Versionsneutrale Socket-Programmierung

Betreuende Person: Prof. Dr. Thomas Schmidt
Protokollanten: Vadim Budagov (ECS), Ömer Kirdas (TI)

Inhaltsverzeichnis

Abbildungsverzeichnis

In dieser Aufgabe handelt es sich um eine Versionsneutrale-Socket-Programmierung, die in der Sprache C realisiert wurde. Dabei soll die Kommunikationsfähigkeit über den Router RNS1 getestet werden. Außerdem soll dies sowohl durch IPv4 als auch IPv6 geschehen. Um den Versuch zu starten, haben wir zunächst den Weg vom Rechner LAB21 zur Gegenstelle LAB30 konfiguriert (Siehe Abbildung 1).

```
networker@lab21:~/Desktop$ sudo route add -net 192.168.18.0 netmask 255.255.255.0 gw 192.168.17.2
```

Abbildung 1: Route von lab21 zu lab30

1. Verbindungsaufbau

Um die Kommunikation zwischen Client und Server aufzubauen, muss zuerst Server ausgeführt werden. Der Server hört auf eingehende Verbindungen (listen). Der Client muss Hostname (Abbildung 2), IPv4 (Abbildung 3) oder IPv6 (Abbildung 4) übernommen bekommen haben.

```
networker@lab21:~/Desktop/rnp/doc/Praktikum/Aufgabe 3/src/Client$ ./client lab30
argc: 2; argv[1]: lab30
Listener-Port: 8080
client: connecting to 141.22.27.109
```

Abbildung 2: Connection Hostname

```
networker@lab21:~/Desktop/rnp/doc/Praktikum/Aufgabe 3/src/Client$ ./client 172.16.1.5
argc: 2; argv[1]: 172.16.1.5
Listener-Port: 8080
client: connecting to 172.16.1.5
```

Abbildung 3: Connection IPv4

```
networker@lab21:~/Desktop/rnp/doc/Praktikum/Aufgabe 3/src/Client$ ./client fe80:
:21b:21ff:fe40:e6e5%eth2
argc: 2; argv[1]: fe80::21b:21ff:fe40:e6e5%eth2
Listener-Port: 8080
client: connecting to fe80::21b:21ff:fe40:e6e5
```

Abbildung 4: Connection IPv6%eth2

```
networker@lab21:~/Desktop/rnp/doc/Praktikum/Aufgabe 3/src/Client$ ./client fe80:
:21b:21ff:fe40:e6e4%eth1
argc: 2; argv[1]: fe80::21b:21ff:fe40:e6e4%eth1
```

Abbildung 5: Connection IPv6%eth1

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.18.133  netmask 255.255.255.0  broadcast 192.168.18.255
        inet6 fd32:6de0:1f69:18:21b:21ff:fe40:e6e4  prefixlen 64  scopeid 0x0<g
obal>
        inet6 fe80::21b:21ff:fe40:e6e4  prefixlen 64  scopeid 0x20<link>
        ether 00:1b:21:40:e6:e4  txqueuelen 1000  (Ethernet)
        RX packets 37  bytes 2422 (2.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 45  bytes 6685 (6.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device memory 0xf7c20000-f7c3ffff
```

Abbildung 6: Gegenstelle eth1

```
eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.1.5  netmask 255.255.255.0  broadcast 172.16.1.255
        inet6 fd32:6de0:1f69:16:21b:21ff:fe40:e6e5  prefixlen 64  scopeid 0x0<gl
obal>
        inet6 fe80::21b:21ff:fe40:e6e5  prefixlen 64  scopeid 0x20<link>
        ether 00:1b:21:40:e6:e5  txqueuelen 1000  (Ethernet)
        RX packets 10736  bytes 816038 (816.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5402  bytes 413878 (413.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device memory 0xf7c00000-f7c1ffff
```

Abbildung 7: Gegenstelle eth2

Diese unterschiedliche Verbindungsaufbaue werden durch Server angenommen, wie in der Abbildung 8 zu sehen ist.

