Socket? Questi sconosciuti!

Dalla teoria alla pratica: Reti Informatiche,
Socket, Multiplexing ed esempi con ...

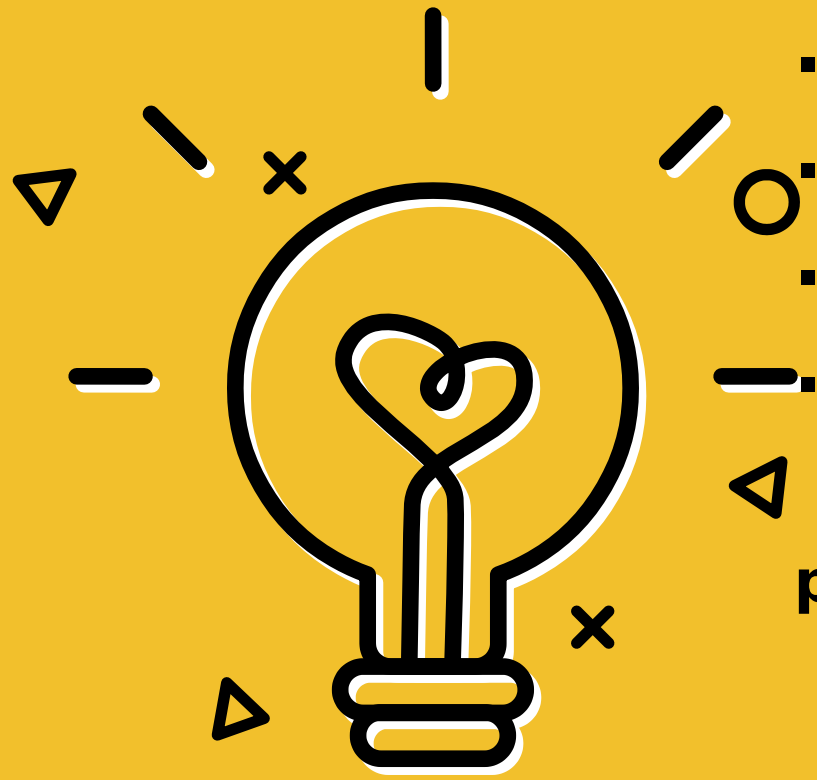
Linux!!

Luca Capisani

Cosa vedremo

- La rete di calcolatori
- Il concetto di pacchetto e di payload
- Cos'è un socket?
- Ciclo di vita (server e client)
- Demo 1: Un client ed un Echo server TCP
- Demo 2: Una simpatica chat: Multiplexing!
- Quali sono le ultime novità?
- Strumenti di sistma e debug su Linux!

Domande?



.....

