Praktikum

Rechnernetze und Verteilte Systeme

Block 2

Sockets

23.11 – 06.12.2020

Präsenzaufgaben

Die folgenden Aufgaben werden im Termin besprochen.

Aufgabe 1:

Was ist der Unterschied zwischen einem Namen und einer Adresse?

Aufgabe 2:

Die Zuordnung von Domainnamen zu IP-Adressen erfolgt im Internet mit Hilfe des Domain Name Systems (DNS). Beschreiben Sie anhand der folgende Fragen den Aufbau und die Funktionsweise von DNS:

(a) Wie sind die Verantwortlichkeiten im Namenssystem aufgeteilt?

(b) Wie funktioniert das rekursive bzw. iterative Auflösen von Namen mit DNS?

(c) Welche Funktion erfüllen die TTL-Einträge?

(d) Überprüfen Sie mittels des Kommandozeilen-Tools dig (Linux / OS X) bzw. nslookup (Windows), welche DNS-Server für die Domain tu-berlin.de sowie deren Subdomains zuständig sind und welche TTL diese Einträge haben.

Aufgabe 3:

Nennen Sie jeweils 3 Vor- und Nachteile der Client-Server-Architektur.

Aufgabe 4:

Nutzen sie das Beispiel des Abrufs von E-Mails durch Clients bei einem Server, um die

folgenden Fragen für das Client-Server-Prinzip im Allgemeinen zu beantworten:

(a) Muss der betreffende Rechner (Client/Server) immer angeschaltet und ans Netz angeschlossen sein? Was hätte es für Konsequenzen, wenn beide Kommunikationspartner gleichwertig sind, also es keinen expliziten Server bzw. Client gibt?

(b) Welche Art von Adresse haben Client und Server? Was hätte es für Konsequenzen, wenn beide Kommunikationspartner gleichwertig sind, also es keinen expliziten Server bzw. Client gibt?

(c) Mit wem kommunizieren Clients in der Regel direkt? Mit wem Server?

Aufgabe 5:

Geben Sie jeweils für die folgenden API-Aufrufe an, was diese genau bewirken:

(a) socket

(b) bind

(c) listen

(d) accept

(e) connect

(f) send

(g) recv

(h) close

Aufgabe 6:

Wozu werden Portnummern verwendet?

Aufgabe 7:

Machen Sie sich mit der Funktion `getaddrinfo()`[1] [2] vertraut und beantworten Sie die folgenden Fragen:

[1] https://beej.us/guide/bgnet/html/#getaddrinfoman

[2] `man getaddrinfo`

(a) Was macht die Funktion?

(b) Was sind die Parameter der Funktion?

(c) Welchen Inhalt hat `struct addrinfo`?

(d) Wie wird technisch umgesetzt, dass `struct sockaddr_storage` [3] [4] sowohl mit IPv4 als auch IPv6 Adressen arbeiten kann?

(e) Was ist das Resultat des Aufrufs?

(f) Erklären Sie die folgenden hints:

```
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
```

[3] https://beej.us/guide/bgnet/html/#structsockaddrman

[4] `man bind`

Praktische Aufgaben

Die praktischen Aufgaben sind in Kleingruppen von i.d.R. 3 Personen zu lösen. Die Ergebnisse besprechen Sie im zweiten Termin mit dem Tutor. Reichen Sie deshalb bitte den Quelltext bzw. Lösungen dieser Aufgaben bis Mittwoch der zweiten Woche (02.12.) 20:00 Uhr per ISIS ein. Aufgaben werden sowohl auf Plagiate als auch auf Richtigkeit hin automatisch getestet, halten Sie sich deshalb genau an das vorgegebene Abgabeformat.

Auf der ISIS-Seite zur Veranstaltung finden Sie zusätzliche Literatur und Hilfen für die Bearbeitung der Aufgaben! Zusätzlich können wir den kostenlosen *Beej's Guide to Network Programming Using Internet Sockets* [5] empfehlen. Es gibt für die Aufgaben keinen Vorgabe-Quelltext, aber es finden sich im verlinkten Guide und auch in zahlreichen weiteren Quellen genug Beispiele. Sie sollen bei der Aufgabe auch lernen, sich selbstständig in ein Thema einzuarbeiten.

