19. März 2014

Steffen Giersch & Maria Lüdemann

HAW Hamburg

Rechnernetze Praktikum 1

Werkzeuge zur Arbeit mit Netzwerken & Netzwerkprogrammiereung

Inhaltsverzeichnis

[Werkzeuge zur Arbeit mit Netzwerken 2](#_Toc384631336)

[Netzwerkprogrammierung 5](#_Toc384631337)

Teil 1

# Werkzeuge zur Arbeit mit Netzwerken

1. Welche Informationen erhält man mit dem Programm /sbin/ifconfig

Sbin/config = Liefert Informationen zu den Netzwerkgeräten z.B iP Adresse, Netzwerkmaske, broadcast, ethernetmaske, Packets Informationen, Device Informationen

1. Messen Sie die Round Trip Time für ein 1000-Datenbyte-Paket nach www.google.de, nach

www.haw-hamburg.de und zum Rechner lab22 jeweils als Durchschnittswerte von 10

Stichproben.

**ping -c 10 -s 1000 www.google.de**  
PING www.google.de (173.194.113.159) 1000(1028) bytes of data.  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=1 ttl=56 time=8.59 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=2 ttl=56 time=8.58 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=3 ttl=56 time=8.59 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=4 ttl=56 time=8.60 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=5 ttl=56 time=8.63 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=6 ttl=56 time=8.57 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=7 ttl=56 time=8.60 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=8 ttl=56 time=8.60 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=9 ttl=56 time=8.58 ms  
1008 bytes from ham02s11-in-f31.1e100.net (173.194.113.159): icmp\_seq=10 ttl=56 time=8.58 ms

**ping -c 10 -s 1000 www.haw-hamburg.de**PING haw01.rz.tu-harburg.de (134.28.219.2) 1000(1028) bytes of data.  
  
--- haw01.rz.tu-harburg.de ping statistics ---  
10 packets transmitted, 0 received, 100% packet loss, time 8999ms

**ping -c 10 -s 1000 lab22**  
PING lab22.cpt.haw-hamburg.de (141.22.27.101) 1000(1028) bytes of data.  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=1 ttl=64 time=0.527 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=2 ttl=64 time=0.278 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=3 ttl=64 time=0.277 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=4 ttl=64 time=0.285 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=5 ttl=64 time=0.279 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=6 ttl=64 time=0.271 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=7 ttl=64 time=0.272 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=8 ttl=64 time=0.283 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=9 ttl=64 time=0.278 ms  
1008 bytes from lab22.cpt.haw-hamburg.de (141.22.27.101): icmp\_seq=10 ttl=64 time=0.270 ms

1. Finden Sie heraus, welches Programm auf Ihrem Rechner den Port 9400 benutzt. Welches Protokoll setzt dieses Programm auf der Transportschicht ein? In welchem Systemaufruf "hängt" das Programm? (Für letzteres brauchen Sie noch das Programm strace mit der Option -p <pid>)

**Programm Name:** rn11serve

**Protokoll:** UDP

/usr/local/ti/rnp11/rn11server

**ps -el | grep 3413**  
0 S 62749  3413  3402  0  80   0 -  1021 skb\_re ?        00:00:00 rn11server

**Elternprozess**: Pid 3402 rn11launcher

Der Prozess hängt im recfrom Systemaufruf

1. Stellen Sie eine TCP-Socketverbindung zwischen der Shell (Standardein-/ausgabe) und dem Server www.bsh.de auf dessen Port 80 her. Senden Sie die Zeichenkette „GET“ und Sie die Antwort des Servers. Um welches Protokoll handelt es sich hier?

