Labor Client/Server-Programmierung

Teil 2: Socket-Programmierung

1. Hintergrundliteratur

- Ausgewählte Vorlesungskapitel zur Socket-Programmierung.
- Linksammlung zu auf `jochenkoegel.de/dhbw` mit
    - Grundlegende Linux-Befehle
    - Link zu „Beej's Guide to Network Programming“
    - Link zu Linux-Manpages

2. Linux-Befehle zu Sockets und TCP/IP

- **`wireshark`**
    - wireshark benötigt root-Rechte
    - start deshalb mit: `sudo wireshark`
- **`netstat`**
  Zeigt bestehende Netzverbindungen oder Listening Sockets an.

  Wichtige Parameter:

    - `-n`: numerische Adressen und Ports (keine Namensauflösung). → immer verwenden
    - `-t`: nur TCP-Sockets
    - `-l`: nur Listening Sockets (per default nur Client-Sockets)
    - `-a`: Listening-Sockets und Client-Sockets
    - `-p`: gibt den Prozess, der den Socket geöffnet hat, aus (als root aufrufen)
    - `-o`: zeigt Timeout-Werte der Sockets an

  Filterung auf Sockets eines bestimmten Ports mit angehängtem pipe + grep möglich.

  Beispiel – Anzeige der Listening TCP-Sockets für Port 12345:

  ```
  netstat -ntl | grep 12345
  ```

- **`netcat/nc`**
  Netcat ist ein einfaches und universelles Server-Client-Tool, mit dem Daten per TCP oder UDP versendet werden können. Je nach Linux-Distribution kann es mit dem Aufruf `nc` oder `netcat` gestartet werden.

  Verwendung als TCP-Client: `netcat <Ziel-Adresse> <Ziel-Port>`
  Beispiel: `netcat 127.0.0.1 12345`

  Per Tastatur (STDIN) eingegebene Daten werden an den anderen Endpunkt gesendet. Empfangene Daten werden ausgegeben (STDOUT).

3. Wichtige Socket-Optionen

3.1. Setzen von Socket-Optionen

Socket-Optionen können in C-Programmen mit `setsockopt` gesetzt werden:

`setsockopt(<sockfd>,<level>,<optname>, <*optval>, <optlen>)`

- `<sockfd>`: file descriptor
- `<level>`: protocol level. In diesen Aufgaben immer `SOL_SOCKET`
- `<optname>`: Name der Option
- `<optval>`: Pointer auf zu setzenden Wert
- `<optlen>`: Länge des Wertes, also sizeof optval

3.2. Address-Reuse-Option

Mit dieser Socket-Option kann ein Listening-Socket geöffnet werden, obwohl für den Port aktuell noch eine Verbindung im Zustand `TIME_WAIT` existiert.

Name: `SO_REUSEADDR`

Aktiv: Wert=1

4. Allgemeine Hinweise

- Versehen Sie alle Ihre Programme mit aussagekräftigen Debug-Ausgaben (`printf`)
- Verwenden Sie ein Unterverzeichnis pro Aufgabe
- Orientieren Sie sich an den Beispielen aus „Beej's Guide to Network Programming“
- Schlagen Sie weitere benötigte Informationen in den man-pages nach

4.1. Fehlerbehandlung

Prüfen Sie die Rückgabewerte von bind und listen auf Fehler.

