

Reti informatiche cod. 545II [9 CFU]

Corso di Laurea in Ingegneria Informatica

Laboratorio e Programmazione di Rete
A. A. 2021/2022

Francesco Pistolesi, PhD
Dipartimento di Ingegneria dell'Informazione
francesco.pistolesi@unipi.it

Cosa impariamo oggi?

- I socket
- Creazione e configurazione di un socket
- Primitive *socket*, *bind* e *listen*

Programmazione distribuita in C



Cooperazione tra processi

Due processi possono essere **indipendenti** o **cooperanti**, su una o più macchine



In questo caso si parla di
sistemi distribuiti

Modalità di cooperazione

Sincronizzazione



Per esempio, tramite **semafori**

Comunicazione



Memoria condivisa



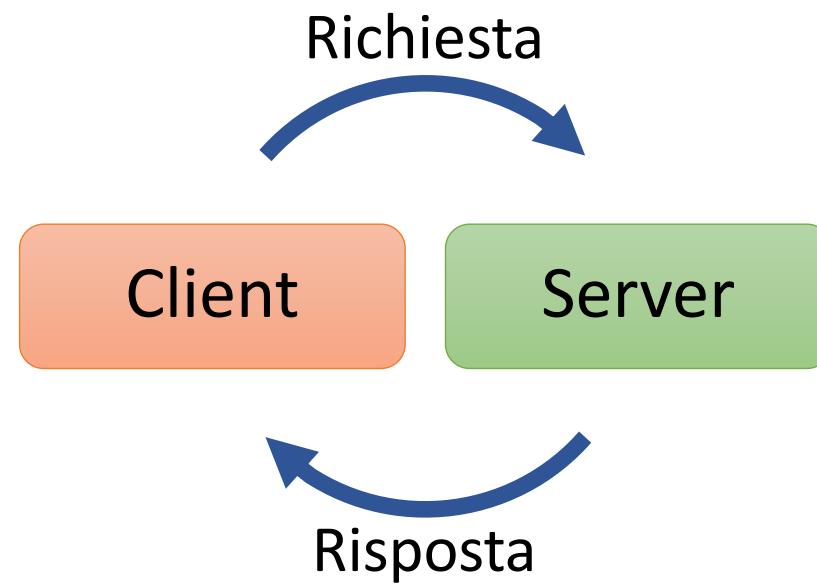
Chiamata a procedura remota



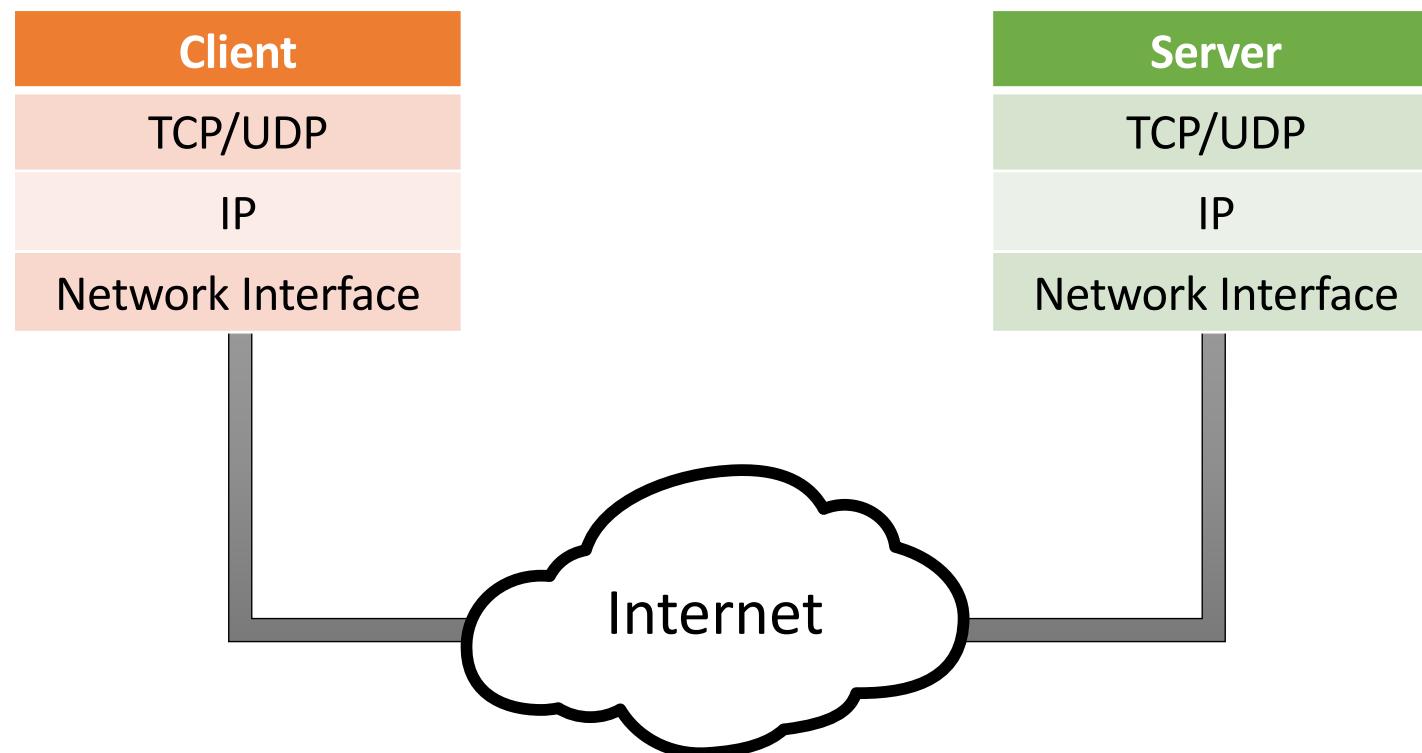
Scambio di messaggi

Modello client–server

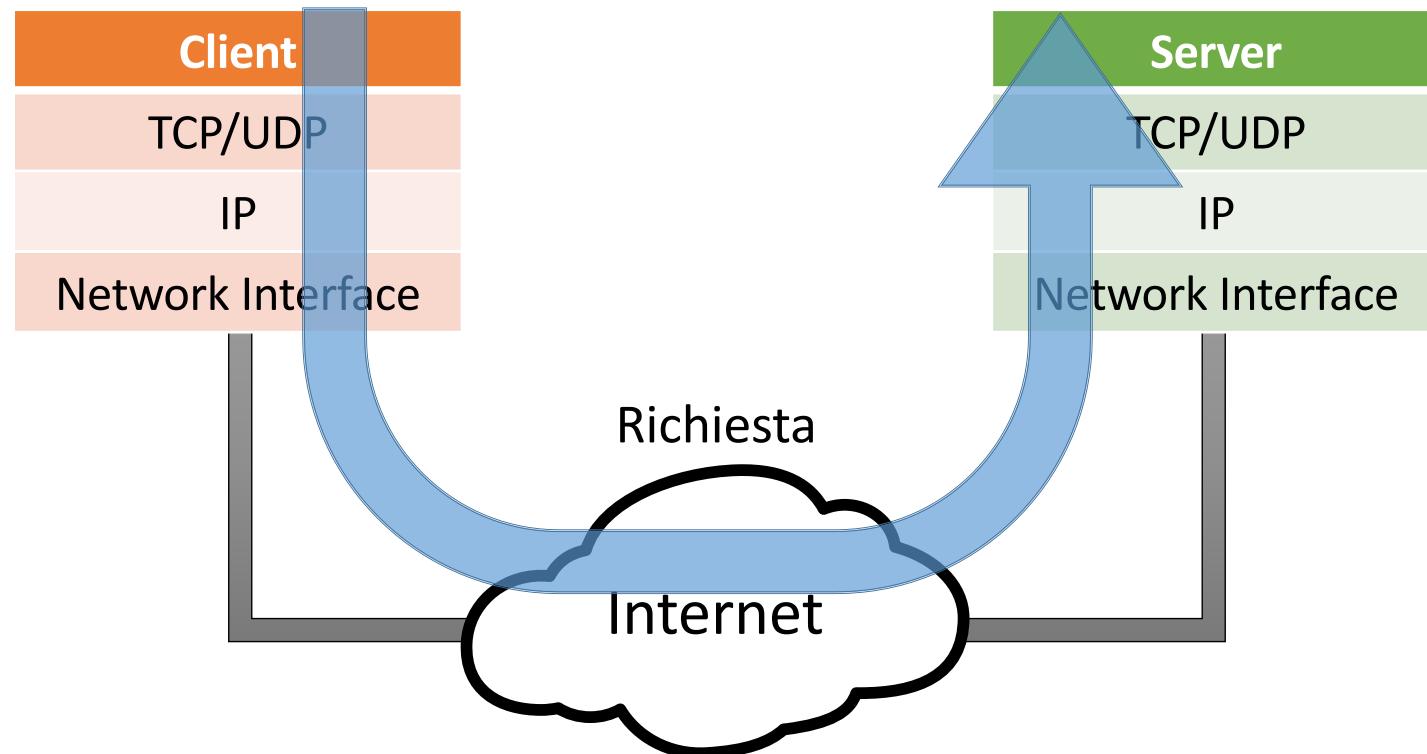
- Paradigma basato su **scambio di messaggi**
- Viene usato principalmente per sistemi distribuiti



