Ausarbeitung zum Versuch IIS 3

Netzwerkprogrammierung

Gruppe 6:

Thomas Espeter

Kai Schmitz-Hofbauer

Protokollführer: Kai Schmitz-Hofbauer

1 Einleitung

1.1 Ziel des Versuches

Ziel dieses Versuches war es, Netzwerkprogrammierung unter Verwendung von Sockets kennenzulernen. Als Programmiersprache wurde Java eingesetzt.

1.2 ISO / OSI - Referenzmodell

Bevor näher auf Sockets eingegangen werden kann, muß zunächst das ISO / OSI - Referenzmodell kurz erwähnt werden.
Das ISO/OSI-Referenzmodell teilt Netzwerkverbindungen in sieben logische Schichten ein, die jeweils eine eigene Aufgabe übernehmen. Alle Schichten sind unabhängig voneinander. Durch dieses Verfahren sind höhere Schichten völlig unabhängig von den physikalischen Gegebenheiten. Anderseits können über eine funktionierende physikalische Verbindung alle Arten von Daten und Protokollen (höhere Schichten) benutzt werden. Das ISO/OSI-Referenzmodell hat folgende Struktur :

1. Bitübertragung
2. Sicherung
3. Vermittlung
4. Transport
5. Kommunikationssteuerung
6. Darstellung
7. Anwendung

Die Schichten 1 - 4 werden der Transportfunktion zugeordnet.
Die Schichten 5 - 7 werden den Anwenderfunktionen zugeordnet.

1.3 Ports

Damit ein Rechner gleichzeitig mehrere Verbindungen bearbeiten kann, müssen diese unterschieden werden. Dazu bedient sich das TCP der sogenannten Ports. Jeder Anwendung, die von einer höheren Schicht das TCP benutzen will, wird ein Port zugeordnet. Es gibt 65536 verschiedene Ports. Sie sind fortlaufend numeriert. Die ersten 1023 Ports sind für spezielle Anwendungen reserviert. Die Ports oberhalb von 1024 sind für jede Anwendung offen.

1.4 Sockets

Ein Socket ist eine Schnittstelle zur Transportschicht des Netzwerkes. Um eine Socketverbindung aufbauen zu können, müssen folgende Parameter angegeben werden :

- das Protokoll der Transportschicht (z. B. TCP oder UDP)
- die Adressen der beiden Rechner, die eine Verbindung eingehen wollen
- die Ports der beiden Rechner, die eine Verbindung eingehen wollen

In der Programmiersprache Java stellt die Klasse `java.net.Socket` Socketfunktionalität zur Verfügung. Mit Hilfe von Sockets ist es auch möglich, Client-Server Anwendungen zu erstellen. Hierzu werden allerdings spezielle Sockets, nämlich Server-Sockets (zugehörige Java-Klasse : `java.net.ServerSocket`), benötigt. Der Server erzeugt zunächst ein ServerSocket und bindet dies an einen bestimmten Port. Das ServerSocket “horcht” an dem angegebenen Port und wartet auf Verbindungsanfragen der Clients. Auf der Client-Seite muß ein Socket, das Serveradresse und Port enthält, erzeugt werden. Im Anschluß daran können Daten zwischen Server und Client ausgetauscht werden. Nach dem Datenaustausch muß die Verbindung geschlossen werden.

2 Versuchsdurchführung

2.1 Netzwerkprogrammierung

In dem erstem Teil des Versuches mußte von einer Client-Anwendung eine Socketverbindung zu einem Echo - Port eines vorhandenen Servers aufgebaut werden. Zu diesem Zweck stand die Java- Anwendung `Netzprogmmierung.java` zur Verfügung. Die Applikation war bis auf den Teil, der die Netzwerkfunktionalität betrifft, bereits vor dem Versuch fertiggestellt. An im Quelltext des Programms markierten Stellen mußte die Netzwerkfunktionalität ergänzt werden. Zunächst wurde mit dem Befehl

```
socket = new Socket (feld_Serveradresse.getText(), 7);
```

