



# LABORATORIO di Reti di Calcolatori

## Programmazione di applicazioni client-server

## Bibliografia

- ❖ slide della docente
- ❖ *testo di supporto*: D. Maggiorini, “Introduzione alla programmazione client-server”, Pearson Ed., 2009
  - ❑ cap.1 (tutto)
  - ❑ cap.4 (tutto)
  - ❑ cap.5 (tutto)
  - ❑ cap.7 (tutto)
  - ❑ cap.8 (tutto)
- ❖ *Link utili*:
  - ❑ <http://docs.oracle.com/javase/tutorial/networking/index.html>
  - ❑ <http://docs.oracle.com/javase/6/docs/>

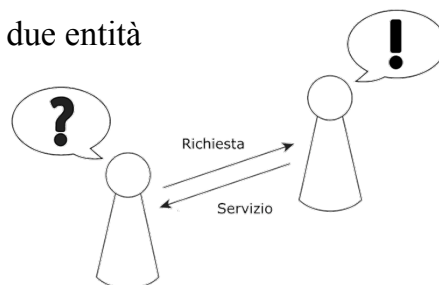
## Modelli di servizio

- ❖ Un modello è una definizione *funzionale* del comportamento di un sistema
  - ❑ Definisco come deve essere strutturata l'erogazione del servizio, non come implementare il servizio
  - ❑ ad un certo livello di *astrazione*
- ❖ L'implementazione sarà dipendente dalla definizione funzionale

## Modello client-server

- ❖ Presuppone l'esistenza di due entità

- ❑ **CLIENT**  
richiede un servizio
- ❑ **SERVER**  
eroga un servizio



- ❖ Esempi

- ❑ un risparmiatore e uno sportello bancomat
- ❑ la segretaria di un'azienda e gli impiegati che ci lavorano
- ❑ un correntista e la sua banca
- ❑ lo sportello bancomat al primo punto e un istituto di credito

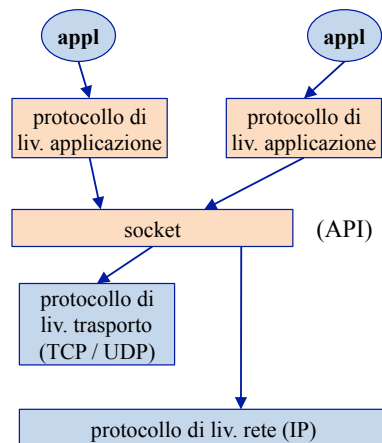
## Struttura applicazioni di rete

- ❑ user agent + protocollo client/server
- ❑ **user agent**: processo software, che si interfaccia con l'utente "sopra" e la rete "sotto".
  - ❑ implementa il protocollo dell'applicazione (**non** il protocollo di livello applicazione...)
  - ❑ Web: browser
  - ❑ E-mail: mail reader
  - ❑ streaming audio/video: media player
  - ❑ ... *ma non capisce nulla della rete!*

## inter-process communication

- ❖ **processi sul medesimo host:** /\* da Sistemi Operativi \*/
  - ❑ C: pipe, FIFO (named pipe), memoria condivisa
  - ❑ Java: RMI (remote method invocation), CORBA (Common Object Request Broker Architecture)
  - ❑ **SOCKET!**
    - più flessibili: bidirezionali, comunicazioni tra più parti, diversi tipi di servizi (con connessione o meno)
- ❖ **processi su host differenti:**
  - ❑ socket!

## Architettura generale



❖ **socket**: insieme di comandi che permettono di richiedere i servizi dei protocolli di rete, e modificarne il comportamento

❖ **multiplexing** sullo stesso host



❖ 2 componenti indirizzo:

- ❑ di rete (IP)
- ❑ locale: numero porta
  - es. 80 per http, 25 per mail

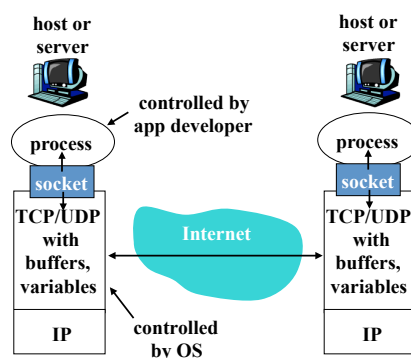
## socket: definizione

❖ “socket” = presa elettrica

➔ punto a cui ci si collega per usufruire di servizio (scambio dati)

