Übung 5 zur Vorlesung "Vernetzte Systeme" WS 1999/2000

Prof. Dr. F. Mattern

Ausgabedatum: 17. Nov. 1999
Abgabedatum: 24. Nov. 1999

20. Aufgabe (Bit-orientierte synchrone Übertragung)

Um einen kontinuierlichen Bit-Strom in einzelne *Frames* unterteilen zu können (die dann einzeln im Fehlerfalle erneut übertragen werden), umgibt man eine variable Anzahl von Datenbits mit öffnenden und schliessenden Markern, z.B. der 8-bit Folge 01111110. Damit zufällig im Datenstrom befindliche Bit-Muster, die mit diesen Markern identisch sind, nicht als Marker missverstanden werden, fügt der Sender nach jeder Serie von 5 aufeinanderfolgenden Einsen eine Null in den Datenstrom ein. Dadurch kann das Marker-Muster 01111110 niemals zwischen den beiden Markern auftreten. Dieses Verfahren ist als *bit-stuffing* bekannt. Auf der Empfängerseite muss dementsprechend jede Null, die nach fünf Einsen empfangen wurde, aus dem Datenstrom entfernt werden.

a) (1 Punkt) Wie sieht der Datenstrom "0111011111101111110" nach erfolgtem *bit-stuffing* aus?

b) (1 Punkt) Wie sieht der Original-Datenstrom aus, wenn folgender (*bit-stuffed*) Bit-String beim Empfänger eintrifft: "01111101101100111110110111110001111110"?

21. Aufgabe (OSI Schichtenmodell)

(4 Punkte) Ordnen Sie die folgenden Operationen, die beim "Surfen" im World-Wide-Web auftreten, der jeweiligen OSI-Schicht zu:

- Übertragung der einzelnen Bits.
- Anfrage in HTTP (z.B. `GET /index.html HTTP/1.0`).
- DNS-Lookup (Übersetzung der Zieladresse von einem String, z.B. *www.ethz.ch*, in eine numerische Adresse, z.B. *129.132.200.35*).

- Host-to-host Routing.
- Umwandlung der zu sendenden Nachricht in kleine Datenpakete.
- Anpassung der Paketgrösse and die für das Netz zulässige Maximalgrösse.
- Anpassung an einen Standard-Zeichensatz (das Betriebsystem verwende einen proprietären Zeichensatz, wie z.B. MS Windows, welches eine spezielle Repräsentation von *carriage return* hat).
- Gleichzeitig laufe ein Online-Chat Programm. Welche Schicht sorgt dafür, dass die ankommenden Pakete an die richtige der beiden Applikationen (also Browser bzw. Chat-Programm) weitergeleitet werden?

22. Aufgabe (Einfacher HTTP-Client in Java)

Nachdem wir letzte Woche mit Hilfe der vordefinierten Java-Klasse `URL` eine Verbindung zum Web-Server geschaffen hatten, sollen Sie diese Schritte nun "per Hand" in Ihrem Java-Programm durchführen. Folgende Hilfsmittel stehen dazu in Java zur Verfügung:

- Die `Socket`-Klasse.
  Diese Klasse ermöglicht es, einen (bidirektionalen) Kommunikationskanal zwischen zwei Programmen auf dem Netzwerk herzustellen. Die Anwendung der Klasse ist analog zu dem aus Aufgabe 10 bekannten `telnet`-Befehl: Nach Angabe des Server-Namen und der *Port*-Adresse stellt `Socket` je einen Ein- und Ausgabekanal ("Stream") zur Verfügung, von dem das Programm dann selbständig lesen bzw. schreiben kann:

  ```
  Socket sock = new Socket(<Server-Name>, <Port-Nummer>);
  ```

  Anschliessend kann mit den Methoden `getInputStream()` bzw. `getOutputStream()` ein Datenstrom zur Ein- oder Ausgabe eröffnet werden:

  ```
  OutputStream out = new OutputStream (sock.getOutputStream());
  ```

- Die `PrintWriter`-Klasse.
  Dies ist eine spezialisierte `OutputStream` Klasse, die zusätzlich einige Methoden zur Ausgabe von Daten zur Verfügung stellt, wie z.B. `print()` und `println()`:

  ```
  PrintWriter out = new PrintWriter (<OutputStream>, true);
  ```
  [1]

  Nun kann man ganze Zeilen (also inklusive abschliessendem Newline) beispielsweise so an den Empfänger schicken:

  ```
  out.println(“Hello World”);
  ```

[1] Das zweite Argument ("`true`") bewirkt hier, dass geschriebene Daten nicht zwischengepuffert, sondern nach jedem Zeilenende (z.B. mit `println()`) zum Empfänger gesendet werden.

Zusammen mit der bereits in Übung 19 verwendeten `InputStreamReader` Klasse haben Sie so alle Teile, die Sie für einen "manuellen" HTTP-Client in Java benötigen.

