Message oriented communication

Thursday, 23 March 2023 08:39

- Web
  - Interazione client-server attraverso chiamate http
    - Client, applicazione per il web lato del client
    - Web browser = user-agent
      - Col compito di Interpretare il codice e visualizzarlo, ed anche esecuz
        Ed il codice non è altro che un insieme di risorse, che sono una sequ
        Risiedenti in un computer identificato da un URL
      - URL è un qualcosa di unico che è costituito da:
        Procollo://IndirizzoIp:porta/PercorsoHost/IdentificazioneRisorsa
        - Vari protocolli hanno diverse funzioni, definiscono:
          - Formato
          - Ordine invio ricezione
          - Tipo dati
          - Azioni da eseguire
        - Indirizzo ip è l'indirizzo della macchine
        - La porta è opzionale siccome certe applicazione hanno porta r
          E questo identifica il processo della risorsa
      - I dati del web possono essere:
        - Standard (Html, XML, Json)
        - Non testuali (Quindi abbiamo una flessibilità
        - Codice
    - Web server = Gestire le risorse
    - Protocollo HTTP
      - Hypertext è un insieme di testi, pagine leggibili tramite hyperlink, ch
      - Unico protocollo per la communicazione web
      - Usa TCP, quindi una socket verso il server con porta 80
      - Stateless: Siccome siamo in un sistema concorrente, ogni messaggio
        Deve contenere tutte le informazioni per l'esecuzione
        Es. Io invio ruota e muovi triangolo, anziché prima ruoto e dopo mu
      - Formato:
        - Start-Line
          Fatto da 3 parti che serve per il parser, finisce con CRLF e spaz
        - Header line
          Coppia header:valore che serve per farti comprendere cosa ti
          Ogni coppia finisce con CRLF, e la fine dell'header è senza hea

Ogni coppia finisce con CRLF, e la fine degli header è senza hea

- Payload
  Ciò che noi vogliamo effettivamente inviare

Es.

- GET percorso/risorsa.html HTTP/1.1
  Esistono diversi metodi:
  - GET
    - Posso eseguirla tante volte voglio, siccome è una
      E' una operazione di lettura
    - Restituisce una rappresentazione di una risorsa
    - Può avere dei parametri con coppie chiave-valore
    - E' cache, quindi possiamo tornare avanti ed indiet
      Qui si utilizza if-modified-since he restituisce 200
      304 se invece c'è stata
    - Non è idempotente, quindi se noi inviamo la stess
      La risposta potrebbero essere diverse
  - POST
    - Qui potremmo modificare dei dati
    - No cache proprio per questo
    - Non è idempotente
  - PUT
    - La PUT è idempotente, quindi
      Se aggiungiamo 2 mario rossi, output deve essere
      Non dobbiamo aggiungere 2 mario rossi nel nostr
      Ma nel primo lo aggiungiamo, nel secondo lo aggi
      L'output è lo stesso
  - DELETE