❖ **socket**: libreria di funzioni (metodi / system call) che permettono di accedere ai servizi di *kernel* di S.O. per la rete

- ❑ forniscono *API*



## socket: dominio

- ❖ determina *contesto* in cui si vuole usare la socket
- ❖ dominio Unix: per comunicazione tra processi sul medesimo host
  - ❑ famiglia protocolli: AF\_UNIX / PF\_UNIX (\*)
  - ❑ indirizzamento: pathname
  - ❑ non trattati in questo corso
- ❖ dominio Internet
  - ❑ famiglia protocolli: AF\_INET / PF\_INET (\*)
  - ❑ dà accesso ai servizi di TCP / UDP / IP
  - ❑ usa *indirizzamento* di rete

(\*) AF per Address Family; PF per Protocol Family

## socket: indirizzi di rete (L3)

- ❖ come sapete indicare un computer remoto?
  - ❑ nomi simbolici: `www.unimi.it`
    - ma le stringhe sono difficili da trattare per i computer...
  - ❑ notazione decimale puntata: `159.172.87.15`
    - ma è ancora una stringa!
  - ❑ formato di rete: **unsigned long di 32 bit**
    - associato ad una interfaccia di rete
    - assegnato da IANA (dept. of ICANN)
    - indirizzo puntato è solo trascrizione “human readable” dei 4 ottetti

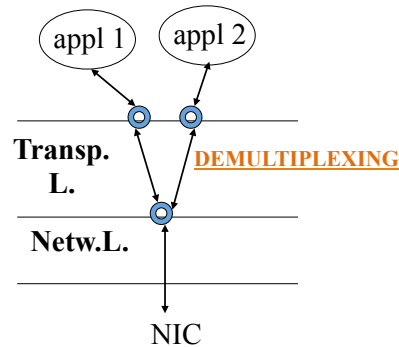
IANA = Internet Assigned Numbers Authority  
ICANN = Internet Corporation for Assigned Names and Numbers

➔ lezioni di Teoria per gestione di nomi e numeri

## socket: indirizzi di trasporto (L4)

- ❖ addressing: connessione tra 2 processi, invece che tra 2 host come a network L.
- ❖ distinzione tra diversi processi src/dest di dati su un host
- ❖ PID non va bene: remoto?
- ❖ **port**: ID unico legato a processo

- ❑ in TCP/IP: *unsigned short (16 bit)*
- ❑ **well-known** per servizi standard (0 ÷ 1023)



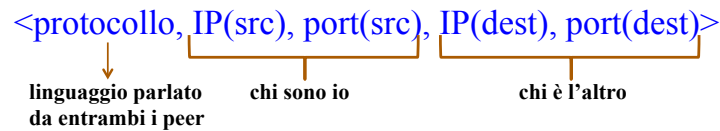
- ❑ **registered** per servizi di rete (1024 ÷ 49151)
- ❑ **dynamic** liberamente usabili (49152 ÷ 65535)

## indirizzo client vs. server

- ❖ il **server** deve **esistere** a *indirizzo facilmente determinabile*
  - ❑ es. tutti i servizi web si trovano alla porta 80
  - ❑ voglio il servizio web di Università di Milano
  - ❑ concatenazione indirizzo rete (IP) + porta → identificazione univoca server (+/- ...)
- ❖ il **client** si presenta con il proprio indirizzo al server *nel momento in cui lo contatta*
- ❖ in sistemi Unix, file **/etc/services** per porte well-known
  - ❑ applicazione indica servizio (peer) con cui vuole parlare
  - ❑ sistema effettua la traduzione

## socket: associazione

- ❖ socket definita da un **quintupla**:



- ❖ protocollo definito a creazione socket → *creazione di una struttura di sistema operativo*
  - richiamiamo l'architettura di slide 4...
- ❖ indirizzo locale definito quando si lega (**binding**) la socket al processo che la usa
- ❖ indirizzo peer: dipende da modello e tipo servizio

