# RSO - Reti e Sistemi Operativi

Sara Angeretti

Ottobre 2023

# Indice


**1 Organizzazione del corso** **4**
- 1.1 Informazioni sul corso . . . 4
  - 1.1.1 Lezioni . . . 4
  - 1.1.2 Personale del corso . . . 5
  - 1.1.3 Modalità di svolgimento del corso . . . 5
  - 1.1.4 Il sito del corso . . . 5
  - 1.1.5 Appelli d'esame . . . 6
  - 1.1.6 Le tempistiche . . . 6
- 1.2 Contatti . . . 7
- 1.3 Obiettivi del corso . . . 8
- 1.4 Materiale e strumenti didattici . . . 9
- 1.5 Programma del corso . . . 10

**2 Sistemi operativi: struttura e servizi** **11**
- 2.1 I sistemi operativi . . . 11
  - 2.1.1 Requisiti per i sistemi operativi . . . 13
- 2.2 Struttura dei sistemi operativi . . . 15
- 2.3 Servizi dei sistemi operativi . . . 15
  - 2.3.1 Principali servizi: . . . 16
- 2.4 Chiamate di sistema e API . . . 17
- 2.5 I programmi di sistema . . . 18
  - 2.5.1 Tipologie di programmi di sistema . . . 18

**3 Introduzione alle reti** **22**
- 3.1 Cos'è Internet? . . . 22
  - 3.1.1 Visione nuts and bolts . . . 22
  - 3.1.2 Visione come infrastruttura di servizi . . . 24
- 3.2 Protocolli . . . 24
  - 3.2.1 Cos'è un protocollo? . . . 25
- 3.3 Da cosa è composta la rete? . . . 25
  - 3.3.1 Network edge . . . 26

3.3.2 Access networks . . . . . 27
3.3.3 Network core . . . . . 27

# Capitolo 1

# Organizzazione del corso

## 1.1 Informazioni sul corso

### 1.1.1 Lezioni

Crediti: 4 crediti lezione (32 ore) + 4 crediti esercitazione (40 ore).
Due turni

- Turno 1: cognomi dalla A alla L
- Turno 2: cognomi dalla M alla Z

Corso in blended e-learning:

- Lezioni frontali in presenza in aula e da remoto (a causa di ristrutturazione delle aule, tenere sempre controllato il calendario)
- Esercitazioni in e-learning asincrono
- Materiale didattico attraverso il sito del corso (su elearning.unimib.it)
  - Slides e video registrazioni delle lezioni in presenza
  - Materiale video e testuale erogato in e-learning
  - Quiz di autovalutazione
  - Materiale di approfondimento (non oggetto di esame)
  - Indispensabile l'interazione attraverso i forum con docenti, esercitatori e compagni

### 1.1.2 Personale del corso

Docenti:

- Pietro Braione: docente per la parte di sistemi operativi e responsabile del corso
- Marco Savi: docente per la parte di reti

Esercitatori:

- Jacopo Maltagliati: esercitatore per la parte di sistemi operativi
- Samuele Redaelli: esercitatore per la parte di reti

Tutor:

- Samuele Redaelli: tutor e-learning

### 1.1.3 Modalità di svolgimento del corso

Le parti di reti e di sistemi operativi si svolgono in contemporanea.
Il calendario del corso, disponibile sul sito, riporta le date delle lezioni in presenza, in remoto a causa di assenza aula, e le date in cui saranno pubblicati i materiali per l'e-learning asincrono.
Il programma del corso (anch'esso disponibile sul sito) riporta le esatte sezioni dei libri di testo da studiare per ciascun argomento del corso, e la distribuzione degli argomenti sulle due prove in itinere.

### 1.1.4 Il sito del corso

Strumento indispensabile, dal momento che il corso è in blended e-learning!
Aperte le iscrizioni spontanee (iscrivetevi il prima possibile se non siete già iscritti).
Le notizie sul corso verranno comunicate attraverso il forum avvisi.
I materiali didattici vengono distribuiti attraverso il sito.
Sono a disposizione dei forum anonimi per interagire con docenti, esercitatori e tra di voi, allo scopo di discutere gli argomenti del corso e di chiarirvi i dubbi: usateli!