**socat tcp:www.bsh.de:80 -**  
GET  
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">  
<html><head>  
<title>302 Found</title>  
</head><body>  
<h1>Found</h1>  
<p>The document has moved <a href="http://www.bsh.de/de/index.jsp">here</a>.</p>  
</body></html>

danach wird die Verbindung direkt wieder vom Server geschossen

1. Loggen Sie sich von Ihrem Arbeitsplatzrechner aus auf dem zugehörigen Gegenstellenrechner (siehe Tabelle) ein und lassen Sie sich dort mit dem Dateimanager (Befehl konqueror von der Kommandozeile) den Inhalt des dortigen /tmp-Verzeichnisses zeigen.

**(abl128@lab35)/tmp>ls -l**total 892  
drwx------ 4 root     root       4096 Mar 28 13:09 0457058169  
drwx------ 4 root     root       4096 Mar 28 13:09 0503165533  
drwx------ 4 root     root       4096 Mar 28 13:09 0586435161  
drwx------ 4 root     root       4096 Mar 28 13:09 1047265725  
drwx------ 4 root     root       4096 Mar 28 13:09 1048788033  
drwx------ 4 root     root       4096 Mar 28 13:09 1203131527  
drwx------ 4 root     root       4096 Mar 28 13:09 1422341448  
drwx------ 4 root     root       4096 Mar 28 13:09 1678341251  
drwx------ 4 root     root       4096 Apr  7 10:45 1816348633  
drwx------ 4 root     root       4096 Mar 28 13:09 1952245645  
-rw------- 1 ahlf\_c   students   2443 Mar 14 11:38 6B9QCX.tmp  
-rw------- 1 lp       lp          191 Mar 14 11:37 HLS7000DN\_latest\_print\_info  
-rw-r--r-- 1     1000 users    138793 Feb 17 16:30 hs\_err\_pid8450.log  
drwx------ 2 ahlf\_c   students   4096 Mar 28 13:09 hsperfdata\_ahlf\_c  
drwxr-xr-x 2     1000 users      4096 Feb 19 09:12 hsperfdata\_inst  
drwxr-xr-x 2 ldaptstb gast       4096 Mar 19 10:09 hsperfdata\_ldaptstb  
drwxr-xr-x 2 root     root       4096 Feb 11 15:34 hsperfdata\_root  
drwx------ 2     1000 users      4096 Feb 11 11:01 icedteaplugin-inst  
-rw-r--r-- 1 root     root       4715 Feb 11 15:29 install4jError5486138452222491627.log  
drwxr-xr-x 2 root     root       4096 Feb 11 10:23 jna  
drwx------ 2 abe201   students   4096 Mar 31 11:09 kde-abe201  
drwx------ 2 ahlf\_c   students   4096 Mar 28 13:09 kde-ahlf\_c  
drwx------ 2     1000 users      4096 Mar 28 13:09 kde-inst  
drwx------ 2 ldaptsta gast       4096 Mar 28 13:09 kde-ldaptsta  
drwx------ 2 ldaptstb gast       4096 Mar 28 13:09 kde-ldaptstb  
drwx------ 3 root     root       4096 Mar 28 13:09 kde-root  
drwx------ 2 abe201   students   4096 Mar 31 13:04 ksocket-abe201  
drwx------ 2 ahlf\_c   students   4096 Mar 14 10:21 ksocket-ahlf\_c  
drwx------ 3     1000 users      4096 Mar 28 13:09 ksocket-inst  
drwx------ 2 ldaptsta gast       4096 Feb 28 15:59 ksocket-ldaptsta  
drwx------ 3 ldaptstb gast       4096 Mar 28 13:09 ksocket-ldaptstb  
drwx------ 3 root     root       4096 Mar 28 13:09 ksocket-root  
-rw-r--r-- 1 ldaptstb gast          0 Mar 19 10:08 netperf.debug  
drwx------ 2 ahlf\_c   students   4096 Jan  1  2034 orbit-ahlf\_c  
drwx------ 2     1000 users      4096 Jan  1  2034 orbit-inst  
drwx------ 2 ldaptstb gast       4096 Jan  1  2034 orbit-ldaptstb  
drwx------ 2 root     root       4096 Jan  1  2034 orbit-root  
drwx------ 2     1000 users      4096 Feb 11 14:48 pulse-8kXzt48vxL04  
-rwxr-xr-x 1     1000 users    557544 Feb 11 15:42 sqlite-3.7.2-libsqlitejdbc.so­  
-rw-r--r-- 1 root     root      43923 Feb 11 15:34 suite\_installer.log  
-rw-r--r-- 1 ldaptstb gast          0 Mar 19 10:09 umlet.tmp  
-rw-r--r-- 1     1000 users      3586 Feb 11 15:45 vpuml\_integration\_installer.log  
-rw------- 1 ahlf\_c   students   2443 Mar 14 11:37 VX6UCX.tmp