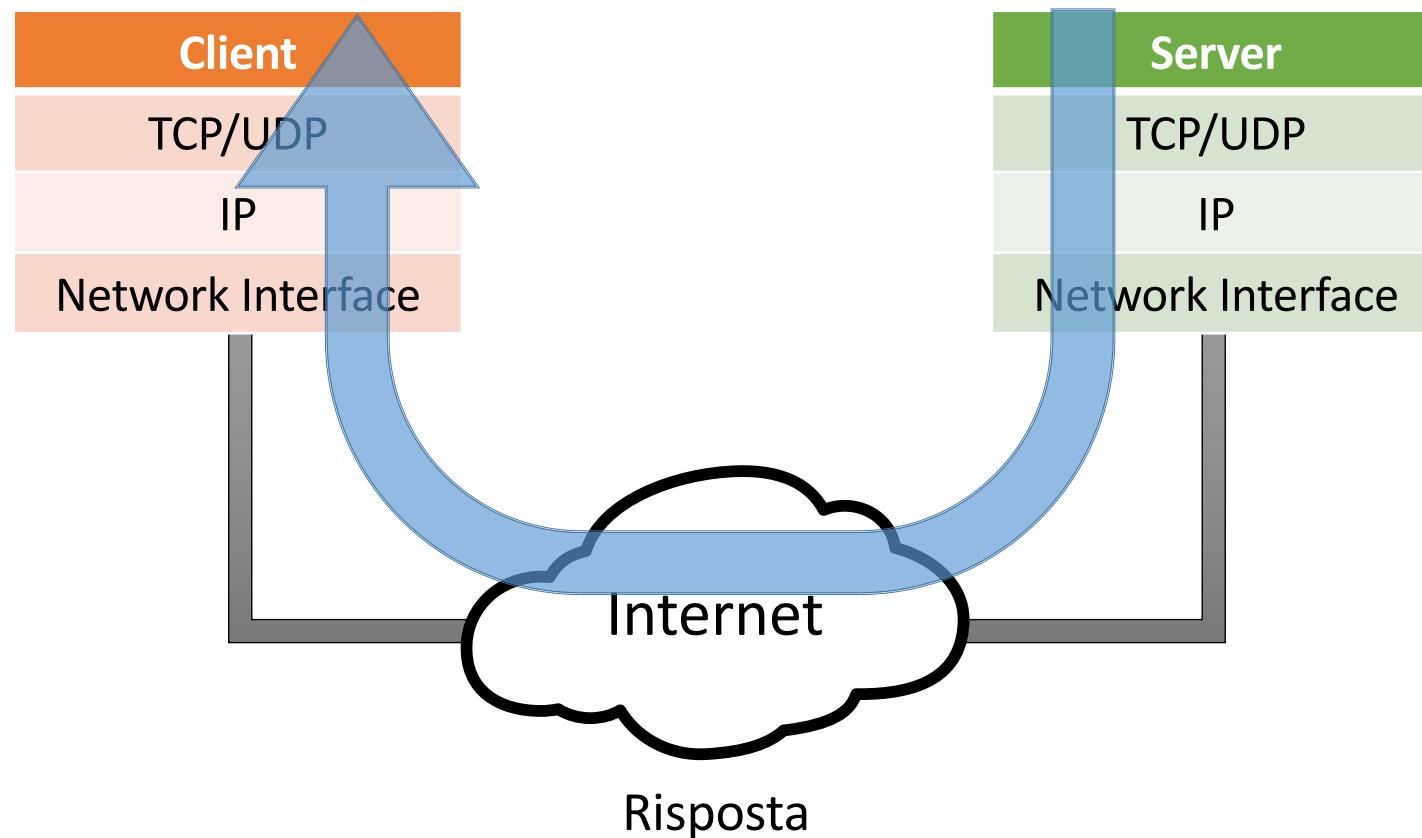
Modello client–server



Modello client–server

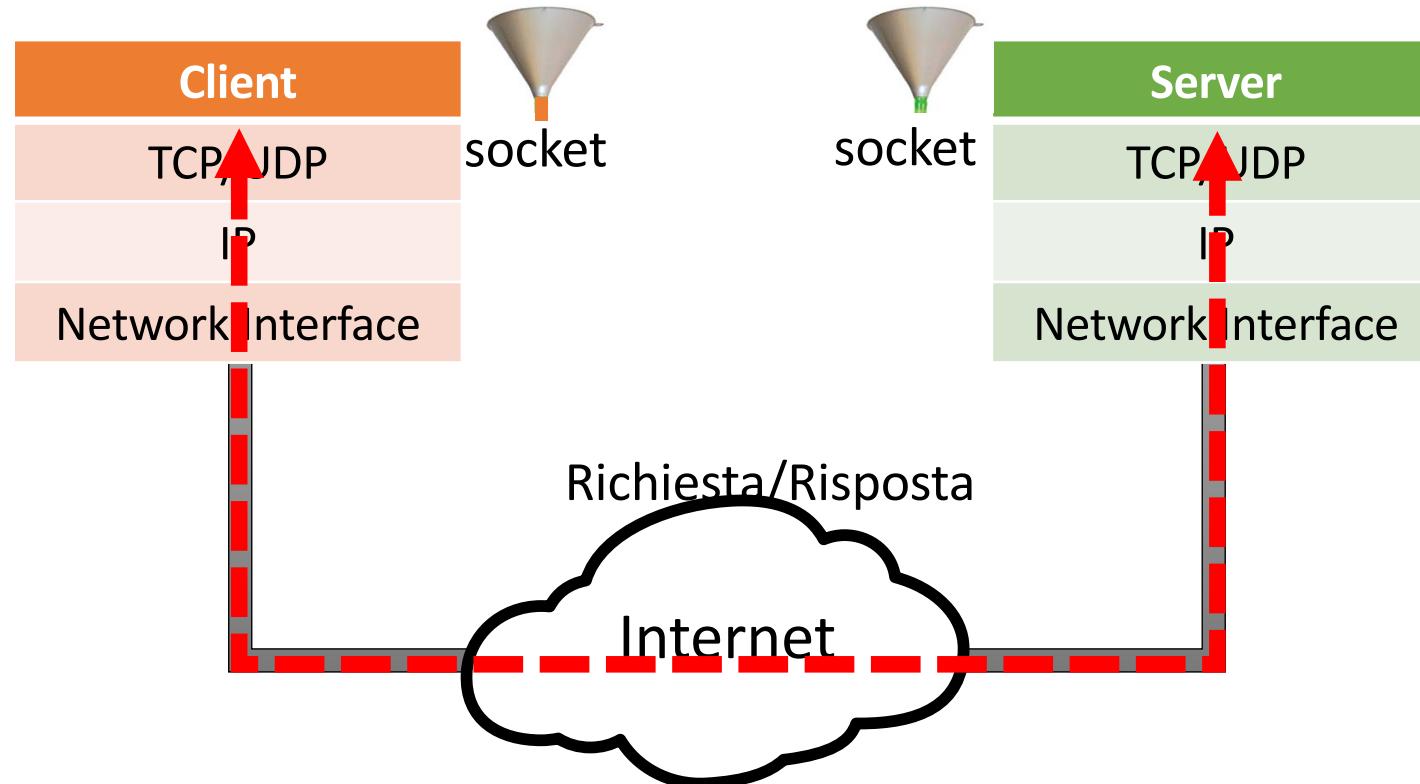


Modello client–server



Socket

Un socket è l'**estremità di un canale di comunicazione** fra due processi (programmi) in esecuzione su macchine connesse in rete

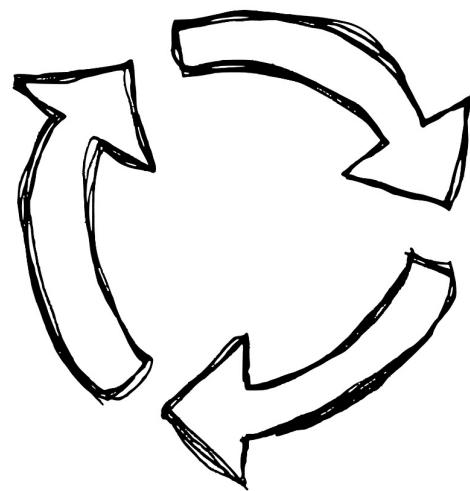


Socket

- Astrazione implementata dal sistema operativo
- Il sistema operativo fornisce le **primitive** per:
 - creare un socket
 - assegnargli un indirizzo
 - connettersi a un altro socket
 - accettare una connessione
 - inviare e ricevere dati attraverso i socket
 - ...



