Lezione n.7
LPR-A-09
TCP Sockets & Multicast

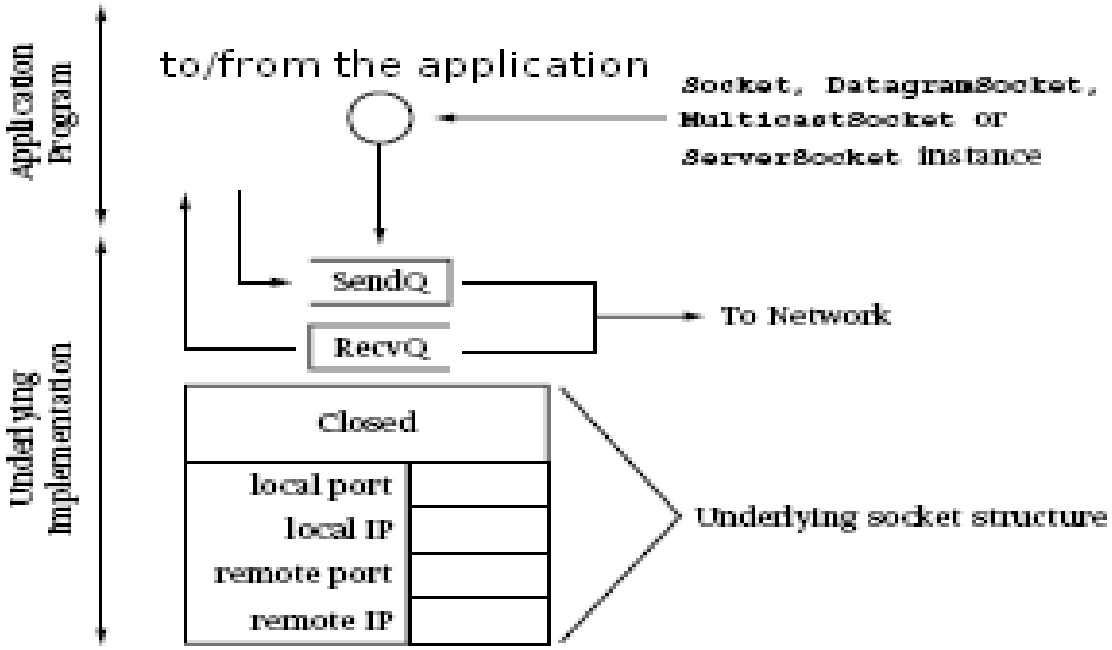
17/11/2009

Vincenzo Gervasi

Sommario

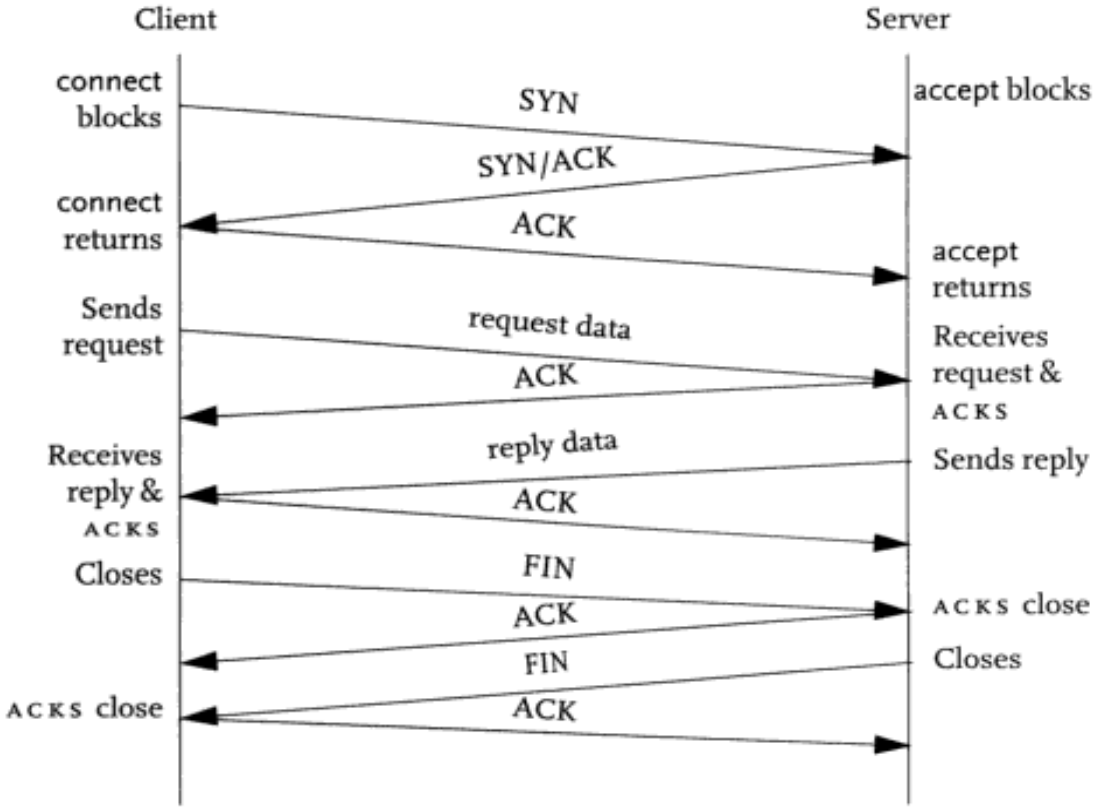
- Ancora sugli stream socket e su three-ways handshaking di TCP
- De-multiplexing di frammenti TCP
- Invio di oggetti tramite TCP con serializzazione (rischio di deadlock)
- Qualcosa sugli esercizi...
- Unreliable Multicast: concetti e API Java