-X beim einloggen tunnelt den Xserver sodass man auch von Entfernung auf Dienste wie Konqueror zugreifen kann. Auf dem Rechner selbst können sie laufen, das heißt aber nicht dass man remote darauf zugreifen kann.

Teil 2

# Netzwerkprogrammierung

Programmierung einer Server ~ Client Anwendung

Beschreibung des Vorgehens:

Die Aufgabe wurde in drei Klassen gelöst, Client, Server und ServerThread die als Server und Client agieren. Es gibt den Clienten der versucht eine Verbindung zu einem Server aufzubauen dessen Verbindungsdaten direkt vom Nutzer abgefragt werden. Der Server nutzt eine WelcomeSocket um verbindungs Versuche anzunehmen, dann überprüft er ob er noch weitere Clienten annehmen möchte, ist dies der Fall öffnet er einen neuen Thread gibt ihm einen neuen Socket auf dem er fortan mit dem Clienten kommuniziert.

Funktionalität Erfolgsfall:

Der Client hat einige Möglichkeiten Befehle an den Server zu senden.

* UPPERCASE <message>
* LOWERCASE <message>
* REVERSE <message>
* BYE
* SHUTDOWN <password>

Für alle Befehl <message> Konstrukte gilt, sie dürfen nicht länger als 255 Bytes sein.

**Uppercase** nimmt den <message> String und wandelt alle kleinen Buchstaben in große um

Bsp. Hallo -> HALLO

**Lowercase** nimmt den <message> String und wandelt alle großen Buchstaben in kleine um

Bsp. Hallo -> hallo

**Reverse:**  nimmt den <message> String und dreht ihn um

Bsp. Hallo -> ollah

**Bye:** wird vom Server erwiedert, danach beendet sich der Client

**Shutdown:** nimmt einen String den er als Passwort übergibt. Der Server überprüft ob das Passwort stimmt, hört wenn dies der Fall ist auf neue Verbindungen anzunehmen und wartet darauf dass sich alle Threads beenden. Dann beendet sich auch der Server.

Funktionalität Misserfolg/Fehler:

Wenn Eingaben länger sind als 255 Bytes wird ein Fehler geworfen und die gesamte Eingabe verworfen und auf eine neue Eingabe gewartet.

ERROR Message to long

Dabei wird der gesamte Buffer vorgespult, sodass alle nachfolgenden Befehle nicht mehr bearbeitet werden.

Bei unbekannten Befehlen oder fehlender <message> wird ein

Unknown command

Zurück gegeben und auf eine neue Eingabe gewartet.

Ein falsches Passwort erzeugt die Rückgabe:

ERROR Wrong Password

Danach ist der Server wieder bereit Befehle entgegen zu nehmen

Client Konsole einfach schließen

Connection lost

Keine Verbindung bekommen

Thread Sicherheit:

Damit der Server beim Shutdown auf alle Threads warten kann nutzen wir eine Variable die mitzählt wie viele Threads ein Server geöffnet hat. Beendet sich ein Client ruft er eine Funktion auf die diese Variable dekrementiert. Dies ist bei Multithreaded Anwendungen immer eine Gefahr. Somit haben wir die Variable auf voltaire gesetzt um sicherzugehen, dass sich die Threads nicht gegenseitig in die Quere kommen können.