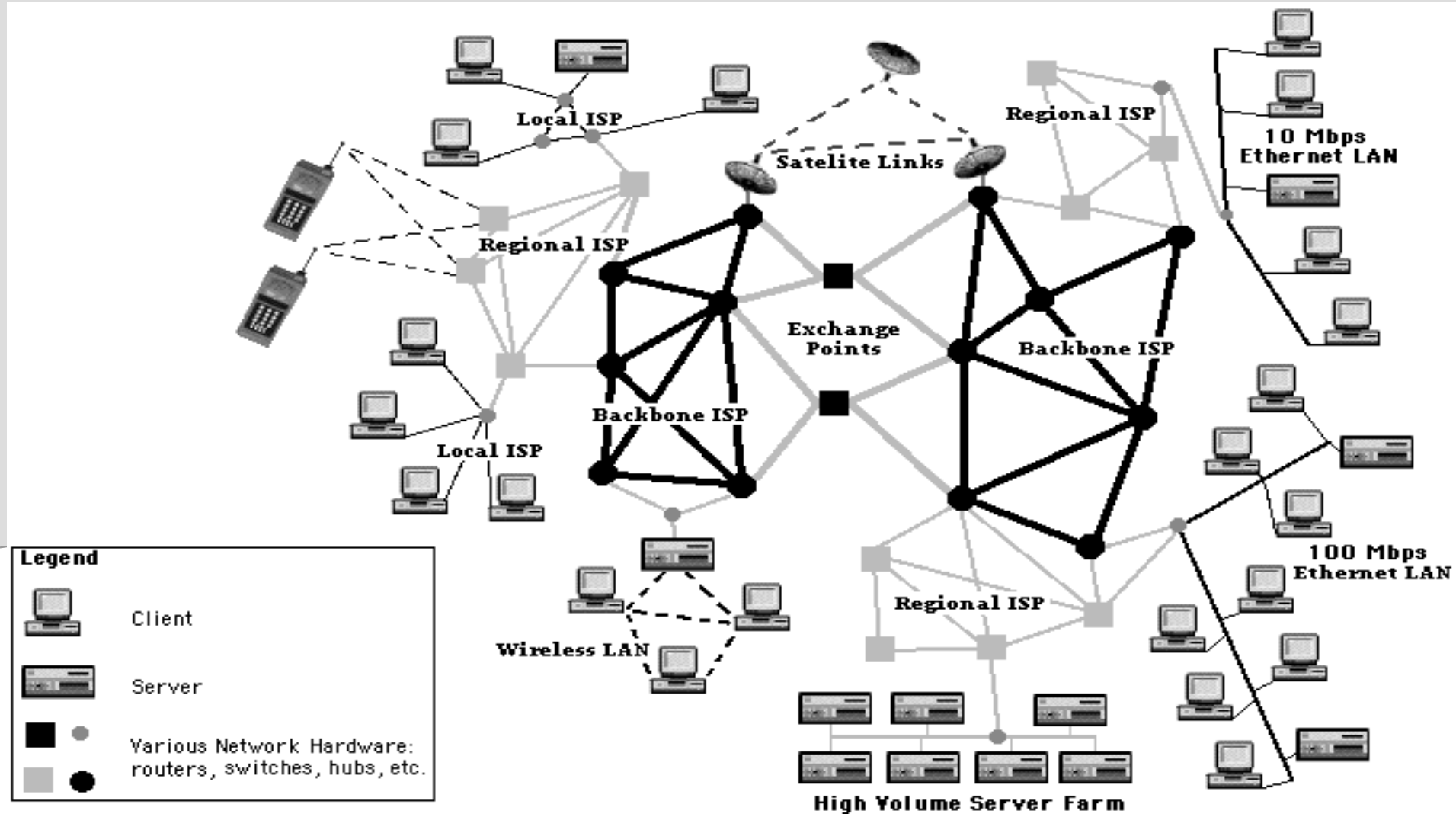
..... **se ci sono dubbi interrompete!**

.....