### 1.1.5 Appelli d'esame

Cinque (o sei?) appelli:

- Due prove in itinere, la prima a novembre 2023 e la seconda a gennaio 2024.
- Due appelli nella sessione di gennaio/febbraio 2024.
- Due appelli nella sessione di giugno/luglio 2024.
- Un appello (o due?) nella sessione di settembre 2024.
- Si può recuperare una (una sola) prova in itinere nel secondo appello della sessione di gennaio/febbraio 2024.

Modalità d'esame:

- Questionario online svolto su computer in laboratorio.
- Le domande possono essere sia teoriche che esercizi che richiedono calcoli, a scelta multipla oppure domande aperte, in qualunque combinazione.
- Ogni prova d'esame comprende sia domande di reti che domande di sistemi operativi: non è possibile sostenere separatamente le parti di reti e di sistemi operativi!
- Regolamento dettagliato di esame disponibile sulla pagina del corso.

### 1.1.6 Le tempistiche

**Parte Sistemi Operativi**

Durata Corso (4 CFU)

- 16 ore di didattica frontale
- 10 ore verranno erogate in presenza (aula), le restanti 6 ore online (sincrono)
- Più due ulteriori lezioni in remoto asincrono
- 20 ore di esercitazioni in blended e-learning
- Video e quiz di autovalutazione caricati sulla pagina Moodle secondo calendario didattico

- Possibile qualche incontro in remoto sincrono e un incontro in presenza fuori orario (per chi è interessato)
- Previste inoltre due sessioni di Q&A prima delle prove in itinere

Orario lezioni: vedi calendario sul sito (verranno comunicate di volta in volta così come se in presenza o da remoto).
Controllare sempre il calendario sul sito per sapere se c'è lezione e quando verranno pubblicati i video/quiz per le esercitazioni!

**Parte Reti**

Durata Corso (4 CFU)

- 16 ore di didattica frontale (in presenza o da remoto a seconda dei giorni)
- Previste inoltre due sessioni di Q&A prima delle prove in itinere
- 20 ore di esercitazioni in blended e-learning
- Video e quiz di autovalutazione caricati sulla pagina Moodle secondo calendario didattico

Orario lezioni: vedi calendario sul sito (verranno comunicate di volta in volta così come se in presenza o da remoto).
Controllare sempre il calendario sul sito per sapere se c'è lezione e quando verranno pubblicati i video/quiz per le esercitazioni!

## 1.2 Contatti

**Parte Sistemi Operativi**

Prof. Pietro Braione.
Ufficio: Edificio U14 (DISCo), secondo piano, stanza 2051.
Email: pietro.braione@unimib.it

- Inserire "[RSO]" prima dell'oggetto dell'email!

Telefono: 0264487915
Orario di ricevimento: appuntamento via email.
Team:

- Jacopo Maltagliati - Esercitatore (email: j.maltagliati@campus.unimib.it)
- Samuele Redaelli - Tutor (email: samuele.redaelli@unimib.it)

**Parte Reti**

Prof. Marco Savi
Ufficio: Edificio U14 (DISCo), Secondo Piano, Stanza 2035
Email: marco.savi@unimib.it

- Inserire "[RSO]" prima dell'oggetto dell'email!

Telefono: 0264487884
Orario di ricevimento: appuntamento via email
Team:

- Samuele Redaelli - Esercitatore (email: samuele.redaelli@unimib.it)
- Samuele Redaelli - Tutor (email: samuele.redaelli@unimib.it)

## 1.3 Obiettivi del corso

**Parte Sistemi Operativi**

Acquisire le conoscenze fondamentali relativi ai sistemi operativi:

- A cosa servono i servizi operativi? Perché sono necessari?
- Che servizi offrono ai programmi e agli utenti di un sistema?
- Come sono implementati i sistemi operativi e i servizi che offrono?

Particolarmente importanti sono le esercitazioni pratiche, nelle quali imparerete ad usare i servizi di un sistema operativo moderno (Linux).

- Necessario un computer laptop
- Sono argomento di esame!

**Parte Reti**

Acquisire le conoscenze fondamentali per comprendere l'architettura e i protocolli principali delle reti di telecomunicazioni basate sullo stack TCP/IP.

- Lo stack TCP/IP è alla base della quasi totalità dei servizi di comunicazione.

Al termine del corso avrete appreso i principi fondamentali del funzionamento di Internet.