Processo server



- **Programma in esecuzione** sulla macchina server (strato applicazione)
- Offre un servizio (o più) e fa uso di **system call** per utilizzare i socket

Le system call utilizzano **primitive del sistema operativo**

Primitiva **socket()**

- **Creazione** di un socket

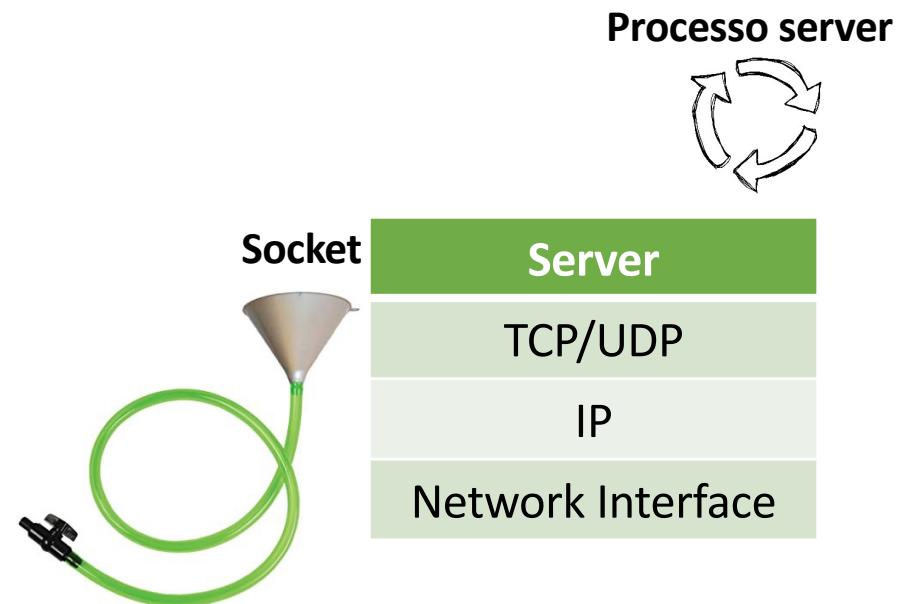
```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int socket(int domain, int type, int protocol);
```

man 7 ip
man 2 socket
man 7 socket

- **domain**: famiglia di protocolli da utilizzare
 - **AF_LOCAL** – Comunicazione locale
 - **AF_INET** – Protocolli IPv4, TCP e UDP
- **type**: tipologia di socket
 - **SOCK_STREAM** – Connessione affidabile, bidirezionale (TCP) - Oggi vedremo questo
 - **SOCK_DGRAM** – Connectionless, invio di pacchetti UDP
- **protocol**: sempre a 0
- La socket() restituisce un *descrittore di file* (-1 in caso di errore) cioè un numero che rappresenta un file, una pipe, o un socket (come in questo caso) aperto dal processo e sul quale il processo può effettuare operazioni.
- **Attenzione**: dopo la chiamata alla socket(), il socket non è ancora associato né a un indirizzo IP né a una porta

Primitiva **socket()**



Struct per gli indirizzi: socket e IP address

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

/* Struttura per memorizzare l'indirizzo di un socket */
struct sockaddr_in {
    sa_family_t    sin_family;   /* Address Family: AF_INET */
    in_port_t      sin_port;     /* Port (in network order) */
    struct in_addr sin_addr;    /* IP Address (istanza di struct in_addr)
*/;
};