- Panoramica su Linux Networking Tools (grazie a Daniele Sgandurra)

STRUTTURA GENERALE DI UN SOCKET

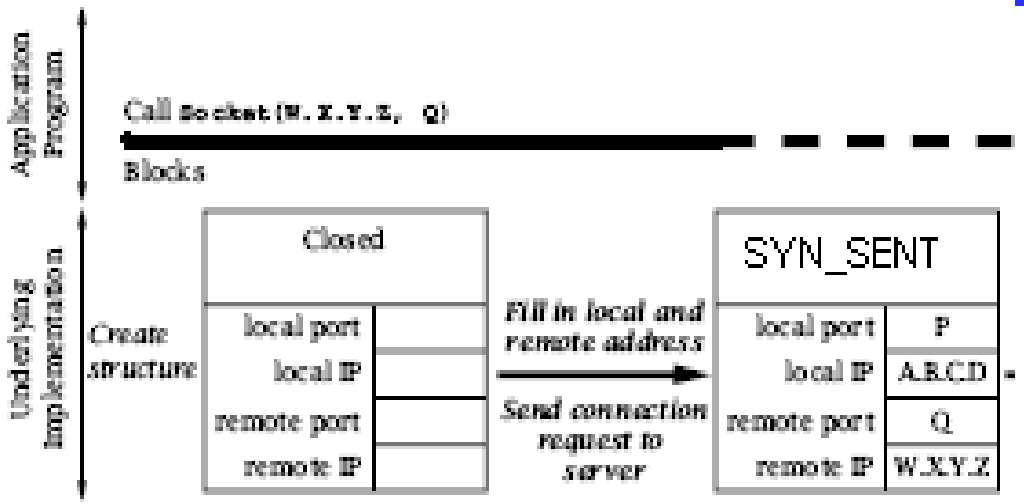


- remote port ed host significative solo per socket TCP
- SendQ, RecQ: buffer di invio/ricezione
- ogni socket è caratterizzato da informazioni sul suo stato (ad esempio closed). Lo stato del socket è visibile tramite il comando netstat