## 1.4 Materiale e strumenti didattici

**Parte Sistemi Operativi**

Libro di riferimento:

- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Sistemi Operativi - Concetti e Esempi, Decima edizione, Pearson, 2019.
- Versione in inglese: Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating Systems Concepts, 10th Edition, John Wiley and Sons, 2018.

Materiale online su Moodle.

- Slides e registrazioni video delle lezioni
- Quiz di autovalutazione
- Video delle esercitazioni

Forum tematico anonimo di discussione su Moodle (indispensabile per le esercitazioni!)

**Parte Reti**

Materiale online su Moodle

- Slides ufficiali del libro di riferimento (rivisitate, in inglese)
- Video sulla parte di esercitazioni

Libro di riferimento

- Jim Kurose, Keith Ross, Reti di calcolatori ed Internet - Un approccio top-down, Ottava edizione, Pearson, 2021
- Versione in inglese: Jim Kurose, Keith Ross, Computer Networking - A Top-Down Approach, 8th Edition, Pearson, 2021

Forum su Moodle per la parte di reti (specialmente utile per la parte di esercitazioni...)

## 1.5 Programma del corso

**Parte Sistemi Operativi**

Sistemi operativi: struttura e servizi
Servizi:

- Processi e thread: i servizi
- Gestione della memoria: i servizi
- File system: i servizi

Struttura:

- Interfaccia e struttura del kernel
- Processi e thread: la struttura
- Gestione della memoria: la struttura
- File system: la struttura

**Parte Reti**

Parti specifiche del libro (da Capitolo 1 a Capitolo 6)

- Capitolo 1: Introduzione alle reti di calcolatori e Internet
- Capitolo 2: Livello di applicazione [cenni]
- Capitolo 3: Livello di trasporto
- Capitolo 4: Livello di rete - Piano dei dati
- Capitolo 5: Livello di rete - Piano di trasporto
- Capitolo 6: Livello di collegamento e reti locali

# Capitolo 2

# Sistemi operativi: struttura e servizi

**Argomenti**

- A che servono i sistemi operativi?
- Requisiti per i sistemi operativi
- Struttura e servizi dei sistemi operativi
- Chiamate di sistema ed API
- I programmi di sistema

## 2.1 I sistemi operativi

Un S.O. è una certa quantità di software che viene dato assieme all'hardware perché quest'ultimo da solo non funziona. Per esempio quando compri un telefonino apri e ti trovi Android o iOS, mentre su un laptop abbiamo Windows, macOS e Linux.
Parliamo del modello teorico della macchina di Von Neumann. Questo modello si basa su cinque componenti fondamentali:

- Unità centrale di elaborazione (**CPU**), che si divide a sua volta in unità aritmetica e logica (ALU o unità di calcolo) e unità di controllo;
- Unità di **memoria**, intesa come memoria di lavoro o memoria principale (RAM, Random Access Memory);
- Unità di **input**, tramite la quale i dati vengono inseriti nel calcolatore per essere elaborati;

- Unità di **output**, necessaria affinché i dati elaborati possano essere restituiti all'operatore;
- **Bus**, un canale che collega tutti i componenti fra loro.

Un computer quando lo accendiamo inizia ad eseguire un programma. Se non c'è un programma da eseguire è solo un mucchio di ferraglia che si blocca e non fa niente. Perciò possiamo dire che un S.O. è il **primo** programma che viene eseguito all'accensione del computer e che permette di eseguire altri programmi. È una cosa molto diversa dalle *applicazioni*, che sono quelle che "fanno le cose utili" cit prof. Infatti la prima cosa che facciamo quando per esempio compriamo un telefono è installare WhatsApp o altre app che ci servono e rendono comomda la vita. Un S.O. di per sé non è *"utile"*, lo diventa in relazione al funzionamento delle app che ci interessano.
Un S.O. fornisce un ambiente a finestre (laptop) o icone (smartphone) che permette di eseguire le applicazioni utili: le possiamo installare, lanciare, far eseguire (anche più di una contemporaneamente), interromperne l'esecuzione...
La macchina di Von Neumann esegue un programma alla volta, ma noi di solito su un computer eseguiamo più di un'applicazione alla volta (es. WebEx per la lezione e VSCode per gli appunti). Questa cosa ci è permessa dal S.O., che ci permette ci gestire $n$ applicazioni contemporaneamente, anche più dei processori di cui dispone il mio computer. Es.: mettiamo di avere un computer con un processore a 32 core, io però posso eseguire anche centinaia di programmi contemporaneamente, non massimo 32.
Il S.O. crea un'ambiente in cui le applicazioni possono collaborare assieme. Se avessi più applicazioni che usano contemporaneamente lo stesso hardware (es. lo schermo)? Il S.O. si occupa di gestire le risorse hardware e di farle usare alle applicazioni. Il S.O. è il software *intermediario* tra le applicazioni e l'hardware.
Un'altra cosa che fa il S.O. è organizzare i nostri file in un sistema ordinato di file e cartelle (anche memorizzandoli su dispositivi secondari di memoria e storaging). Il S.O. si occupa di gestire i file e le cartelle, di crearli, cancellarli, rinominarli, spostarli...