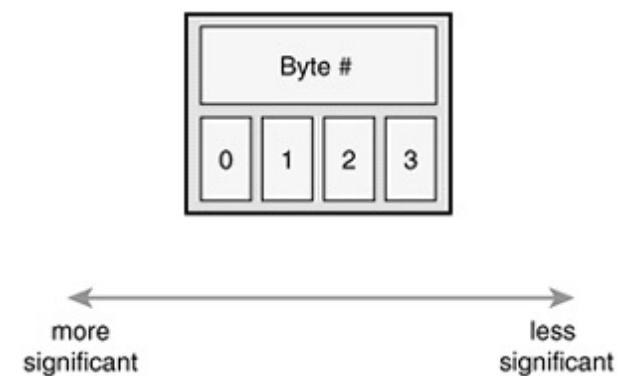
/* Struttura per memorizzare un IP address*/
struct in_addr {
    uint32_t s_addr;           /* IP Address (in network order)*/
};
```

man 7 ip

Esiste anche la struttura **sockaddr** (usata da alcune funzioni descritte nel seguito), ma non la useremo direttamente.

Endianness (a.k.a. ordine dei byte)

Calcolatori diversi possono usare modi diversi di
posizionare i byte all'interno di una *word*



Endianness (a.k.a. ordine dei byte)

Esempio: numero **422990**, in 32 bit (4 byte)

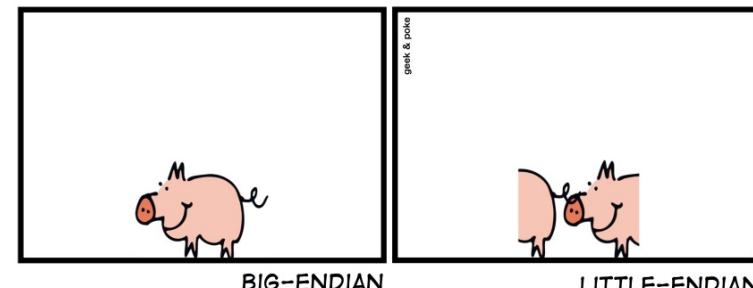
Memorizzazione *big-endian* - Per primo il byte più significativo (MSB)

Indirizzo A	Indirizzo A+1	Indirizzo A+2	Indirizzo A+3
0000 0000	0000 0110	0111 0100	0100 1110

Memorizzazione *little-endian* - Per primo il byte meno significativo (LSB)

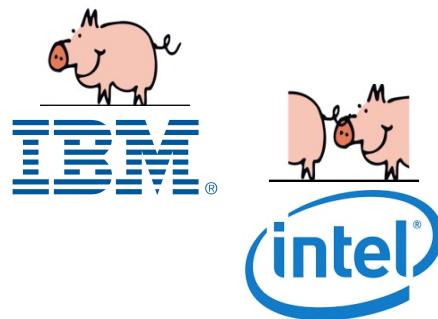
Indirizzo A	Indirizzo A+1	Indirizzo A+2	Indirizzo A+3
0100 1110	0111 0100	0000 0110	0000 0000

SIMPLY EXPLAINED



Endianness in reti e host

Il formato usato in rete (*network order*) è ***big-endian***,
quello dell'host (*host order*) dipende dall'host



Per esempio, Intel usa little-endian,
mentre IBM usa big-endian

Funzioni di conversione

Esistono **funzioni di conversione**:

```
#include <arpa/inet.h>
```

man 3 byteorder

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Funzioni
ninja ☺

Attenzione: i tipi `uint16_t` e `uint32_t` sono **interi senza segno** che occupano **sempre** 16 e 32 bit, rispettivamente, a prescindere dall'host (calcolatore) usato per compilare. Sono utili quando c'è bisogno di avere il controllo sulla dimensione delle variabili. Sono definiti nell'header `<stdint.h>`.

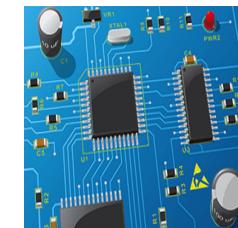
Formato degli indirizzi IP

Un indirizzo IP può essere rappresentato in formato **presentation** o in formato **numeric**

usato dal developer



usato dalla macchina



Presentation vs numeric

Presentation

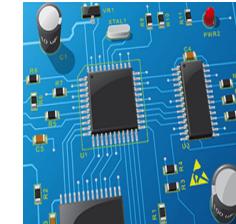
192.168.1.4



Stringa in notazione
decimale puntata

Numeric

3232235780



Intero a 32 bit

Conversione

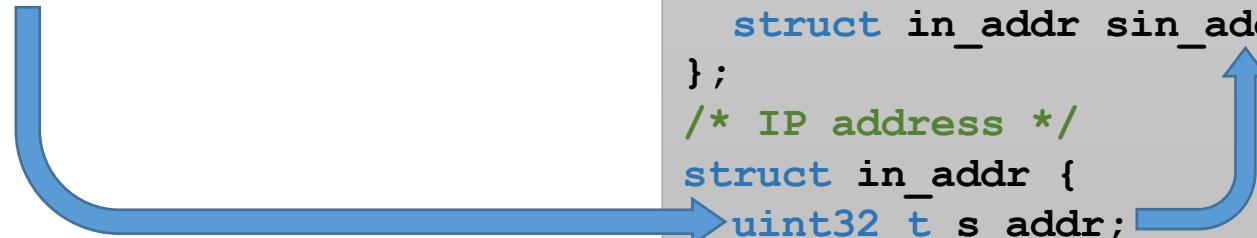
Presentation

192.168.1.4



Numeric

3232235780



Funzioni di conversione

`man inet_pton`
`man inet_ntop`

```
int inet_pton(int af, const char* src, void* dst);
```

- **af**: famiglia (`AF_INET`)
- **src**: stringa del tipo "`xxx.xxx.xxx.xxx`"
- **dst**: puntatore a un'istanza di struttura `in_addr`

Presentation



Numeric