Aufgabe 8:

Zuerst sollen Sie einen Client für das im RFC 865 beschriebene *Quote of the Day Protocol* [6] programmieren. Zunächst soll ihr Client TCP benutzen, um die Daten abzurufen. Ihr Client soll (wie `netcat`) vom Benutzer über die Kommandozeile die IP Adresse bzw. den DNS-Namen als ersten Parameter und die Port-Nummer des Servers als zweiten Parameter übergeben bekommen. Der Client soll auf der Konsole die Antwort des Servers ausgeben. Sie können ihren fertigen Client mit einem der folgenden öffentlichen Server testen:

DNS-Name	Port
alpha.mike-r.com	17
djxmmx.net	17
cygnus-x.net	17

Achten sie bei der Implementierung besonders auf die folgenden Punkte:

[5] https://beej.us/guide/bgnet

[6] https://tools.ietf.org/html/rfc865

(a) Sie sollen in Ihrem Projekt CMake benutzen. Das Projekt sollte folgende Struktur aufweisen:

```
Block2
+--- CMakeLists.txt
+--- client.c
```

Der folgende Aufruf soll im Unterverzeichnis `build` eine ausführbare Datei mit dem Namen `client` erstellen:

```
$ mkdir build; cd build
$ cmake .. && make
```

(b) Ihr Client soll **sowohl** mit einer IP-Adresse als auch mit einem Hostnamen als Parameter umgehen können.

(c) Im RFC ist angegeben, dass der Server nicht mit mehr als 512 Zeichen antworten **sollte**. Deshalb muss Ihr Client auch mit beliebig größeren Antworten umgehen können und diese korrekt und vollständig ausgeben. Beachten Sie auch, dass ein einmaliger Funktionsaufruf von `recv()` nicht den Empfang aller vom Sender gesendeten Daten garantiert.

(d) Sie sollen die Daten genauso ausgeben, wie sie der Server geschickt hat. **Fügen Sie insbesondere selbst keine weiteren Zeichen (z.B. Zeilenumbrüche) hinzu!** Achten Sie auch darauf, dass Bytes, die keinem Zeichen entsprechen, trotzdem korrekt ausgegeben werden, z.B. Null-Bytes.

Aufgabe 9:

Im zweiten Teil dieser Aufgabe sollen Sie einen *einfachen* Stream-Socket Server implementieren, der das im RFC 865 beschriebene *Quote of the Day Protocol* implementiert. Es genügt, wenn der Server mit einem Client gleichzeitig kommunizieren kann. Der Server soll über die Kommandozeile den Port und den Dateinamen einer Textdatei übergeben bekommen und jedem Client zufällig ein Zeile aus dieser Datei zurückgeben. Sie können sich bei Ihrer Textdatei im Internet inspirieren lassen. [7].

[7] https://github.com/HashanP/qotd-server/blob/master/qotd.txt

Achten sie bei der Implementierung besonders auf die folgenden Punkte:

(a) Ihr Server soll prinzipiell **auf allen IP-Adressen** des Systems antworten (Stichwort: `AI_PASSIVE`).

(b) Wir legen fest, dass am Ende jeder Zeile der Eingabedatei das line feed (LF) (`\n`) Symbol steht. Dieses soll **nicht** mit übertragen werden. Genauso wenig sollen zusätzliche Null-Bytes übertragen werden. Sind spezielle Bytes in der Eingabedatei enthalten, sollen sie aber mitgeschickt werden. Sollte eine Datei nicht mit dem line feed (LF) (`\n`) Symbol enden, sollten alle Zeichen zwischen dem letzten Zeilenumbruch und dem Ende der Datei ignoriert werden.

(c) Als genereller Hinweis: Ihre Programme sollten auch außerhalb der Spezifikationen nicht unkontrolliert abstürzen. Stattdessen sollten sie einen entsprechenden Fehler zurückgeben. Achten Sie z.B. darauf, dass auch eine leere Datei übergeben werden könnte oder eine ohne line feed (LF) (`\n`) Symbol.

(d) Sie sollen in Ihrem Projekt weiterhin CMake benutzen. Das Projekt sollte nun mit der vorherigen Aufgabe zusammen folgende Struktur aufweisen:

```
Block2
+--- CMakeLists.txt
+--- client.c
+--- server.c
```

Der folgende Aufruf soll im Unterverzeichnis `build` eine ausführbare Datei mit dem Namen `client` und eine mit dem Namen `server` erstellen:

```
$ mkdir build; cd build
$ cmake .. && make
```

Erstellen Sie aus Ihren **beiden** Lösung ein tar.gz Archiv, z.B. so:

```
tar -czvf Block2.TXXGYY.tar.gz Block2
```

Laden Sie dieses Archiv bei ISIS bei der entsprechenden Abgabe hoch.

Vertiefungsaufgaben

Diese Aufgaben sind zu Ihrer eigenen Vertiefung in Hinblick auf die Klausurvorbereitung gedacht:

Aufgabe 10:

Beantworten Sie folgende Fragen rund um den TCP/IP Stack:

(a) Welche Layer gibt es im TCP/IP Modell?

(b) Angenommen Sie öffnen einen Browser und senden eine Web-Anfrage (HTTP) über TCP über IP über Ethernet. In welcher Reihenfolge werden die Header gesendet?

(c) Angenommen Sie haben zwei Browser gleichzeitig geöffnet und rufen vom gleichen Server gleichzeitig über HTTP das selbe Dokument ab. Wie unterscheidet der Server zwischen den beiden Anwendungen?

(d) Was bringt das *Hourglass Modell* zum Ausdruck?