TCP: GESTIONE DELLE CONNESSIONI



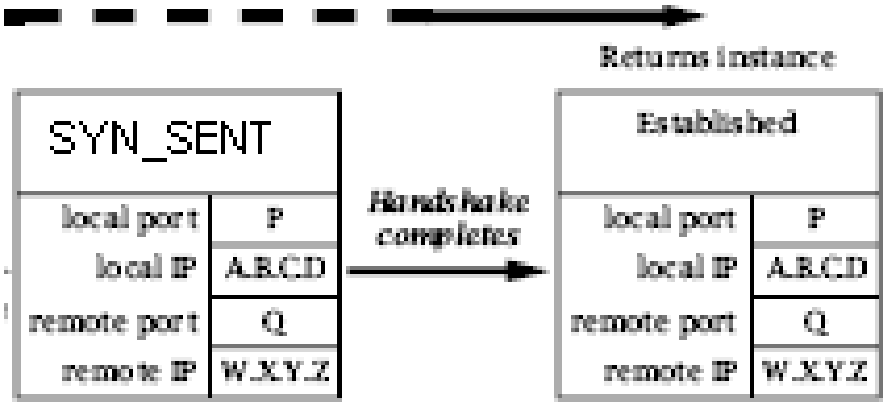
CONNESSIONE LATO CLIENT: STATO DEL SOCKET



Quando il client invoca il costruttore Socket( ).

- lo stato iniziale del socket viene impostato a Closed, la porta (P) e l'indirizzo locale (A.B.C.D) sono impostate dal supporto
- dopo aver inviato il messaggio iniziale di handshake, lo stato del socket passa a SYN_SENT (inviato segmento SYN)
- il client rimane bloccato fino a che il server riscontra il messaggio di handshake mediante un ack

CONNESSIONE LATO CLIENT: STATO DEL SOCKET

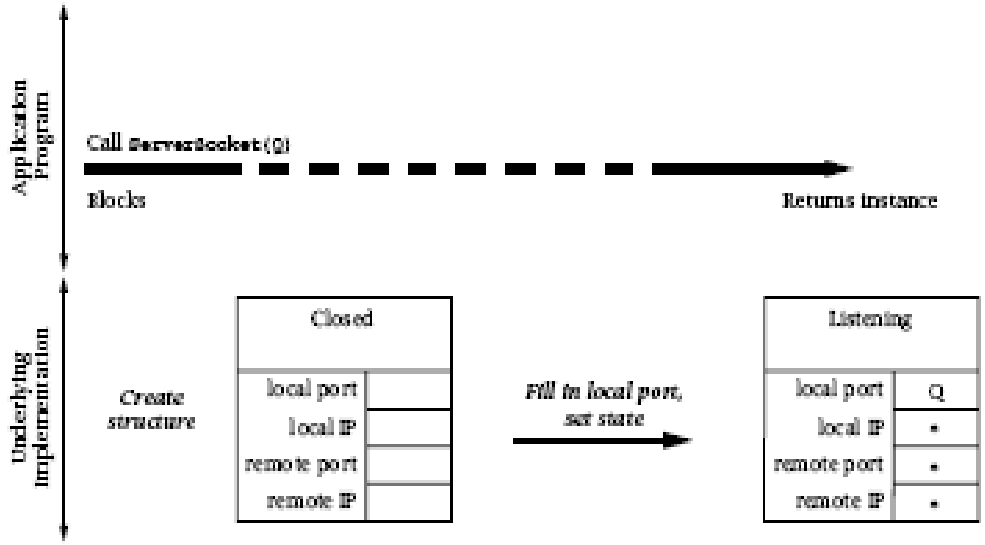


- il messaggio di handshake può venire trasmesso più volte
- il client può rimanere bloccato per un lungo periodo.