```
const char* inet_ntop(int af, const void* src,
                      char* dst, socklen_t size);
```

- **af**: famiglia (`AF_INET`)
- **src**: puntatore a un'istanza di struttura `in_addr`
- **dst**: puntatore a un buffer di caratteri di lunghezza **size**
- **size**: dimensione dell'indirizzo del socket, dove il tipo `socklen_t` è usato per rendere il codice indipendente dall'architettura che può avere CPU a 16, 32, o 64 bit. Come valore, si può usare `INET_ADDRSTRLEN` (16 byte).

Numeric



Presentation

Creazione e inizializzazione di un socket

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

int main () {
    /* Creazione socket */
    int sd = socket(AF_INET, SOCK_STREAM, 0);

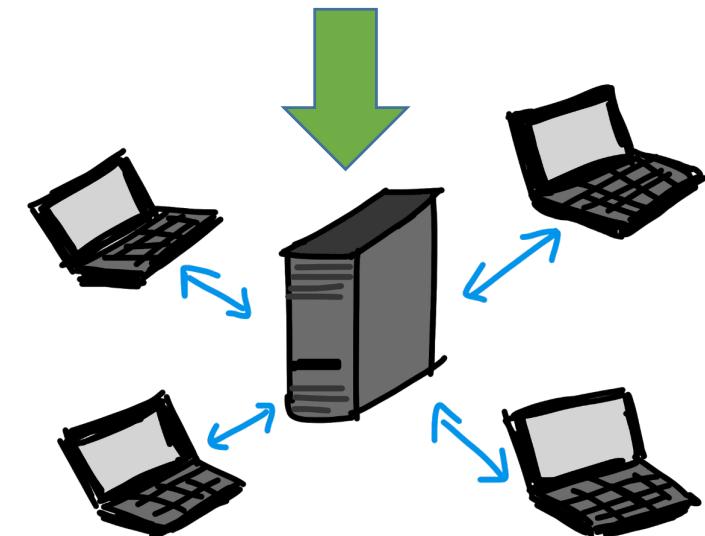
    /* Creazione indirizzo del socket */
    struct sockaddr_in indirizzo;

    memset(&indirizzo, 0, sizeof(indirizzo)); // Pulizia
    // Address family (IPv4)
    indirizzo.sin_family = AF_INET;
    // Port inizializzata a 4242, in network order
    indirizzo.sin_port = htons(4242);
    // IP address "192.168.4.5", convertito in numeric format
    inet_pton(AF_INET, "192.168.4.5", &indirizzo.sin_addr);

    // to be continued...
```

Programmazione distribuita

Lato server



Primitiva **bind()**

- **Associa** un socket a un indirizzo
 - L'indirizzo del socket è un'istanza di struct `sockaddr_in` che specifica IP e porta dove il server **riceve richieste di connessione**

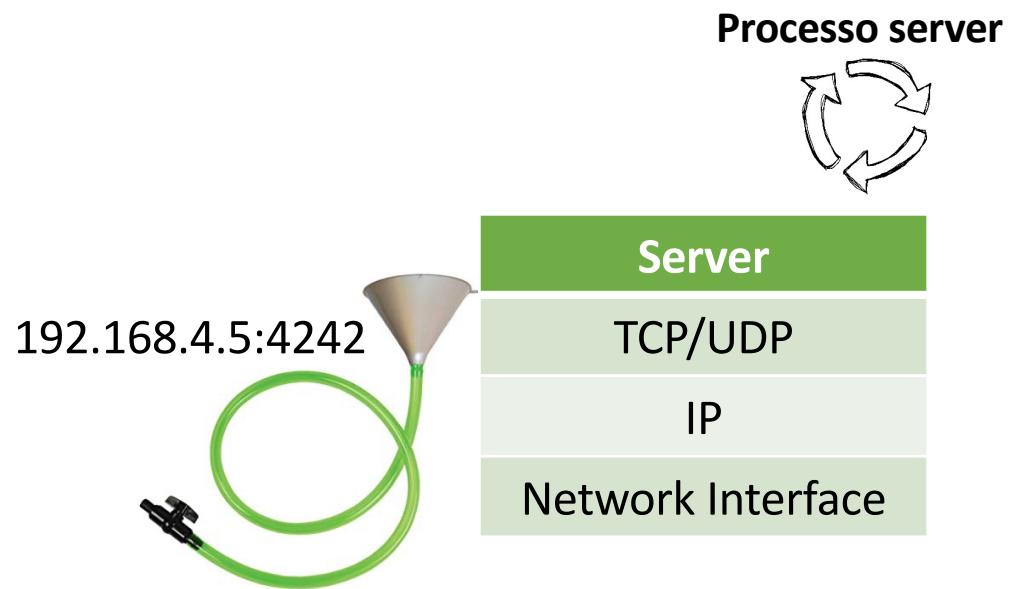
```
int bind(int sockfd, const struct sockaddr *addr,  
        socklen_t addrlen);
```

man 2 bind

- **sockfd**: descrittore del socket
- **addr**: puntatore a struttura di tipo **sockaddr**
N.B.: usiamo `sockaddr_in`, occorre quindi un cast del puntatore
- **addrlen**: dimensione di **addr**
- La primitiva **bind()** restituisce 0 se ha successo, -1 se si verifica un errore

```
ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
```