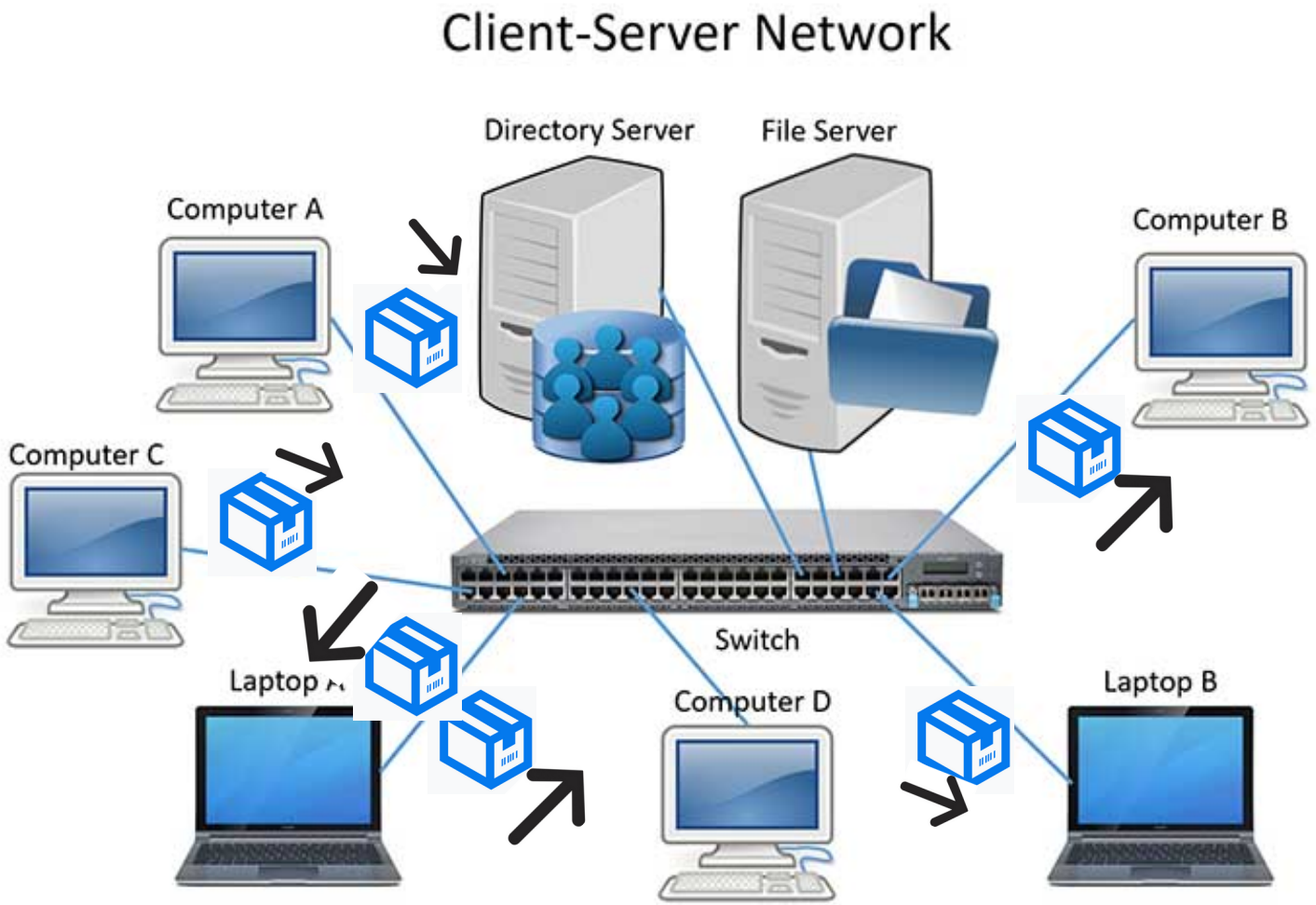
.....

per questioni più difficili ne parliamo dopo!

Ecco a voi una rete informatica!



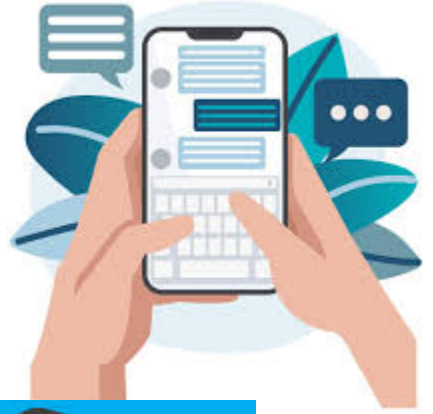
Semplifichiamo un po' :-)



Perchè parliamo di rete?

- La rete è un sistema distribuito che offre ai suoi ***host*** la ***possibilità*** di ***scambiare messaggi***!

