Socket: cosa sono e come funzionano

Un **socket** è un ***oggetto software*** che **permette l'invio e la ricezione di dati, tra host remoti** (tramite una rete) **o tra processi locali** (Inter-Process Communication).
Più precisamente, il concetto di socket si basa sul modello Input/Output su file di Unix, quindi sulle operazioni di ***open***, ***read***, ***write*** e ***close***; l'utilizzo, infatti, avviene secondo le stesse modalità, aggiungendo i parametri utili alla comunicazione, quali **indirizzi**, **numeri di porta** e **protocolli**.
Socket locali e remoti in comunicazione formano una **coppia** (pair), composta da **indirizzo** e **porta** di client e server; tra di loro c'è una connessione logica.
Solitamente i sistemi operativi forniscono delle API per permettere alle applicazioni di controllare e utilizzare i socket di rete.
Possiamo vedere i socket come degli **intermediari** tra il **livello applicazione** e di **trasporto** nello stack TCP/IP. Infatti la funzione dei socket è quella di **indirizzamento dei processi**.

Formato di un Socket Address

Dato che sui sistemi interlocutori possono esserci molti processi, bisogna avere un modo per **indirizzare** precisamente il processo con cui si sta dialogando.
Per questo si usano le **porte**: dei numeri che identificano i processi in esecuzione.
Gli interlocutori, quindi, memorizzano **indirizzo e porta** della controparte, in un **indirizzo socket**, formato così:

- Indirizzo IP: 32 bit;
- Numero di porta: 16 bit.

I **numeri di porta** sono stati definiti dalla IANA, e si dividono in:

- Well-known (riservate a protocolli specifici): 20, 23, 25, 80, 110,…;
- Non usate: 0;
- Riservate per processi well-known: 1-255;
- Riservate per altri processi: 256-1023;
- Altre applicazioni: 1024-65535;

Normalmente i numeri di porta assegnati ai processi (non well-known) sono a discrezione del sistema operativo. Si dice che il sistema operativo assegna delle porte *effimere*.

Famiglie di socket

I tipi di **protocolli** utilizzati dal **socket**, ne definiscono la **famiglia** (o dominio).
Possiamo distinguere, ad esempio, due importanti famiglie:

- ***AF_INET***: comunicazione tra host remoti, tramite Internet;

- ***AF_UNIX***: comunicazione tra processi locali, su macchine Unix. Questa famiglia è anche chiamata *Unix Domain Socket*.

Tipi di socket

All'interno della famiglia possiamo distinguere il **tipo di socket**, a seconda della modalità di connessione. Abbiamo:

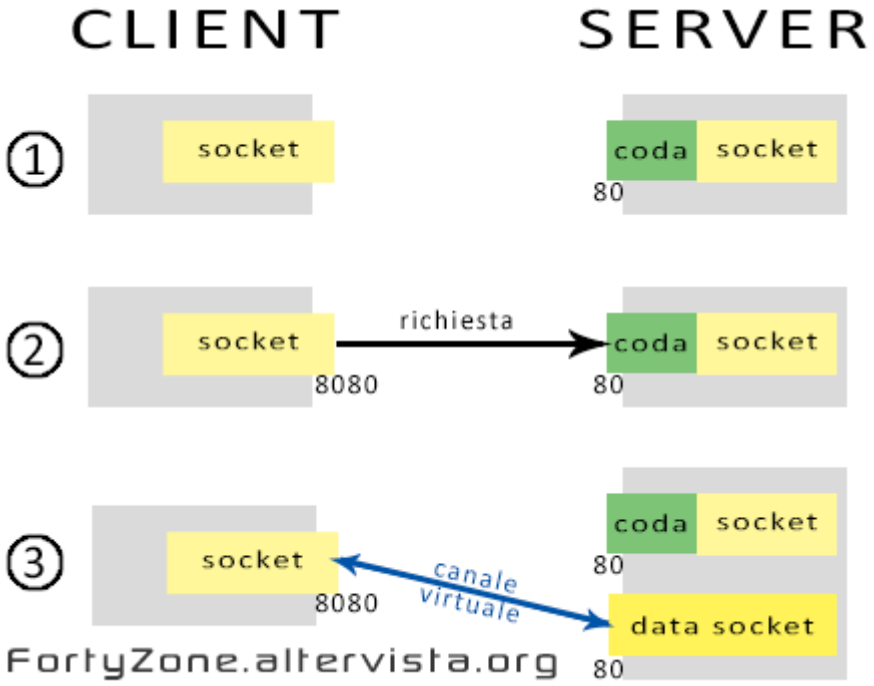
- **Stream socket**: orientati alla connessione (connection-oriented), basati su protocolli affidabili come TCP o SCTP;
- **Datagram socket**: non orientati alla connessione (connectionless), basati sul protocollo veloce ma inaffidabile UDP;
- **Raw socket** (raw IP): il livello di trasporto viene bypassato, e l'header è accessibile al livello applicativo.