### Cos'è un sistema opertativo?

È un insieme di programmi (software) che gestiscono gli elementi fisici di un computer (hardware).
Fornisce una piattaforma di sviluppo per le applicazioni, che permette loro

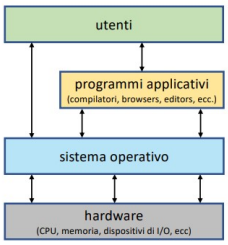
di condividere ed astrarre le risorse hardware.

Agisce da intermediario tra utenti e computer, permettendo agli utenti di controllare l'esecuzione dei programmi applicativi e l'assegnazione delle risorse hardware ad essi.
Protegge le risorse degli utenti (e dei loro programmi) dagli altri utenti (e dai loro programmi) e da eventuali attori esterni.

Un S.O. in primo luogo è 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, e di condividerle:

- Da un lato, il S.O. **astrae** le risorse hardware, presentando agli sviluppatori dei programmi applicativi una visione delle risorse hardware più facile da usare e più potente rispetto alle risorse hardware «native».
- Dall'altro, il S.O. **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.

#### Componenti di un sistema di elaborazione




Utenti: persone, macchine, altri computer...
Programmi applicativi: risolvono i problemi di calcolo degli utenti.
S.O.: coordina e controlla l'uso delle risorse hardware.
Hardware: risorse di calcolo (CPU, periferiche, memorie di massa...).

### 2.1.1 Requisiti per i sistemi operativi

#### Cosa si richiede ad un S.O.?

Oggigiorno i computer sono ovunque: vi sono molteplici tipologie di computer utilizzati in scenari applicativi diversi (i nostri laptop, i nostri smartphones, i computer detti "embedded" che non hanno lo scopo di interagire con persone ma sono "cyber-fisici", cioè che fanno parte di servizi ad es. quelli che

controllano le automobili ad es. il sistema ABS che fa in modo che la ruota non si blocchi e quindi non slitti quando noi inchiodiamo e freniamo a fondo, etc. . . ).
In quasi tutti i tipi di computer si tende ad installare un S.O. allo scopo di gestire l'hardware e semplificare la programmazione.
Ma ogni scenario applicativo in cui viene usato un computer richiede che il S.O. che vi viene installato abbia caratteristiche ben determinate (es. laptop molto diverso dai sistemi embedded). Che cosa si richiede ad un S.O. per supportare un certo scenario applicativo?
A seconda che sia:

**Server, mainframe:** massimizzare la performance, rendere equa la condivisione delle risorse tra molti utenti

**Laptop, PC, 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)

### 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 S.O. deve supportare un insieme di scenari applicativi troppo ampio, non sarà in grado di supportare nessuno di tali scenari particolarmente bene. Praticamente io quando cerco di dare più di un esame per sessione.
Questo si è visto con OS/360, il primo S.O. che doveva supportare una famiglia di computer diversi (la linea 360 dell'IBM).
Quella **maledizione della generalità** non avviene sempre e necessariamente, è un potenziale rischio; può essere tuttavia aggirata, ma non ho capito come.