  - Ciascun host ha al proprio interno ***molti task*** che necessitano di scambiare messaggi con altri *task*, sia ***locali*** che ***remoti***
  - Le procedure di scambio dei messaggi sono *complesse* => i ***protocolli di comunicazione*** all'interno del sistema operativo si occupano di attuarle!
    - **LA RETE È CONOSCIUTA DA TUTTI, MA NON È L'UNICO MEZZO PER SCAMBIARE MESSAGGI TRA TASK…**



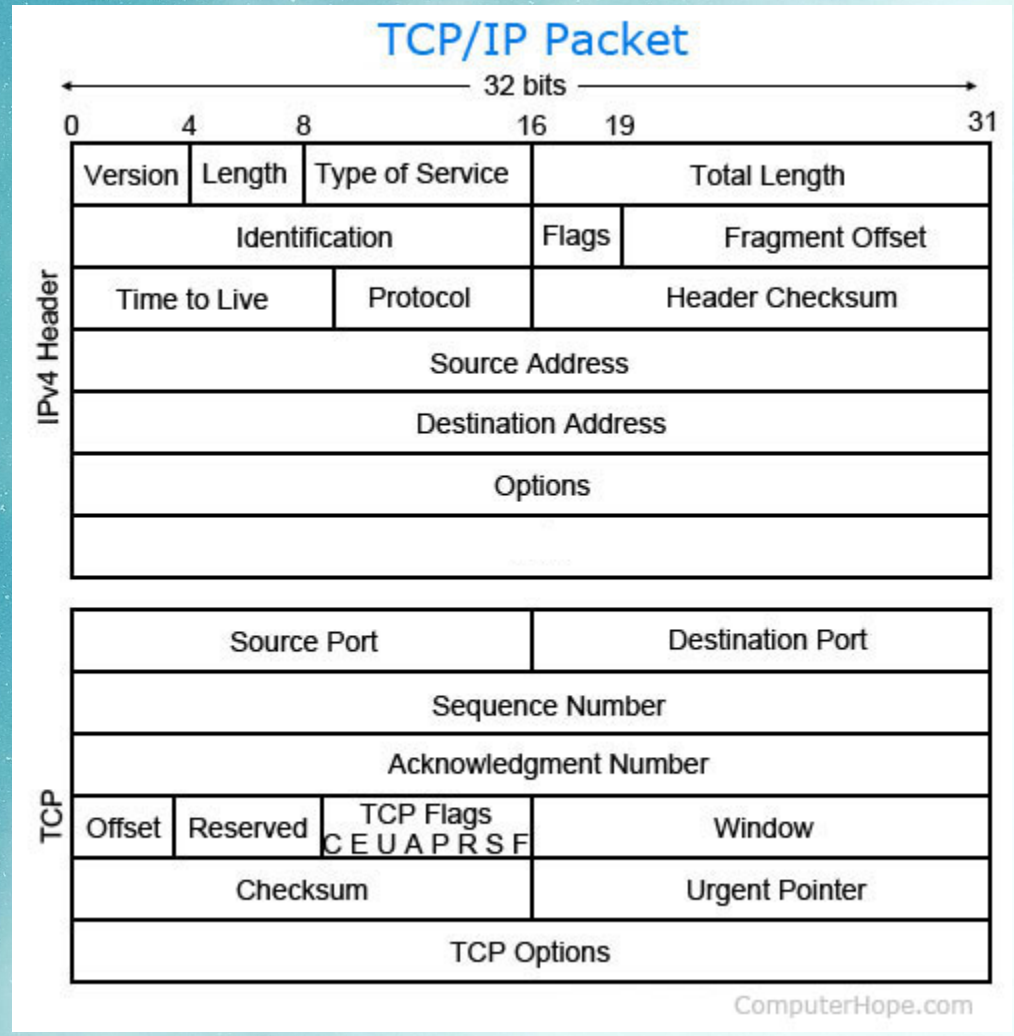
pacchetto “dati”

Contiene un sacco di informazioni sconosciute..

servono per **garantire che arrivi a destinazione**

sano e salvo assieme ai suoi *amici* :-D

ma... il messaggio da trasmettere dov’è??

