Esercizi su Invio di Messaggi su TCP

Esercitazione di Laboratorio di Programmazione di Rete A

Daniele Sgandurra

Università di Pisa

19/11/2008

Framing

- Dopo aver inviato i dati sul socket, l'informazione deve essere recuperata in maniera corretta dal destinatario a partire dalla sequenza di byte trasmessi.
- Con il termine framing ci si riferisce al problema di come permettere al destinatario di individuare l'inizio e la fine di un messaggio.
- Il protocollo a livello dell'applicazione deve definire le modalità tramite le quali il destinatario riconosce di aver ricevuto tutto il messaggio.

Framing su UDP e TCP

- Se un messaggio completo viene inviato come payload di un `DatagramPacket`, il problema è semplice: dato che il payload ha una lunghezza definita, il ricevente sa esattamente dove il messaggio termina.
- Su TCP (stream di byte) la situazione è diversa, in quanto TCP non ha la nozione di delimitatore di messaggio:
  - se il messaggio ha un numero fissato di campi di lunghezza fissa, la dimensione è nota: il ricevente legge dal socket il numero atteso di byte in un buffer di `byte[]`;
  - se il messaggio ha lunghezza variabile, il destinatario non sa a priori quanti byte leggere dal socket.

Senza Framing su TCP

- Se il destinatario prova a leggere dal socket più byte di quelli presenti nel messaggio inviato, possono accadere una delle seguenti condizioni:
  - se nessun altro messaggio è sul socket, il destinatario rimarrà bloccato sulla `read`, senza possibilità di processare il messaggio ricevuto; se il mittente è inoltre bloccato in attesa di una risposta, ci sarà una situazione di deadlock;
  - se invece il mittente ha inviato un altro messaggio, il ricevente leggerà parte dei nuovi dati come se fossero del primo messaggio, causando errori nel protocollo.

Framing su TCP

- Due tecniche generali che possono essere usate per determinare la fine di un messaggio sono:
  1. la fine del messaggio viene indicata da un delimitatore unico, ad esempio una sequenza di byte nota che il mittente trasmette subito dopo aver inviato il messaggio;
  2. il messaggio viene preceduto da un campo (di lunghezza fissa) che indica di quanti byte sarà composto il messaggio successivo.

Delimitatore

- Questo approccio è spesso usato quando i messaggi sono codificati come testo.
- Un carattere particolare, o una sequenza di caratteri, marca la fine di un messaggio.
- Il ricevente, legge i caratteri di input ricevuti cercando il delimitatore: tutto quello che sta prima del delimitatore forma il messaggio.
- Problema: il messaggio stesso non deve contenere il delimitatore.

Prefisso con Indicatore di Lunghezza (1)

- Approccio più semplice, ma richiede un limite superiore alla lunghezza del messaggio.
- Il limite superiore per la lunghezza determina il numero di byte richiesti per codificare la lunghezza complessiva del messaggio.
- Ad esempio, un byte se il messaggio è sempre minore di 256 byte, due byte se il messaggio è sempre minore di 65.536 byte, e così via.

Prefisso con Indicatore di Lunghezza (2)

- Il mittente, deve calcolare la lunghezza del messaggio, codificarla come un intero, e inviarla prima del messaggio.
- Il destinatario:
  1. legge dal socket il numero di byte usati per codificare il messaggio;
  2. li decodifica in un intero indicante la lunghezza in byte del messaggio;
  3. infine legge dal socket tanti byte quanti indicati in precedenza: questi formano il messaggio.

Esempio di Prefisso con Indicatore di Lunghezza (1)

```
public class LengthFramer
{
   public static final int MAX_MESSAGE_LENGTH = 65535;
   public static final int BYTEMASK = 0xff;
   public static final int SHORTMASK = 0xffff;
   public static final int BYTESHIFT = 8;
   private DataInputStream in;

   public LengthFramer(InputStream in) throws IOException
   {
      this.in = new DataInputStream(in);
   }

   public void sendMsg(byte[] message, OutputStream out) throws IOException
   {
      if(message.length > MAX_MESSAGE_LENGTH)
      {
         throw new IOException("messaggio troppo lungo");
      }
      //il prefisso del messaggio specifica la sua lunghezza:
      //due byte in formato big endian (prima invio quello di indirizzo maggiore)
      out.write((message.length » BYTESHIFT) & BYTEMASK);
      out.write(message.length & BYTEMASK);
      //invio il messaggio
      out.write(message);
      out.flush();
   }
```

Esempio di Prefisso con Indicatore di Lunghezza (2)

```
    public byte[] getMsg() throws IOException
    {
        int length;
        try
        {
            length = in.readUnsignedShort(); // legge 2 byte
        }catch(EOFException e) return null;
        // 0 <= length <= 65535
        byte[] msg = new byte[length];
        in.readFully(msg); // legge fino alla lunghezza max dell'array
        return msg;
    }
}
```

Costruzione del Messaggio

- Una volta determinato il formato del messaggio (rappresentato da un oggetto in Java), e scelto il framing, bisogna definire come codificare il messaggio per l'invio:
  1. rappresentazione testuale (ad esempio HTTP): si inviano i campi dell'oggetto come stringhe, separati da un carattere speciale (metodo più generico);
  2. rappresentazione binaria: ad es., ogni messaggio ha una dimensione fissa, e si inviano tutti i campi dell'oggetto in un ordine prestabilito, usando un `DataOutputStream`;
  3. sfruttando la serializzazione di Java (funziona solo con applicazioni Java).

L'interfaccia `Serializable`

- Java fornisce un meccanismo automatico di serializzazione, che richiede che l'oggetto implementi l'interfaccia `java.io.Serializable`:
  - in questi casi, Java gestisce la codifica/decodifica e il framing internamente.
- Il metodo di codifica utilizzato da Java consiste nella traduzione di ogni campo dell'oggetto in un stream di byte.
- Ogni campo dell'oggetto da inviare deve essere serializzabile (eccetto i campi marcati come `transient`).
- Alcuni oggetti, come Thread, Socket, ObjectOutputStream, etc, non sono serializzabili: controllare le API.

Esempio: Serializzazione

```
public class Person implements Serializable
{
   private String name;
   private int id;

   public Person(String s, int n)
   {
      this.name = s;
      this.id = n;
   }
  ...
}

//per inviare su socket
p = new Person(...);
objOut = new ObjectOutputStream(socket.getOutputStream());
objOut.writeObject(p);
objOut.flush();

//per ricevere da socket
objIn = new ObjectInputStream(socket.getInputStream());
p = (Person)objIn.readObject();

//per salvare su file
ObjectOutputStream objstream = new ObjectOutputStream(new FileOutputStream(filename));
objstream.writeObject(p);
objstream.close();

//per leggere da file
ObjectInputStream objstream = new ObjectInputStream(new FileInputStream(filename));
p = (Person)objstream.readObject();
objstream.close();
```

Esercizio 1

- 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 1

- Il server deve attivare un thread diverso per ogni client che intende partecipare all'asta. La comunicazione tra client 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,
  - la serializzazione offerta da JAVA in modo da scambiare oggetti tramite la connessione TCP.

Soluzioni

Inviare la soluzione degli esercizi (solo i file .java) a :

`ricci@di.unipi.it`
`sgandurra@di.unipi.it`

Tra due settimane saranno disponibili le soluzioni.