti *out.txt* viene creato. L'immagine sopra rappresenta una possibile struttura del codice; le invocazioni delle API sono riportate in grassetto. *cp* è implementato come programma di sistema.

## 2 Processi e thread: i servizi

Un sistema operativo esegue un certo numero di programmi sullo stesso sistema di elaborazione. Il numero di programmi da eseguire può essere arbitrariamente elevato, di solito è infatti molto maggiore del numero di CPU del sistema. A tale scopo, il sistema operativo realizza e mette a disposizione un'astrazione detta **processo**. Un processo è quindi un'entità attiva astratta definita dal sistema operativo allo scopo di eseguire un programma. Per il momento, assumiamo che l'esecuzione di un processo sia sequenziale, tuttavia **rilasseremo presto questa assunzione**.

### 2.1 Programmi e processi

È fondamentale notare la differenza tra programma e processo!

- Un programma è un'entità passiva (un insieme di istruzioni, tipicamente contenuto in un file sorgente o eseguibile)
- Un processo è un'entità attiva (è un **esecutore di un programma** o un **programma in esecuzione**)

Uno stesso programma può dare origine a **diversi processi**:

- Diversi utenti eseguono lo stesso programma
- Uno stesso programma viene eseguito più volte, anche contemporaneamente, dallo stesso utente

### 2.2 Struttura di un processo