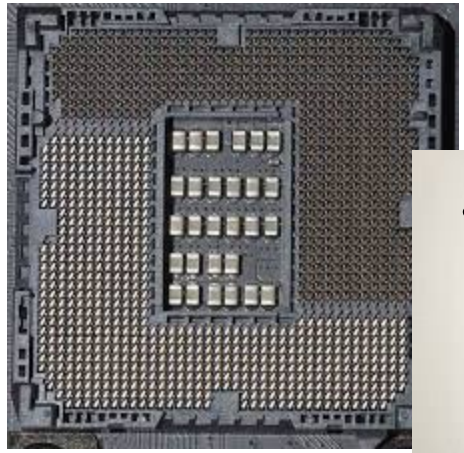


Cos'è un socket

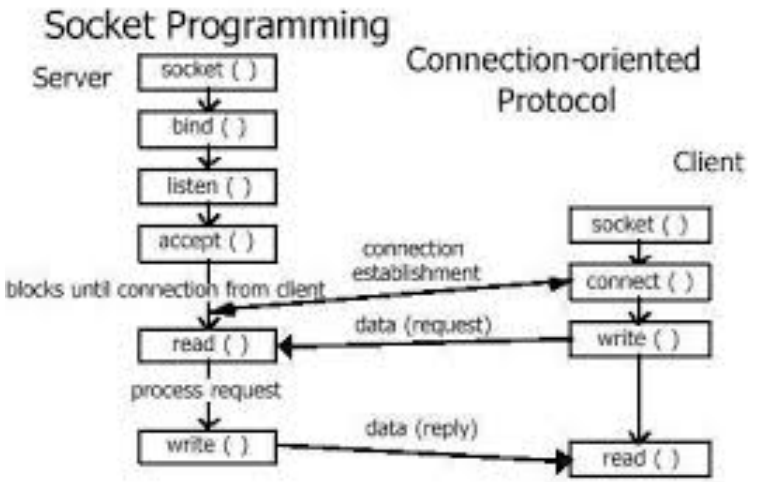
E' un *meccanismo* che permette di *interfacciarci* con un sistema ***mediante regole precise ed universali...***

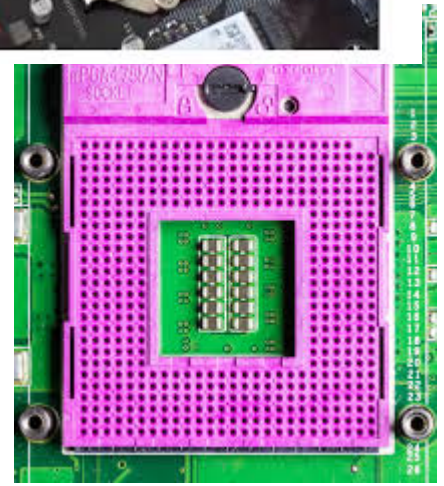
<u>Non è necessario conoscere il funzionamento del sistema</u> stesso, ci basta conoscere le **regole di interfacciamento**

Tutti i sistemi operativi moderni consentono lo scambio di informazioni mediante socket ‼️



LGA775
LGA1700





Cosa possiamo fare con Linux

SISTEMA OPERATIVO
(Kernel - librerie)

SOCKET LOCALI (AF_UNIX)

Scambio messaggi tra task LOCALI

non utilizza i servizi di rete

SOCKET DI RETE (AF_INET)

Scambio messaggi tra task LOCALI o REMOTI

utilizza i servizi di rete

Interfaccia: standard BSD sockets (POSIX)

