

## Übungsblatt 3 – Einfache verteilte System mit Sockets und HTTP

### Aufgabe A – HTTP

Nachdem wir in der letzten Übung bereits erste Erfahrungen mit *HTTP* gemacht haben, wollen wir etwas tiefer in Richtung verteilte Systeme vorstoßen. Dazu nutzen wir weiterhin *HTTP*.

1. *HTTP* arbeitet mit Nachrichten im Request-Response-Schema. D.h. ein Client sendet Anfragen an den Server, der Server antwortet auf die eingehenden Anfragen.
  - a) Im Folgenden ist ein *HTTP*-Request abgebildet:

*<request-line>*

*<general-headers>*

*<request-headers>*

*<entity-headers>*

*<empty-line>*

*[<message-body>]*

*[<message-trailers>]*

GET /index.html HTTP/1.1	Request Line	HTTP Request
Date: Fri, 19 May 2006 21:12:55 GMT Connection: close	General Headers	
Host: www.myfavoriteamazingsite.com From: joebloe@somewebsitesomewhere.com Accept: text/html, text/plain User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)	Request Headers	
	Entity Headers	
	Message Body	

Erläutern sie den Aufbau solch einer *HTTP*-Nachricht.

- b) Im Folgenden ist eine *HTTP*-Response abgebildet.

`<status-line>`  
`<general-headers>`  
`<response-headers>`  
`<entity-headers>`  
`<empty-line>`  
`[<message-body>]`  
`[<message-trailers>]`

HTTP/1.1 200 OK	Status Line	HTTP Response
Date: Fri, 19 May 2006 21:12:58 GMT Connection: close	General Headers	
Server: Apache/1.3.27 Accept-Ranges: bytes	Response Headers	
Content-Type: text/html Content-Length: 170 Last-Modified: Wed, 17 May 2006 10:14:49 GMT	Entity Headers	
<html> <head> <title>Welcome to the Amazing Site!</title> </head> <body> <p>This site is under construction. Please come back later. Sorry!</p> </body> </html>	Message Body	

Erläutern sie den Aufbau der prototypischen *HTTP*-Nachricht.

2. *HTTP* ist gehört zu zustandslosen Protokollen. Jedoch kodiert der Server bei jeder Antwort die Art des Response.

*HTTP* hält generell fünf Typen von Status-Codes (Response-Codes) vor, die erste Ziffer gibt den Typ des Codes an. Recherchieren sie kurz, was folgenden Status-Codes bedeuten:

- a) *1yy*
- b) *2yy*
- c) *3yy*
- d) *4yy*
- e) *5yy*

## Aufgabe B – Scapy

In der kommenden Übung arbeiten wir unter anderem mit *Scapy*. Mithilfe von *Scapy* können einfach Pakete gebaut, analysiert und verändert werden. So auch *HTTP*-Pakete.

Der Vorteil ist, dass sie nicht die darunter liegenden Schichten abarbeiten müssen, dass erledigt *Scapy* für sie transparent.

1. Innerhalb *Scapys* können sie sich den Aufbau verschiedener Paketen anzeigen lassen. Geben hierfür einfach `load_layers(NAME)` ein.
2. Lesen sie den nachfolgenden Code und kommentieren sie sich, wie ein *HTTP*-Request und -Response umgesetzt werden sollten.

```
>>> load_layer("http")
>>> HTTPRequest().show()
###[ HTTP Request ]###
Method      = 'GET'
Path        = '/'
Http_Version= 'HTTP/1.1'
A_IM        = None
Accept      = None
Accept_Charset= None
Accept_Datetime= None
Accept-Encoding= None
Accept-Language= None

[...]
```

```
>>> HTTPResponse().show()
###[ HTTP Response ]###
Http_Version= 'HTTP/1.1'
Status_Code= '200'
Reason_Phrase= 'OK'
Accept_Patch43= None
Accept_Ranges= None

[...]
```

## Aufgabe C – Python

Im Folgenden ist ein einfacher Chat-Server in der Programmiersprache *Python* implementiert worden. Ihre Aufgabe ist es den Code an den markanten Stellen zu verstehen.

1. Lesen sie den Code zunächst einmal von oben nach unten. Markieren sie sich ggf. erste Funktionalitäten.
2. Ein `#` markiert den Beginn eines Kommentars. Fügen sie an den entsprechen Stellen Kommentare ein, die erklären, was der Code macht (D.h dort wo „`# COMPLETE ME`“ steht).

Kommentieren sie den Code des Chat-Servers!

```
#!/usr/local/bin/python3.8
import socket
import select
```

```
import threading
import sys

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
"""
the first argument AF_INET is the address domain of the socket. This is
used when we have an Internet
Domain

with any two hosts
The second argument is the type of socket. SOCK_STREAM means that data
or characters are read in a
continuous flow
"""
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

IP_address = "IP_ADDR"
Port = PORT
server.bind((IP_address, Port))
#binds the server to an entered IP address and at the specified port
number. The client must be aware
of these parameters

server.listen(100)
#listens for 100 active connections. This number can be increased as per
convenience

list_of_clients=[]

def clientthread(conn, addr):
    conn.send("Welcome to this chatroom!".encode())
    #sends a message to the client whose user object is conn
    while True:
        try:
            message = conn.recv(2048)
            if message:
                print("<" + addr[0] + "> " + message)
                message_to_send = "<" + addr[0] + "> " + message
                broadcast(message_to_send, conn)
                #prints the message and address of the user who just
                sent the
                message on
                the server
                terminal

            else:
                remove(conn)
        except:
            continue

def broadcast(message, connection):
    for clients in list_of_clients:
        if clients!=connection:
            try:
                clients.send(message)
            except:
```

```
        clients.close()
        remove(clients)

def remove(connection):
    if connection in list_of_clients:
        list_of_clients.remove(connection)

while True:
    conn, addr = server.accept()
    """
    Accepts a connection request and stores two parameters, conn which
    is a socket object for that
    user, and addr which contains
    the IP address of the client that just connected
    """
    list_of_clients.append(conn)
    print(addr[0] + " connected")
    #maintains a list of clients for ease of broadcasting a message to
    all available people in the
    chatroom

    #Prints the address of the person who just connected
    threading.Thread(target=clientthread(conn,addr))
    #creates and individual thread for every user that connects

conn.close()
server.close()
```

3. Analog zu letzten Aufgabe: Lesen und kommentieren sie den Client-Code.

```
#!/usr/local/bin/python3.8
# Python program to implement client side of chat room.
import socket
import select
import sys

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

IP_address = "IP_ADDR"
Port = PORT
server.connect((IP_address, Port))

while True:
    sockets_list = [sys.stdin, server]
    read_sockets, write_socket, error_socket = select.select(sockets_list
        , [], [])

    for socks in read_sockets:
        if socks == server:
            message = socks.recv(2048)
            print(message)
        else:
            message = sys.stdin.readline()
            server.send(message.encode())
            sys.stdout.write("<You>")
```

```
        sys.stdout.write(message)
        sys.stdout.flush()
server.close()
```