Stream Socket

Essendo basati su protocolli a livello di trasporto come **TCP**, garantiscono una comunicazione **affidabile**, **full-duplex**, **orientata alla connessione**, e con un ***flusso di byte*** di lunghezza variabile.
La comunicazione mediante questo socket, si compone di queste fasi:

#1 – Creazione dei socket

Client e server creano i loro rispettivi **socket**, e il **server** lo pone in **ascolto** su una **porta**.



Dato che il server può creare più connessioni con client diversi (ma anche con lo stesso), ha bisogno di una **coda** per gestire le varie richieste.

#2 – Richiesta di connessione

Il **client** effettua una **richiesta di connessione** verso il server.
Da notare che possiamo avere due numeri di porta diversi, perchè una potrebbe essere dedicata solo al traffico in uscita, l'altra solo in entrata; questo dipende dalla configurazione dell'host. In sostanza, non è detto che la porta locale del client coincida con quella remota del server
Il **server** riceve la richiesta e, nel caso in cui sia accettata, viene creata una **nuova connessione**.

#3 – Comunicazione

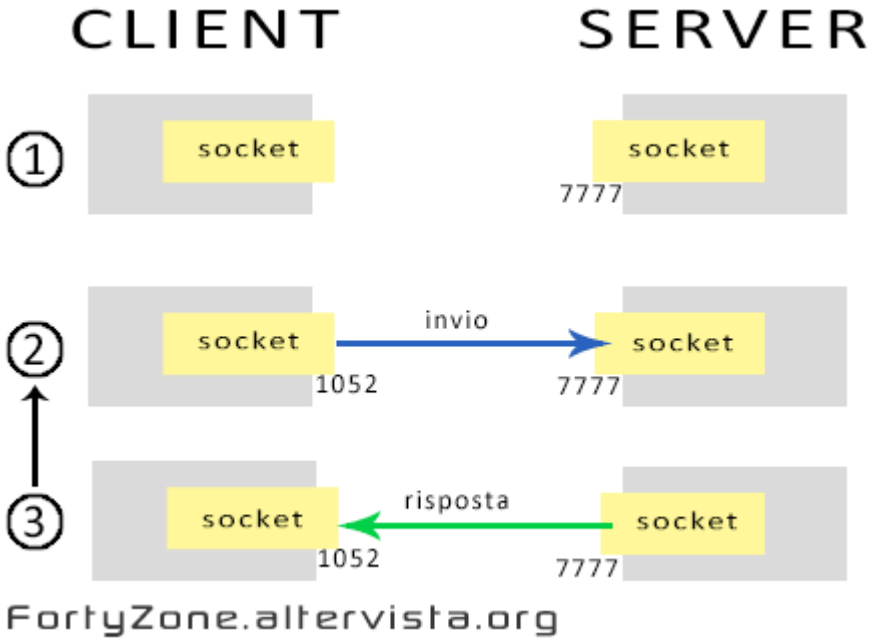
Ora client e server comunicano attraverso un **canale virtuale**, tra il socket del primo, ed uno nuovo del server, creato appositamente per il flusso dei dati di questa connessione: **data socket**.
Coerentemente a quanto accennato nella prima fase, il server crea il data socket perchè il primo serve esclusivamente alla gestione delle richieste. È possibile, quindi, che ci siano molti client a comunicare con il server, ciascuno verso il **data socket** creato dal server per loro.

#4 – Chiusura della connessione

Essendo il TCP un protocollo orientato alla connessione, quando non si ha più la necessità di comunicare, il client lo comunica al server, che ne **deistanzia il data socket**. La connessione viene così chiusa.

Datagram Socket

Sono basati su **UDP**, un protocollo a livello di trasporto che garantisce comunicazioni a bassa latenza (ideali per videochat, ad esempio), a discapito dell'affidabilità dei dati. Non esiste, infatti, controllo di flusso (ordinamento dei datagrammi e controllo degli errori).



Dato che l'UDP non è un protocollo orientato alla connessione, non esiste la fase di connessione vista poco fa, ma il client comunica direttamente con il server, quando vuole.

#1 – Creazione dei socket

Come nel tipo precedente, client e server creano i loro rispettivi **socket**, e il **server** lo pone in **ascolto** su una **porta**. Il socket del server, stavolta, non ha bisogno di una coda, in quanto i dati in entrata e in uscita non devono essere circoscritti all'interno di una connessione, quindi **la comunicazione con diversi client si svolte sulla stessa interfaccia**.

#2 – Invio dei dati

Il **client** invia direttamente i datagrammi al server. Anche in questo caso vale quanto detto in precedenza riguardo i numeri di porta.

#3 – Risposta del server

Il **server** manda al client un eventuale **risposta**. La comunicazione è quindi, costituita da un **loop che dura finchè ci sono dati da inviare** e, ovviamente, finchè gli host sono raggiungibili.