Cosa possiamo fare con Linux -2

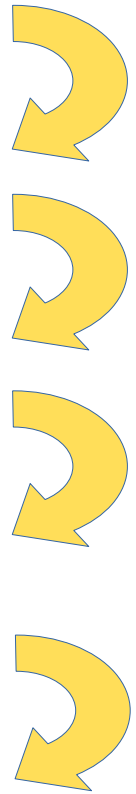
Creare semplici applicazioni che comunicano tra di loro
ANCHE IN REMOTO
ad es. macchine/sistemi operativi molto diversi tra loro

Eseguire test e debug di applicativi di terze parti

Interagire con applicativi di terze parti

Automatizzare task di scambio ed elaborazione dati

Utilizzare i potentissimi strumenti di diagnosi
... e molto altro!



NETSTAT

WIRESHARK

TCPDUMP

STRACE

SS

curl, wget nc,
telnet....

Schema concettuale di interazione

Applicazione A (user space)

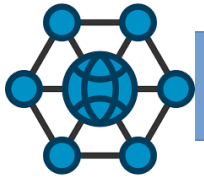
send()/recv()

Librerie OS / C / JAVA / PYTHON ecc

sys call

Kernel Linux (TCP/IP stack)

NIC



Network

NIC

Kernel Linux (TCP/IP stack)

sys call

send()/recv()

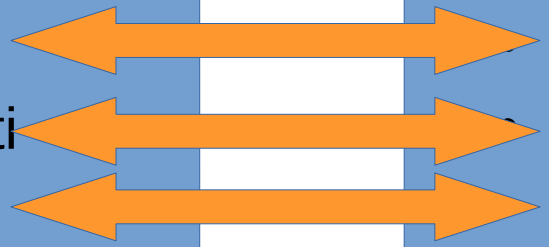
Librerie OS / C / JAVA / PYTHON ecc

Applicazione B (user space)

Socket TCP: ciclo di vita!!

- **SERVER**

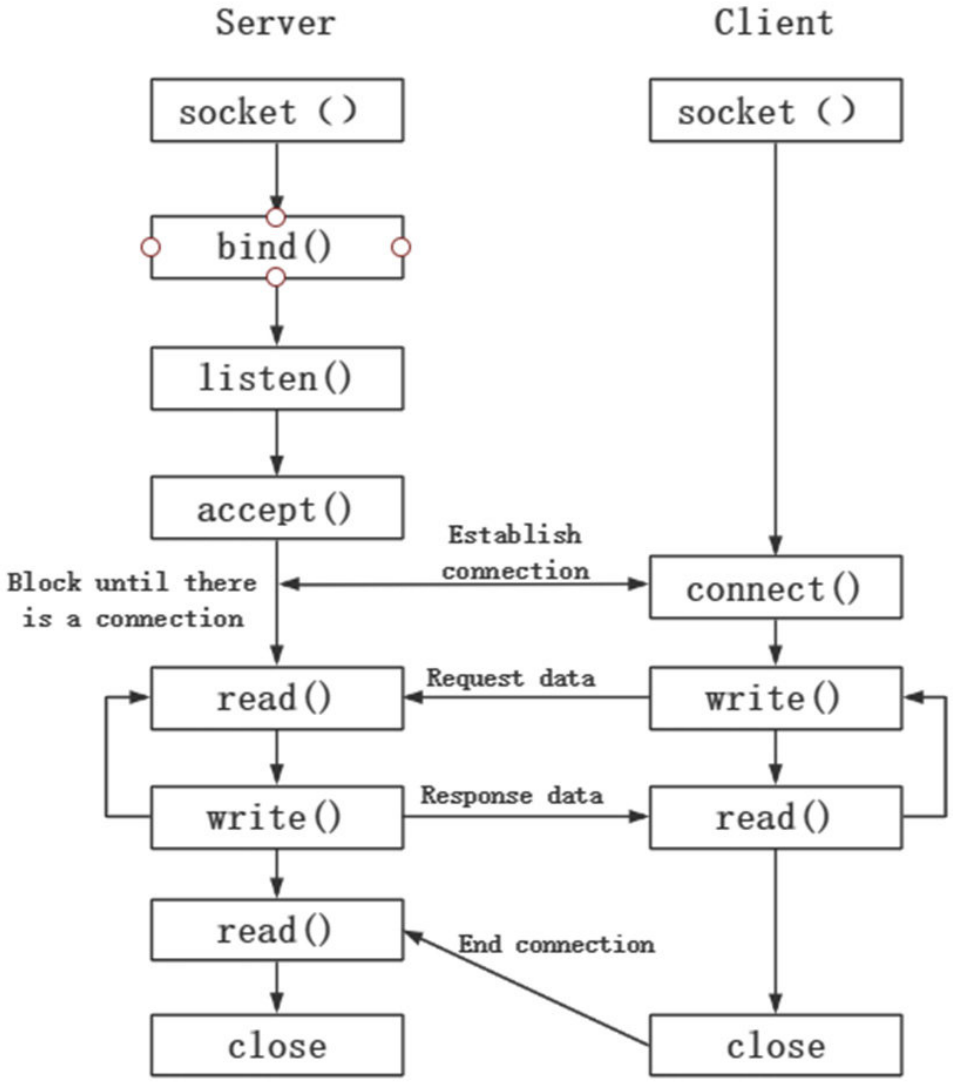
- socket() → crea il file descriptor
- bind() → associa indirizzo/porta
- listen() → mette in ascolto
- accept() → accetta un client
- recv()/send() → scambia dati
- close() → singolo client
- close() → quando avviene? :-)



