Esercitazione n.5b
LPR-A-09
Il protocollo UDP: DatagramPacket

10/11/2009

Vincenzo Gervasi

ESERCIZIO 1: UNA CLASSE AUSILIARIA PER TRASMETTERE OGGETTI

Scrivere la classe **Obj2DP** che fornisce due metodi statici:

- **public static** Object **dp2obj**(DatagramPacket dp)

  che restituisce l'oggetto contenuto nel pacchetto passato per argomento, deserializzandolo

- **public static** DatagramPacket **obj2dp**(Object obj)

  che restituisce un pacchetto contenente l'oggetto passato per argomento, serializzato, nel payload.

- Semplificare le classi UDP_SendObject e UDP_ReceiveObject viste prima usando i metodi della classe Obj2DP, senza più usare le classi ObjectOutput/InputStream e ByteOutput/InputStream.
- Usare la classe **Obj2DP** per i prossimi esercizi, trasmettendo oggetti serializzati con UDP invece di dati di tipi primitivi.

ESERCIZIO 2: MiniTalk Server e Client con UDP (1)

Si realizzino un Server e un Client UDP che realizzano un semplice Instant Messanger: ogni linea scritta nella shell del Server viene copiata nella shell del Client e viceversa. La trasmissione delle linee inizia dopo una semplice fase di handshaking, descritta di seguito.

- Il Server viene lanciato da shell, fornendo il numero di porta su cui ricevere i pacchetti UDP.
- Il Client viene lanciato da un'altra shell (eventualmente su un altro computer), fornendo host e porta del Server e porta locale su cui ricevere pacchetti UDP.
- Il Client manda una richiesta di connessione al Server, indicando nel messaggio la propria porta. Se non riceve un messaggio di ack entro 3 secondi, riprova a mandare la richiesta altre 5 volte, poi termina. Se invece riceve l'ack, inizia la trasmissione delle linee scritte nella shell locale e la ricezione e stampa delle linee scritte nella shell del Server.

ESERCIZIO 2: MiniTalk Server e Client con UDP (2)

- Quando il Server riceve una richiesta di connessione, recupera indirizzo e porta del Client dalla richiesta e manda un messaggio di ack al Client, quindi comincia la trasmissionex e ricezione delle stringhe come descritto per il Client.
- Il Client decide quando disconnettersi in base a una condizione a vostra scelta (per esempio, allo scadere di un timeout, oppure se la linea da mandare è vuota, oppure se la stringa è "CLOSE",...).
- Per disconnettersi, il Client manda una richiesta di disconnessione al Server e aspetta un ack, quindi termina l'esecuzione.
- Quando il Server riceve una richiesta di disconnessione interrome la trasmissione delle linee, manda un messaggio di ack, e si rimette in attesa di una richiesta di connessione.

Il Client e il Server devono scambiarsi **unicamente** oggetti della classe **TalkMsg**, usando i metodi della classe per crearne istanze e per ispezionare i messaggi arrivati.

ESERCIZIO 3: MiniTalk Messanger con UDP

Riusando il più possibile il codice sviluppato per l'esercizio precedente, realizzare un programma Messanger che offre le stesse funzionalità, ma in cui non si distinguono un Server e un Client.

Due istanze di Messanger devono essere lanciate in due shell diverse, fornendo ad ognuna tre dati: la porta locale, e l'host e la porta dell'altro Messanger. Ideare un opportuno protocollo di handshaking, che permetta di stabilire una connessione (concettuale) tra le due istanze di Messanger.

I messaggi scambiati devono essere tutti oggetti di una stessa classe. Usare la classe **TalkMsg**, oppure estenderla o definirne una analoga se necessario.

ESERCIZIO 4: TFTP con UDP (Trivial File Transfer Protocol) [1]

Questa è la specifica di TFTP da WIKIPEDIA:

- L'host A invia un pacchetto RRQ (read request) o WRQ (write request) all'host B, contenente il nome del file e la modalità di trasferimento.
- B risponde con un ACK (acknowledgement) packet, che serve anche a dire ad A quale porta sull'host B dovrà usare per i restanti pacchetti.
- L'host di origine invia dei pacchetti DATA numerati all'host di destinazione, tutti tranne l'ultimo contenenti un blocco di dati completo. L'host di destinazione risponde con un pacchetto ACK numerato per ogni pacchetto DATA.
- Il pacchetto DATA finale deve contenere un blocco di dati non pieno ad indicare che si tratta dell'ultimo. Se la dimensione del file trasferito è un multiplo esatto della dimensione dei blocchi, la sorgente invia un ultimo pacchetto di dati contente 0 byte di dati.

ESERCIZIO 4: TFTP con UDP (Trivial File Transfer Protocol) [2]

Realizzare un Server TFTP che implementa il comportamento dell'host B e un Client TFTP che implementa l'host A. In particolare:

- Client e Server devono scambiarsi solo oggetti di una classe, **TFTPmsg**, usati sia per messaggi di servizio (RRQ, WRQ, ACK) che per i pacchetti DATA: definire opportunamente la classe **TFTPmsg**.
- Per il trasferimento dei file, considerarli come file binari, usando quindi opportuni Output/InputStreams (e non Writer/Reader).
- Inviare le porzioni di file in array di byte all'interno di un'istanza di **TFTPmsg**.

Per testare il programma:

- Confrontare il file originale spedito dal mittente con quello ricevuto dal destinatario e scritto nel file system.
- Usare la classe UnreliableDatagramSocket per controllare che i pacchetti persi vengano reinviati correttamente.