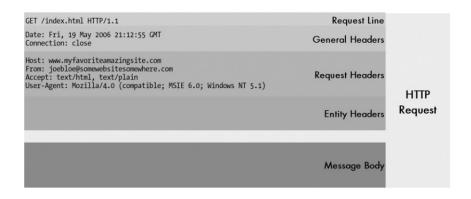
Übungsblatt 3 – Einfache verteilte System mit Sockets und HTTP

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* arbeiten mit Nachrichten in einem Request-Response-Schema. D.h. ein Client sendet Anfragen an den Server, der Server antwortet auf die eingehenden Anfragen.
 - a) Im Folgenden ist eine HTTP-Request abgebildet. Erläutern sie den Aufbau

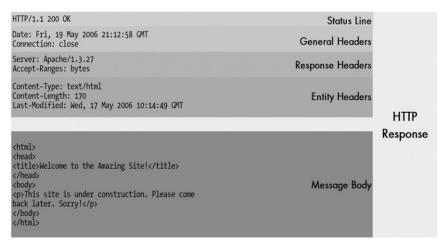
<request-line>
<general-headers>
<request-headers>
<entity-headers>
<empty-line>
[<message-body>]
[<message-trailers>]



solch einer HTTP-Nachricht.

b) Im Folgenden ist eine $HTTP ext{-}$ Response abgebildet.





Erläutern sie den Aufbau solch einer HTTP-Nachricht.

2. HTTP ist selbst zustandslos. Jedoch kodiert der Server bei den Antwort auf eine Anfrage die Art der Nachricht.

HTTP hat hierbei generell fünf Typen von Codes festgelegt, 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 Pakte gebaut, analysiert und verändert werden. So auch *HTTP*-Pakete.

- 1. Innherhalb *Scapy* können sie sich den Aufbau von verschiedene Paketen anzeigen lassen.
- 2. Lesen sie den nachfolgenden Code und kommentieren sie sich, wie ein HTTP-Request und Response umgesetzt werden sollen.

```
>>> load_layer("http")
>>> HTTPRequest().show()
###[ HTTP Request ]###
 Method = 'GET'
           = 1/1
 Path
 Http_Version= 'HTTP/1.1'
         = None
 A_IM
 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 programmiert.

- 1. Lesen sie den Code zunächst einmal von oben nach unten.
- 2. Ein # markiert den Beginn eines Kommentars. Fügen sie an den entsprechen Stellen kommentare ein, die erklären, was der Code macht (an den Stellen, wo "# COMPLETE ME" steht).

```
import socket # import socket lib -> IP+TCP/UDP
import select # import I/O wait/await etc.
import sys # import OS functionality
from thread import * # threading lib

"""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 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# COMPLETE ME
IP_address = "IP_SERVER"
# takes second argument from command prompt as port number
Port = int(PORT)
#COMPLETE ME -> What does 'bind' mean?
server.bind((IP_address, Port))
# Max 4 clients can connect to our server
server.listen(4)
# list of client, starting with 0
list_of_clients = []
#def for function start
def clientthread(conn, addr): # COMMENT ME: what does this function do?
    # COMMENT ME
   conn.send("Welcome to this chatroom!")
  # COMMENT ME
   while True:
            try:
                message = conn.recv(2048) # COMMENT ME
                if message:
                    # COMMENT ME
                    print ("<" + addr[0] + "> " + message)
                    # COMMENT ME
                    message_to_send = "<" + addr[0] + "> " + message #
                                                           COMMENT ME
                    broadcast(message_to_send, conn) # COMMENT ME
                else: # COMMENT ME
                    """message may have no content if the connection
                    is broken, in this case we remove the connection"""
                    remove(conn)
            except:
               continue
# COMMENT ME
def broadcast(message, connection):
   for clients in list_of_clients: # COMMENT ME
        if clients!=connection: # COMMENT ME
            try:
                clients.send(message) # COMMENT ME
```

```
clients.close() # COMMENT ME
                # if the link is broken, we remove the client
                remove(clients)
"""The following function simply removes the object
from the list that was created at the beginning of
the program"""
def remove(connection):
    if connection in list_of_clients:
        list_of_clients.remove(connection)
while True:
    """ \mbox{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"""
   conn, addr = server.accept()
   """Maintains a list of clients for ease of broadcasting
   a message to all available people in the chatroom"""
   list_of_clients.append(conn)
   # prints the address of the user that just connected
   print (addr[0] + " connected")
   # creates and individual thread for every user
    # that connects
   start_new_thread(clientthread,(conn,addr))
conn.close()
server.close()
```

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

```
# Python program to implement client side of chat room.
import socket
import select
import sys

# COMMENT ME
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# COMMENT ME
IP_address = "IP_CLIENT" # COMMENT ME
Port = int("PORT") # COMMENT ME
server.connect((IP_address, Port)) # COMMENT ME
while True:

# maintains a list of possible input streams
```

```
sockets_list = [sys.stdin, server]
    """ There are two possible input situations. Either the \,
    user wants to give manual input to send to other people,
    or the server is sending a message to be printed on the
    screen. Select returns from sockets_list, the stream that
    is reader for input. So for example, if the server wants
    to send a message, then the if condition will hold true % \left( t\right) =\left( t\right) \left( t\right) 
    below. If the user wants to send a message, the else
    condition will evaluate as true"""
    read_sockets,write_socket, error_socket = select.select(sockets_list
                                              ,[],[])
  # COMMENT ME
    for socks in read_sockets:
        if socks == server: # COMMENT ME
            message = socks.recv(2048) # COMMENT ME
            print (message)
        else:
            message = sys.stdin.readline() # COMMENT ME
            server.send(message) # COMMENT ME
            sys.stdout.write("<You>") # COMMENT ME
            sys.stdout.write(message) # COMMENT ME
            sys.stdout.flush() # COMMENT ME
server.close() # COMMENT ME
```