3 Hausaufgaben

3.1 Echo as a service (6 Punkte)

Schreiben Sie ein Server-Programm, das einen TCP-Socket erzeugt und auf Port 44445 auf eingehenden TCP-Verbindungen wartet. Es soll jede neu ankommende Verbindung annehmen und dann Eingaben von dieser Verbindung akzeptieren – allerdings ohne dabei auf diese Verbindung zu blockieren! Sie müssen also auf mehrere Sockets gleichzeitig reagieren können, z.B. mittels `poll()`. Alle Zeichen, die auf einer Verbindung empfangen werden, werden wieder an an Absender zurückgeschickt.

Wenn auf einer Verbindung für 5 Sekunden keine Daten empfangen wurden, wird diese vom Server mit `close()` geschlossen. Ebenso soll die Verbindung beendet werden, wenn sie von der anderen Seite geschlossen wird. Wenn Ihr Server 10 Sekunden lang gar nichts zu tun hat ("Idle-Timeout"), also weder eine neue Verbindungsanfrage bekommt, noch Daten auf einem Socket empfängt, soll er terminieren.

Schreiben Sie Ihren Server so, dass er mehrere (bis zu 10) Verbindungen gleichzeitig behandeln kann. Hierzu können Sie Ihre Listenverwaltung verwenden und eine einfache Datenstruktur erzeugen, die Sie für jede Verbindung erzeugen und dann mittels der Liste verwalten. Beispielsweise kann diese wie folgt aussehen.

```
struct connection {
    int                 sd;
    struct sockaddr_in  sin;
    struct timeval      timeout;
};
```

Denken Sie daran, dass Sie einen Timeout pro Verbindung verwalten und diesen nach jeder Interaktion mit der Verbindung neu starten müssen. Dasselbe gilt auch für den Idle-Timeout Ihres Servers. Auch für die Timeouts empfielt sich Ihre Listenverwaltung. Sie können die obige Verbindungs-Datenstruktur in mehreren Listen verwalten, eine für die Verbindungen und eine für die Timeouts. Wie in der Vorlesung angedeutet, bietet es sich an die Timeouts in aufsteigender Reihenfolge in der Liste zu verwalten (also frühester Timeout zuerst).

Wenn Sie ein Element in der Liste suchen wollen, können Sie hierzu `list_find()` mit einer entsprechenden Vergleichfunktion für den Socket-Deskriptor verwenden, oder aber Sie iterieren manuell durch Ihre Liste.

Nutzen Sie das Kommando `nc(1)`, um Ihren Server zu testen.

```
$ ./server &
$ nc localhost 44445
Hello, world.
Hello, world.
Ich bekomme meine Zeichen zurueck.
Ich bekomme meine Zeichen zurueck.
^C$
```

Sie können mit folgenden Kommandos Ihren Server auch mit umfangreichen Eingaben testen, indem Sie eine Eingabedatei vorbereiten und diese an den Server senden, die vom Server empfangenen Ausgaben in eine *andere* Datei schreiben und diese anschließend mittels `diff(1)` vergleichen.

```
$ ./server &
$ nc localhost 44445 < 1.in > 1.out
$ diff 1.in 1.out
$
```

Hinweis: Lesen Sie vom Socket in einen Puffer fester Größe von 1024 Bytes. (Das ist auch wichtig für die nächste Aufgabe.) Achten Sie darauf, dass der Rückgabewert von `read()` bzw. `recv()` anzeigt, wie viele Bytes tatsächlich gelesen worden sind.

Quellen: server.c list.c list.h

Executable: server

3.2 Echo/Hexdump as a service (4 Punkte)

Erweitern Sie Ihren obigen Echo-Server so, dass er gleichzeitig auf einem zweiten Port (44446) gleichzeitig Verbindungen annimmt. Auf diesem Port eingehenden Verbindungen sollen einen hexdump über die Eingaben zurücksenden. Verändern Sie Ihre `hexdump()`-Funktion so, dass sie als ersten Parameter keinen FILE-Pointer, sondern den Socket- Deskriptor erhält. Schreiben Sie die Funktion intern so um, dass sie die Ausgaben zeilenweise in einen Puffer schreibt (`sprintf()` statt `fprintf()` und senden Sie diesen Puffer dann zeilenweise über den Socket (etwa mittels `write()`).