## 2.2 Struttura dei sistemi operativi

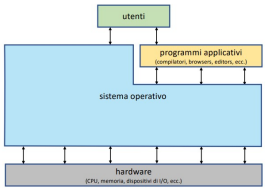
Non c'è una definizione universalmente accettata di quali programmi fanno parte di un S.O..
In generale però un S.O. comprende almeno:

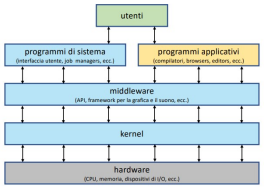
**Kernel:** il "programma sempre presente", che si "impadronisce" dell'hardware, lo gestisce, ed offre ai programmi i servizi per poterlo usare in maniera condivisa ed astratta.

**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 jobs e processi, interfaccia utente...)

Alcuni sistemi operativi forniscono anche dei programmi applicativi (editor, word processor, fogli di calcolo...), ma non li considereremo parti del S.O. stesso.




## 2.3 Servizi dei sistemi operativi

Un S.O. offre un certo numero di **servizi**:

**Per i programmi applicativi:** perché possano eseguire sul sistema di elaborazione usando le risorse astratte esposte dal S.O..

**Per gli utenti:** per gestire l'esecuzione dei programmi e stabilire a quali risorse hardware i programmi (e gli altri utenti) hanno diritto.

Per garantire che il sistema di elaborazione funzioni in maniera efficiente.

Gli utenti però interagiscono con il S.O. attraverso i programmi di sistema...
...i quali utilizzano gli stessi servizi dei programmi applicativi...
Quindi, in definitiva, il S.O. ha bisogno di esporre i suoi servizi esclusivamente ai programmi (applicativi o di sistema).

### 2.3.1 Principali servizi:

**Controllo processi:** questi servizi permettono di caricare in memoria un programma, eseguirlo, identificare la sua terminazione e registrarne la condizione di terminazione (normale o erronea).

**Gestione file:** questi servizi permettono di leggere, scrivere, e manipolare files e directory

**Gestione dispositivi:** questi servizi permettono ai programmi di effettuare operazioni di input/output, ad esempio leggere da/scrivere su un terminale.

**Comunicazione tra processi:** 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.

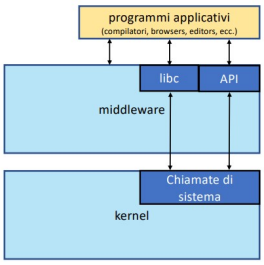
**Allocazione delle risorse:** alloca le risorse hardware (CPU, memoria, dispositivi di I/O) ai programmi in esecuzione in maniera equa ed efficiente.

**Rilevamento errori:** gli errori possono avvenire nell'hardware o nel software (es. divisione per zero); quando avvengono il S.O. 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, ovvero fare sì che un programma o un utente non usi troppe risorse sottraendole ad altri programmi o utenti.

## 2.4 Chiamate di sistema e API

I programmi applicativi accedono a questi servizi che abbiamo appena elencato tramite chiamate di sistema o API.
Un programma applicativo che vuole accedere ai servizi offerti dal kernel non chiama direttamente dal kernel ma passa da un Middleware.
Il kernel offre i propri servizi ai programmi come chiamate di sistema, ossia di 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 Program Interface (API) implementate invocando le chiamate di sistema.
Questo perché le chiamate di sistema dipendono dal linguaggio mentre le API sono più standardizzate (es. le POSIX, tipicamente in C). 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 una parte implicita dell'API.

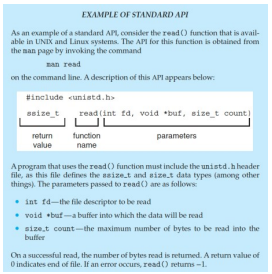



### Differenza tra chiamate di sistema ed API

Le API sono esposte dal middleware, le chiamate di sistema dal kernel.
Le API usano le chiamate di sistema nella loro implementazione.
Le API sono standardizzate (esempio: standard POSIX e Win32), le chiamate di sistema no (ogni kernel ha le sue).
Le API sono stabili, le chiamate di sistema possono variare al variare della versione del S.O..
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.

**Esempio confronto API Win32 e POSIX**


**Esempio API POSIX**



*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.

## 2.5 I programmi di sistema

La maggior parte degli utenti utilizza i servizi del S.O. 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.

### 2.5.1 Tipologie di programmi di sistema

**Interfaccia 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 file e directory.

**Modifica dei file:** editor di testo, programmi per la manipolazione del contenuto dei file.

**Visualizzazione e modifica informazioni di stato:** data, ora, memoria disponibile, processi, utenti... fino informazioni complesse su prestazioni, 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, debugger.