- **CLIENT**

-
- socket()
- connect() → apre la connessione
- send()/recv()
- close() → da non dimenticare

Socket TCP: ciclo di vita!!



Echo client TCP in C (semplificato)

```
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

int main() {
    int sock; struct sockaddr_in server_addr; char buffer[1024]; ssize_t n;

    sock = socket(AF_INET, SOCK_STREAM, 0);

    server_addr.sin_family = AF_INET; server_addr.sin_port = htons(8080);
    inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

    connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr));

    strcpy(buffer, “ciaoooooo”);

    send(sock, buffer, strlen(buffer), 0);
    n = recv(sock, buffer, sizeof(buffer) - 1, 0);
    if (n > 0) { buffer[n] = '\0'; printf("Risposta dal server: %s\n", buffer);}
    else if (n == 0) printf("Connessione chiusa dal server.\n");

    close(sock);
}
```

Echo server TCP in C (semplificato)

```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include ...
#include <sys/socket.h>

int main(){
  int s = socket(AF_INET, SOCK_STREAM, 0);
  struct sockaddr_in addr = {0};
  addr.sin_family = AF_INET;
  addr.sin_port = htons(8080);
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  bind(s, (struct sockaddr*)&addr, sizeof(addr));
  listen(s, 5);
  printf("Listening on 127.0.0.1:8080\n");
  int c = accept(s, NULL, NULL);
  char buf[1024];
  ssize_t n = recv(c, buf, sizeof(buf), 0);
  send(c, buf, n, 0);
  close(c);
  close(s);
}
```

SI TRATTA DI UN SERVER MOLTO LIMITATO

PERCHE'??

Compilazione e test rapido

Compiliamo, eseguiamo il nostro **server**!

```
$ gcc -Wall server.c -o server
$ ./server
```

Compiliamo, eseguiamo il nostro **client**!

```
$ gcc -Wall client.c -o client
$ ./client
```

Eseguiamo un test con netcat!!

```
$ nc localhost 8080
```

Sotto il cofano con strace

```
$ strace -e trace=network ./server
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(8080), ...}, 16) = 0
listen(3, 5) = 0
accept(3, {sa_family=AF_INET, sin_port=htons(49834), ...},[16]) = 4
recv(4, "hello", 5, 0) = 5
send(4, "hello", 5, 0) = 5
```

Perchè conoscere i socket?