Il costruttore può sollevare una eccezione se,

- non esiste il servizio richiesto sulla porta selezionata
- il messaggio di handshake non viene riscontrato entro un certo intervallo di tempo(timeout)

CONNESSIONE LATO SERVER: STATO DEL SOCKET



Il Server crea un server socket sulla porta P

- se non viene specificato alcun indirizzo IP (wildcard = *), il server può ricevere connessioni da una qualsiasi delle sue interfacce
- lo stato del socket viene posto a Listening: questo indica che il server sta attendendo connessioni da una qualsiasi interfaccia, sulla porta P

CONNESSIONE LATO SERVER: STATO DEL SOCKET

- il server si sospende sul metodo accept( ) in attesa di una nuova connessione
- quando riceve una richiesta di connessione dal client, crea una nuova struttura che implementa il nuovo socket creato. In tale struttura
  - indirizzo e porta remoti vengono inizializzati con l'indirizzo IP e la porta ricevuti dal client che ha richiesto la connessione
  - L'indirizzo locale viene settato con l'indirizzo dell'interfaccia da cui è stata ricevuta la connessione.
  - La porta locale viene inizializzata con quella a cui associata al server socket
  - Lo stato del nuovo socket è SYN_RCVD

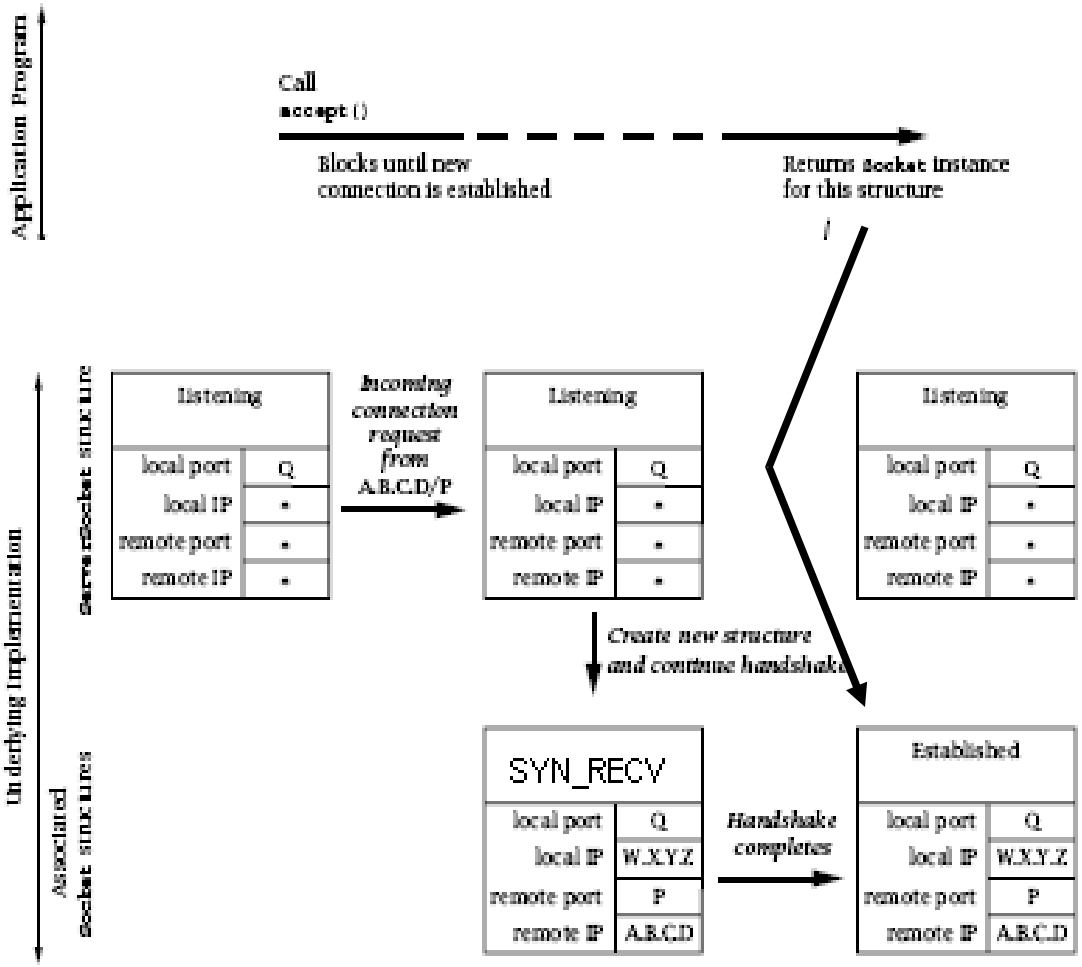
    è stato inviato il SYN/ACK al client e si sta attendendo l'ACK dal client, per terminare il 3-way handshake