Wichtig: Lesen Sie auch hier maximal in Blöcken von 1024 Bytes und übergeben Sie den Puffen nach jedem Lesen an `hexdump()`. Das bedeutet, dass Ihre Offsets bei jeder Ausgabe immer wieder von vorne beginnen, was in diesem Fall so gewollt ist.

```
$ ./server &
$ nc localhost 44446
Hello
000000 : 48 65 6c 6c 6f 0a                                      Hello.
Ein etwas laengerer Text zum Testen.
000000 : 45 69 6e 20 65 74 77 61 73 20 6c 61 65 6e 67 65        Ein etwas laenge
000010 : 72 65 72 20 54 65 78 74 20 7a 75 6d 20 54 65 73        rer Text zum Tes
000020 : 74 65 6e 2e 0a                                         ten..
^C$
```

Auch hier gilt: wenn auf einer Verbindung für 5 Sekunden keine Daten empfangen wurden, wird diese durch den Server geschlossen. Achten Sie darauf, dass zu beiden Ports Verbindungen parallel aufgebaut werden können und dass diese auch anschließend parallel Daten austauschen können. Aber Sie müssen nicht mehr als 10 Verbindungen gleichzeitig unterstützen.

```
$ ./server &
$ cat 0.in
Auch hier gilt: wenn auf einer Verbindung f\"ur 5 Sekunden keine Daten
empfangen wurden, wird diese durch den Server geschlossen.  Achten Sie
darauf, dass zu beiden Ports Verbindungen parallel aufgebaut werden
k\"onnen und dass diese auch anschlie{\ss}end parallel Daten austauschen
k\"onnen.  Aber Sie m\"ussen nicht mehr als 10 Verbindungen
gleichzeitig unterst\"utzen.

$ nc localhost 44445 < 0.in
Auch hier gilt: wenn auf einer Verbindung f\"ur 5 Sekunden keine Daten
empfangen wurden, wird diese durch den Server geschlossen.  Achten Sie
darauf, dass zu beiden Ports Verbindungen parallel aufgebaut werden
k\"onnen und dass diese auch anschlie{\ss}end parallel Daten austauschen
k\"onnen.  Aber Sie m\"ussen nicht mehr als 10 Verbindungen
gleichzeitig unterst\"utzen.

^$ nc localhost 44446 < 0.in
000000 : 41 75 63 68 20 68 69 65 72 20 67 69 6c 74 3a 20   Auch hier gilt:
000010 : 77 65 6e 6e 20 61 75 66 20 65 69 6e 65 72 20 56   wenn auf einer V
000020 : 65 72 62 69 6e 64 75 6e 67 20 66 5c 22 75 72 20   erbindung f\"ur
000030 : 35 20 53 65 6b 75 6e 64 65 6e 20 6b 65 69 6e 65   5 Sekunden keine
000040 : 20 44 61 74 65 6e 0a 65 6d 70 66 61 6e 67 65 6e    Daten.empfangen
```

```
000050 : 20 77 75 72 64 65 6e 2c 20 77 69 72 64 20 64 69   wurden, wird di
000060 : 65 73 65 20 64 75 72 63 68 20 64 65 6e 20 53 65   ese durch den Se
000070 : 72 76 65 72 20 67 65 73 63 68 6c 6f 73 73 65 6e   rver geschlossen
000080 : 2e 20 20 41 63 68 74 65 6e 20 53 69 65 0a 64 61   .  Achten Sie.da
000090 : 72 61 75 66 2c 20 64 61 73 73 20 7a 75 20 62 65   rauf, dass zu be
0000a0 : 69 64 65 6e 20 50 6f 72 74 73 20 56 65 72 62 69   iden Ports Verbi
0000b0 : 6e 64 75 6e 67 65 6e 20 70 61 72 61 6c 6c 65 6c   ndungen parallel
0000c0 : 20 61 75 66 67 65 62 61 75 74 20 77 65 72 64 65    aufgebaut werde
0000d0 : 6e 0a 6b 5c 22 6f 6e 6e 65 6e 20 75 6e 64 20 64   n.k\"onnen und d
0000e0 : 61 73 73 20 64 69 65 73 65 20 61 75 63 68 20 61   ass diese auch a
0000f0 : 6e 73 63 68 6c 69 65 7b 5c 73 73 7d 65 6e 64 20   nschlie{\ss}end
000100 : 70 61 72 61 6c 6c 65 6c 20 44 61 74 65 6e 20 61   parallel Daten a
000110 : 75 73 74 61 75 73 63 68 65 6e 0a 6b 5c 22 6f 6e   ustauschen.k\"on
000120 : 6e 65 6e 2e 20 20 41 62 65 72 20 53 69 65 20 6d   nen.  Aber Sie m
000130 : 5c 22 75 73 73 65 6e 20 6e 69 63 68 74 20 6d 65   \"ussen nicht me
000140 : 68 72 20 61 6c 73 20 31 30 20 56 65 72 62 69 6e   hr als 10 Verbin
000150 : 64 75 6e 67 65 6e 0a 67 6c 65 69 63 68 7a 65 69   dungen.gleichzei
000160 : 74 69 67 20 75 6e 74 65 72 73 74 5c 22 75 74 7a   tig unterst\"utz
000170 : 65 6e 2e 0a 0a                                    en...
^C$
```

Quellen: server.c hexdump.c list.c list.h

Executable: server