**Ambienti di supporto alla programmazione:** compilatori, assemblatori, debugger, interpreti per diversi linguaggi di programmazione.

**Comunicazione:** forniscono i meccanismi per creare connessioni 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 file...

**Servizi in 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...

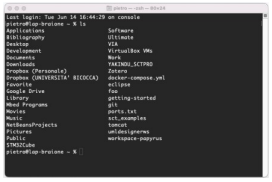
### Interfaccia utente: l'interprete dei comandi

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

**Built-in:** l'interprete esegue direttamente il comando (tipico nell'interprete di comandi di Windows)

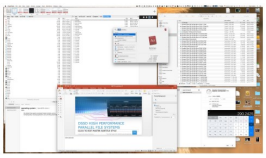
**Come programma di sistema:** l'interprete manda in esecuzione il programma (tipico delle shell Unix e Unix-like)

Spesso riconosce un vero e proprio linguaggio di programmazione con variabili, condizionali, cicli...



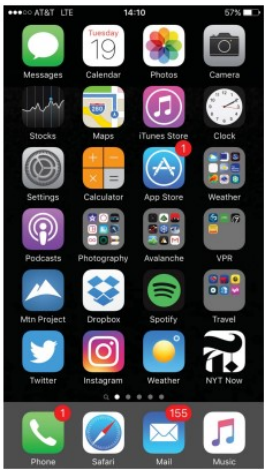

### Interfaccia utente: le interfacce grafiche

L'interfaccia grafica (GUI) è di solito basata sulla metafora della scrivania, delle icone e delle cartelle (corrispondenti alle directory).
Nate dalla ricerca presso lo Xerox PARC lab negli anni 70, popolarizzate dal computer Apple Macintosh negli anni 80.
Su Linux le più popolari sono Gnome e KDE.



### Interfaccia utente: le interfacce touch-screen

I dispositivi mobili richiedono interfacce di nuovo tipo.
Nessun dispositivo di puntamento (mouse)
Uso dei gesti (gestures)
Tastiere virtuali
Comandi vocali

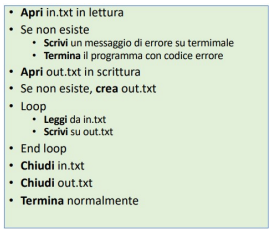



### L'implementazione dei programmi di sistema

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:

- cp in.txt out.txt
- 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

È implementato come programma di sistema.
Una possibile struttura del codice è riportata sulla destra (le invocazione delle API sono riportate in grassetto).


• **Apri** in.txt in lettura
• Se non esiste
  • **Scrivi** un messaggio di errore su termimale
  • **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

# Capitolo 3

# Introduzione alle reti

## 3.1 Cos'è Internet?

Due prospettive:

- Visione degli ingranaggi (dadi e bulloni), delle componenti della rete.
- Visione della rete come un'infrastruttura che fornisce servizi.

Nell'immagine che schematizza, sui bordi della rete troviamo un grandissimo numero di devices (cell, laptops, ...) che eseguono *applicazioni di rete* che richiedono uno scambio di dati e che la rete si occupa di collegare. Alcuni nodi di questa applicazione eseguono parte di un'applicazione e altri nodi interconnessi eseguono altre parti di applicazioni (sistemi distribuiti).

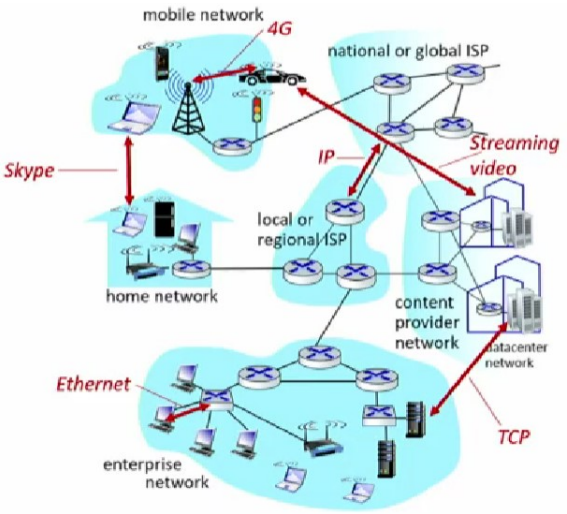
### 3.1.1 Visione nuts and bolts

**hosts** , sono dispositivi *end system* ovvero terminali.

**packet switches** o *commutatori di pacchetto*, che sono *routers* e *switches*, dispositivi che si occupano di trasferire pacchetti (unità) di informazione.
I pacchetti sono unità in cui l'informazione viene scomposta ed etichettata per essere trasferita da un dispositivo all'altro.