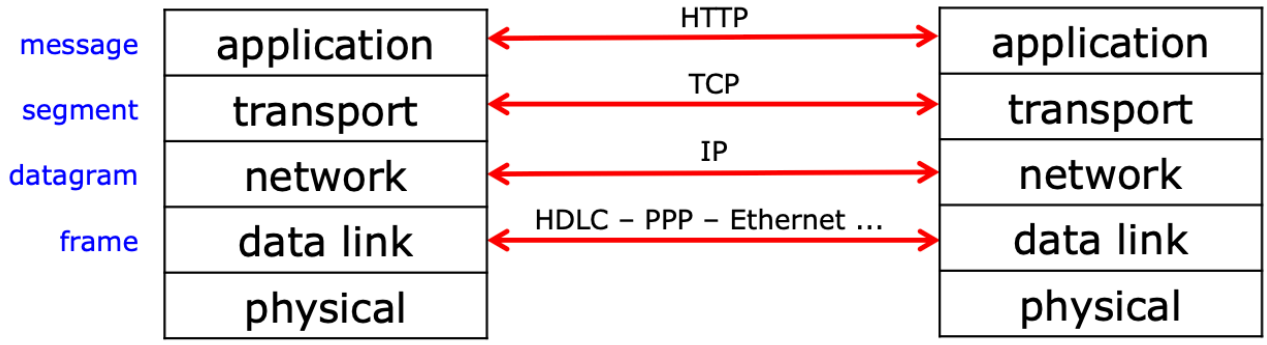
  Ed esistono diversi codici di stato:
  - 1**: Richiesta ricevuta (probabilmente ignorata), Inform
  - 2**: Successo
  - 3**: Redirect
  - 4**: Client error
  - 5**: Server error
- Host: HOST
  Connection: close -> Richiede chiusura richiesta
  User-agent: Mozilla/4.0 -> Qualifica il richiedente
  Accept: text/html, image/gif, image/jpeg -> La risposta che co
  I tipi non possono essere inventati, sono predefini
  Accept-language: fr -> La risposta che vogliamo
  Cookie: per tenere dei dati persistenti
  -> Il server invia un cookie al client con header set-cooki
  Authenticate: Ci identifica

Authenticate: Ci identifica
-> Spazio vuoto per dire la fine dell'header

Es:

- HTTP/1.1 200 OK
- Connection: close
  Date: Thu 06
  Server: Apache/1.3.0
  Last modified: mon 22 jun
  Content-Len: 6821 ----------_\-/ --> Non possiamo avere un carat
  Terminatore
  Content-type: text/html -> Obbligatorio se abbiamo body
- Data: body

- Message vs Stream Communcation



Applicazione
Invio messaggio come stream di byte al layer di trasporto, e legge lo stream per ricostr

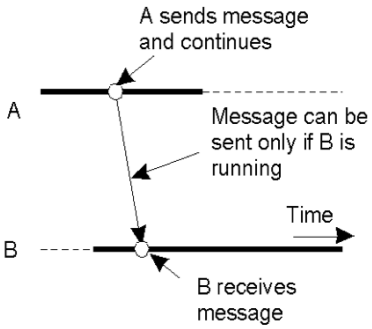
- UDP scompone lo stream di byte in segmenti ed invia i segmenti
- TCP fa come UDP, però ogni segmento garantisce: Riordinamento, controllo per

Il messaggio viene ricevuto dal web attraverso http da un altro application

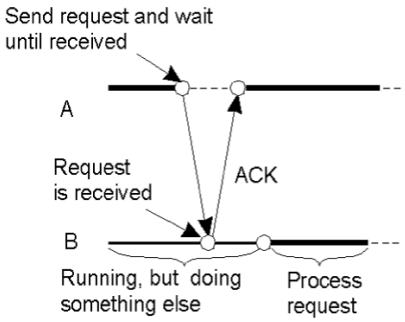
- Tipi di communicazioni
  - Sincrona/Asincrona
    Per la sincronizzazione abbiamo:
    - All'inizio, quando inviamo la richiesta
    - Alla fine, quando la richiesta viene ricevuta
    - Alla risposta del server dopo aver processato il messaggio |-> Chiamata di

    Questi sono sia punti di sincronizzazione ma anche punti di controllo
    \-> Es uno non risponde più

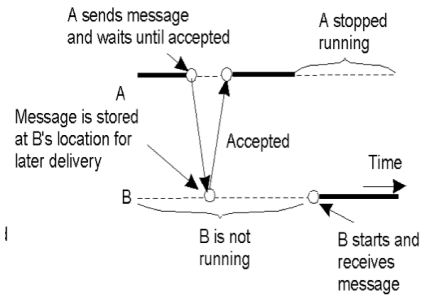
    Praticamente da ora vediamo la sincronia quando avviene, come avviene
    In quali stage temporali.
  - Transiente
    - Transito asyncrono
      Io sono A, mando un messaggio e vado avanti
      B riceve un messaggio e vado avanti, ed il problema è che non so se l'ha le