ein neues Socket erzeugt. Mit der Methode `feld_Serveradresse.getText();` wurde die Serveradresse von einem Eingabefeld der graphischen Benutzeroberfläche gelesen. Das Socket wurde an Port 7 gebunden, da dies der Echo - Port ist. In das Textfeld `feld_Eingabe` konnte ein beliebiger Text eingegeben werden. Durch Betätigen eines Schalters sollten die Eingaben gesendet werden können. Zu diesem Zweck mußte der Sendemechanismus implementiert werden. Zum Senden der Daten wurde der OutputStream des Sockets benutzt. Zunächst wurde der OutputStream über die Anweisung `OutputStream out = socket.getOutputStream();` ermittelt. Um die Eingabe zu senden, mußte der Inhalt des Eingabefeldes in den OutputStream geschrieben werden. Hierzu mußte das Programm um die Zeile

```
out.write(feld_Eingabe.getText().getBytes());
```

ergänzt werden. Im nächsten Schritt galt es den Empfangsmechanismus zu implementieren. Hier wurde analog zur Implementierung des Sendemechanismus verfahren. Zunächst wurde über die Anweisung `InputStream in = socket.getInputStream();` der InputStream des Sockets ermittelt. Die vom

Server gesendeten Daten werden über den InputStream eingelesen und anschließend in das Ausgabefeld der graphischen Benutzeroberfläche geschrieben.

```
byte[] b = new byte[in.available()];
in.read(b);
String antwort = new String(b);
feld_Ausgabe.setText(antwort);
```

Nach Übertragung der Daten wurde das Socket mit dem Befehl `socket.close()` geschlossen. Das Programm konnte kompiliert, gestartet und anschließend benutzt werden.

2.2 IISChat

Im zweiten Teil des Versuches war es Aufgabe, ein bereits vorhandenes Chat-Programm um die Netzwerkfunktionalität zu ergänzen. Im Unterschied zum vorangegangenen Versuchsteil stand hier nicht die clientseitige Implementierung im Vordergrund, sondern die der Serverseite. Auch hier wurde an im Quelltext des Programms markierten Stellen Programmcode hinzugefügt. Das Chat Programm bestand aus den Klassen `IISChat.java` und `Server.java`. Die Klasse `Server.java` kapselt die Serverfunktionalität, während die Klasse `IISChat.java` das eigentliche Chat-Programm beinhaltet.

2.2.1 Server.java

Hier mußte zuerst ein ServerSocket erzeugt werden. Zu diesem Zweck wurde die Zeile `serverSocket = new ServerSocket(ServerPort)` hinzugefügt. Bei Serverport handelte sich um eine Konstante mit dem Wert 2000. Durch Ergänzung des Aufrufs `socket = serverSocket.accept()` wurde es dem Server ermöglicht, Verbindungen mit Clients einzugehen. Die Methode `accept()`

gibt ein Socket für die Verbindung zurück.

2.2.2 IISChat.java

Hier mußte im ersten Schritt eine Socketverbindung zum Server aufgebaut werden (`socket = new Socket(host,port)`). Hierbei beinhaltete host die Serveradresse und port die Portnummer 2000. Voraussetzung für einen fehlerfreien Ablauf war hier, daß der Server vorher ein serverSocket erzeugt hat und die Methode `accept()` aufgerufen wurde. Weiterhin sollte eine zeichenweise Übertragung realisiert werden. Sobald ein Zeichen eingegeben wurde, wurde dies von einem DocumentListener registriert und das Zeichen in der String-Variable `text` gespeichert. Mit Hilfe des OuputStreams des Sockets konnte das Zeichen zum Server gesendet werden.

```
socket.getOutputStream().write(text.getBytes());
```

Nach der Datenübertragung wurde das Socket wieder geschlossen (`socket.close();`).

Die Anwendung konnte kompiliert, gestartet und anschließend benutzt werden. Das Chatten funktionierte sowohl auf dem lokalen Rechner als auch mit anderen Rechnern im Netzwerk.

3 Diskussion der Ergebnisse

Der Versuch hat gezeigt, daß Netzwerkprogrammierung mit Java-Sockets bei kleineren und überschaubaren Projekten relativ unkompliziert ist. Auf eine Sache sei an dieser Stelle allerdings noch hingewiesen: Hat sich bei der Implementierung einer verteilten Anwendung / Netzwerkanwendung ein Fehler eingeschlichen, so kann dieser Fehler nicht unbedingt auf Anhieb ausfindig gemacht werden. Falls die Kommunikation zwischen einem Sender und ei-

nem Empfänger fehlerhaft ist, kann sich der Fehler sowohl beim Sender als auch beim Empfänger befinden. Um den Fehler zu lokalisieren, muß u. U. der Netzverkehr mit zusätzlicher Software überwacht werden. Zu diesem Zweck eignet sich z. B. die Sniffer-Software *Ethereal*.