## indirizzo host: class InetAddress

- ❖ classe senza costruttore; ha metodi per convertire tra i diversi formati degli indirizzi IP (IPv4 o IPv6)
  - `InetAddress InetAddress.getByName(hostName)`
    - da nome simbolico a indirizzo numerico
  - `InetAddress[] InetAddress.getAllByName(hostName)`
    - tutti gli indirizzi associati a quel nome simbolico
  - `InetAddress InetAddress.getLocalHost()`
    - indirizzo dello host locale
- ❖ gestiscono struttura utile per lavorare con le socket
  - `byte[] InetAddress.getAddress()`
    - estrae indirizzo IP dalla struttura (ma serve conversione!)
  - `String InetAddress.getHostAddress()`
    - estrae indirizzo IP decimale puntato dalla struttura
  - `String InetAddress.getHostName()`
    - estrae nome simbolico host dalla struttura
- ❖ struttura a partire da indirizzo numerico fornita da
  - `InetAddress InetAddress.getByAddress(addr)`

## proviamoci un po'...

```
1 import java.net.InetAddress;
2 import java.net.UnknownHostException;
3
4 // codice gestione indirizzi IP / nomi simbolici
5
6 public class indirizzi
7 {
8     public static void main(String[] args)
9     {
10         String nome = "www.unimi.it"; // da convertire in indirizzo IP
11
12         try {
13             InetAddress ia = InetAddress.getByName(nome);
14             byte[] ndp = ia.getAddress();
15             System.out.println("Indirizzo: " + (ndp[0] & 0xff) + "." +
16                                 (ndp[1] & 0xff) + "." + (ndp[2] & 0xff) +
17                                 "." + (ndp[3] & 0xff));
18         }
19         catch (UnknownHostException uhe) {
20             uhe.printStackTrace();
21         }
22     }
23 }
```

convertiamo lo unsigned long (32 bit)  
in 4 ottetti di interi senza segno

Elena Pagani

LABORATORIO Reti di Calcolatori – A.A. 2019/2020

15 of 18

## homework

1. modificare codice in modo da estrarre l'indirizzo IP dello host locale su cui si lavora
  2. proviamo a usare *getAllByName* per estrarre tutti gli indirizzi IP per il dominio *www.google.com*
- ☐ serve ciclo lungo `<nome_struttura>.length`

```
10 String nome = "www.google.com"; // da convertire in indirizzo IP
11
12 try {
13     InetAddress[] iaa = InetAddress.getAllByName(nome);
14
15     for (int i=0; i < iaa.length; i += 1)
16     {
17         System.out.println("Indirizzo " + iaa[i].getHostName() +
18                             " --> " + iaa[i].getHostAddress());
19     }
20 }
21 catch (UnknownHostException uhe) {
22     uhe.printStackTrace();
23 }
```

❖ come funziona? interroga il *resolver* (→ Teoria!)

Elena Pagani

LABORATORIO Reti di Calcolatori – A.A. 2019/2020

16 of 18



## costruzione indirizzo per socket

- ❖ abbiamo visto che serve indirizzo host + #port
- ❖ per server: numero porta ben noto
  - ❑ ... o comunicato ai potenziali client
- ❖ per client: numero qualsiasi, anche scelto da S.O.
  - ❑ è iniziatore: al 1° messaggio dà #port a server per risposta
- ❖ class `InetSocketAddress` con costruttori:
  - ❑ `InetSocketAddress(InetAddress addr, int port)`
  - ❑ `InetSocketAddress(String hostname, int port)`
  - ❑ `InetSocketAddress(int port)`
- ❖ e con metodi utili:
  - ❑ `InetAddress getAddress()`
  - ❑ `String getHostName()`
  - ❑ `int getPort()`

## proviamo un po'...

```
1 import java.net.Socket;
2 import java.net.InetAddress;
3 import java.net.InetSocketAddress;
4 import java.net.UnknownHostException;
5
6 // codice client per servizio connection-oriented (TCP)
7
8 public class esempio1
9 {
10     public static void main(String[] args)
11     {
12         Socket sClient;
13         InetAddress ia; // IP address client
14         InetSocketAddress isa; // socket address client
15
16
17         sClient = new Socket();
18         try {
19             ia = InetAddress.getLocalHost();
20             isa = new InetSocketAddress(ia, 50000);
21         }
22         catch (UnknownHostException uhe) {
23             uhe.printStackTrace(); }
24     }
25 }
```

scelta porta in codice  
...ma potrebbe essere già in uso!