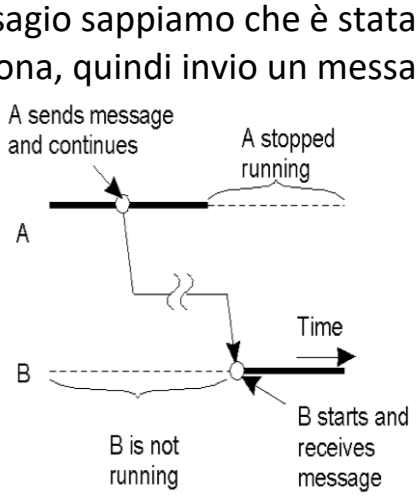
- Receipt-based transito sincrono
  Noi inviamo a B, ed aspettiamo la risposta da B
  B dice che l'ha ricevuto, però farà la processione dopo



- Persistente
  Vuol dire che, anche se non possiamo leggere ora i messaggi
  Appena possiamo li possiamo leggere, aka mantengo memoria
  E qui abbiamo:
  - Sincrona, quindi invio un messaggio, ed aspetto la risposta per potere con

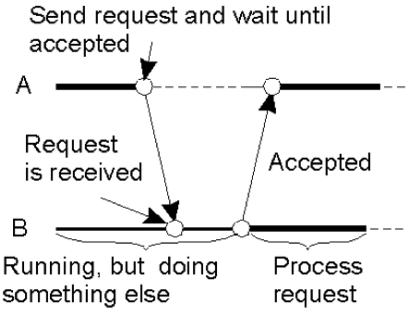


    Il messagio sappiamo che è stata ricevuta, e che prima o poi verrà letta
  - Asincrona, quindi invio un messaggio ed io continuo fottendomene se il ti

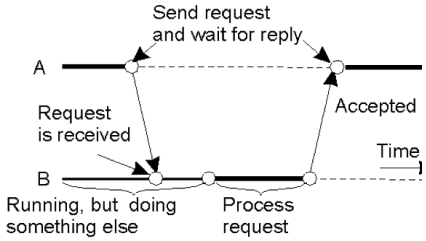


    Quindi qui abbiamo un buffer

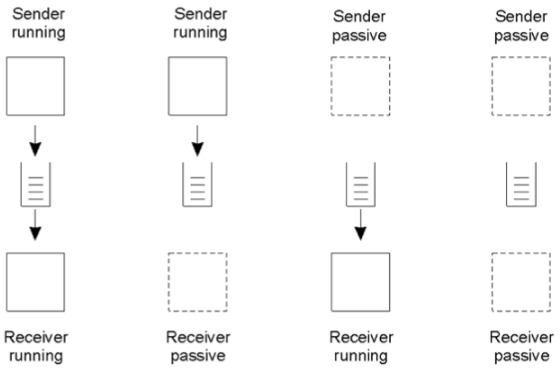
- Delivery
  - Delivert-based, quindi riceviamo notifica quando il server inizia a processa



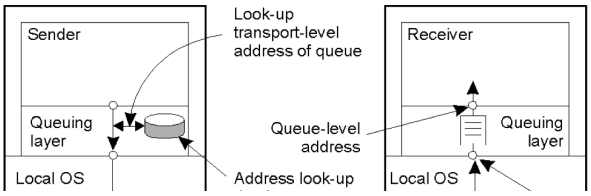
  - Response-based, qui invece dopo che il messaggio è stato processato

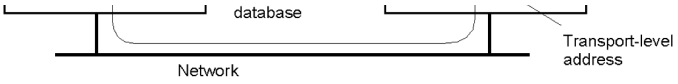


- Per implementare molti di questi sistemi abbiamo bisogno di un sistema di messaggi a
  Che ci permette di avere uno storage dei messaggi, quindi senza obbligare il client/ser
  Attivi durante la communicazione.
  Esistono 4 tipologie:
  - Client e server sono tutti e due attivi per forza e si passano i messaggi con la cod
  - Il client invia, ed il server può aspettare
  - Il client diventa l'entità passiva, quindi invia i messaggi nella coda, e prima o poi
  - Sia il server che il client sono passivi