CONNESSIONE LATO SERVER: STATO DEL SOCKET

Dopo aver creato il nuovo socket, il server

- riscontra il **SYN** inviato dal client mediante un **SYN/ACK**
- quando riceve, a sua volta, il riscontro dal client (**ACK**, terzo messaggio del 3-way handshake)
  - imposta lo stato del socket ad ESTABLISHED
  - inserisce il socket creato in una lista di socket associata al ServerSocket da cui è stata ricevuta la richiesta di connession
  - Solo a questo punto, l'esecuzione del metodo accept( ) termina e restituisce un puntatore alla struttura creata
- Anche il client imposta lo stato del proprio socket ad ESTABLISHED dopo aver terminato il 3-way handshake

CREAZIONE DI CONNESSIONI LATO SERVER



DEMULTIPLEXING DEI SEGMENTI TCP

- Tutti i sockets associati allo stesso ServerSocket 'ereditano' da esso
  - la porta di ascolto
  - l' indirizzo IP da cui è stata ricevuta la richiesta di connessione
- Questo implica che sullo stesso host possano esistere più sockets associati allo stesso indirizzo IP ed alla stessa porta locale (il **Server Socket** e tutti i **Sockets** associati,.....)
- **Meccanismo di demultiplexing**: utilizzato per decidere a quale socket è destinato un segmento TCP ricevuto su quella interfaccia e su quella porta
- La conoscenza dell'indirizzo e porta destinazione non risulta più sufficiente per individuare il socket a cui è destinato il segmento

DEMULTIPLEXING DEI SEGMENTI TCP

Definizione del meccanismo di demultiplexing:

- La porta locale riferita nel socket deve coincidere con quella contenuta nel segmento TCP
- Ogni campo del socket contenente una wildcard (*), può essere messo in corrispondenza con qualsiasi valore corrispondente contenuto nel segmento
- Se esiste più di un socket che corrisponde al segmento in input, viene scelto il socket

  che contiene il minor numero di wildcards.
- in questo modo un segmento spedito dal client viene ricevuto sul socket S associato alla connessione con quel client, piuttosto che sul serversocket perchè S risulta 'più specifico' per quel segmento

INVIO DI OGGETTI SU CONNESSIONI TCP

- Per inviare oggetti su una connessione TCP basta usare la serializzazione: gli oggetti inviati devono implementare l'interfaccia Serializable
- Si possono usare i filtri ObjectInputStream / ObjectOutputStream per incapsulare gli stream ottenuti invocando getInputStream() / getOutputStream() sul socket
- Quando creo un ObjectOutputstream viene scritto lo stream header sullo stream. In seguito scrivo gli oggetti che voglio inviare sullo stream
- L'header viene letto quando viene creato il corrispondente ObjectInputStream
- L'invio/ ricezioni degli oggetti sullo/dallo stream avviene mediante scritture/letture sullo stream (writeObject(), readObject())