```
networker@lab30:~/Desktop/rnp/doc/Praktikum/Aufgabe 3/src/Server$ ./server
anfang main
vor readfile in main
vor while loop
selectserver: new connection from ::ffff:141.22.27.100 on socket 4
Port: 48540
CLIENTNAME: lab21.cpt.haw-hamburg.de
```

Abbildung 8: Server/Verbindung angenommen

2. Put-Befehl

Mit dem Befehl Put<Dateiname> verschickt der Client eine Text-Datei (siehe Abbildung 9) an Server, damit er diese bei sich ablegen kann. Dabei kann man den Dateinamen, die Anzahl der verschickten Bytes, den Hostnamen, das Host IP und den Zeitstempel von erfolgreich gesendeter Datei ablesen (siehe Abbildung 10).

Abbildung 9: eine Text-Datei

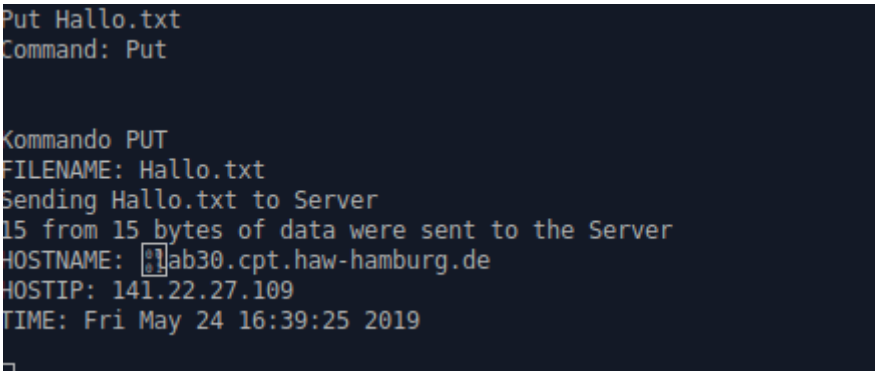

Abbildung 10: Put<dateiname>

3. Get-Befehl

Mit dem Befehl Get<Dateiname> fordert der Client eine Datei, die beim Server abgelegt ist. Nach dem Erhalt der angeforderten Datei wird beim Client auf der Konsole die Größe, der Zeitstempel und der Inhalt ausgegeben.

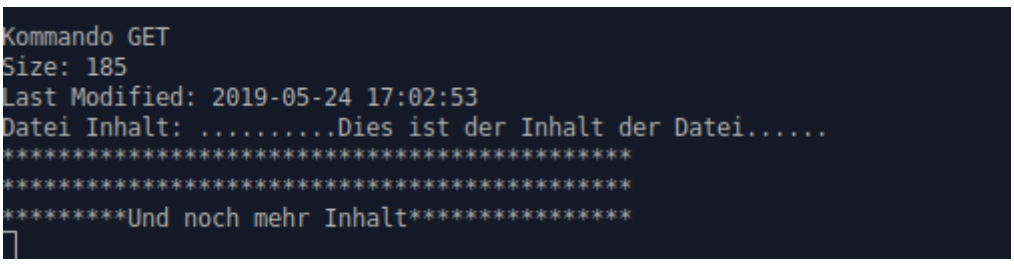

Abbildung 11: Get<dateiname>

4. List-Befehl

Der Client gibt den Befehl List auf der Konsole ein. Damit erhält der Client eine Liste mit allen verbundenen Clients mit dem Server. In der Abbildung 12 sieht man zunächst zwei

verbundene Clients. In der Abbildung 13 sieht man, dass ein Client 7 sich mit dem Server verknüpft hat. Außerdem kann man da auch sehen, dass der Client 5 die Verbindung mittels Quit abgebrochen hat.

```
List
Command: List


Kommando LIST
CLIENTFD: 4
Hostname: lab21.cpt.haw-hamburg.de
ClientPort: 48540


CLIENTFD: 5
Hostname: lab21.cpt.haw-hamburg.de
ClientPort: 42820
```

Abbildung 12: Liste mit zwei Clients

```
Command: List


Kommando LIST
CLIENTFD: 4
Hostname: lab21.cpt.haw-hamburg.de
ClientPort: 48540


CLIENTFD: 6
Hostname: lab21.cpt.haw-hamburg.de
ClientPort: 47920


CLIENTFD: 7
Hostname: lab21.cpt.haw-hamburg.de
ClientPort: 48560
```

Abbildung 13: Neue Liste nach Quit und Connect

5. Quit-Befehl

Mit dem Quit-Befehl beendet ein Client seine Verbindung zum Server (Siehe Abbildung 14). Auf der Server Seite (Abbildung 15) wird die FD-Nummer des Clients ausgegeben und damit bestätigt, dass die Verbindung mit dem jeweiligen Client beendet wurde. Der Client wird aus der Liste entfernt.

```
Quit
Command: Quit


client: received ''
networker@lab21:~/Desktop/rnp/doc/Praktikum/Aufgabe 3/src/Client$
```

Abbildung 14: Client/ Quit

Abbildung 15: Server/ Quit wurde bestätigt