**Communication links** o *collegamenti di comunicazione*, sono i canali che collegano i nodi della rete. Li disegnamo come righe che uniscono i vari nodi. possiamo crearli usando diversi mezzi trasmissivi, filo di rame, onde radio, fibra ottica, ... Ognuno di questi link è definito da una **banda**, la quantità di informazione che il link può trasferire in un secondo.

**Networks** , Internet è una rete di reti, ovvero una rete di dispositivi che sono collegati tra loro e che a loro volta sono reti di dispositivi collegati tra loro. Quindi tutta la rete è una connessione globale di reti locali (es. la rete dell'ufficio, la rete domestica...)




Sempre più dispositivi oggi giorno si sono evoluti fino ad adattarsi all'uso di Internet, come ad esempio le auto, i frigoriferi, le lavatrici...


Quindi anche Internet deve evolversi e adattarsi per poter gestire un numero sempre maggiore di dispositivi con esigenze sempre più diverse.

### Internet come rete di reti

Abbiamo detto che Internet è viibile appunto come una rete di reti, un *insieme interconnesso* di **ISPs** (Internet Service Providers), le organizzazioni che gestiscono la rete. A volte ISPs viene usato anche come sinonimo della rete vera e propria.
Gli ISPs comunicano fra loro tramite l'uso di **protocolli**. I protocolli nella

rete sono ovunque, controllano il modo in cui i messaggi vengono inviati e ricevuti tra i dispositivi. Alcuni esempi sono HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, Ethernet...
Questi protocolli si basano su **standards**.
Un lavoro fondamentale a riguardo è svolto da **enti di standardizzazione** (il più famoso è IETF, *Internet Engineering Task Force*) che stilano documenti (RFC, *Request for Comments*) che spiegano come i protocolli devono essere implementati e come i dispositivi che implementano questi protocolli devono comportarsi.
Gli standard sono fondamentali per la comunicazione univoca tra i dispositivi, senza di essi non ci sarebbe interoperabilità tra i dispositivi.

### 3.1.2 Visione come infrastruttura di servizi

Internet può essere visto come un'**infrastruttura** che fornisce **servizi** alle applicazioni distribuite.
È una vista importante perché la rete sottostante è fondamentale per quando dobbiamo per esempio programmare delle applicazioni o dei servizi: da questo pov è molto importante il concetto di socket, un'interfaccia che ci permette di interfacciarci alla rete senza necessariamente sapere come la rete sotto funziona.

## 3.2 Protocolli

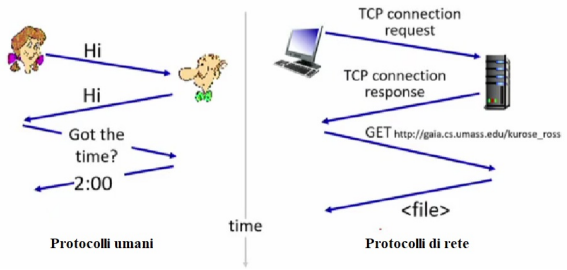
**Protocolli umani** I protocolli umani sono le regole che seguono gli esseri umani quando comunicano tra loro.
Esempio: incontro una persona, scambio di convenevoli, poi chiedo: "Che ore sono?" e in base alla risposta ho delle reazioni diverse.
Altro esempio: "Ho una domanda" durante una lezione, così che per non interrompere il professore, lui possa finire il suo discorso e poi rispondere alla domanda.
Altro esempio: un giro di tavolo per fare le presentazioni, seguo delle convenzioni di discorso.

**Protocolli di rete** Emulano il funzionamento dei protocolli umani.
Tutte le comunicazioni di rete sono gestite da protocolli.