Primitiva **bind()**



Primitiva `listen()`

- Specifica che il socket è **usato per ricevere richieste di connessione (socket passivo)**
 - si possono mettere in attesa solo i socket **SOCK_STREAM**

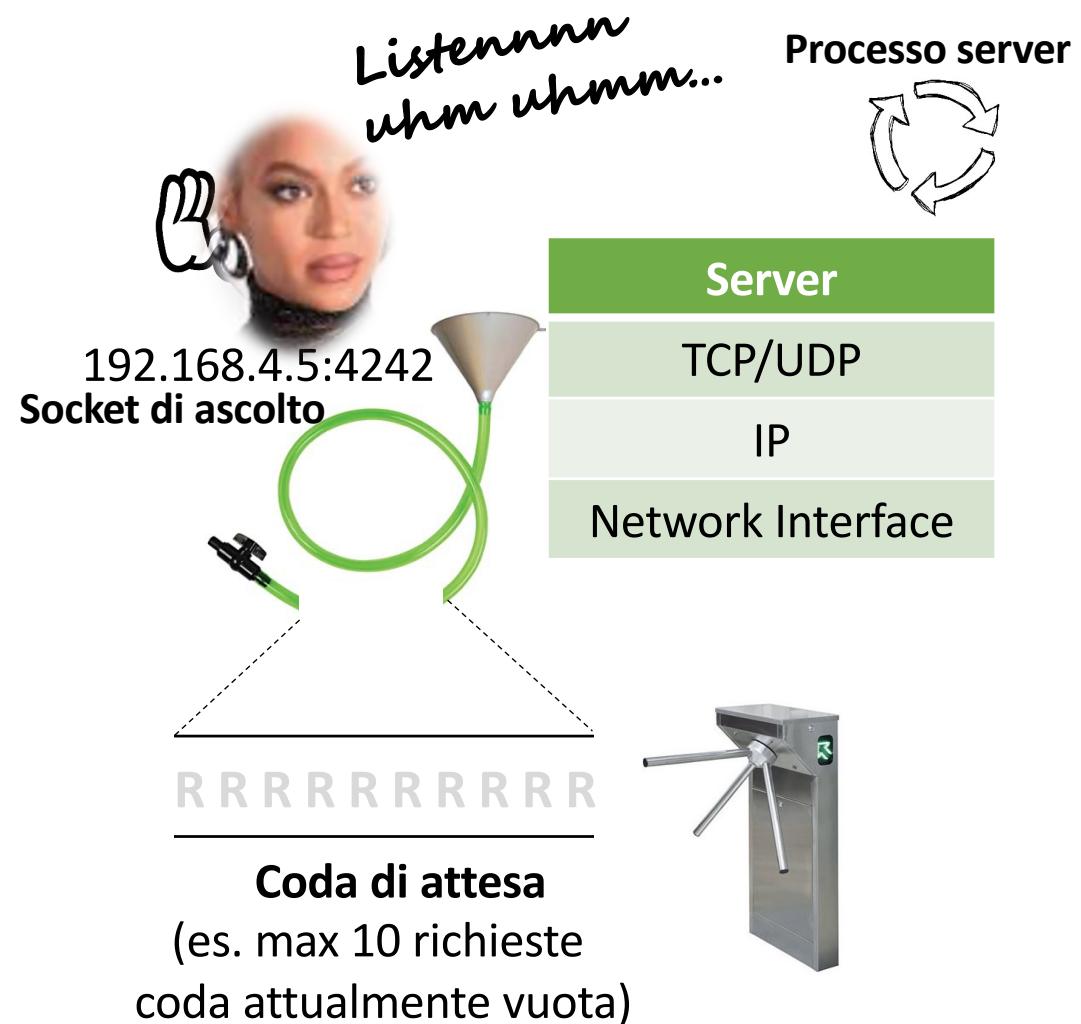
```
int listen(int sockfd, int backlog);
```

man 2 listen

- **sockfd**: descrittore del socket
- **backlog**: dimensione della coda, cioè il numero massimo di richieste (dai client) possono essere messe in attesa di essere gestite
- La funzione restituisce 0 se ha successo, -1 su errore

```
ret = listen(sd, 10);
```

Primitiva `listen()`



Dopo `listen()`