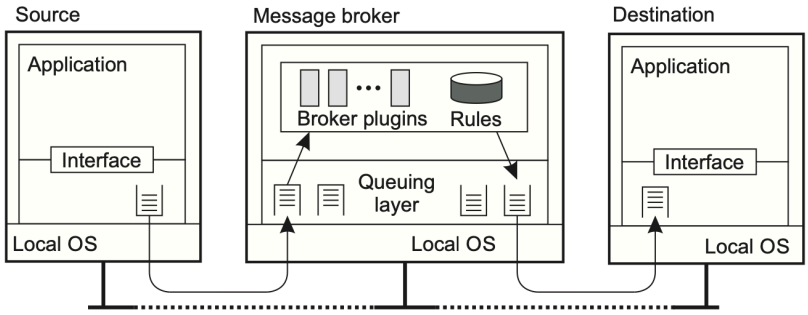


  Ed ogni coda ha le seguenti primitive:
  - Put: Aggiungiamo messaggio in queue
  - Get: Ti fermi se la queue è vuota, prende quando 1 elemento, rimuovi
  - Poll: Vediamo se c'è un messaggio, se c'è restituisco, non è bloccante
  - Notify: Veniamo notificati quando un messaggio viene aggiunto



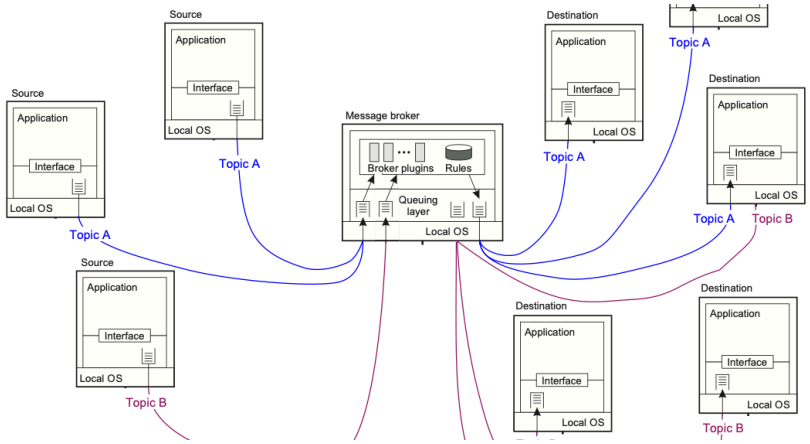


Spesso abbiamo un middleware nella message queue chiamato broker
Che smista i messaggi e li riordina



Quindi il client ed il server non comunicano direttamente:

- Abbiamo un disaccoppiamento, aka indipendenza tra i componenti
  Es potremmo avere un server per argomento, ed il client sceglie l'argomento, ec
  A seconda dell'argomento sceglie il server
- Ed una maggiore scalabilità siccome abbiamo una concorrenza
  Aka si può creare comunicazione molti-molti
  E comunicazioni persistenti



- Un esempio è il MQTT, ed esistono di 3 tipi:
  0. Il client non riceve la notifica di ricezione dal server
  1. Assicurazione del messaggio, quindi se il messaggio si è perso lo rinvio, e chi rice
     potrebbero essere dei doppioni che devono essere gestiti
  2. Sta volta 1 ed 1 solo messaggio viene ricevuto, e vengono scambiati i certificati p

  E qui abbiamo diverse politiche:
  - Sessioni persistenti
    Quindi mi vengono tenuti da parte i messaggi se richiesto
  - E nota che, il broke in tutti e due i casi mantiene sempre almeno 1 messaggio (d
    inviare quando il client si riconnette .
    Di norma si invia sempre 1 messaggio all'inizio, il messaggio di welcome, che pu
    oppure una welcome/informazioni
  - LTW
    Dico ad un altro middleware che messaggio inviare a MQTT che messaggio invia

Nota: Si usano semprei middleware durante la communicazione
Il client ha un middleware, il server ne ha uno, che permettono la communicazio