Comunicazione tramite socket

Comunicazione tra processi

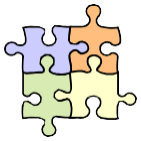
- La spina dorsale dei sistemi distribuiti è la **comunicazione tra processi (interprocesso)** – **IPC**, *interprocess communication*
  - i sistemi operativi offrono uno o più servizi di base per la comunicazione interprocesso
    - ad esempio, pipe e socket
  - alcuni servizi di IPC sono basati su protocolli di Internet
    - ad esempio, TCP e UDP

Comunicazione interprocesso di base

- Ci concentriamo sulla comunicazione interprocesso di tipo unicast, ovvero: un mittente e un destinatario.
  - una coppia di processi indipendenti – che possono essere sullo stesso computer o su computer diversi si scambiano dati (*messaggi*)
  - è necessario basare la comunicazione su un *protocollo* – formato da messaggi e da regole, anche di sincronizzazione – accettato da entrambe le parti.

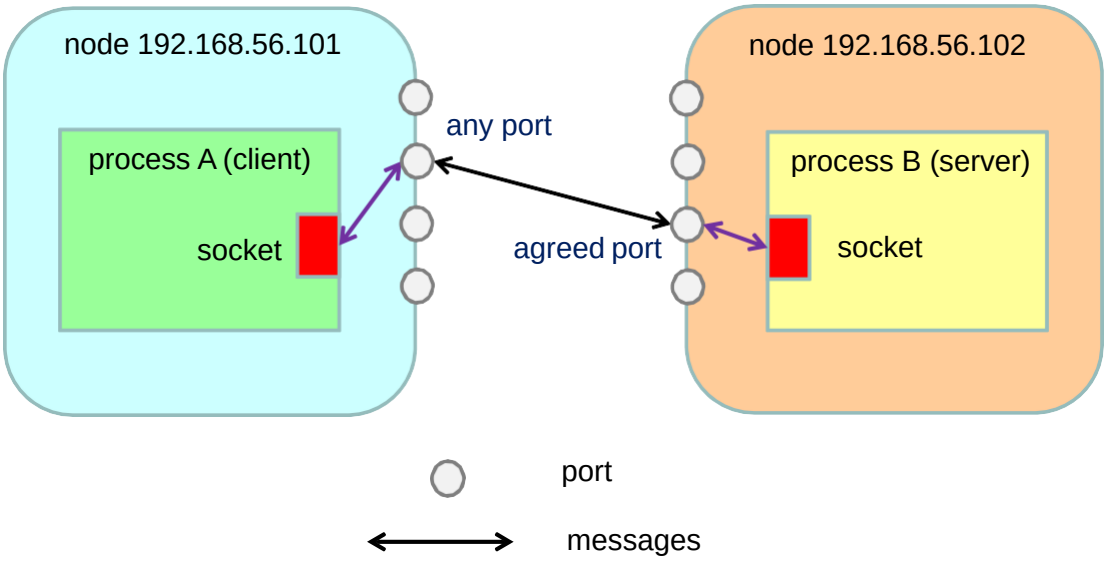
Socket

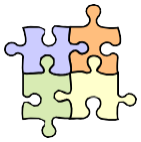
- Nel seguito faremo riferimento ai **socket** – un'astrazione di programmazione (ovvero, un'API standard) per lo scambio di messaggi in rete tramite UDP e/o TCP
  - un socket fornisce un punto (endpoint) per la comunicazione traprocessi – letteralmente, socket = connettore, presa
  - l'astrazione di comunicazione interprocesso fornita dai socket consiste nella possibilità di inviare un messaggio tramite un socket di un processo e ricevere il messaggio tramite un socket di un altro processo



Socket – modello fisico

- In pratica, ciascun socket deve essere legato a una porta di un computer (in rete)
  - ogni socket ha un indirizzo (*indirizzo IP*, *numero di porta*)