Beispiel:

```
if(bind(...) < 0)
{
     perror("Bind: ");
     exit(1);
}
```

4.2. Header-Files

Folgende Include-Statements werden benötigt:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
```

5. Vorbereitungsaufgaben

1. Über welches Transportprotokoll und welchen Port läuft HTTP?
2. In welchem Zustand des TCP-Zustandsautomaten befindet sich ein Webserver, der auf eingehende Verbindungen wartet? In welchem Zustand des TCP-Zustandsautomaten befindet sich eine Client-Verbindung des Web-Browsers nach dem das erste TCP-Paket an den Server gesendet wurde?
3. Wieviele und welche Pakete werden mindestens bei einer korrekt auf- und abgebauten TCP-Verbindung gesendet?
4. Mit welchem Systemaufruf wird ein Socket erzeugt?
5. Welchen ausgehenden Port verwendet ein TCP-Client, der einen Socket öffnet standardmäßig und wie kann dieser Port explizit gesetzt werden?
6. Mit welchem Socket-Primitiv wird auf eine eingehende Verbindung gewartet? Was geschieht mit einem Server-Socket, nachdem eine Verbindung eingegagen ist? Was ist die Bedeutung des dabei erzeugen Client-Sockets?
7. Beschreiben Sie in der Notation
   {Lokale-IP-Adresse: Lokaler-Port, Entfernte-IP-Adresse:Entfernter-Port}
   - Den Socket, auf dem ein Webserver an der Adresse 85.214.99.212 hört.
   - Den Socket eines Web-Browsers, des sich von 1.1.1.1, Port 42000 mit dem Webserver auf 85.214.99.212 verbunden hat.
   - Den Socket eines Webservers, der auf allen Adressen des Systems Verbindungen entgegennimmt.
   - Den Socket des Webservers, auf 85.214.99.212, der die Verbindung zu einem Web-Browser an der Adresse 1.2.3.4, port 424242 beschreibt.

6. Aufgabe 1: Echo-Server

6.1. Szenario

Es soll ein TCP-Serverprozess erstellt werden, der einen Client bedienen kann. Nach erfolgreichem Verbinden, soll ein Begrüßungstext an den Client gesendet werden. Danach soll der Server vom Client gesendete Daten an diesen zurückschicken. Die Funktionalität soll schrittweise implementiert werden.

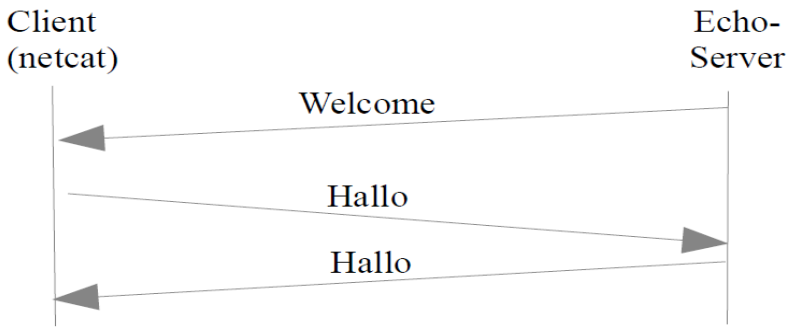

Abbildung 6.1 Echo-Server

Weitere Anforderungen

- Listening-Port: 12345
- Der Serverprozess soll nach dem Neustart sofort wieder den Socket öffnen können.

6.2. Aufgaben

1. Laden Sie sich das Codegerüst von `http://jochenkoegel.de/dhbw/lab2-code-skeleton/echoserver.c` herunter. Ändern bzw. ergänzen sie die mit `TODO` markierten Stellen und kompilieren Sie das Pogramm.
2. Überprüfen Sie mit `netstat`, ob der Listening-Socket angelegt wird.
3. Implementieren Sie den Aufruf von `accept()` und das Senden der Begrüßungsnachricht anhand der Codebeispiele aus dem Vorlesungsskript. Danach soll der Prozess warten. Testen Sie die Funktionalität mit `netcat`, verfolgen Sie den Zustand der TCP-Verbindung mit `netstat` und die gesendeten Pakete mittels `wireshark`.
4. Implementieren Sie die Echo-Funktionalität anhand der Codebeispiele aus dem Vorlesungsskript. Testen Sie mit `netcat`.
5. Beenden Sie den Server mit Strg-C, nachdem Sie dem Server Daten per `netcat` geschickt haben. Starten Sie danach den Server neu. Welches Problem tritt dabei auf? Erweitern Sie den Code so, dass der Server sofort neu gestartet werden kann.
6. Versuchen Sie sich mit einem weiteren parallelen `netcat`-Aufruf zum Server (z.B. in einem anderen Terminal) zu verbinden. Warum erhalten Sie keine Begrüßungsnachricht?

7. Aufgabe 2: Weiterentwicklung zum Webserverser

7.1. Szenario

Der Server soll einem Web-Browser auf einen GET-request mit einer einfachen Webseite antworten. Dazu muss laut HTTP-Standard (https://tools.ietf.org/html/rfc2616) der Response-Code gefolgt von zwei Zeilenumbrüchen der eigentlichen Nachricht vorausgehen.

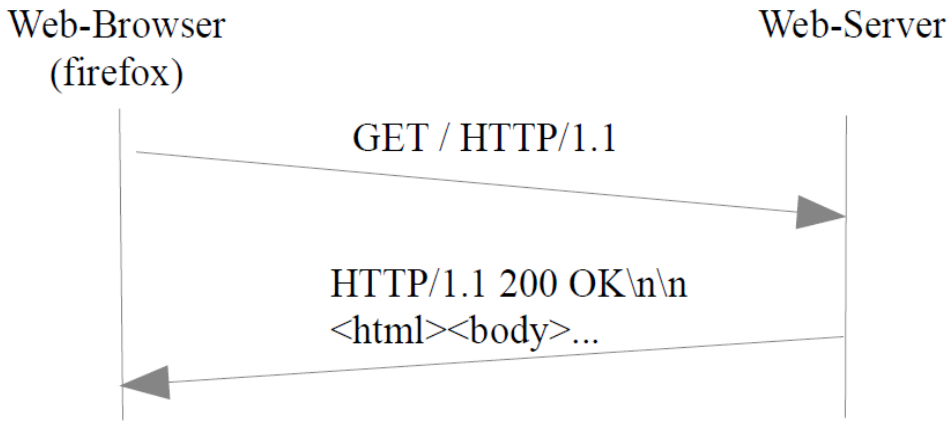

Abbildung 7.1 Web-Server

7.2. Aufgaben

1. Stellen sie den Listening-Port des Echo-Servers auf den HTTP-Port um und verbinden sie sich mit einem Web-Browser auf den Server. Was geschieht und warum stellt der Browser zunächst nichts dar?
2. Senden sie dem Browser den Response-Code und eine einfache Webseite zurück. Danach soll die Verbindung geschlossen werden und ein weiterer Request verarbeitet werden können. Erweitern Sie den Code entsprechend.
3. Erweitern sie die Webseite so, dass dem Client seine IP-Adresse und Port angezeigt wird.
4. Der Web-Server soll 10 Sekunden lang jede Sekunde eine weiteren Teil der Webseite an den Web-Browser schicken, bevor die Verbindung geschlossen wird.
5. Verbinden sie sich mit einem zweiten Browser-Fenster auf den Webserver, warum erhalten sie keine Antwort?

8. Aufgabe 3: Weiterentwicklung zum Multiclient-Server

8.1. Szenario

Der Server soll pro Client-Verbindung einen neuen Prozess starten.

8.2. Aufgaben

1. Implementieren Sie den MultiClient-Server unter der Verwendung des `fork()`-Aufrufs.
2. Die Webseite soll jeweils anzeigen, der wievielte Client sich gerade verbunden hat.
3. Die Webseite soll selbst beim gleichzeitigen Zugriff mehrerer Clients korrekt anzeigen, wieviele Bytes der Webserver insgesamt gesendet hat.