Socket (TCP/IP)

M. Danelutto
LPRb A.A. 2007-2008

Socket (TCP/IP)

M. Danelutto
LPRb A.A. 2007-2008

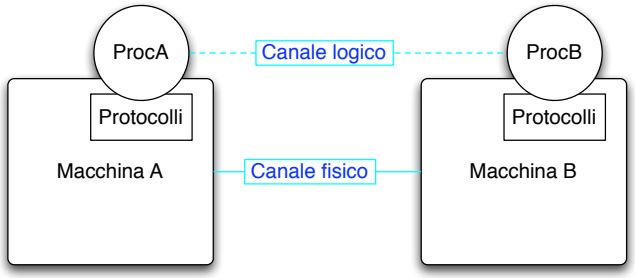
Concetto di socket

- Astrazione "presa di rete"
- Comunicazione fra processi su macchine diverse
  - O fra processi della stessa macchina
- Supportato da protocolli di tipo diverso

Cenni (reti)

- Protocolli TCP/IP (trasporto/rete)
  - astrazione di stream bidirezionale
- Protocolli UDP/IP
  - astrazione a scambio di messaggi
- Li vedrete abbondantemente nel corso di reti

Comunicazione (stream)



Modello di comunicazione

- Orientato alla connessione
  - apertura della connessione
    - connessione + accettazione
  - flusso bidirezionale (Input/OutputStream)
  - chiusura della connessione

Apertura della connessione

- Asimmetria
  - server: "pubblica" un socket (indirizzo)
  - client: "si connette" ad un socket
  - server: "accetta" la connessione
- Connessione può fallire
  - indirizzo sbagliato, server non pronto, ...

Indirizzi da utilizzare

- Indirizzi IP (fwd: corso di reti)
  - IPv4: 32 bit (4 byte: 131.114.11.234)
  - IPv6: 128 bit (8 * 16 bit hex 1000:23ef: 1e10:3432::1f1f)
- Numero di porta (16 bit)
- Indirizzo = < IPnumber, port >

Numeri di porta

- 0-1024 : riservate (di sistema)
  - 80 http, 443 https, 22 ssh, 21 ftp, 25 smtp, ...
- 1024-65535 :"effimere"
  - alcune sono utilizzate comunque
    - 8080 proxy web

Indirizzi: IPnumber

- Associazione unica mantenuta fra nomi e IP
- da autorità mondiale (evita conflitti)
- Dal nostro punto di vista
  - InetAddress ipNumber = InetAddress.getByName ("nomeMacchina");
  - throws UnknownHostException

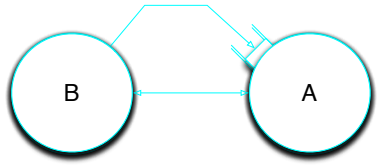
Utilizzo connessione

- Client e server
  - ottengono InputStream e OutputStream
  - read/write sugli stream esattamente come se fossero stream su disco
  - possibilità di sovrapporre diversi tipi di I/O
    - buffered, object, ...

Chiusura della connessione

- operazioni esplicite
- sui socket
  - sottintendono chiusura stream associati

Come funziona (logico)



API Java (java.net)

- Pubblicazione di un socket (lato server)
- ServerSocket ss =
  new ServerSocket(int port)
- rende disponibile un socket (server) su cui si possono attaccare altri processi (clienti)

Varianti

- new ServerSocket()
  - porta scelta dal sistema
  - ss.getLocalPort()

Java API (java.net)

- accettazione della connessione
- ServerSocket ss = ... ;
  Socket s = ss.accept();
- chiamata bloccante
  - ritorna quando qualcuno si è connesso al server socket

Java API (java.net)

- Connessione a un processo che ha pubblicato un server socket
- Socket s =
  new Socket(InetAddress ia, int port);
- throws IOException in caso di errore

Fase di connessione

- ***server*** ***client***
- ServerSocket ss =
  new ServerSocket(port);
- Socket s = ss.accept();
- Socket s =
  new Socket(ip,port);