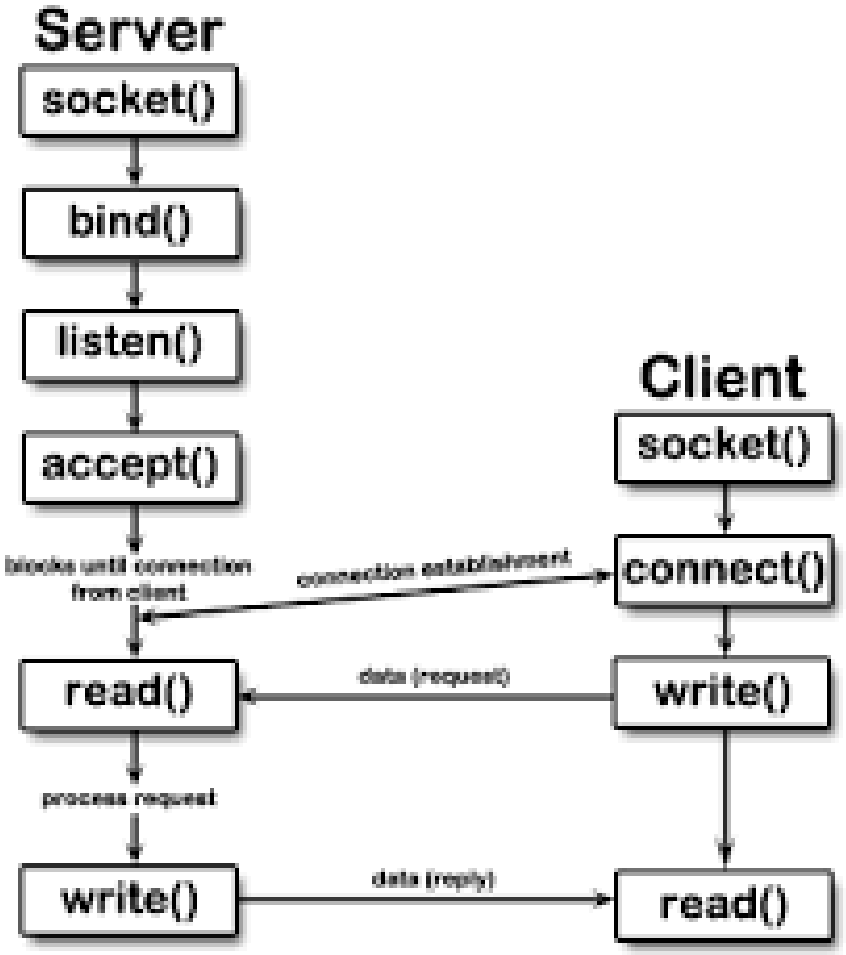
UDP e TCP

- I socket consentono la comunicazione tra processi con riferimento ai protocolli (a livello di trasporto) UDP e TCP
- *UDP* è un protocollo per il trasporto di datagrammi – senza ack e ritrasmissione
  - di tipo “best effort” – ovvero, non offre garanzie di consegna – i datagrammi potrebbero venire persi (a causa di errori di rete o scartati per se la rete è congestionata) o consegnati fuori ordine
  - connectionless – l’overhead è basso – non è richiesto setup
- *TCP* è un servizio di trasporto più sofisticato
  - consegna “affidabile” (?, vedi dopo) di uno stream ordinato di dati – uso di numerazione di pacchetti, checksum, ack e ritrasmissione
  - connection-oriented – richiesto il setup di un canale di comunicazione bidirezionale – l’overhead è più alto

Primitive connect e disconnect

- Due primitive per la comunicazione tra processi con connessione:
  - ***connect*** – per stabilire una connessione logica tra due processi
    - request-to-connect + accept-connection
    - per allocare delle risorse per gestire la connessione
  - ***disconnect*** – per terminare una connessione logica su entrambi i lati della comunicazione
    - per deallocare le risorse per la gestione della connessione
- Avvertenza
  - la comunicazione connection-oriented è più costosa sia in termini di occupazione di memoria che di consumo di CPU
  - questo può anche limitare la scalabilità di un servizio – ovvero il numero massimo di sessioni/connessioni che possono essere attive nello stesso momento – a meno che non vengano implementate soluzioni opportune, ad es., connection pooling

Il sistema operativo i socket



Il Sistema Operativo mette a disposizione per le applicazioni delle primitive (sostanzialmente funzioni-funzionalità) che consentono di instaurare la connessione tra due processi e il conseguente scambio dati. Il Server inizia le operazioni istanziando un socket tramite la primitiva socket() e successivamente collega tale istanza all'indirizzo di rete tramite bind(). Con la primitiva listen(), il processo server si mette in ascolto in attesa di richiesta di connessione.
Quando un client necessita di effettuare una richiesta al server, istanza un socket con la primitiva socket(), e richiede una connessione invocando la primitiva connect(). A questo punto, il server riceve la richiesta di connessione, la accetta, e comunica l'accettazione al client, il quale inizia la trasmissione della richiesta con write(); il server userà quindi la primitiva read() per leggere la richiesta e appena avrà elaborato una risposta, la invierà al client tramite la primitiva write().