Aufgabe 1: Sockets & Protocol Stack

Wenn nicht anders angegeben, verwenden Sie pro Frage etwa 2–4 Sätze.

1. Was ist das **Intranet**, **Internet**, was ist das **World Wide Web (WWW)**? Grenzen Sie die drei Begriffe klar voneinander ab.

2. Was bedeutet **Protokoll** im Kontext Internet; welche Protokolle werden verwendet?

3. Das Konzept **Protokoll Stack** ist ein Möglichkeit die Funktionsweise des Internets zu modellieren. Wir haben zwei Modelle in diesem Kontext kennengelernt; welche sind das, wie unterscheiden diese sich und welche Vor- bzw. Nachteile haben Sie jeweils?

   Bilden Sie die Realität genau ab?

4. IPv6 wurde von der Internet Engineering Task Force (IETF) bereits 1996 standardisiert[1]. Welche Hürden gab (und gibt) es bei der Einführung von IPv6, was macht dieses Protokoll anders als sein Vorgänger?

5. Sockets sind unter UNIX eine Möglichkeit, dem Programmierer Kommunikationsschnittstellen zwischen Prozessen – u. U. auf unterschiedlichen Rechnern – bereitzustellen. Sockets sind dabei Filedeskriptoren auf Dateien bzw. Inodes vom Typ `S_IFSOCK`. Das bedeutet, wir können mit den regulären Funktionen `write` bzw. `read` auf Sockets zugreifen und damit Daten „senden“ respektive „empfangen“.

   Es gibt aber auch die speziellen Funktionen `send` und `recv` die speziell für Sockets noch mehr Optionen bieten.

   Wie unterscheiden sich **Stream Sockets** und **Datagram Sockets**?

6. In jedem Fall muss einem Socket eine Adresse (`struct sockaddr`) zugeordnet werden, bevor es benutzt werden kann.

   Was für eine „Art“ Adresse das ist, hängt von dem konkreten Protokoll ab, das verwendet wird.

   Über welche Protokolle können wir mit Sockets kommunizieren und was für Adressen werden für diese jeweils benutzt?

[1] `http://www.ietf.org/rfc/rfc2460.txt`

Aufgabe 2: UNIX-Domain-Sockets

Wir haben bereits rudimentäre Inter-Prozess-Kommunikation mittels Signalen kennengelernt. Jetzt möchten wir tatsächlich Datenpakete zwischen einzelnen Prozessen auf unserem Rechner senden. Eine Möglichkeit dafür sind die Named Pipes bzw. FIFOs (`man 7 fifo`), geteilte Speicherbereiche zwischen Prozessen (`man 7 shm_overview`) aber auch bidirektionale Kommunikation über lokale UNIX-Domain-Sockets (`man 7 unix`). Für diese Aufgabe sollen Sie sich mit der letztgenannten Möglichkeit auseinandersetzen.

Implementieren Sie einen Server und einen Client, wobei der Client von der Standardeingabe Benutzereingaben einliest und an den Server schickt. Der Server soll jede eingehende Nachricht mittels `"OK"` an den Client bestätigen. Der Client soll die Bestätigung abwarten und dem Benutzer mitteilen, bevor er wieder auf Benutzereingaben wartet.

Verwenden Sie verbindungsorientierte Sockets (`SOCK_STREAM`).

Die beiden Programme sollen folgendermaßen aufgerufen werden:

```
$ ./server socketname
Waiting for client ...
$ ./client socketname
User input:
```

Server Der Server muss ein Socket erstellen (`socket()`) und dieses Socket an den übergebenen Dateinamen binden (`bind()`). Erst dadurch ist es möglich, dass ein Client sich mit dem Server verbinden kann. Der Server soll auf ankommende Verbindungen auf diesem Socket lauschen (`listen()`) und diese akzeptieren (`accept()`). Über das dadurch erhaltene zusätzliche Socket findet die eigentliche Kommunikation mit dem Client statt, beispielsweise über `recv()` und `send()`.

Wenn der Client die Nachricht `"QUIT"` schickt, soll die Verbindung zu ihm getrennt werden (`close()`). Danach soll wieder eine neue Verbindung eines neues Clients akzeptiert werden.

Client Auch der Client muss ein Socket erstellen und sich dann über die übergebene Datei zum Server verbinden (`connect()`). Danach wird mit dem Server kommuniziert und schließlich die Verbindung geschlossen.

Hinweis: Sie können bevor Sie das jeweils andere Programm schreiben, Ihr Programm mit dem Tool `nc` testen. Wenn Sie bereits den Server geschrieben haben, verbinden Sie sich mit `nc -U socketname` auf den Server. Wenn Sie bereits den Client geschrieben haben, erstellen Sie einen Server mit `nc -l -U socketname` und testen dann die Funktionsweise.

Als Ressource empfehlen wir den „Beej's Guide to UNIX Interprocess Communication“: `http://beej.us/guide/bgipc/`