**DEFINIZIONE**

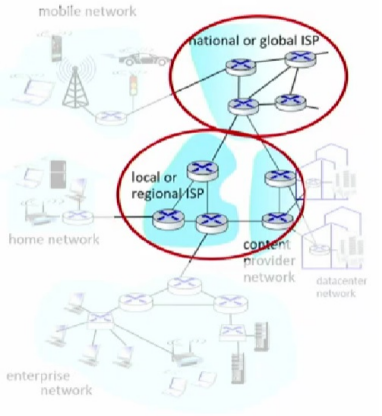
I protocolli di rete definiscono le regole per i messaggi inviati in rete. In particolare definiscono il formato dei messaggi, l'ordine in cui vengono inviati e ricevuti tra le entità di rete (host, commutatori, ...) e le azioni da intraprendere una volta che questi messaggi vengono inviati e ricevuti.

### 3.2.1 Cos'è un protocollo?




## 3.3 Da cosa è composta la rete?

Andiamo più nel dettaglio.




#### Network edge

Primo segmento della struttura generale della rete. Ci troviamo sui bordi della rete (quindi c'è un po' una diatriba sul fatto che appartengano o meno alla rete, in ogni caso sono molto importanti), dove si trovano gli **hosts** (client, server). Sono intesi in senso un po' lato, ovvero:

- I *client* sono intesi come dispositivi con una *bassa* capacità computazionale.
- I *server* sono intesi come dispositivi con una *alta* capacità computazionale.
  Infatti di solito si trovano nei data center.

#### Access networks

Addentrandoci ancora più nella rete, abbiamo le reti di accesso. Tipicamente hanno dei collegamenti di comunicazione che possono essere *wired* (cavi, stoppini, fibre ...) e *wireless* (4G, onde radio...). Sono molto importanti perché accolgono il traffico dagli utenti e lo portano verso la rete. O viceversa accolgono il traffico di dati da passare al client. Per accedere alla rete bisogna acquistare un accesso alla rete per mezzo di un provider, un ISP.
Sono importanti perché evolvono molto rapidamente nel tempo, sostengono sempre più il cambiamento e l'evoluzione del traffico generato dagli utenti a loro volta in costante evoluzione.
Non parleremo delle reti di accesso.

#### Network core

La rete di core è una rete con una maglia di dispositivi solitamente molto performanti (quindi in grado di gestire una gran quantità di informazione) e che si trovano al centro della rete. Permettono di implementare quella rete di reti di cui abbiamo parlato prima.
Per mezzo delle reti di core garantisco che ogni utente che si trova in rete possa comunicare con ogni altro utente che si trova nella stessa rete. Ovviamente non è sempre concretamente possibile.
Internet è una rete connessa: da un nodo posso raggiungere un qualsiasi altro nodo. Questo è quello di cui si occupano le reti core: permettono di mettere in comunicazione reti più al limitare della rete di altre.

### 3.3.1 Network edge

L'host, abbiamo detto, ha il compito di inviare pacchetti di dati, prende un messaggio applicativo che deve inviare e lo scompone in pacchetti di lunghezza pari ad $L$ (per semplicità ora li consideriamo tutti della stessa lunghezza, ma di solito hanno dimensioni di lunghezza variabile). Questi pacchetti vengono inviati dall'host in rete dalle reti di accesso che forniscono l'accesso ad un rate trasmissivo (banda, capacità...) pari a $R$ (è in bit/s). Questo vuol dire che l'host può inviare pacchetti di lunghezza $L$ a velocità $R$.

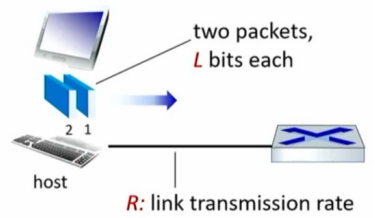
*Ovviamente più è alta la banda, più è alta la velocità di trasmissione.*

Ma come si calcola il **ritardo di trasmissione**?

**DEFINIZIONE**

Il *ritardo di trasmissione* è il ritardo che intercorre tra l'invio del primo bit di un pacchetto di $L$ bit alla ricezione dall'altro lato del link dell'ultimo bit dello stesso pacchetto.

Essendo $L$ la lunghezza del pacchetto e $R$ la velocità di trasmissione, il *ritardo di trasmissione* sarà pari a $L/R$.




### 3.3.2 Access networks

### 3.3.3 Network core