Hinweise zur Abgabe: Bitte senden Sie Ihre Lösungen im Quelltext per Email an Ihren Tutor.[2] Bitte verwenden Sie ein separates Attachment für jede der abgegebenen Teilaufgaben. Die Lösungen müssen sich mittels `javac` kompilieren lassen und bei Ausführung die gewünschte Ausgabe erzeugen.

a) (4 Punkte) Schreiben Sie das Programm aus Aufgabe 19.a so um, dass es anstatt der `URL`-Klasse nun die `Socket`-Klasse verwendet. Der Einfachheit halber dürfen Sie statt eines einzelnen URL-Parameters (wie in dem Programm aus Aufgabe 19) in Ihrer Lösung zwei getrennte Parameter für den Hostnamen und den Dokument-Pfad verwenden.[3] Ebenso wollen wir in dieser Aufgabe keinen Unterschied zwischen dem eigentlichen Text des Dokumentes und den (durch das Protokoll bedingten) *Header*-Zeilen machen, die der Web-Server vor dem eigentlichen Dokument-Text sendet: Geben Sie einfach alle Daten aus, die der Server als Antwort zurücksendet. Tip: Wenn vom Web-Server keine Antworten zu kommen scheinen, könnte ihre Anfrage unvollständig sein. Probieren Sie ein wenig (wie in Aufgabe 10 beschrieben) mit `telnet`, um zuerst das korrekte Format einer HTTP-Anfrage herauszufinden!

b) (4 Punkte) Mit Hilfe der `BufferedReader`-Klasse können Sie statt einzelner Bytes ganze Zeilen einlesen:[4]

```
BufferedReader in = new BufferedReader (
     new InputStreamReader(mySocket.getInputStream()));
```

Die `readLine()`-Methode der Klasse puffert dabei die eingehenden Zeichen solange, bis ein Zeilenende-Zeichen empfangen wird, und liefert dann die gesamte Zeile (ohne abschliessendes Zeilenende-Zeichen!) als `String` zurück.[5] Verwenden Sie diese Klasse nun, um in Ihrem HTTP-Client die Header-Zeilen bei der Ausgabe des Web-Dokumentes zu überspringen. Tip: In HTTP sind Header und Dokumenttext durch eine einzelne Leerzeile[6] voneinander getrennt.

c) (6 Punkte) Hinter manchen Web-Adressen befinden sich keine Text-Dokumente, wie z.B. HTML-Seiten oder unformatierter Text, sondern Multimedia-Objekte wie Audio-, Video- oder Bild-Dateien. Der *Typ* eines solchen Objektes wird dabei vom Web-Server in seiner Antwort explizit

[2]Die Email Adressen der Tutoren finden Sie unter `http://www.inf.ethz.ch/department/IS/vs/lectures/WS9900/VS/`

[3]Also z.B. `java UserAgent www.yahoo.com /index.html` statt wie zuvor `java UserAgent http://www.yahoo.com/index.html`

[4]Wie im Beispiel gezeigt benötigt der Konstruktor dieser Klasse statt eines `InputStream` jedoch einen `InputStreamReader` als Argument!

[5]Am Dateiende wird die spezielle Konstante `null` von der Methode zurückgegeben.

[6]Also ein `String` mit `length()==0`.

im Header angegeben. Der betreffende Eintrag wird mit “`Content-Type:`“ eingeleitet, gefolgt von einem standardisierten Typ-Bezeichner. Bezeichner für Text-Objekte beginnen dabei immer mit “`text/`”, wie z.B. “`text/html`” für HTML-Dokumente oder “`text/plain`” für unformatierten Text. Audio-Objekte beginnen dementsprechend mit “`audio/`”, Bild-Objekte mit “`image/`” (also z.B. “`image/gif`” oder “`image/jpeg`”), etc. Zum Beispiel erhält man für `http://www.inf.ethz.ch/pics/Logo.gif`:

```
HTTP/1.1 200 OK
Date: Tue, 16 Nov 1999 13:29:48 GMT
Server: Apache/1.3.4 (Unix)
Last-Modified: Tue, 24 Feb 1998 10:32:23 GMT
Content-Length: 2112
Content-Type: image/gif

[Binäre Bildaten folgen hier]
```

Erweitern Sie Ihren Web-Client so, dass er den Typ-Bezeichner der zum Download angegebenen Web-Ressource erkennt und *nur* im Falle eines “`text/...`” Objektes auch anschliessend (wie in Teilaufgabe **b**) das eigentliche Dokument ausgibt. Alle anderen Objekt-Typen (wie z.B. Bild-Daten, etc.) sollen mit einer Fehlermeldung, wie z.B. “Kann Objekte vom Typ ... nicht anzeigen” quittiert werden, ohne den Inhalt auszugeben.

Nützliche Java-Methoden zum Analysieren von Textzeilen werden bereits vom Datentyp `String` bereitgestellt:

- `boolean startsWith(String prefix)`
  liefert den Wert `true`, wenn der String mit der Zeichenkette `prefix` beginnt. Zum Beispiel:

  ```
  String foo = "Hello world"
  if (foo.startsWith("Hello"))
      System.out.println ("match!");
  ```

- `String trim()`
  entfernt eventuell vorhandene Leerzeichen an Anfang und Ende des Strings. Zum Beispiel:

  ```
  String foo = "  Wh it es pace  ";
  System.out.println (foo.trim()); // schreibt "Wh it es pace"
  ```

- `String substring(int beginIndex)`
  liefert den Teilstring, der an der Position `beginIndex` beginnt und bis zum Ende des Strings reicht. Zum Beispiel:

  ```
  String foo = "OneTwoThree";
  System.out.println(foo.substring(3)); // schreibt "TwoThree"
  ```