Meccanismo standard di programmazione
per lo scambio di informazioni

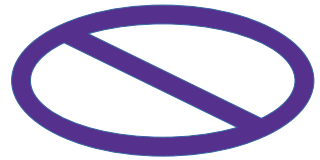
Presenti in tutti i principali sistemi operativi:
le funzioni principali sono quasi del tutto equivalenti

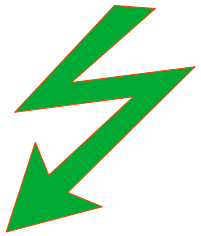
La programmazione di rete, soprattutto in un contesto
di concorrenza/multiplexing, non è semplice

L’errata implementazione delle logiche di comunicazione è
spesso la causa di PROBLEMI

Tante le problematiche *non banali* da affrontare:
connessioni instabili * memory leak * buffer overflow...

Nota importante

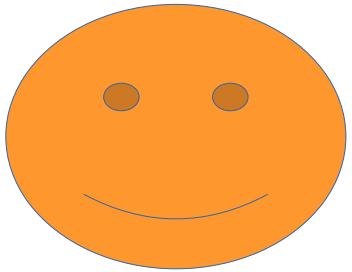




Le connessioni DEVONO essere considerate INSTABILI

Non ha senso scrivere un software di comunicazione che non tenga conto delle comuni casistiche di interruzione della comunicazione

Streaming audio-video, sw videoconferenza, websocket, applicazioni remote mostrano spesso malfunzionamenti su connessioni instabili ‼️



Perché multiplexing?

Un server reale gestisce più client simultanei ==> **complessità MOLTO maggiore**

- **select**(), poll(), epoll, **thread**, async I/O
- **Attendere** su più descrittori **senza bloccare**
- Massimizzare l'**efficienza** del software
- Mantenere semplice la **gestione della concorrenza** delle richieste
- Conciliare molte **attività asincrone** di vario tipo **simultanee**

Multiplexing => Multitasking?

- (implica il)

Mini chat con SELECT

```
import socket, select
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind(("127.0.0.1", 8081)); server.listen()
sockets = {server}
while True:
    r, _, _ = select.select(list(sockets), [], [])
    for s in r:
        if s is server:
            c, _ = server.accept(); sockets.add(c)
        else:
            data = s.recv(1024)
            if not data: sockets.remove(s); s.close(); continue
            for t in sockets:
                if t not in (server, s): t.sendall(data)
```

SELECT: quali vantaggi?

- L’implementazione di un SERVER quasi sempre richiede la possibilità di attuare il MULTIPLEXING:
  - Gestire un task dedicato, o addirittura un processo dedicato per ciascuna richiesta proveniente da client è estremamente inefficiente
  - Spesso le attività di richiesta client si concludono in pochi millisecondi, sono molto frequenti nuove richieste. Quasi sempre il server non può imporre al client un profilo temporale delle connessioni
    - ==> CONTEXT SWITCH da evitare!

Dare un'occhiata ai pacchetti (tcpdump o wireshark? :-)

```
$ sudo tcpdump -i lo port 8080 -n
listening on lo, link-type EN10MB (Ethernet)
12:00:01 IP 127.0.0.1.49834 > 127.0.0.1.8080: Flags [S], seq 0, win 65535,
12:00:01 IP 127.0.0.1.8080 > 127.0.0.1.49834: Flags [S.], ack 1, ...
...
```

Strumenti e debug

```
$ ss -tuln
```

Porte e stati

```
$ netstat -anp
```

Porte e stati

```
$ strace -e trace=network <cmd>
```

Syscall di rete

```
$ wireshark
```

```
$ tcpdump -ni lo
```

Sniffing pacchetti

```
$ nc
```

Generazione traffico

Efficientare la Select?

`select()`

`poll()`

`epoll()`

LUGMan