INVIO DI OGGETTI SU UNA CONNESSIONE TCP

```
import java.io.*;

    public class Studente implements Serializable {
    private int matricola;
    private String nome, cognome, corsoDiLaurea;
    public Studente (int matricola, String nome, String cognome,
                        String corsoDiLaurea) {
        this.matricola = matricola; this.nome = nome;
        this.cognome = cognome; this.corsoDiLaurea = corsoDiLaurea;}
    public int getMatricola () { return matricola; }
    public String getNome () { return nome; }
    public String getCognome () { return cognome; }
    public String getCorsoDiLaurea () { return corsoDiLaurea; }   }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP- LATO SERVER

```
import java.io.*; import java.net.*;
public class TCPObjectServer {
public static void main (String args[]) {
try { ServerSocket server = new ServerSocket (3575);
      Socket clientsocket = server.accept();
      ObjectOutputStream output =
                  new ObjectOutputStream (clientsocket.getOutputStream ());
      output.writeObject("<Welcome>");
      Studente studente = new Studente (14520,"Mario","Rosso","Informatica");
      output.writeObject(studente);  output.writeObject("<Goodbye>");
      clientsocket.close();
      server.close();
} catch (Exception e) { System.err.println (e); } } }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP-LATO CLIENT

```
import java.io.*; import java.net.*;
public class TCPObjectClient { public static void main (String args[ ]) {
try { Socket socket = new Socket ("localhost", 3575);
ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
String beginMessage = (String) in.readObject();
Studente stud = (Studente) in.readObject();
System.out.println(beginMessage);
System.out.print(stud.getMatricola() + " - " + stud.getNome() + " ");
System.out.println(stud.getCognome() + " - " + stud.getCorsoDiLaurea());
String endMessage = (String) in.readObject();
System.out.println (endMessage); socket.close();} catch (Exception e)
   { System.out.println (e); } } }
```

INVIO DI OGGETTI SU UNA CONNESSIONE TCP- LATO CLIENT

Stampa prodotta lato Client

<Welcome>

14520 - Mario Rossi - Informatica

<Goodbye>

OBJECT INPUT/OUTPUT STREAM: RISCHIO DI DEADLOCK

- La creazione dell'ObjectInputStream cerca di leggere lo header. Se questo non è stato ancora creato, si blocca.
- Quindi **si verifica una situazione di deadlock** se i due partner della connessione eseguono le istruzioni nel seguente ordine (s è il socket locale):

```
ObjectInputStream in = new ObjectInputStream(s.getInputStream( ));
ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream( ));
```

Infatti,

- entrambi tentano di leggere l'header dello stream dal socket
- l'header viene generato quando viene viene creato l'ObjectOutputStream
- nessuno dei due è in grado di generare l'ObjectOutputStream, perchè bloccato
- E' sufficiente invertire l'ordine di creazione degli stream in almeno uno dei partner

ESERCIZIO:ASTA ELETTRONICA

Sviluppare un programma client server per il supporto di un'asta elettronica. Ogni client possiede un budget massimo B da investire. Il client può richiedere al server il valore V della migliore offerta pervenuta fino ad un certo istante e decidere se abbandonare l'asta, oppure rilanciare. Se il valore ricevuto dal server supera B,l'utente abbandona l'asta, dopo aver avvertito il server. Altrimenti, il client rilancia, inviando al server un valore maggiore di V. Il server invia ai client che lo richiedono il valore della migliore offerta ricevuta fino ad un certo momento e riceve dai client le richieste di rilancio. Per ogni richiesta di rilancio, il server notifica al client se tale offerta può essere accettata (nessuno ha offerto di più nel frattempo), oppure è rifiutata.

ESERCIZIO:ASTA ELETTRONICA

Il server deve attivare un thread diverso per ogni client che intende partecipare all'asta.

La comunicazione tra clients e server deve avvenire mediante socket TCP. Sviluppare due diverse versioni del programma che utilizzino, rispettivamente una codifica testuale dei messaggi spediti tra client e server oppure la serializzazione offerta da JAVA in modo da scambiare oggetti tramite la connessione TCP