Java API (java.net)

- instaurazione del canale bidirezionale di tipo stream
- lato client e lato server
- Socket s = ... ; // new Socket(...) o accept()
  InputStream is = s.getInputStream();
  OutputStream os = s.getOutputStream();

Scrittura dati su socket

- write(byte[] buffer, int offset, int len);
  - scrive len byte di buffer a partire da offset
- write(byte[] buffer);
- scrittura non bufferizzata
- protocollo TCP/IP garantisce il delivery (a meno di eccezioni)

Lettura dati da socket

- int read(byte[] buffer, int offset, int len);
- legge nel buffer un massimo di len bytes a partire da offset (BLOCCANTE!)
- restituisce il numero di byte letti o -1 se non c'e' nient'altro da leggere
- non necessariamente (su InputStream da socket) legge len bytes !!!

Lettura (2)

- while (letti != tutti) {
  lettiOra = read(buffer, offset, maxbuf);
  if(lettiOra > 0) {
  letti += lettiOra;
  offset+=lettiOra;
  maxbuf -= lettiOra;
  } else break;
  }

BufferedStream

- BufferedReader in =
  new BufferedReader(
  new InputStreamReader(s.getInputStream());
- BufferedWriter out =
  new BufferedWriter(
  new OutputStreamReader(s.getOutputStream
  ());

BufferedStream (2)

- String s = in.readLine();
- ATTENZIONE
  - buffering in generale non controllabile
  - su socket sono implementati con buffering diverso ...

Chiusura della connessione

- Socket s = ... ;
  s.close();
- chiude la connessione corrente
- va fatta lato client e lato server

Applicazioni client/server

- N clienti
- 1 server
- Server: pubblica ServerSocket
  accetta richieste di connessione
  forka un thread per ogni richiesta di connessione

Ruolo del server

- Parte importante dell'applicazione
  - senza il server non hanno senso i clienti
- Protocollo di comunicazione del server
  - definisce la modalità di fruizione del servizio

Client

- Devono seguire il protocollo del server
- possono essere sviluppati indipendentemente
- Esempio: client HTTP
  - stabilisce connessione su porta 80 e manda un "HTTP 1.0 GET filename" (vedi corso di reti)

Server tipico (singlethreaded)

- ServerSocket ss = ...
  while(true) {
  Socket s = ss.accept();
  InputStream is = s.getInputStream();
  OutputStream os = s.getOutputStream
  ();
  // legge richiesta dal client
  // prepara risposta e la scrive
  s.close();

Server tipico (multithreaded)

- ServerSocket ss = ...
  while(true) {
  Socket s = ss.accept();
  Thread t = new ThreadHandler(s);
  t.start();
  }
- normalmente utilizza un threadPool ...

Caratteristiche del canale

- Tutto quello che scrivo su un socket arriva sul socket destinazione
- Se ho errori, in generale avrò un'eccezione
- Non garanzie sui tempi di delivery
- Problemi di ottimizzazione con TCP (fwd)

Flush

- scrittura su OutputSocket o BufferedStream
- se buffered
  - stream.flush();
    provoca la spedizione effettiva dei byte scritti fino a quel momento (vedi reti fra un pò)

Appl tipiche: trasferimento file

- Server: pubblica socket e fa accept
- Client: si connette e invia nomefile CR file
- Server: legge nome file, lo apre e poi entra in un ciclo di copia da socket a file
- più file => accept in un ciclo ...

Appl tipiche: erogazione di un servizio

- Server: pubblica un socket, entra in un ciclo che comincia con accept e prosegue con la fork di un thread che tratta la connessione
- Client: si connette, manda il messaggio di richiesta e aspetta il messaggio di risposta

Applicazione tipica: cliente di servizio std

- Cliente: si connette al server sulla well know port
- manda la richiesta
- attende la risposta
  - TELNET

Documentazione

- java.net
  - Socket, ServerSocket, InetAddress, accept, close
- java.io
  - InputStream, OutputStream, BufferedReader/Writer, read, write, readline, flush